X Window Graphical Analysis Program for Rock Friction Data

Dept. of Electrical Engineering and Computer Science
Advanced Undergraduate Project

Project Supervisor: Prof. Chris Marone
Phone: 253-4352
Office: 54-724

Student: Lokman Alwi
Phone: 225-6196
MIT ID: 616-38-4267

Abstract

The objective of this project is to develop an X Window based graphical analysis program for analyzing rock friction data. The program is based on an existing text based program called Look. The new X Window version, named Xlook, has enhanced graphics capabilities. XView Toolkit is used to provide an Open Look compliant user interface.

MAY 1995

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Program Overview

This application is a graphical analysis program for studying rock friction laws and the mechanics of earthquakes and geological faulting. It is an X Window version of an existing text based program called "look"; hence the name Xlook. The original program uses Unix's standard input for its text input and has limited graphical capabilities based on Suntools.

The objective of porting this program is to give the *look* program a standard user interface as in regular X Window applications. XView Toolkit is used to provide the standard Open Look compliant user interface. Therefore, even though this program is closely modelled after the original text based program, the user interface is completely Open Look compliant.

1.2. Xlook's Workspace

Xlook's workspace can be divided into two main areas. The first part is a **main window** where inputs are typed and various messages are displayed. The second part are **plot windows** where plots are displayed. During the lifetime of Xlook, there can be only one main window. However, there can be up to ten windows for data plots. The plot windows can be created and destroyed at any time.

1.2.1. Main Window

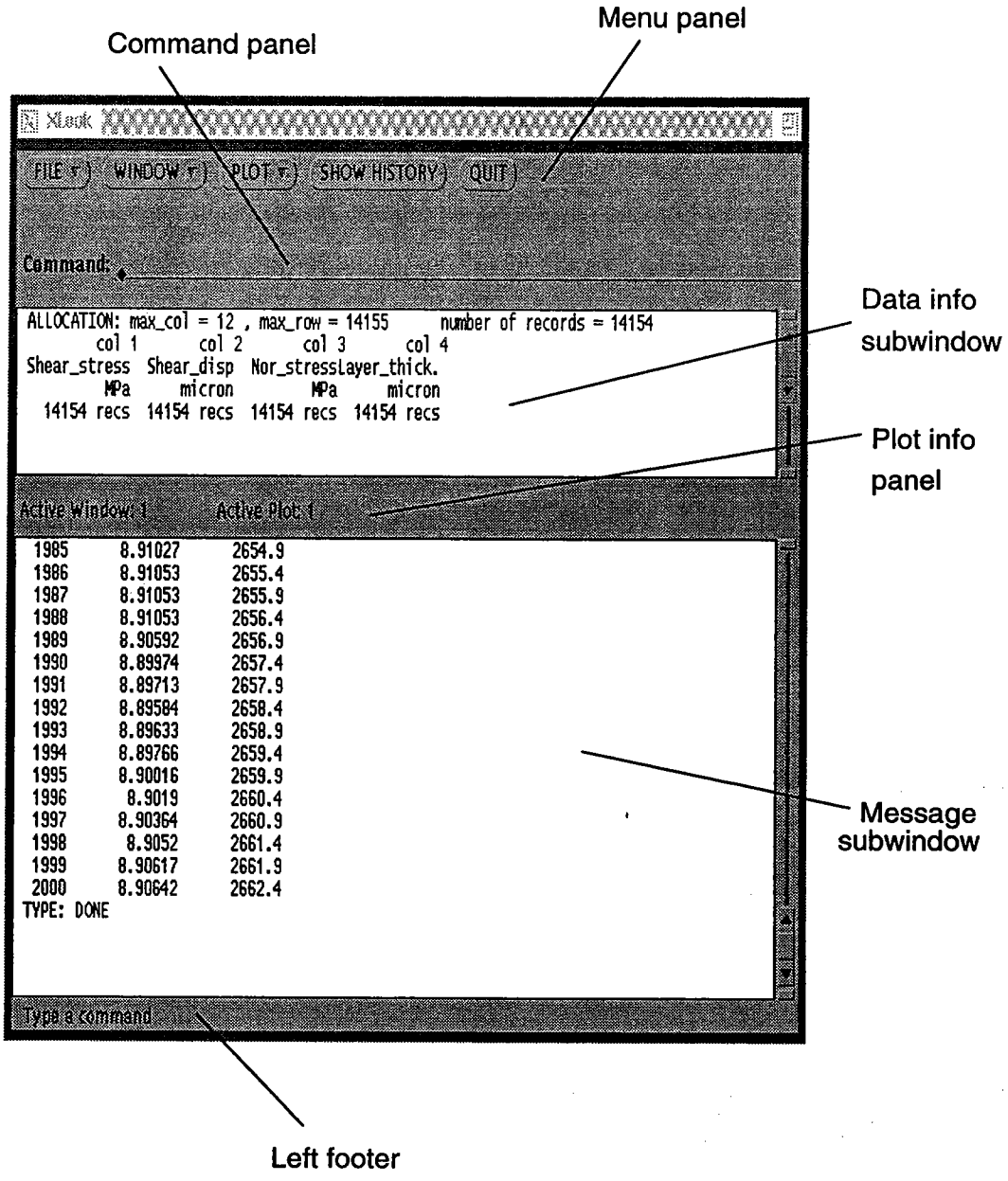

Figure 1. Main window.

The main window can be subdivided into six areas, as shown in figure 1. The topmost area is a panel for menu buttons. The area below the **menu panel** is a text panel for typing commands or input. This is called the

command panel. Data info subwindow, which is used to display the current state of the data, is right below the command panel. The fourth area is the **message subwindow**. This subwindow is used for displaying various kinds of messages to the user. For example, it is used for displaying error messages, command completions and input format for commands with many arguments. At the bottom of the main window is a panel used for displaying the input format for commands with a small number of arguments (as long as the message fits in this area, which, by default, is about 80 characters). In XView, this is called a **left footer**. Sandwiched between the data info subwindow and the message subwindow is a **plot info panel** for displaying the currently active plotting window and the currently active plot in that window.

1.2.2. Plot Windows

A plot window, shown in figure 2, contains mainly a **canvas** for plotting data. A user can click anywhere on the canvas to determine the exact coordinate of that point. Other operations possible on the canvas are determining the distance between two selected points, drawing a line between two points, drawing a vertical line or determining the row number of a point. The appropriate information is displayed in an **info panel** above the canvas. At the top of the window is a **menu panel** for the plot window.

Each plot window can have a maximum of ten plots at any particular time. The active plot is defined as the last plot being worked on by the user. Hence, any operation performed is acted upon the active plot.

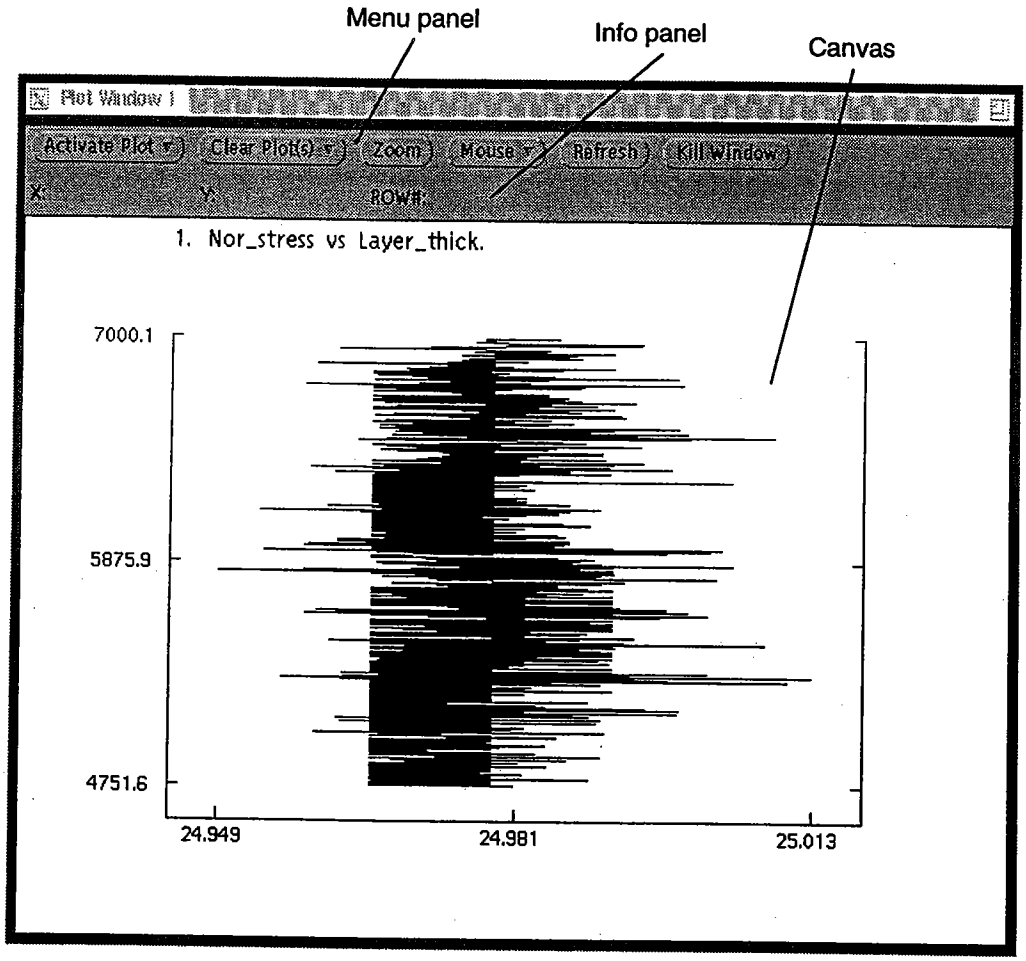

Figure 2. Plot window

1.3. Programming Approach

This project was done in three phases. In the first phase, I designed the user interface. Next, I programmed the main control structure and the user interface. Then, in the final phase, I added all *look* commands one by one.

In designing the user interface, I first studied the *look* program to determine what are the objects necessary for the user interface and what

type of graphics capabilities are required. In the second phase, I studied an XView program called *xvgr* and *ice* to learn about event driven programming. These programs were used as models for the main control structure of Xlook. For creating the user interface, I used the XView Programming Manual, XView Reference Manual and Xlib Programming Manual. The latter was used primarily as a reference for graphics related operations, which are not supported directly by the XView Toolkit. Once the underlying structure of Xlook was complete, adding *look* commands was less involving because most of the original code could be reused without major modifications.

To check the program's correctness, the output of each command was compared with the output of the same command executed in *look*. Since the script file handler was completed in the beginning of the third phase, script files were used extensively for these checks.

This project proved very helpful to me. First of all, I learned X Window programming, which is a very sought after experience in the industry. I also learned a lot about event driven programming, which is the programming model used by all graphics oriented environment such as Microsoft Windows and the Macintosh operating system. Finally, this project also taught me how to plan, manage and organize a big software project.

2. PROGRAM STRUCTURE

The original *look* program is a conventional sequential program even though it was written for Suntools. Since X Windows requires that all programs be event driven, the *look* program needs to be rewritten to follow the event driven paradigm that X Windows requires. This entails a major modification of the main control structure of the *look* program.

Xlook can be divided into two major parts: X Window related and non-X Window related parts. The non-X Window related parts are parts that are mostly copied or modified from *look;* the procedures for commands that do not require any graphics manipulations for example. Even though most of these codes are modified from *look*, they perform almost exactly the same as before. For example, in *look*, almost all the commands are handled as switches in main(). In Xlook, it is not possible to do this due to limitations enforced by XView. Each of the commands is now handled by at least one procedure that is called by another procedure that acts as a switcher. Therefore, even though the codes are separated into different procedures, most of the codes are still the same.

The parts that are X Window related include the procedures that create XView objects such as the main window, plot windows, menus and buttons. In XView, every object created needs a callback function that handles the actions to be done when the object is selected. For example, the callback function for the **Plotall** menu item will set up Xlook so that it is ready to accept the arguments for **plotall** command, just as if the user had typed "plotall" in the command panel.

Creating objects such as windows, menus and buttons can be programmed easily using the XView Toolkit (for a complete description of XView objects, please refer to the XView Reference Manual by O'Reilly and Associates). However, when there are many objects floating around on the desktop, they are not as easy to manipulate. Complex data structures are needed to keep track of and to manipulate the windows and the individual plots in the windows.

2.1. Xlook Data Structures

Xlook uses many data structures to maintain its current state and data. All the data structures and some of the global variables in *look* are still used unmodified in Xlook. New data structures and variables are needed, however, in order to handle the complexity of having multiple plots and windows.

2.1.1. Look Data Structures

Xlook uses all the data structures used in *look* for maintaining the data. Therefore, they will not be discussed in detail here. The data structures used by *look* are **channel, header, plot_sum, statistics** and **rs_parameters**.

channel is used for storing the general information about a column of data. It includes the name and unit of the column, the gain, the number of

data elements in the column and an optional comment field. Each column of data has an associated channel structure.

header is tagged to every plot. Among others, it has the title of the plot, the number of channels in it, a channel structure and the number of records.

plot_sum stores the summary of each plot. It includes the number of rows, the maximum and the minimum values of each column.

statistics keeps track of the mean, max, min, standard deviation and record number of a column of data. This structure is consulted by the plot commands.

rs_parameters is used by certain commands for storing the relevant parameters required by these commands. **scm** and **simplex**, for example, use this structure internally.

2.1.2. X Window Related Variables And Data Structures

Since Xlook supports multiple windows and plots, new data structures and variables are needed. The most important global variables are **action, active_window and active_plot**. These variables are used extensively throughout the program. **action** is a variable that keeps track of what Xlook is currently doing. When it is idle, action is set to MAIN. Then, when the user types **kill_window**, for example, **action** is set to KILL_WINDOW. By checking **action**, Xlook now knows that it has to prompt for the window

number the user wants to destroy. Once given, Xlook destroys the window, sets the **active_window** variable accordingly, sets **action** back to MAIN and waits for input again. **active_window** keeps track of the currently active window. **active_plot** is the plot number of the currently active plot in this window.

The structures maintained by Xlook to keep track of the plots and windows are **wininfo, plotarray** and **canvasinfo**. These data structures contain all the information about every window and every plot in Xlook.

windows_info consists of an array of ten integers, **windows**, whose elements are each mapped to a plot window. **windows** array is used to track the existence of plot windows. The indices of this array are the same as the window numbers they map to. The values in the array are either 0 or 1. A 0 means that the plot window indexed by this element does not exist. Conversely, a 1 means that a plot window indexed by this element exists. This one to one mapping works well because a unique number (from 1 to 10) is assigned to a window only during this window's lifetime. This number is freed when the window is destroyed so that it can be assigned to a new window.

```
typedef struct windows_info
{
 int windows[10];
 canvasinfo *canvases[10];
} wininfo;
```

windows_info also has an array of pointers to a **canvas_info** structure. **canvas_info** is the structure that actually contains all the

information about a particular window. Therefore, every plot window has a corresponding **canvas_info** data. The information in **canvas_info** includes pointers to the window's Canvas object, Xv Window object, Window object, number of plots in the window, the active plot in that window and various coordinates for the plotting area. It also has an array called **alive_plots** that works exactly like the **windows** array in **windows_info**. However, instead of mapping the elements to windows, it maps the elements to the plots in a window.

```
typedef struct canvas_info
{
 Canvas canvas;
 Xv_Window xvwin;
 Window win;
 plotarray *plots[10];
 int canvas_num;
 int active_plot;
 int total_plots;
 int alive_plots[10];
 int start_x;
 int start_y;
 int end_x;
 int end_y;
 int start_xaxis;
 int start_yaxis;
 int end_xaxis;
 end_yaxis;
} canvasinfo;
```

canvas_info also maintains an array of pointers to **plot_array**. **plot_array** is a structure that stores the information of a plot. It has two pointers to the arrays of x and y data for the plot. Other information stored in **plot_array** are the minimun and maximum x and y values, the number of

data points in the plot, the scale of x and y to the screen coordinates, the label type, the current mouse mode for this plot and various coordinates used by the mode.

```
typedef struct plot_array
{
  float *xarray;        /* x coords */
  float *yarray;        /* y coords */
  int nrows;
  int col_x;            /* x column */
  int col_y;            /* y column */
  int begin;            /* begin row */
  int end;              /* end row */
  float xmin;
  float xmax;
  float ymin;
  float ymax;
  float scale_x;
  float scale_y;
  int label_type;
  int mouse;       /* mode  */
  int x1;     /* points coords */
  int y1;
  int x2;
  int y2;
  int p1;     /* point 1 picked */
  int p2;     /* point 2 picked */
  int zp1;    /* zoom point 1 picked */
  int zp2;    /* zoom point 2 picked */
}plotarray;
```

When Xlook draws a plot in a window, it finds the attributes of the window in **canvasinfo** and uses the data in **plotarray** to plot the points.

2.2. Xlook Main Procedures

Xlook has three main procedures: xv_main_loop(), command_handler() and process_action().

2.2.1. xv_main_loop()

The main() procedure in Xlook() takes care of X Windows initialization. It creates the main window and all the objects in the main window such as the menus and command panel. The last line in main() is a call to xv_main_loop().

In X Window, all user actions are associated to certain events. When a user performs an action, a specific event is triggered. This event is stored in an event queue, which the application can later retrieve for processing. For example, when the user presses the left mouse button, an event is put into the queue. The event structure stores the x and y coordinate where the button was clicked, the button number and various other data. An X application must check the event queue from time to time, extract an event from the queue one by one and do the appropriate processing. All these must be done in an infinite loop. In a complex application, this can be very tedious and complicated.

XView hides all these details from the programmer by providing a nice interface. In XView, when an object is created, a callback function has to be associated to the object if it requires specific actions to be done when certain

events are triggered. Then, xv_main_loop() must be called at the end of main(). xv_main_loop will do all the queue checkings and event dispatchings automatically. This is a procedure defined in XView and is required in any application that uses XView.

2.2.2. command_handler()

When the user types something in the command panel, the text string is passed to command_handler() for processing. command_handler() first checks if the **action** variable equals MAIN (see X Window Related Variable and Data Structures for a description of the **action** variable). If not, this means that the user had previously typed a command and Xlook needs to parse the arguments. Therefore, it passes the string to process_action() which handles argument parsing and displays a new prompt if necessary. Otherwise, command_handler() checks the string if it matches any of the commands. If it does and the user had also given all the required arguments, command_handler() calls the procedure that handles the command and passes the string to it. The procedure will then execute the command if all the arguments are acceptable. When the procedure returns, command_handler() surrenders control back to xv_main_loop(). However, if insufficient arguments are given, command_handler() will set **action** to the appropriate value and return control to xv_main_loop(). Therefore, when the user next types something, the string would be passed to command_handler(), which then passes it to process_action() because **action** is not equal to MAIN.

2.2.3. process_action()

process_action() is always called by command_handler() whenever **action** is not equal to MAIN. In process_action(), **action** is compared to all the possible values and if it matches any of them, it does the appropriate processing. A command may require several calls to process_action() to get all the arguments it needs. Therefore, process_action prompts the user for the appropriate arguments needed by a command. This is repeated until it gets all the arguments it needs. When all the arguments have been given, process_action() calls the procedure that handles the command. Hence, its function is more or less like command_handler(). However, it bases its action on the **action** variable instead of the command.

Example:

To make this clear, consider the command **plotscale**. It requires a column for the x-axis, another column for the y-axis, the beginning and ending record number and the scale to be used. Let's assume that Xlook is now waiting for input. Therefore, **action** equals MAIN. Now, we type **plotscale** in the command panel. Somewhere in xv_main_loop(), it detects this and it calls the callback function for command panel which, in turn, calls command_handler() and passes the string "plotscale" that we just typed to it. Command_handler() finds a match to the command in one of its if-else blocks. Then, it prompts for the x and y column, sets **action** to PLOT_GET_XY and returns, which ultimately goes back to xv_main_loop().

Now, we see the prompt and type the column number for the x and y axes. xv_main_loop() detects this again and calls command_handler(), which then passes the string to process_action(). process_action() finds a match to PLOT_GET_XY. It stores the arguments to a global variable, prompts for the begin and end record number, sets action to PLOT_GET_BE and returns.

Next, we type the begin and end record number. Xlook goes through the same series of procedures again. This time, process_action() finds a match to PLOT_GET_BE. It concatenates the new arguments to the previous one, prompts for the scale, sets action to PLOT_GET_SCALE and returns again.

This time, we type the scale and Xlook follows the same path again. In process_action, it concatenates the scale to the previous arguments and passes the concatenated string, which contains the plotscale command and its arguments, to do_plot(). This procedure processes the arguments, sets up the plot, draws the plot, sets action back to MAIN and returns.

As we can see, command_handler() only handles the topmost level of all commands. It just parses the command and lets process_action() handle the arguments. Therefore, if a command requires 3 levels of arguments, as in the above example, process_action() will be called three times. The last call to process_action() will pass all the arguments to the procedure that handles the command, which will then execute the command using the arguments passed to it by process_action().

Command_handler() could be used as the callback function for command panel. However, I chose to have a separate callback function which

actually calls command_handler(). This is necessary because we also want to use command_handler() for processing inputs from a script file. If it were used as a callback function, it is not possible to call it from regular procedures because a callback function requires certain data types internal to XView as its arguments.

2.3. Xlook Commands Procedures

For each Xlook command there is always a corresponding procedure, normally called do_<command>(), that actually executes the command using the arguments passed to it. This procedure is always called by command_handler() or process_action(). In some cases, it is called directly by a callback function if its arguments are trivial. For example, the procedure to quit Xlook, quit(), can be called by command_handler() or the callback function of the Quit button in main window.

2.4. XView Objects Callback Functions

Every XView object that allows a user to interact with it requires a callback function that xv_main_loop will call when the user actually acts on the object. For example, when the user selects a menu item, the callback function for that menu item will be called by xv_main_loop().

There are many objects in Xlook that have callback functions. Among these are all the menu items, the buttons, the canvases and the command panel. Most of the menu items and buttons have simple callback functions. These callback procedures usually just call the appropriate procedures directly, bypassing command_handler(). Simple callback functions are not discussed here. The XView Reference Manual already treats this subject in great detail. However, the canvases and the Clear Plot menus in the plotting windows are more complicated and worthy of some discussion. Among these are the canvas handler, the Select Plot menu item and the Activate Plot menu item.

2.4.1. Canvas Callback Function

The canvases are interested in mouse button clicks. The callback function for the canvases is event_proc(). It first checks the current mode of the plot and then executes the appropriate handler for each of the buttons. Some of the variables in the plot data structure (plotarray) are sets to store the current state of each canvas. Therefore, it is possible, for instance, to be in the zoom mode in one window and in distance mode in another.

Besides handling mouse events, event_proc() also handles window resizing. Whenever a plotting window is resized, the canvas and the plots inside it need to be resized too. Therefore, when Xlook receives a resize event, it gets the new window size and rescales the canvas and the plots accordingly.

2.4.2. Menu Callback Function

In XView, a callback function is attached to each menu item, instead of to the menu object itself, when it is created. This is adequate for most cases. However, to implement dynamically bounded items, i.e., items that only show up in the menu when certain conditions are met, this simple implementation does not work. In Xlook, except for the Select Plot and Activate Plot menus, all menus use the simple implementation. The extra complexity of handling the other two menus are described below.

The first time the Select Plot menu item is selected, a subwindow is created with ten selectable items in it. As explained previously, a subwindow is used, instead of a regular menu, because XView does not support dynamically bounded non-exclusive menu items. Then, each item is bounded to the currently existing plots in the window. Those without corresponding items are greyed out and labeled as No plot. This subwindow is shared by all the windows however. Therefore, when a Select Plot menu in another window is selected while one is already displayed by another window, all the items in this subwindow are remapped to the plots in the latest window which calls it. If the user wants to select a plot from the previous window, he or she can just set that window active. The subwindow will then be relabelled for this window.

When the user selects the Done button, Xlook checks all the items. Checked items will then have '0' in the alive_plot array in that window's canvasinfo. Then, Xlook just clears the window and redraws the plots that are not checked (still have '1' in alive_plot).

This subwindow is shared by all the windows and not destroyed until Xlook quits because this menu may be called many times. It is more efficient to change the labels in the subwindow and remap it to the screen than to create a new subwindow for every plotting window whenever the Select Plot menu item is selected and destroy this subwindow upon completion.

Unlike the Select Plot menu, Activate Plot menu uses a regular menu item because its items are exclusives, which means that only one can be selected at any time. However, like the Select Plot menu, Activate Plot menu is also dynamically bounded. Therefore, this menu only shows the plots that actually exist in the window as its menu items. To implement this, a callback function is attached to the menu instead of the menu items. This callback function finds all the plots in this window (from alive_plot array in canvasinfo) and creates a menu item for each alive plot. All the items in this menu use the same callback function. To determine the number of the selected plot, Xlook compares the items one by one with the item passed to it. Then, the selected plot is set to active.

3. USING XLOOK

The *look* program has many commands for executing various functions related to rock friction analysis. Almost all of these commands are duplicated in Xlook. Xlook tries to retain the format of the original commands as much as possible. Therefore, most of the original *look* commands are unchanged.

Unfortunately, to support the new graphical environment, some of the commands are changed and new commands are added. Conversely, there are also commands that are removed since they are not appropriate in a graphical environment. For a list of all the commands and the arguments needed for each command, refer to appendix A. All changes to the original command set are noted in the appendix.

3.1. Using The Command Panel

Xlook has more than one hundred commands. Since it is not possible to assign a button or a menu item for each of these commands, the main method of user input is still by typing.

In Xlook, a user types commands in the command panel. If a command requires arguments and the user did not give the arguments along with the command, Xlook will prompt the user for the arguments. Arguments are also typed in the command panel. If the command requires many arguments, they will be displayed in the message subwindow. Otherwise, it is displayed in the left footer (the left footer is only used for displaying short messages because the physical space for it is limited.)

If a user wants to cancel the last command issued, he or she can press the 'return' key when asked for the arguments. This will give an incorrect number of arguments to the command, which will cause an abort. Note, however, that some commands are executed in multiple stages. Therefore, it

is possible for a command to abort halfway through its execution. This may leave the data in an unpredictable state.

3.2. Using Xlook Menus

Some of the commonly used Xlook commands are duplicated as menu items in either the main window or the plot windows. In the main window, there is a **File menu, Window menu and Plot menu**. There are also a **Show History** and **Quit** buttons on the menu panel. On the plot window, there is an **Activate Plot menu, Clear Plot menu, Zoom button, Mouse menu, Refresh button** and **Kill Window button**.

The **File** menu on the main window has **Read File** and **Write File** menu items. The **Window** menu has **New Window, Kill Window** and **Activate Window** menu items. All the plot commands are duplicated as menu items in the Plot menu. All these menu items perform exactly as if the user had typed the equivalent command in the command panel. Therefore, Xlook will prompt for the appropriate arguments when an item is selected. **Show History** will open up a window with all the commands typed so far as selectable items. If an item is selected, the text will be copied to the command panel. This can be useful for re-entering long commands. The **Quit** button is used to quit Xlook.

The **Kill** button in a plot window is to destroy the window. This is equivalent to selecting the **Kill Window** menu item in the main window and

giving it the plotting window's number as the argument. The **Zoom** button sets Xlook in zoom mode. The **Activate Plot** menu has all the plots in the window as its menu items. These items are dynamically bounded. Therefore, this menu is rebuilt whenever it is selected. The **Clear Plot** menu consists of a **Clear All** and **Select Plot** items. **Clear Plot** will clear all the plots in the window. Select Plot will display a subwindow with ten selectable items in it. It is possible to select more than one of these items. The items are labelled with the labels of the plots in the window. Therefore, the items without corresponding plots are greyed out and not selectable. This menu uses a subwindow because XView does not support non-exclusive dynamically bounded menus (**Activate Plot** menu is exclusive because it only allows one item to be selected). Selecting the **Done** button in this subwindow will clear all the selected plots and unmap the subwindow. Selecting the **Cancel** button ignores the selection and unmaps the subwindow.

3.3. Manipulating Plot Windows

Three commands related to plotting window manipulations are included in Xlook. They are **new_window, kill_window <win_num>** and **activate_window <win_num>. new_window** will create a new plotting window. The number is automatically generated by Xlook. A window will not be created If there are already ten plot windows opened. The user will be notified of this failure to create a new window. **kill_window** destroys the specified plot window and the data associated to the plots in that window. **activate_window** sets the active window to the plot window specified. Any

plot command issued thereafter will be directed to this active window. It is also possible to activate a window by clicking anywhere in the canvas of the desired window. The active plot in this window will automatically be the currently active plot.

The plot windows and the main window can be resized using regular resizing operation provided by a window manager. If a plot window is resized, all the plots in that window are rescaled to fit in the new window size. If the main window is resized, all the subwindows in it are also rescaled.

3.4. Manipulating Plots

Plots are created whenever any of the plot commands are issued either through the command panel or the Plot menu in the main window. All the plot commands in look are present in Xlook. Xlook also has a new plot command, **pa**, which works exactly like **plotauto**, but uses a different labeling scheme.

Besides the plot commands, two new commands are added in Xlook. They are **activate_plot <plot_num>** and **kill_plot <plot_num>**. **activate_plot** sets <plot_num> as the active plot in the currently active window while **kill_plot** clears <plot_num> in the currently active window. **kill_plot** also removes the data associated to the plot from memory.

There are other manipulations that can be done on the plots. For example, a user can zoom on a particular area of a plot, draw a vertical line on a particular position on a plot, draw a straight line connecting two points on a plot and find the distance between two points on a plot. In order to do any of these, the user has to first enter the appropriate mode so that the mouse buttons can be bounded to the appropriate functions. The possible modes are **Normal, Zoom, Line, Vertical Line** and **Distance**. Note that these modes are only active in the respective plot. They do not affect any other plots in the window, nor do they affect the main window. This means that the user can still issue any Xlook command in the main window or be in a different mode on the other plots.

3.4.1. Normal Mode

By default, the user is in the Normal mode. In this mode, whenever the user clicks the left or middle mouse button, the point that is actually selected is the data point closest to the actual point on the screen. On the other hand, if the right button is clicked, the point selected is the actual point on the screen. A crosshair is drawn on the point and the x and y corrdinates are displayed in the data info panel when any of the buttons are clicked. When the left button is clicked, the row number of the data is also displayed in the data info panel. The middle button does the same thing as the left button. However, in addition to that, it also duplicates the information displayed the data info panel in the message window. The right button displays the information in the info panel, the plot and the message window.

3.4.2. Zoom Mode

When the Zoom button in a plotting window is pressed, Xlook enters the zoom mode. In this mode, the left button is used for selecting the first limit for the zooming area and the middle button is for the second limit. The point with a smaller x value is automatically selected as the lower limit. It is possible to press either of the buttons as many times as the user wishes. However, only the last set of limits selected will be used as the limits for the zooming area. When the right button is pressed, Xlook draws the data points bounded by the two limits, leaves the zoom mode and enters the normal mode.

3.4.3. Line Mode

The mouse enters the line plot mode when the user selects the Line Plot item in the Mouse menu of a plot window. In this mode, the left button is for selecting the starting point for the line to be drawn. Once the first point is selected, a rubber-band line that follows the mouse movement is drawn. This line connects the first point to the current point pointed by the mouse. When the middle button is pressed, the second point is selected and a permanent line is drawn connecting the first point to the second point. Xlook leaves this mode when the right button is pressed.

3.4.4. Vertical Line Mode

To draw vertical lines on the plot, the user must select the Vertical Line item in the Mouse menu of a plot window. Once in this mode, a vertical line is drawn whenever the left button is pressed. The line is drawn using the x coordinate of the point selected by the left button. Pressing the middle button does nothing. The right button is used for exiting this mode and enters the Normal mode.

3.4.5. Distance Mode

To determine the distance between two points on a plot, the user must enter the distance mode first. To do this, select the Distance item in the Mouse menu of a plotting window. The left button selects the first point and the middle button selects the second point. Once the second point is selected, the x distance, the y distance and the hypotenus distance are displayed in the data info panel of that plot window. The right button is used for leaving this mode, which automatically puts the user in the Normal mode.

3.5. Using A Script File

Like look, Xlook allows the use of script files. A script file contains Xlook commands complete with their arguments. These commands are executed as

if they were typed directly in the command panel by the user. This is very useful for executing a command sequence that must repeated many times.

To execute a script file, type "doit <script filename>" in the command panel. Xlook allows up to ten script files opened at one time. Therefore, care must be taken when nesting script files, i.e., calling a script file from another script file. Note, that there are currently two commands that use a script file internally. These commands are r_trend_o and interpolater. Do not forget to add this to the total number of script files opened when using these commands in a script file.

A script file must begin with a "begin" string on a line by itself and end with a "end" string. Any command can be put in between the begin and end strings. The arguments can all be put on the same line with the command itself or on different lines just like in the interactive mode.

Comments can be included in the script file. However, every line must begin with a '#' character. Everything after the '#' character to the end of the line will be ignored. Unrecognized commands will also be ignored by Xlook.

It is important that the sequence of commands in the script file does not generate any error because even if an error occurs, the script file will continue to be executed. Therefore, it may produce a different result than expected. Test the script file before using it. The following is an example of a script file.

Example:

```
begin
#this is a comment
read u27gs
plotall 1, 3
# done
end
```

4. ADDING FUNCTIONS TO XLOOK

It is possible to add more commands to perform new functions in Xlook. This section gives a general guideline on how to add commands and XView objects to Xlook.

Adding a command that does not require or manipulate graphics is simple. Section 4.1, "Adding A New Command", describes how to add a command that does not use graphics. The only procedures that must be modified are command_handler() and process_action() (both are in event.c). An appropriate procedure that actually performs the function must also be added. Currently, there are two files where functions reside. They are cmds.c and cmds1.c. cmds.c have most of the Xlook procedures that implement Xlook functions. Special functions, such as reading a file, writing a file, the doit procedure (script file handler) and the plotting procedures are in cmds1.c. global.h file, which contains the constant definitions needed in

command_handler() and process_action(), should also be modified as necessary.

The section 'Adding XView Objects' describes how to add simple objects in Xlook. However, for a complete description on how to create XView objects, refer to the XView Programming Manual. To add graphics related commands, please refer to the Xlib Programming Manual. Graphics programming is not covered in the XView manuals because XView does not support graphics directly. Graphics programming in XView must be done using Xlib.

4.1. Adding A New Command

To add a new command to Xlook, first determine what are the inputs required. If possible, get all the inputs before the procedure is actually called. This will make the procedure very simple in terms of input handling. Next, determine how many levels of inputs are required. It is possible to just have one level even if the command requires many inputs. However, it is better to split the inputs into different levels so that they are grouped logically. For example, if a function requires a column for x-axis, another one for y-axis and an integer, it is better to ask for the x and y-axis columns together and ask for the integer in the next level. Therefore, this function has two levels of input.

Next, add a checking block in command_handler. This block must have a 'if (strncmp(cmd, "xxx", n) == 0)' clause where "xxx" is the name of this new command or function and n is the length of this command. See the example below.

After this, check if the user actually gives all the input. If this is true, pass the command directly to the procedure that actually performs this command (if possible, try to name the procedure do_xxx() to make it consistent with the names of Xlook procedures that already exist.) If this is not true, display the prompt and explain what the required inputs are, either in the left footer or in the message window. Then, set **action** to a variable defined in global.h. This takes care of the first level of input.

If the command requires more than one level of input, modify process_action() next. First, add the variable name as one of the switch cases. In this block, store the command string and the argument passed to process_action in tmp_cmd. Then, set command prompt and left footer to ask for the next inputs, set **action** to another variable and define this in global.h again. Continue this process for every level of input. However, the last level must pass tmp_cmd to the procedure that performs this function. By now, tmp_cmd should already have the complete arguments and the command name.

Example:

The following is an example of how to add a command 'add' to Xlook. This command adds column1 and column2 and put the new values in

column3. The format is 'add col1 col2 col3'. **nargs**, the number of arguments, is counted at the top of command_handler() and process_action().

in command_handler():

```
...
else if (strncmp(cmd, "add", 3) == 0)
{
 if (nargs == 3)  /* got the complete arguments */
      do_add(arg);
 else /* only the command was typed */
 {
      set_left_footer("Type the columns to add");
      set_cmd_prompt("Col1 Col2: ");
      action = ADD;
 }
}
...
```

in process_action():

```
switch...
 ...
 case ADD:
      /* handles first level input */
      sprintf(tmp_cmd, "add %s ", arg);
      set_left_footer("Type the column to add to");
      set_cmd_prompt("Col3: ");
      action = ADD1;
      break;
 case ADD1:
      /* handles second level input and call do_add with
the                arguments */
      strcat(tmp_cmd, arg);
      do_add(tmp_cmd);
      break;
 ...
 ...
```

Then define ADD and ADD1 in global.h. Use numbers that are not already used.

```
...
```

```
#define ADD (x)
#define ADD1 (y)
...
```

Finally, write the procedure that implements the new command. It does not have not be in cmds.c. However, since there are many variables defined in cmds.c, it may be easier to have all the functions in this file.

The procedure should check for the correct number of arguments. But first, it must call nocom() which removes all commas from the input. This is necessary because functions expect arguments to be separated by spaces instead of commas. nocom() replaces commas with spaces in the input string. Whenever this procedure wants to return prematurely because of not enough arguments or for any other reason, **action** must be set to MAIN and the command prompt set appropriately.

There are many global variables in cmds.c used by most of the procedures. Please reuse these variables as much as possible. The global variables are necessary because many of the functions require several procedures. Therefore, the variables must be global so that the other procedures for the same function can use them.

The procedure must not request any input because only xv_main_loop() can do this. The simplest way is to ask for all the inputs before this procedure is even called. However, if it is not possible, then the only way is to split this procedure into several parts. Each part would relinquish control back to xv_main_loop. This is done by setting action to another variable (instead of MAIN) and returns. process_action() should also be changed accordingly (but

leave out command_handler()), as described above. Now, if the user types something, the string will be passed to the correct switch case in process_action().

The above modifications should be sufficient to support a new command in Xlook. However, to make it more robust and fool proof, there are more checks done by command_handler() and process_action(). For example, most of the commands actually allow the user to give more arguments than is required. The arguments that are not needed are not passed to process_action. Instead, they are removed from the string and placed on the command panel so that the user does not have to retype them when they are prompted for in the next level. This is a rather complicated mechanism. To get into the details, please examine the source code.

4.2. Adding XView Objects

To bind a new command to a button or a menu item, first decide in which window this object will reside. If it is to be in the main window, modify main() in main.c to add the codes that create this object. Else, put the codes in canvas.c. If the object requires a callback function, put this function in main.c. The following is an example on how to have a button in the main window that calls the **add** command.

First, create the button object in the menu panel. Put this code in main.c. Object placement are ordered according to the order the objects are

created. Assume we want the button to be on the left of the quit button. So, we put the code right before the quit button is declared. Do not forget to declare the callback function, which in this case is add_proc(), in main().

```
/* callback function declaration */
void add_proc();

/* create button object for the ADD button */
(void) xv_create(main_frame_menu_panel, PANEL_BUTTON,
            PANEL_LABEL_STRING, "ADD",
            PANEL_NOTIFY_PROC, add_proc,
            NULL);
```

Then, define the callback function, which acts exactly like the if-else block in command_handler(). This procedure asks for the arguments, sets action to ADD and returns control to xv_main_loop(). When the user gives the arguments, process_action() will be called automatically. Put the callback function in main.c since that is where callback functions reside.

```
void add_proc(item, event)
 Panel_item item;
 Event *event;
{
 set_left_footer("Type the columns to add");
 set_cmd_prompt("Col1 Col2: ");
 action = ADD;
}
```

Many of the procedures that manipulate the windows and the plots are separated from their callback functions. This is to allow the procedures to be called from process-action or command_handler() instead of the buttons or menus in Xlook. If everything is done by the callback function, then a separate procedure must be defined if the same function is to have a

command defined in Xlook because every callback function has arguments predefined by XView. Most of the arguments are XView structures or objects which are not easily replicated manually. Hence, we eventually have two separate procedures that do the same exact function. Having the callback function calls a separate procedure allows the procedure to be shared by the callback function and process_action() or command_handler().

For example, the **Clear All** menu item in a plot window has a callback function clear_win_proc(). This procedure actually calls clr_all(), which is also called by process_action when the **clear_win** command is issued by the user. kill_win_proc() removes all the data of the plots in that window and then clears the plots. The clr_all() procedure may be used by other procedures also. For instance, if we want to modify the zoom command to clear the window of all plots before drawing the zoomed area, we could call clr_all() to do just that.

REFERENCES

1. Heller, Dan [1991]. *XView Programming Manual,* O'Reilly & Associates, Inc.

2. Van Raalte, Thomas [1991]. *XView Reference Manual,* O'Reilly & Associates, Inc.

3. Nye, Adrian [1992]. *Xlib Programming Manual,* O'Reilly & Associates, Inc.

4. Jones, Oliver [1989]. *Introduction to the X Window System,* Prentice-Hall, Inc.

APPENDICES

Appendix A: List of commands

The following is the list of available commands in Xlook. A line preceded by '*' is a comment.

- acos col, new_col, func
- activate_plot plot_num
  * New command
- activate_window window_num
  * New command
- all y/n
- append filename
- asin col, new_col, func
- atan col, new_col, func
  * func: (one of) sin, cos, tan, asin, acos, atan
- cgt del_h, h_initial, disp_col, new_col
- chisqr col, new_col
- cm model_disp_col, disp_col, mu_col, vs_row, end_row, model_mu_col, stiffness, Sigma_n, v_initial, v_final, mu_initial, mu_final, a, b1, dc1, dc2
- col_power power_col, col, new_col
- comment col, text
- compress col, new_col
- cos col, new_col, func

- cs — axial_disp_col, layer_thick_col, output_col, first, last
- curv — col, new_col, dx
- decimat — col, new_col, increment, first, last
- decimat_r
- deriv — x_col, y_col, new_col, first, last
- doit — filename
- ec — axial_disp_col, shear_stress_col, new_col, first, last, (e/l)
- end — * Removed
- er_bar — * Removed
- erase — * Removed
- ev — * Use the zoom button in plot window
- examin — filename
- exp — col, new_col
- ExpLin — col, new_col
- findint — col, first, last
- genexp — col, new_col
- gensin — col, new_col
- getaschead — filename
- head — filename/s
- interpolate — x_col, y_col, new_xcol, new_ycol, first, last, dx_inc/default
- interpolater — x_col, y_col, new_xcol, new_ycol, first, last, dx_inc/default
- kill_window — window_num
  * New command
- kill_plot — plot_num
  * New command
- lineplot — * Removed
  * Use the mouse menu in plot window

- ln col, new_col
- log col, new_col
- math x_col, opr, y_col, type, new_col
  * opr: (one of) +, -, *, /
- math_int x_col, opr, y_col, type, new_col, first, last
- median_smooth col, new_col, first, last, window_size
- mem x_col, y_col, freq_col, power_col, first, last, #freqs, #poles, (l/n), (w/n)
- metafile * Removed
- mouse/mp/mousemu/distance * Removed
  * Use the mouse menu in plot window
- name col, new_name, unit
- new_window * New command
- normal col, new_col
- offset col1, rec1, col2, rec2
- offset_int col, rec1, rec2
- pa x_axis, y_axis, first, last
  * New command
- pdf col, new_col_prob, new_col_bin, #bins, max, min
- pdfauto col, new_col_prob, new_col_bin, #bins
- peak col, new_col,
- plotall x_axis, y_axis
- plotauto x_axis, y_axis, first, last
- plotlog x_axis, y_axis, first, last
- plotover x_axis, y_axis
- plotsame x_axis, y_axis, first, last
- plotscale x_axis, y_axis, first, last, x_max, x_min, y_max, y_min

- plotsr x_axis, y_axis
- Poly4 col, new_col
- polyfit x_col, y_col, new_col, order, first, last, y/n, row/0
  * Give 0 as the last argument if previous argument is n, else give the row number.
- polyfit_i x_col, y_col, new_col, order, first, last, dx/default, row/0
  * Give 0 as the last argument if previous argument is default, else give the row number.
- positive col, new_col
- power power, col, new_col
- Power1 col, new_col
- Power2 col, new_col
- qi disp_col, mu_col, mu_fit_col, first_row, vs_row, last_row, weight_row, lin_term, conv_tol, lambda, wc, stiff, v_initial, v_final, mu_initial, a, bl, dc1, b2, dc2
- quit
- r_col col
- r_mean col, new_col
- r_row col, first, last
- r_row_col col
- r_spike col, row
- r_trend x/y, input, x_col, y_col, new_col, slope, y_int, first, last
  x/y, comp, x_col, y_col, new_col, first, last
- r_trend_o x/y, input, x_col, y_col, new_col, slope, y_int, first, last
  x/y, comp, x_col, y_col, new_col, first, last
- raster * Removed
- rclow col, new_col

- rcph col, new_col
- read filename
- recip col, new_col
- rgt del_h, disp_col, gouge_thick_col, new_col
- rsm disp_col, mu_col, vs_row, end_row, model_mu_col, stiffness, Sigma_n, v_initial, v_final, mu_initial, mu_final, a, b1, dc1, dc2
- scchisqr col, new_col
- scm disp_col, mu_col, vs_row, last_row, mu_fit_col, stiff, sig_n, v_initial, v_final, mu_initial, mu_final, a, b1, dc1, dc2
- set_path pathname
- simplex func, first, last, x_col
  *func: (one of) Power1, Power2, normal, chisqr, scchisqr, genexp, Poly4, gensin, Explin, rclow, rcph
- sin col, new_col, func
- slope x_col, y_col, new_col, first, last, window_size
  *Used to be called o_slope
  *Original slope command is removed
- slope_diff * Removed
  * Use the mouse menu in plot window
- smooth col, new_col, first, last, window_size
- sort/order col, first, last
- stat col, first, last
- stat_a col
- stdasc filename
- summation col, new_col
- tan col, new_col, func

- tasc i/f filename
- trend x_col, y_col, first, last
- trend_a x_col, y_col
- trig col, new_col, func
- type first_row, last_row, first_col, last_col, filename/screen
- typeall filename/screen
- vc vol_strain, shear_stress_col, new_col, first, last, comp
- verify * Removed
- write filename
- z_max col, new_col
- z_min col_new_col
- zero col, rec
- zero_all rec

Appendix B: Makefile for Xlook

```
SOURCES = main.c event.c canvas.c menus.c drawwin.c \
      cmds.c filtersm.c nrutil.c sort.c special.c \
      array.c median.c polyfit.c func.c tpulsel.c \
      perrfl.c mem.c fq.c look_funcs.c lookio.c \
      simplexl.c strcmd.c cmds1.c mouse.c qi_look.c \
      notices.c messages.c

OBJECTS = main.o event.o canvas.o menus.o drawwin.o \
      cmds.o filtersm.o nrutil.o sort.o special.o \
      array.o median.o polyfit.o func.o tpulsel.o \
      perrfl.o mem.o fq.o look_funcs.o lookio.o \
      simplexl.o strcmd.o cmds1.o mouse.o qi_look.o \
      notices.o messages.o

LIBS =      -lX11 -lolgx -lxview -lm
CFLAGS =
LDFLAGS =

debug := CFLAGS= -g

xlook: $(OBJECTS)
      cc -o $@ $(OBJECTS) $(CFLAGS) $(LIBS)
```

Appendix C: Source code

This appendix contains the complete source code required to build Xlook. The first part lists all the header files and the second part lists all the source files. The files are organized in alphabetical order.

```
#include        <stdio.h>
#include  <math.h>
#include        <string.h>

#define TRUE  1
#define  FALSE  0
#define  MAX_COL  17
#define  MAX_ROW  100000
#define  BELL  '\007'
#define  ALPHA  1.0  /*  reflection  factor  */
#define  BETA  0.5  /*  contraction  factor  */
#define  GAMMA  2.0  /*  expansion  factor  */
#define  MAX_PARAM  10  /*  max  #  of  parameters  to  be  optimised  */
#define  TWOPI  6.283185308
#define  PI  3.141592654
#define  TRUE  1
#define  FALSE  0
#define  SMALL  1e-10
#define  SQR(a)  (a*a)
#define  WELCH_WINDOW(j,a,b)  (1.0-SQR((((j)-1)-(a))*(b)))  /*see  p  445  of  numerical  rec
s  in  c*/
#define  HALF  0.5000
#define  ROUND(A)  (  (  (A-floor(A)  )  >  HALF  )  ?  ceil(A)  :  floor(A)  )

/*  §§§§§§§§§§§§§  GLOBAL  DECLARATIONS  §§§§§§§§§§§§§  */

/*-----------------------------------------------------------------*/
struct  channel
{
  char  name[13];
  char  units[13];
  float gain;
  char  comment[50];
  int  nelem;
};

struct  header
{
  char  title[20];
  int   nchan;
  int  nrec;
  int   swp;
  float dtime;
  struct        channel ch[MAX_COL];
  float extra[5];
};
struct  header  head  ;

/*-----------------------------------------------------------------*/
struct  plot_sum
{
  int  nrow[MAX_COL]  ;
  float  max[MAX_COL]  ;
  float  min[MAX_COL]  ;
};
struct  plot_sum  plot_info  ;

struct  statistics
{
  int  rec  ;
  float  mean  ;
  float  max  ;
  float  min  ;
  float  stddev  ;
```



```
};
struct  statistics  col_stat  ;

struct  rs_parameters
{
  char  law;                    /*d=Dieterich,  r=Runia,  j=Rice,  p=Perrin*/
  int   one_sv_flag;            /*true  if  doing  a  1sv  case*/
  int   op_file_flag;   /*  use  bits,  see  lookv4.c  for  key*/
  int   vs_row;
  int   first_row;
  int   last_row;
  int   mu_col;
  int   disp_col;
  int   mu_fit_col;
  int   weight_row;
  int   end_weight_row;
  int   weight_pts;
  int   weight_control;
  int   peak_row;
  int   added_pts;
  double  *disp_data;
  double  *mu_data;
  double  *model_mu;
  double        stiff;
  double        sig_n;
  double        vo;
  double        vf;
  double        muo;
  double        muf;
  double        amb;
  double        a;
  double        a_er;
  double        a_step;
  double        b1;
  double        b1_er;
  double        b1_step;
  double        b2;
  double        b2_er;
  double        b2_step;
  double        dc1;
  double        dc1_er;
  double        dc1_step;
  double        dc2;
  double        dc2_er;
  double        dc2_step;
  double        total_er;
  double        weight;
  double  vo_dc1;
  double  vo_dc2;
  double  k_vf;
  double  lin_term;
  double  vel_list[50];
  int   n_vsteps;
  int   vs_row_list[50];
};
struct  rs_parameters  rs_param  ;


/*-----------------------------------------------------------------*/

float  *darray[MAX_COL]  ;
int  max_col  ;
int  max_row  ;
int  n_param;
```



```
double  simp[MAX_PARAM+1][MAX_PARAM+1],temp[MAX_PARAM+1];

float   *arrayx,  *arrayy;

FILE  *command  ,  *com_file[10]  ,  *temp_com_file;


typedef  struct  plot_array
{
  float  *xarray;
  float  *yarray;
  int  nrows_x;  /*  probably  need  just  one  nrows  */
  int  nrows_y;
  int  col_x;
  int  col_y;
  int  begin;
  int  end;
  float  xmin;
  float  xmax;
  float  ymin;
  float  ymax;
  float  scale_x;
  float  scale_y;
  int  label_type;
  int  mouse;
  int  x1;
  int  y1;
  int  x2;
  int  y2;
  int  p1;
  int  p2;
  int  zp1;
  int  zp2;

}plotarray;


#include  <xview/canvas.h>
typedef  struct  canvas_info
{
  Canvas  canvas;
  Xv_Window  xvwin;
  Window  win;
  plotarray  *plots[10];
  int  canvas_num;
  int  active_plot;
  int  total_plots;
  int  alive_plots[10];
  int  start_x;
  int  start_y;
  int  end_x;
  int  end_y;
  int  start_xaxis;
  int  start_yaxis;
  int  end_xaxis;
  int  end_yaxis;
}  canvasinfo;

struct  windows_info
{
  int  windows[10];
  canvasinfo  *canvases[10];
 /*  int  active_window  =  0;  easier  to  declare  as  global  var...  */
};
```



```
struct  windows_info  wininfo;

#define  GC_TICK  (1)
#define  GC_TITLE  (2)
#define  CAN_X  (1)
#define  CAN_Y  (2)
#define  CAN_ROW  (3)
#define  CAN_NUM  (4)
#define  GRAF_FRAME  (5)
#define  MSG_LENGTH  300


/************************  action  DEFINE  variables  ********************/
#define  MAIN  (0)
#define  SET_PATH  (5)
#define  DOIT  (6)
#define  STDIN  (7)
#define  COM_FILE  (8)
#define  ALL  (9)
#define  READ  (10)
#define  APPEND  (11)
#define  WRITE  (12)
#define  TASC  (13)
#define  STDASC  (14)
#define  HEAD  (15)
#define  GETASCHEAD  (16)
#define  EXAMIN  (17)
#define  OVERWRITE  (18)
#define  ALL_NEED_ROW_COL  (20)
#define  PLOT_GET_BE  (21)
#define  PLOT_GET_XY  (22)
#define  PLOT_OTHERS  (23)
#define  PLOT_SCALE  (24)
#define  KILL_PLOT  (25)
#define  KILL_WIN  (26)
#define  CREATE_WIN  (27)
#define  SET_ACTIVE_WINDOW  (28)
#define  INTERPOLATE  (29)
#define  INTERPOLATE_GET_F_L  (30)
#define  INTERPOLATE_GET_INC  (31)
#define  MATHINT  (32)
#define  MATHINT_F_L  (33)
#define  MATH  (34)
#define  R_MEAN  (35)
#define  R_SPIKE  (36)
#define  POSITIVE  (37)
#define  COMPRESS  (38)
#define  X_MAX  (39)
#define  X_MIN  (40)
#define  TREND_XY  (41)
#define  TREND_F_L  (42)
#define  TREND_A  (43)
#define  FINDINT  (44)
#define  SUMMATION  (45)
#define  CURV  (46)
#define  PEAK  (47)
#define  DECIMAT  (48)
#define  DECIMAT_R  (49)
#define  DECIMAT_F_L  (50)
#define  PDF  (51)
#define  PDFAUTO  (52)
#define  R_ROW_COL  (53)
#define  R_ROW_COL_F_L  (54)
```

```
#define  R_ROW_F_L  (55)
#define  COMMENT_COL  (56)
#define  COMMENT_STR  (57)
#define  NAME  (58)
#define  NAME_GET_NAME  (59)
#define  NAME_GET_UNIT  (60)
#define  R_COL  (61)
#define  ZERO_ALL  (62)
#define  ZERO  (63)
#define  OFFSET  (64)
#define  OFFSET_INT  (65)
#define  OFFSET_INT_GET_YN  (66)
#define  SIMPLEX_GET_FN  (67)
#define  SIMPLEX_GET_F_L  (68)
#define  SIMPLEX_GET_COLS  (69)
#define  SCM_GET_ARGS  (70)
#define  EXAMIN_GET_FILENAME  (71)
#define  HEAD_GET_FILENAME  (72)
#define  GETASCHEAD_GET_FILENAME  (73)
#define  TASC_GET_FILENAME  (74)
#define  TASC_GET_FI  (75)
#define  STDASC_GET_FILENAME  (76)
#define  STDASC_GET_FI  (77)
#define  STDASC_GET_CR  (78)
#define  INTERPOLATE_1  (79)
#define  INTERPOLATE_2  (80)
#define  MATH_FINAL_1  (81)
#define  INTERPOLATE_R  (82)
#define  COL_POWER  (83)
#define  POWER  (84)
#define  RECIP  (85)
#define  LOG  (86)
#define  EXPLIN  (87)
#define  POLY4  (88)
#define  GENSIN  (89)
#define  GENEXP  (90)
#define  RCLOW  (91)
#define  RCPH  (92)
#define  SCCHISQR  (93)
#define  CHISQR  (94)
#define  NORMAL  (95)
#define  POWER2  (96)
#define  POWER1  (97)
#define  LN  (98)
#define  EXP  (99)
#define  DERIV  (100)
#define  DERIV_GET_INT  (101)
#define  VC  (102)
#define  VC_GET_INT  (103)
#define  VC_GET_F  (104)
#define  CGT  (105)
#define  RGT  (106)
#define  EC  (107)
#define  EC_GET_INT  (108)
#define  EC_GET_F  (109)
#define  CS  (110)
#define  CS_GET_INT  (111)
#define  POLYFIT  (112)
#define  POLYFIT_GET_INT  (113)
#define  POLYFIT_GET_INC  (114)
#define  POLYFIT_I  (115)
#define  POLYFIT_I_GET_INT  (116)
#define  POLYFIT_I_GET_INC  (117)
#define  POLYFIT_I_GET_ROW  (118)
```



```
#define  SORT  (119)
#define  SLOPE  (120)
#define  O_SLOPE  (121)
#define  RSM  (122)
#define  TRIG  (123)
#define  CM  (124)
#define  MEDIAN_SMOOTH  (125)
#define  SMOOTH  (126)
#define  MEM  (127)
#define  STAT  (128)
#define  TYPE  (129)
#define  TYPE_FL  (130)
#define  TYPE_S  (131)
#define  TYPEALL  (132)
#define  R_TREND  (133)
#define  R_TREND_TYPE  (134)
#define  R_TREND_INPUT  (135)
#define  R_TREND_COMP  (136)
#define  R_TREND_COMP_FL  (137)
#define  SIMP_FUNC_GET_MAX_ITER  (138)
#define  SIMP_FUNC_GET_INIT_GUESSES  (139)
#define  SIMP_FUNC_GET_INIT_STEP_SIZE  (140)
#define  SIMP_FUNC  (141)
#define  SCM_SIMP_WEIGHT_L2  (142)
#define  SCM_SIMP_WEIGHT  (143)
#define  SCM_1  (144)
#define  QI_MVS  (145)
#define  QI_WC  (146)
#define  QI  (147)
#define  SIMPLEX  (148)
#define  SCM  (149)
#define  SET_ACTIVE_PLOT  (150)



/*  name  DEFINE  variables  */
#define  SLOPE_NAME  (1001)
#define  TRIG_NAME  (1002)
#define  RCPH_NAME  (1003)
#define  POWER_NAME  (1004)
#define  RECIP_NAME  (1005)
#define  LOG_NAME  (1006)
#define  EXPLIN_NAME  (1007)
#define  POLY4_NAME  (1008)
#define  GENSIN_NAME  (1009)
#define  GENEXP_NAME  (1010)
#define  RCLOW_NAME  (1011)
#define  SCCHISQR_NAME  (1012)
#define  CHISQR_NAME  (1013)
#define  LN_NAME  (1014)
#define  NORMAL_NAME  (1015)
#define  POWER1_NAME  (1016)
#define  POWER2_NAME  (1017)
#define  EXP_NAME  (1018)
#define  DERIV_NAME  (1019)
#define  VC_NAME  (1020)
#define  RGT_NAME  (1021)
#define  CGT_NAME  (1022)
#define  EC_NAME  (1023)
#define  CS_NAME  (1024)
#define  POLYFIT_NAME  (1025)
#define  INTERPOLATE_NAME  (1026)
#define  MATH_NAME  (1027)
#define  MATHINT_NAME  (1028)
```



```
#define  POSITIVE_NAME  (1029)
#define  COMPRESS_NAME  (1030)
#define  X_MIN_NAME  (1031)
#define  X_MAX_NAME  (1032)
#define  SUMMATION_NAME  (1033)
#define  DECIMAT_NAME  (1034)
#define  CURV_NAME  (1035)
#define  PEAK_NAME  (1036)
#define  R_MEAN_NAME  (1037)
#define  SMOOTH_NAME  (1038)
#define  MEDIAN_SMOOTH_NAME  (1039)
#define  R_TREND_COMP_NAME  (1040)
#define  R_TREND_INPUT_NAME  (1041)
#define  INTERPOLATE_NAME1  (1042)
#define  INTERPOLATE_NAME2  (1043)
```



```
typedef  struct  FCOMPLEX  {float  r,i;}  fcomplex;
typedef  struct  IMMENSE  {unsigned  long  l,r;}  immense;
typedef  struct  GREAT  {unsigned  short  l,c,r;}  great;

#define  TRADITIONAL  1  /*  but  leave  LINT_ARGS  and  ANSI  undefined  */
#ifdef  LINT_ARGS
        void  adi(double  **,  double  **,  double  **,  double  **,
                  double  **,  double  **,  double  **,  double  **,
                  int,  int,  double,  double,  double);
        void  amoeba(float  **,  float  *,  int,  float,  float  (*)(),  int  *);
        void  anneal(float  *,  float  *,  int  *,  int);
        void  avevar(float  *,  int,  float  *,  float  *);
        void  balanc(float  **,  int);
        void  bcucof(float  *,  float  *,  float  *,  float,
                  float,  float  **);
        void  bcuint(float  *,  float  *,  float  *,  float  *,  float,
                  float,  float,  float,  float,  float,  float  *,
                  float  *,  float  *);
        float  bessi(int,  float);
        float  bessi0(float);
        float  bessi1(float);
        float  bessj(int,  float);
        float  bessj0(float);
        float  bessj1(float);
        float  bessk(int,  float);
        float  bessk0(float);
        float  bessk1(float);
        float  bessy(int,  float);
        float  bessy0(float);
        float  bessy1(float);
        float  beta(float,  float);
        float  betacf(float,  float,  float);
        float  betai(float,  float,  float);
        float  bico(int,  int);
        void  bksub(int,  int,  int,  int,  int,  float  ***);
        float  baldev(float,  int,  int  *);
        float  brent(float,  float,  float,  float  (*)(),  float,  float  *);
        void  bsstep(float  *,  float  *,  int,  float  *,  float,
                  float,  float  *,  float  *,  float  *,  void  (*)());
        void  caldat(long,  int  *,  int  *,  int  *);
        float  cel(float,  float,  float,  float);
        void  chder(float,  float,  float  *,  float  *,  int);
        float  chebev(float,  float,  float  *,  int,  float);
        void  chebft(float,  float,  float  *,  int,  float  (*)());
        void  chebpc(float  *,  float  *,  int);
        void  chint(float,  float,  float  *,  float  *,  int);
        void  chsone(float  *,  float  *,  int,  int,  float  *,
                  float  *,  float  *);
        void  chstwo(float  *,  float  *,  int,  int,  float  *,
                  float  *,  float  *);
        void  cntab1(int  **,  int,  int,  float  *,  float  *,
                  float  *,  float  *,  float  *);
        void  cntab2(int  **,  int,  int,  float  *,  float  *,
                  float  *,  float  *,  float  *,  float  *,  float  *,  float  *);
        void  convlv(float  *,  int,  float  *,  int,  int,  float  *);
        void  correl(float  *,  float  *,  int,  float  *);
        void  cosft(float  *,  int,  int);
        void  covsrt(float  **,  int,  int  *,  int);
        void  crank(int,  float  *,  float  *);
        float  dbrent(float,  float,  float,  float  (*)(),  float  (*)(),
                  float,  float  *);
        void  ddpoly(float  *,  int,  float,  float  *,  int);
        void  des(immense,  immense,  int  *,  int,  immense  *);
        void  ks(immense,  int,  great  *);
```

```
void cyfun(unsigned long, great, unsigned long *);
float dfidim(float);
void dfpmin(float *, int, float, int *, float *, float (*)(),
        void (*)());
void difeq(int, int, int, int, int, int, int *, int,
        float **, float **);
void dlinmin(float *, float *, int, float *, float (*)(), void (*)());
void eclass(int *, int, int *, int *, int);
void eclazz(int *, int, int (*)());
void eigsrt(float *, float **, int);
float el2(float, float, float, float);
void elmhes(float **, int);
float erf(float);
float erfc(float);
float erfcc(float);
void eulsum(float *, float, int, float *);
float evlmem(float, float *, int, float);
float expdev(int *);
float fidim(float);
float factln(int);
float factrl(int);
void fgauss(float, float *, float *, float *, int);
void fit(float *, float *, int, float *, int, float *,
        float *, float *, float *, float *, float *);
void fixrts(float *, int);
void fleg(float, float *, int);
void flmoon(int, int, long *, float *);
void four1(float *, int, int);
void fourn(float *, int *, int, int);
void fpoly(float, float *, int);
void frprmn(float *, int, float, int *, float *, float (*)(),
        void (*)());
void ftest(float *, int, float *, int, float *, float *);
float gamdev(int, int *);
float gammln(float);
float gammp(float, float);
float gammq(float, float);
float gasdev(int *);
void gauleg(double, double, double *, double *, int);
void gaussj(float **, int, float **, int);
void gcf(float *, float, float, float, float *);
float golden(float, float, float, float (*)(), float, float *);
void gser(float *, float, float, float *);
void hqr(float **, int, float *, float *);
void hunt(float *, int, float, int *);
void indexx(int, float *, int *);
int irbit1(unsigned long int *);
int irbit2(unsigned long int *);
void jacobi(float **, int, float *, float **, int *);
long julday(int, int, int);
void kendl1(float *, float *, int, float *, float *, float *);
void kendl2(float **, int, int, float *, float *, float *);
void ksone(float *, int, float (*)(), float *, float *);
void kstwo(float *, int, float *, int, float *, float *);
void laguer(fcomplex *, int, fcomplex *, float, int);
void lfit(float *, float *, float *, int, float *, int,
        int *, int, float **, float *, void (*)());
void linmin(float *, float *, int, float *, float (*)());
void locate(float *, int, float, int *);
void lubksb(float **, int, int *, float *);
void ludcmp(float **, int, int *, float *);
void mdian1(float *, int, float *);
void mdian2(float *, int, float *);
void medfit(float *, float *, int, float *, float *, float *);
```



```
void memcof(float *, int, int, float *, float *);
float midexp(float (*)(), float, float, int);
float midinf(float (*)(), float, float, int);
float midpnt(float (*)(), float, float, int);
float midsql(float (*)(), float, float, int);
float midsqu(float (*)(), float, float, int);
void mmid(float *, float *, int, float, float, int, float *,
        void (*)());
void mnbrak(float *, float *, float *, float *,
        float *, float *, float (*)());
void mnewt(int, float *, int, float, float);
void moment(float *, int, float *, float *, float *,
        float *, float *, float *);
void mprove(float **, float **, int, int *, float *, float *);
void mrqcof(float *, float *, float *, int, float *, int,
        int *, int, float **, float *, float *, void (*)());
void mrqmin(float *, float *, float *, int, float *, int,
        int *, int, float **, float **, float *, void (*)(),
        float *);
void odeint(float *, int, float, float, float, float,
        float, int *, int *, void (*)(), void (*)());
void pcshft(float, float, float *, int);
void pearsn(float *, float *, int, float *, float *, float *);
void piksr2(int, float *, float *);
void piksrt(int, float *);
void pinvs(int, int, int, int, int, int, float ***, float **);
float plgndr(int, int, float);
float poldev(float, int *);
void polcoe(float *, float *, int, float *);
void polcof(float *, float *, int, float *);
void poldiv(float *, int, float *, int, float *, float *);
void polin2(float *, float *, float **, int, int, float,
        float, float *, float *);
void polint(float *, float *, int, float, float *, float *);
void powell(float *, float **, int, float, int *, float *, float (*)());
void predic(float *, int, float *, int, float *, int);
float probks(float);
void pzextr(int, float, float *, float *, float *, int, int);
void qcksrt(int, float *);
float qgaus(float (*)(float), float, float);
float qromb(float (*)(float), float, float);
float qromo(float (*)(float), float, float, float (*)());
void qroot(float *, int, float *, float *, float);
float qsimp(float (*)(float), float, float);
float qtrap(float (*)(float), float, float);
float quad3d(float (*)(float), float, float);
float ran0(int *);
float ran1(int *);
float ran2(long *);
float ran3(int *);
float ran4(int *);
void rank(int, int *, int *);
void ratint(float *, float *, int, float, float *, float *);
void realft(float *, int, int);
void red(int, int, int, int, int, int, int, int, int, int,
        int, float ***, float **);
void rk4(float *, float *, int, float, float, float *, void (*)());
void rkdumb(float *, int, float, float, int, void (*)());
void rkqc(float *, float *, int, float *, float, float,
        float *, float *, float *, void (*)());
float rofunc(float);
float rtbis(float (*)(), float, float, float);
float rtflsp(float (*)(), float, float, float);
float rtnewt(void (*)(), float, float, float);
```



```
float rtsafe(void (*)(), float, float, float);
float rtsec(float (*)(), float, float, float);
void rzextr(int, float, float *, float *, float *, int, int);
void scrsho(float (*)());
void shell(int, float *);
void shoot(int, float *, float *, int, float, float, float,
        float, float, float *, float *);
void shootf(int, float *, float *, float *, float *, int, int,
        float, float, float, float, float, float, float *,
        float *, float *);
void simp1(float **, int, int *, int, int, int *, float *);
void simp2(float **, int, int *, int, int *, int, float *);
void simp3(float **, int, int, int, int);
void simplx(float **, int, int, int, int, int, int,
        int *, int *, int *);
void sinft(float *, int);
void smooft(float *, int, float);
void sncndn(float, float, float *, float *, float *);
void solvde(int, float, float, float *, int *, int, int,
        int, float **, float ***, float **);
void sor(double **, double **, double **, double **, double **,
        double **, double **, int, double);
void sort(int, float *);
void sort2(int, float *, float *);
void sort3(int, float *, float *, float *);
void sparse(float *, int, float *, float *);
void spctrm(FILE *, float *, int, int, int);
void spear(float *, float *, int, float *,
        float *, float *, float *, float *);
void splie2(float *, float *, float **, int, int, float **);
void splin2(float *, float *, float **, float **, int, int,
        float, float, float *);
void spline(float *, float *, int, float, float, float *);
void splint(float *, float *, float *, int, float, float *);
void svbksb(float **, float *, float **, int, int,
        float *, float *);
void svdcmp(float **, int, int, float *, float **);
void svdfit(float *, float *, float *, int, float *, int,
        float **, float **, float *, float *, void (*)());
void svdvar(float **, int, float *, float **);
void toeplz(float *, float *, float *, int);
void tptest(float *, float *, int, float *, float *);
void tqli(float *, float *, int, float **);
float trapzd(float (*)(float), float, float, int);
void tred2(float **, int, float *, float *);
void tridag(float *, float *, float *, float *, float *, int);
void ttest(float *, int, float *, int, float *, float *);
void tutest(float *, int, float *, int, float *, float *);
void twofft(float *, float *, float *, float *, int);
void vander(float *, float *, float *, int);
int zbrac(float (*)(), float *, float *);
void zbrak(float (*)(), float, float, int, float *, float *, int *);
float zbrent(float (*)(), float, float, float);
void zroots(fcomplex *, int, fcomplex *, int);
#endif
#ifdef ANSI
void adi(double **a, double **b, double **c, double **d, double **e,
        double **f, double **g, double **u, int jmax, int k,
        double alpha, double beta, double eps);
void amoeba(float **p, float *y, int ndim, float ftol,
        float (*funk)(float *), int *iter);
void anneal(float *x, float *y, int *iorder, int ncity);
void avevar(float *data, int n, float *ave, float *svar);
void balanc(float **a, int n);
```



```
void bcucof(float *y, float *y1, float *y2, float *y12, float d1,
        float d2, float **c);
void bcuint(float *y, float *y1, float *y2, float *y12, float x1l,
        float x1u, float x2l, float x2u, float x1, float x2,
        float *ansy, float *ansy1, float *ansy2);
float bessi(int n, float x);
float bessi0(float x);
float bessi1(float x);
float bessj(int n, float x);
float bessj0(float x);
float bessj1(float x);
float bessk(int n, float x);
float bessk0(float x);
float bessk1(float x);
float bessy(int n, float x);
float bessy0(float x);
float bessy1(float x);
float beta(float z, float w);
float betacf(float a, float b, float x);
float betai(float a, float b, float x);
float bico(int n, int k);
void bksub(int ne, int nb, int jf, int k1, int k2, float ***c);
float bnldev(float pp, int n, int *idum);
float brent(float ax, float bx, float cx, float (*f)(float), float tol,
        float *xmin);
void bsstep(float *y, float *dydx, int nv, float *xx, float htry,
        float eps, float *yscal, float *hdid, float *hnext,
        void (*derivs)(float,float *,float *));
void caldat(long julian, int *mm, int *id, int *iyyy);
float cel(float qqc, float pp, float aa, float bb);
void chder(float a, float b, float *c, float *cder, int n);
float chebev(float a, float b, float *c, int m, float x);
void chebft(float a, float b, float *c, int n, float (*func)(float));
void chebpc(float *c, float *d, int n);
void chint(float a, float b, float *c, float *cint, int n);
void chsone(float *bins, float *ebins, int nbins, int knstrn,
        float *df, float *chsq, float *prob);
void chstwo(float *bins1, float *bins2, int nbins, int knstrn,
        float *df, float *chsq, float *prob);
void cntab1(int **nn, int nl, int nj, float *chisq, float *df,
        float *prob, float *cramrv, float *ccc);
void cntab2(int **nn, int ni, int nj, float *h, float *hx, float *hy,
        float *hygx, float *hxgy, float *uygx, float *uxgy,
        float *uxy);
void convlv(float *data, int n, float *respns, int m, int isign,
        float *ans);
void correl(float *data1, float *data2, int n, float *ans);
void cosft(float *y, int n, int isign);
void covsrt(float **covar, int ma, int *lista, int mfit);
void crank(int n, float *w, float *s);
float dbrent(float ax, float bx, float cx, float (*f)(float),
        float (*df)(float), float tol, float *xmin);
void ddpoly(float *c, int nc, float x, float *pd, int nd);
void des(immense inp, immense key, int *newkey, int isw, immense *out);
void ks(immense key, int n, great *kn);
void cyfun(unsigned long ir, great k, unsigned long *iout);
float dfidim(float x);
void dfpmin(float *p, int n, float ftol, int *iter, float *fret,
        float (*func)(float *), void (*dfunc)(float *,float *));
void difeq(int k, int k1, int k2, int jsf, int is1, int isf,
        int *indexv, int ne, float **s, float **y);
void dlinmin(float *p, float *xi, int n, float *fret,
        float (*func)(float *), void (*dfunc)(float *,float *));
void eclass(int *nf, int n, int *lista, int *listb, int m);
```

```
void eclazz(int *nf, int n, int (*equiv)(int,int));
void eigsrt(float *d, float **v, int n);
float el2(float x, float qqc, float aa, float bb);
void elmhes(float **a, int n);
float erf(float x);
float erfc(float x);
float erfcc(float x);
void eulsum(float *sum, float term, int jterm, float *wksp);
float evlmem(float fdt, float *cof, int m, float pm);
float expdev(int *idum);
float fldim(float x);
float factln(int n);
float factrl(int n);
void fgauss(float x, float *a, float *y, float *dyda, int na);
void fit(float *x, float *y, int ndata, float *sig, int mwt, float *a,
  float *b, float *siga, float *sigb, float *chi2, float *q);
void fixrts(float *d, int npoles);
void fleg(float x, float *pl, int nl);
void flmoon(int n, int nph, long *jd, float *frac);
void four1(float *data, int nn, int isign);
void fourn(float *data, int *nn, int ndim, int isign);
void fpoly(float x, float *p, int np);
void frprmn(float *p, int n, float ftol, int *iter, float *fret,
  float (*func)(float *), void (*dfunc)(float *,float *));
void ftest(float *data1, int n1, float *data2, int n2, float *f,
  float *prob);
float gamdev(int ia, int *idum);
float gammln(float xx);
float gammp(float a, float x);
float gammq(float a, float x);
float gasdev(int *idum);
void gauleg(double x1, double x2, double *x, double *w, int n);
void gaussj(float **a, int n, float **b, int m);
void gcf(float *gammcf, float a, float x, float *gln);
float golden(float ax, float bx, float cx, float (*f)(float), float tol,
  float *xmin);
void gser(float *gamser, float a, float x, float *gln);
void hqr(float **a, int n, float *wr, float *wi);
void hunt(float *xx, int n, float x, int *jlo);
void indexx(int n, float *arrin, int *indx);
int irbit1(unsigned long int *iseed);
int irbit2(unsigned long int *iseed);
void jacobi(float **a, int n, float *d, float **v, int *nrot);
long julday(int mm, int id, int iyyy);
void kendl1(float *data1, float *data2, int n, float *tau, float *z,
  float *prob);
void kendl2(float **tab, int i, int j, float *tau, float *z,
  float *prob);
void ksone(float *data, int n, float (*func)(float), float *d,
  float *prob);
void kstwo(float *data1, int n1, float *data2, int n2, float *d,
  float *prob);
void laguer(fcomplex *a, int m, fcomplex *x, float eps, int polish);
void lfit(float *x, float *y, float *sig, int ndata, float *a, int ma,
  int *lista, int mfit, float **covar, float *chisq,
  void (*funcs)(float,float *,int));
void linmin(float *p, float *xi, int n, float *fret, float (*func)(float));
void locate(float *xx, int n, float x, int *j);
void lubksb(float **a, int n, int *indx, float *b);
void ludcmp(float **a, int n, int *indx, float *d);
void medfit(float *x, float *y, int ndata, float *a, float *b,
  float *abdev);
```



```
void memcof(float *data, int n, int m, float *pm, float *cof);
float midexp(float (*funk)(float), float aa, float bb, int n);
float midinf(float (*funk)(float), float aa, float bb, int n);
float midpnt(float (*func)(float), float a, float b, int n);
float midsql(float (*funk)(float), float aa, float bb, int n);
float midsqu(float (*funk)(float), float aa, float bb, int n);
void mmid(float *y, float *dydx, int nvar, float xs, float htot,
  int nstep, float *yout,
  void (*derivs)(float,float *,float *));
void mnbrak(float *ax, float *bx, float *cx, float *fa, float *fb,
  float *fc, float (*func)(float));
void mnewt(int ntrial, float *x, int n, float tolx, float tolf);
void moment(float *data, int n, float *ave, float *adev, float *sdev,
  float *svar, float *skew, float *curt);
void mprove(float **a, float **alud, int n, int *indx, float *b,
  float *x);
void mrqcof(float *x, float *y, float *sig, int ndata, float *a, int ma,
  int *lista, int mfit, float **alpha, float *beta, float
  *chisq, void (*funcs)(float,float *,float *,float *,int));
void mrqmin(float *x, float *y, float *sig, int ndata, float *a,
  int ma, int *lista, int mfit, float **covar, float **alpha,
  float *chisq, void (*funcs)(float,float *,float *,float *,
  int),float *alamda);
void odeint(float *ystart, int nvar, float x1, float x2, float eps,
  float h1, float hmin, int *nok, int *nbad,
  void (*derivs)(float,float *,float *),
  void (*rkqc)(float *,float *,int,float *,float,float,float
  *,float *,float *,void (*)(float,float *,float *)));
void pcshft(float a, float b, float *d, int n);
void pearsn(float *x, float *y, int n, float *r, float *prob, float
void piksr2(int n, float *arr, float *brr);
void piksrt(int n, float *arr);
void pinvs(int ie1, int ie2, int je1, int jsf, int jcl, int k,
  float ***c, float **s);
float plgndr(int l, int m, float x);
float poidev(float xm, int *idum);
void polcoe(float *x, float *y, int n, float *cof);
void polcof(float *xa, float *ya, int n, float *cof);
void poldiv(float *u, int n, float *v, int nv, float *q, float *r);
void polin2(float *x1a, float *x2a, float **ya, int m, int n, float x1,
  float x2, float *y, float *dy);
void polint(float *xa, float *ya, int n, float x, float *y, float *dy);
void powell(float *p, float **xi, int n, float ftol, int *iter,
  float *fret, float (*func)(float *));
void predic(float *data, int ndata, float *d, int npoles,
  float *future, int nfut);
float probks(float alam);
void prsxtr(int iest, float xest, float *yest, float *yz, float *dy,
  int nv, int nuse);
void qcksrt(int n, float *arr);
float qgaus(float (*func)(float), float a, float b);
float qromb(float (*func)(float), float a, float b);
float qromo(float (*func)(float), float a, float b,
  float (*choose)(float (*)(float),float,float,int));
void qroot(float *p, int n, float *b, float *c, float eps);
float qsimp(float (*func)(float), float a, float b);
float qtrap(float (*func)(float), float a, float b);
float quad3d(float (*func)(float,float,float), float x1, float x2);
float ran0(int *idum);
float ran1(int *idum);
float ran2(long *idum);
float ran3(int *idum);
float ran4(int *idum);
void rank(int n, int *indx, int *irank);
```



```
void ratint(float *xa, float *ya, int n, float x, float *y, float *dy);
void realft(float *data, int n, int isign);
void red(int iz1, int iz2, int jz1, int jz2, int jm1, int jm2, int jmf,
  int ic1, int jc1, int jcf, int kc, float ***c, float **s);
void rk4(float *y, float *dydx, int n, float x, float h, float *yout,
  void (*derivs)(float,float *,float *));
void rkdumb(float *vstart, int nvar, float x1, float x2, int nstep,
  void (*derivs)(float,float *,float *));
void rkqc(float *y, float *dydx, int n, float *x, float htry,
  float eps, float *yscal, float *hdid, float *hnext,
  void (*derivs)(float,float *,float *));
float rofunc(float b);
float rtbis(float (*func)(float), float x1, float x2, float xacc);
float rtflsp(float (*func)(float), float x1, float x2, float xacc);
float rtnewt(void (*funcd)(float,float *,float *), float x1, float x2,
  float xacc);
float rtsafe(void (*funcd)(float,float *,float *), float x1, float x2,
  float xacc);
float rtsec(float (*func)(float), float x1, float x2, float xacc);
void rzextr(int iest, float xest, float *yest, float *yz, float *dy,
  int nv, int nuse);
void scrsho(float (*fx)(float));
void shell(int n, float *arr);
void shoot(int nvar, float *v, float *delv, int n2, float x1, float x2,
  float eps, float h1, float hmin, float *f, float *dv);
void shootf(int nvar, float *v1, float *v2, float *delv1, float *delv2,
  int n1, int n2, float x1, float x2, float xf, float eps,
  float h1, float hmin, float *f, float *dv1, float *dv2);
void simp1(float **a, int mm, int *ll, int nll, int iabf, int *kp,
  float *bmax);
void simp2(float **a, int n, int *l2, int nl2, int *ip, int kp,
  float *q1);
void simp3(float **a, int i1, int k1, int ip, int kp);
void simplx(float **a, int m, int n, int m1, int m2, int m3,
  int *icase, int *izrov, int *iposv);
void sinft(float *y, int n);
void smooft(float *y, int n, float pts);
void sncndn(float uu, float emmc, float *sn, float *cn, float *dn);
void solvde(int itmax, float conv, float slowc, float *scalv,
  int *indexv, int ne, int nb, int m, float **y, float ***c,
  float **s);
void sor(double **a, double **b, double **c, double **d, double **e,
  double **f, double **u, int jmax, double rjac);
void sort(int n, float *ra);
void sort2(int n, float *ra, float *rb);
void sort3(int n, float *ra, float *rb, float *rc);
void sparse(float *b, int n, float *x, float *rsq);
void spctrm(FILE *fp, float *p, int m, int k, int ovrlap);
void spear(float *data1, float *data2, int n, float *d, float *zd,
  float *probd, float *rs, float *probrs);
void splie2(float *x1a, float *x2a, float **ya, int m, int n,
  float **y2a);
void splin2(float *x1a, float *x2a, float **ya, float **y2a, int m,
  int n, float x1, float x2, float *y);
void spline(float *x, float *y, int n, float yp1, float ypn, float *y2);
void splint(float *xa, float *ya, float *y2a, int n, float x, float *y);
void svbksb(float **u, float *w, float **v, int m, int n, float *b,
  float *x);
void svdcmp(float **a, int m, int n, float *w, float **v);
void svdfit(float *x, float *y, float *sig, int ndata, float *a,
  int ma, float **u, float **v, float *w, float *chisq,
  void (*funcs)(float,float *,int));
void svdvar(float **v, int ma, float *w, float **cvm);
void toeplz(float *r, float *x, float *y, int n);
```



```
void tptest(float *data1, float *data2, int n, float *t, float *prob)
void tqli(float *d, float *e, int n, float **z);
float trapzd(float (*func)(float), float a, float b, int n);
void tred2(float **a, int n, float *d, float *e);
void tridag(float *a, float *b, float *c, float *r, float *u, int n);
void ttest(float *data1, int n1, float *data2, int n2, float *t,
  float *prob);
void tutest(float *data1, int n1, float *data2, int n2, float *t,
  float *prob);
void twofft(float *data1, float *data2, float *fft1, float *fft2,
  int n);
void vander(float *x, float *w, float *q, int n);
int zbrac(float (*func)(float), float *x1, float *x2);
void zbrak(float (*fx)(float), float x1, float x2, int n, float *xb1,
  float *xb2, int *nb);
float zbrent(float (*func)(float), float x1, float x2, float tol);
void zroots(fcomplex *a, int m, fcomplex *roots, int polish);
#endif
#ifdef TRADITIONAL
void adi();
void amoeba();
void anneal();
void avevar();
void balanc();
void bcucof();
void bcuint();
float bessi();
float bessi0();
float bessi1();
float bessj();
float bessj0();
float bessj1();
float bessk();
float bessk0();
float bessk1();
float bessy();
float bessy0();
float bessy1();
float beta();
float betacf();
float betai();
float bico();
void bksub();
float bnldev();
float brent();
void bsstep();
void caldat();
float cel();
void chder();
float chebev();
void chebft();
void chebpc();
void chint();
void chsone();
void chstwo();
void cntab1();
void cntab2();
void convlv();
void correl();
void cosft();
void covsrt();
void crank();
float dbrent();
void ddpoly();
```

```
void des();
void ks();
void cyfun();
float dfldim();
void dfpmin();
void dlinmin();
void difeq();
void eclass();
void eclazz();
void eigsrt();
float el2();
void elmhes();
float erf();
float erfc();
float erfcc();
void eulsum();
float evlmem();
float expdev();
float fldim();
float factln();
float factrl();
void fgauss();
void fit();
void fixrts();
void fleg();
void flmoon();
void four1();
void fourn();
void fpoly();
void frprmn();
void ftest();
float gamdev();
float gammln();
float gammp();
float gammq();
float gasdev();
void gauleg();
void gaussj();
void gcf();
float golden();
void gser();
void hqr();
void hunt();
void indexx();
int irbit1();
int irbit2();
void jacobi();
long julday();
void kendl1();
void kendl2();
void kscone();
void kstwo();
void laguer();
void lfit();
void linmin();
void locate();
void lubksb();
void ludcmp();
void mdian1();
void mdian2();
void medfit();
void memcof();
float midexp();
float midinf();
```



```
float midpnt();
float midsql();
float midsqu();
void mmid();
void mnbrak();
void mnewt();
void moment();
void mprove();
void mrqcof();
void mrqmin();
void odeint();
void pcshft();
void pearsn();
void piksr2();
void piksrt();
void pinvs();
float plgndr();
float poldev();
void polcoe();
void polcof();
void poldiv();
void polin2();
void polint();
void powell();
void predic();
float probks();
void pzextr();
void qcksrt();
float qgaus();
float qromb();
float qromo();
void qroot();
float qsimp();
float qtrap();
float quad3d();
float ran0();
float ran1();
float ran2();
float ran3();
float ran4();
void rank();
void ratint();
void realft();
void red();
void rk4();
void rkdumb();
void rkqc();
float rofunc();
float rtbis();
float rtflsp();
float rtnewt();
float rtsafe();
float rtsec();
void rzextr();
void scrsho();
void shell();
void shoot();
void shootf();
void simp1();
void simp2();
void simp3();
void simplx();
void sinft();
void smooft();
```



```
void sncndn();
void solvde();
void sor();
void sort();
void sort2();
void sort3();
void sparse();
void spctrm();
void spear();
void spline2();
void splin2();
void spline();
void splint();
void svbksb();
void svdcmp();
void svdfit();
void svdvar();
void toeplz();
void tptest();
void tqli();
float trapzd();
void tred2();
void tridag();
void ttest();
void tutest();
void twofft();
void vander();
int zbrac();
void zbrak();
float zbrent();
void zroots();
#endif
```



```
float *vector();
float **matrix();
float **convert_matrix();
double *dvector();
double **dmatrix();
int *ivector();
int **imatrix();
float **submatrix();
void free_vector();
void free_dvector();
void free_ivector();
void free_matrix();
void free_dmatrix();
void free_imatrix();
void free_submatrix();
void free_convert_matrix();
void nrerror();
```

```
#include "global.h"
extern int action;
extern char msg[128];


double sin(),cos(),tan(),asin(),acos(),atan();
/* ------------------------------------------------------------------------*/
void zero(col,rec)
int *col , *rec ;
{
   static int i ;
   static float temp_fl ;
   temp_fl = darray[*col][*rec] ;
   for( i = 0; i < head.ch[*col].nelem; ++i)
   {
      darray[*col][i] -= temp_fl ;
   }
}
/* ------------------------------------------------------------------------*/
void offset_int(rec,col,rec2,ts)
int *rec , *col , *rec2;
char *ts;
{
   static int i ;
   static float temp_fl ;
   temp_fl = darray[*col][*rec2] - darray[*col][*rec] ;
   if(*ts == 'y')
                     /*set values between rec1 and rec2 equal to rec 1*/
   {
     for( i = *rec; i < *rec2; ++i)
        darray[*col][i] = darray[*col][*rec];
   }

   for( i = *rec2; i < head.ch[*col].nelem; ++i)
   {
      darray[*col][i] -= temp_fl ;
   }
}
/* ------------------------------------------------------------------------*/
void offset(rec,col,rec2,col2)
int *rec , *col , *rec2 , *col2;
{
   static int i ;
   static float temp_fl ;
   temp_fl = darray[*col2][*rec2] - darray[*col][*rec] ;
   for( i = 0; i < head.ch[*col].nelem; ++i)
   {
      darray[*col][i] += temp_fl ;
   }
}
/* ------------------------------------------------------------------------*/
void summation(old,new)
int *old , *new ;
{
   static int i ;
   for( i = 1; i < head.ch[*old].nelem; ++i)
   {
       darray[*new][i] = darray[*old][i] + darray[*new][i-1] ;
   }
}
/* ------------------------------------------------------------------------*/
void expvec(old,new)
int *old , *new ;
{
```



```
   static int i ;
   for( i = 0; i < head.ch[*old].nelem; ++i)
   {
      darray[*new][i] = exp(darray[*old][i]) ;
   }
}
/* ------------------------------------------------------------------------*/
void lnvec(old,new)
int *old , *new ;
{
   static int i ;
   for( i = 0; i < head.ch[*old].nelem; ++i)
   {
      darray[*new][i] = log(darray[*old][i]) ;
   }
}
/* ------------------------------------------------------------------------*/
void sinvec(old,new)
int *old , *new ;
{
   static int i ;
   for( i = 0; i < head.ch[*old].nelem; ++i)
   {
      darray[*new][i] = sin((double)darray[*old][i]) ;
   }
}
/* ------------------------------------------------------------------------*/
void cosvec(old,new)
int *old , *new ;
{
   static int i ;
   for( i = 0; i < head.ch[*old].nelem; ++i)
   {
      darray[*new][i] = cos((double)darray[*old][i]) ;
   }
}
/* ------------------------------------------------------------------------*/
void tanvec(old,new)
int *old , *new ;
{
   static int i ;
   for( i = 0; i < head.ch[*old].nelem; ++i)
   {
      darray[*new][i] = tan((double)darray[*old][i]) ;
   }
}
/* ------------------------------------------------------------------------*/
void asinvec(old,new)
int *old , *new ;
{
   static int i ;
   for( i = 0; i < head.ch[*old].nelem; ++i)
   {
      darray[*new][i] = asin((double)darray[*old][i]) ;
   }
}
/* ------------------------------------------------------------------------*/
void acosvec(old,new)
int *old , *new ;
{
   static int i ;
   for( i = 0; i < head.ch[*old].nelem; ++i)
   {
      darray[*new][i] = acos((double)darray[*old][i]) ;
```



```
   }
}
/* ------------------------------------------------------------------------*/
void atanvec(old,new)
int *old , *new ;
{
   static int i ;
   for( i = 0; i < head.ch[*old].nelem; ++i)
   {
      darray[*new][i] = atan((double)darray[*old][i]) ;
   }
}
/* ------------------------------------------------------------------------*/
void logvec(old,new)
int *old , *new ;
{
   static int i ;
   for( i = 0; i < head.ch[*old].nelem; ++i)
   {
      darray[*new][i] = log10(darray[*old][i]) ;
   }
}
/* ------------------------------------------------------------------------*/
void recipvec(old,new)
int *old , *new ;
{
   static int i ;
   for( i = 0; i < head.ch[*old].nelem; ++i)
   {
      darray[*new][i] = 1.0/darray[*old][i] ;
   }
}
/* ------------------------------------------------------------------------*/
void powcvec(old,new,power_col)
int *old , *new ;
int *power_col;
{
   static int i ;
   for( i = 0; i < head.ch[*old].nelem; ++i)
   {
      darray[*new][i] = (float)pow((double)darray[*old][i],(double)darray[*power_col][i])
;
   }
}
/* ------------------------------------------------------------------------*/
void powvec(old,new,power)
int *old , *new ;
float *power;
{
   static int i ;
   for( i = 0; i < head.ch[*old].nelem; ++i)
   {
      darray[*new][i] = (float)pow((double)darray[*old][i],(double)*power) ;
   }
}
/* ------------------------------------------------------------------------*/
void
elastic_corr(ad_c, tau_c, new_col, first, last, E_on_L)

int ad_c, tau_c, new_col;                    /*ad_c= column that contains axial di
ement*/
int first, last;
float E_on_L;
{
```



```
   int i,n_data;
   int row, nrb;
   float *disp;

        n_data = last-first;
        row = first;
        nrb = row-1;
                                          /*first point of the corrected disp  co
n is the same as that from the uncorrected */

        disp = (float *) calloc(n_data,sizeof(float));  /* in case they chose same
r op */
        for(i=first; i<=last;++i)
                disp[i-first] = darray[ad_c][i];

        darray[new_col][row] = darray[ad_c][row];

        for(i=0;i<n_data;++i)
        {
                nrb++;
                row++;
                darray[new_col][row] = ( darray[new_col][nrb] + (disp[i+1]-disp[i]*
                                         (darray[tau_c][row]-darray[tau_c][nrb])/E_on_L
;

        }
        cfree(disp);
}
/* ------------------------------------------------------------------------*/
void
shear_strain(disp_col, thick_col, strain_col, first, last)

int disp_col, thick_col, strain_col;
int first, last;
{

  int i,n_data;
  int row, orb;
  float *disp;

        n_data = last-first;
        orb = row = first;

        disp = (float *) calloc(n_data,sizeof(float)); /* in case they chose s
ame col for op */
        for(i=first; i<=last;++i)
                disp[i-first] = darray[disp_col][i];

        darray[strain_col][row] = darray[disp_col][row]/darray[thick_col][row];

        for(i=0;i<n_data;++i)
        {
                row++;
                darray[strain_col][row] = ( darray[strain_col][orb] +
                                (disp[i+1]-disp[i]) /
                                ((darray[thick_col][row]+darray[thick_col][orb])/2.00) );
                orb++;

        }
        cfree(disp);
}
/* ------------------------------------------------------------------------*/
void
```

```
calc_geom_thin(disp_col, new_col, L, h_h )

int disp_col, new_col;
float L, h_h;
{

  int i;
  double del_h = 0;


        /* calculate geometric thinning during direct shear test for */
        /* calculation is: del_h = h dx/2L ; where h is initial thickness, dx i
crement and L is length of the sliding block parallel to slip */
        /* see lookv3.c "cgt" for more info */

        L *= 2.00000;

        darray[new_col][0] = h_h;

        for(i=1; i<head.ch[disp_col].nelem; ++i)
        {
          h_h -= (darray[disp_col][i]-darray[disp_col][i-1]) * h_h/L;
          darray[new_col][i] = h_h;
        }



}

/* ------------------------------------------------------------------------*/
void
geom_thin(disp_col, gouge_thick_col, new_col, L)

int disp_col, gouge_thick_col, new_col;
float L;
{

  int i;
  double del_h = 0;


        /* correct horizontal displacement measurement during direct shear test for
tric thinning" */
        /* correction is: del_h = h dx/2L ; where h is thickness, dx is slip i
nd L is length of the sliding block parallel to slip */
        /* see lookv3.c "rgt" for more info */

        L *= 2.00000;

        darray[new_col][0] = darray[gouge_thick_col][0];

        for(i=1; i<head.ch[disp_col].nelem; ++i)
        {
          del_h += (darray[disp_col][i]-darray[disp_col][i-1]) * darray[gouge_thick_col][i
-1]/L;
          darray[new_col][i] = darray[gouge_thick_col][i-1] + del_h;
        }



}

/* ------------------------------------------------------------------------*/
void
```



```
vol_corr(vs_c, tau_c, new_col, first, last, SV_on_V)

int vs_c, tau_c, new_col; /*vs_c= column that contains axial displac
ement*/
                                        /* SV_on_V = S*Va/V */
int first, last;
float SV_on_V;
{

  int i,n_data;
  int row, nrb;

        n_data = last-first;
        row = first;
        nrb = row-1;
                                                /*first point of the corrected vs colu
s
the same as that from the uncorrected */

/* corrected vs = vs - dtau*S*Va/V NOTE that dtau=dPc for the constant normal
ing conditions and a gouge layer at 45 degrees to sigma_1 */

        darray[new_col][row] = darray[vs_c][row];

        for(i=0;i<n_data;++i)
        {
                nrb++;
                row++;
                darray[new_col][row] = ( darray[new_col][nrb] +
                                         darray[vs_c][row]-darray[vs_c][nrb] -
                                         (darray[tau_c][row]-darray[tau_c][nrb])*SV_on_V )
  ;

        }
}
/* ------------------------------------------------------------------------*/
void deriv(xcol,ycol,new_col,first,last)
int *xcol, *ycol, *new_col;
int first, last;

{
 double denom;
 float *next[2];
 float aa[2], bb[2], cc[2], dydx;
 int   i, j, n_data;

        n_data = (last - first ) +1;
        next[0] = &(darray[*xcol][first]);
        next[1] = &(darray[*ycol][first]);

        for(j=0; j<2; ++j){
                aa[j] = *next[j];
                next[j] += 1;
                bb[j] = *next[j];
                next[j] += 1;
                cc[j] = *next[j];
        }


/*form is:     [ dydx = ep2*ep2*f(x+ep1) - ep1 * ep1* (f(x-ep2) - (ep2*ep2-ep1*e
  ] / (ep2*ep2*ep1 + ep1*ep1*ep2)*/

/* ep1          = cc[0] - bb[0] */
/* ep2          = bb[0] - aa[0] */
```



```
/* f(x+ep1)     = cc[1]     */
/* f(x)  = bb[1]            */
/* f(x-ep2)     = aa[1]     */

        denom=(fabs(cc[0]-bb[0])*(bb[0]-aa[0])*(bb[0]-aa[0]))+(fabs(bb[0]-aa[0])*(cc[0]-bb
[0])*(cc[0]-bb[0]));

        dydx = ((bb[0]-aa[0])*(bb[0]-aa[0])*cc[1] - ((cc[0] -bb[0])*(cc[0]-bb[0]))*aa[1])
  ((bb[0]-aa[0])*(bb[0]-aa[0])-(cc[0]-bb[0])*(cc[0]-bb[0]))*bb[1] ) / denom;

        darray[*new_col][first] = dydx;
        darray[*new_col][first+1] = dydx;

        n_data -= 3;    /*go until 1 before the last point*/

        for(i=0; i<n_data; ++i)
        {
           for(j=0; j<2; ++j)
           {
                  next[j] -= 1;
                  aa[j] = *next[j];

                  next[j] += 1;
                  bb[j] = *next[j];

                  next[j] += 1;
                  cc[j] = *next[j];
           }
           denom=(fabs(cc[0]-bb[0])*(bb[0]-aa[0])*(bb[0]-aa[0]))+(fabs(bb[0]-aa[0])*(cc[0]
-bb[0])*(cc[0]-bb[0]));

           dydx = ((bb[0]-aa[0])*(bb[0]-aa[0])*cc[1] - ((cc[0] -bb[0])*(cc[0]-bb[0]))*aa[
) - ((bb[0]-aa[0])*(bb[0]-aa[0])-(cc[0]-bb[0])*(cc[0]-bb[0]))*bb[1] ) / denom;

 /* fprintf(stderr,"x= %f %f\n",bb[0], dydx); */

           darray[*new_col][first+i+2] = dydx;

        }

        darray[*new_col][last] = dydx;

}
/* ------------------------------------------------------------------------*/
void add(a,b,f,c,to,first,last)
int *a , *b, *c ;
char *to ;
float *f ;
int first , last ;
{
int i ;

switch(*to)
{
   case '=':for ( i = first; i <= last; ++i)
            {
            darray[*c][i] = darray[*a][i] + (*f) ;
            }
            break ;
   case ':':for ( i = first; i <= last; ++i)
            {
            darray[*c][i] = darray[*a][i] + darray[*b][i] ;
            }
            break ;
```



```
}
}
/* ------------------------------------------------------------------------*/
void sub(a,b,f,c,to,first,last)
     int *a , *b, *c ;
     char *to ;
     float *f ;
     int first , last ;
{
  int i ;

  switch(*to)
    {
    case '=':
      for ( i = first; i <= last; ++i)
        {
          darray[*c][i] = darray[*a][i] - (*f) ;
        }
      break ;
    case ':':
      for ( i = first; i <= last; ++i)
        {
          darray[*c][i] = darray[*a][i] - darray[*b][i] ;
        }
      break ;
    }
}
/* ------------------------------------------------------------------------*/
void prod(a,b,f,c,to,first,last)
int *a, *b, *c ;
char *to ;
float *f ;
int first , last ;
{
int i;

switch(*to)
{
   case '=':for (i = first; i <= last; ++i)
            {
            darray[*c][i] = darray[*a][i] * (*f) ;
            }
            break ;
   case ':':for ( i = first; i <= last; ++i)
            {
            darray[*c][i] = darray[*a][i] * darray[*b][i] ;
            }
            break ;
}
}
/* ------------------------------------------------------------------------*/
void div(a,b,f,c,to,first,last)
int *a, *b, *c ;
char *to ;
float *f ;
int first , last ;
{
int i;

switch(*to)
{
```

```
    case '*':for (i = first; i <= last; ++i)
             {
             darray[*c][i] = darray[*a][i] / (*f) ;
             }
             break ;
    case ':':for ( i = first; i <= last; ++i)
             {
             darray[*c][i] = darray[*a][i] / darray[*b][i] ;
             }
             break ;
}
}
```



```
#include <math.h>
#include <X11/Xlib.h>
#include <xview/xview.h>
#include <xview/canvas.h>
#include <xview/panel.h>
#include <xview/xv_xrect.h>
#include <xview/font.h>
#include <xview/cursor.h>
#include <X11/olcursor.h>

#include "global.h"

extern int active_window;
extern int total_windows;
extern int old_active_window;
extern int tickfontheight, titlefontheight;
extern char msg[MSG_LENGTH];

extern Frame main_frame;
Xv_Cursor xhair_cursor;



create_canvas()
{
  Frame graf_frame;
  Canvas canvas;
  Xv_Window xvwin;
  Window win;

  Panel canvas_menu_panel;
  Panel canvas_info_panel;
  Panel_item X, Y, ROW;
  canvasinfo *can_info;
  int win_num;
  int i;
  Menu clr_plot_menu, act_plot_menu, mouse_menu;

  extern void canvas_event_proc(), redraw_proc(), refresh_win_proc();
  extern void clr_plots_proc(), clr_all_plots_proc(), clr_plots_notify_proc();
  extern void clear_win_proc(), kill_graf_proc(), zoom_plot_proc();
  extern void create_act_plot_menu_proc(), get_act_item_proc();
  extern void line_plot_proc(), mouse_mu_proc(), dist_proc();


  win_num = get_new_window_num();

  if (win_num == -1)
    {
      sprintf(msg, "Reached limit of 10 windows. Ignored!\n");
      print_msg(msg);
      total_windows--;
      return;
    }

  sprintf(msg, "Plot Window %d", win_num+1);

  graf_frame = (Frame)xv_create(main_frame, FRAME, FRAME_LABEL, msg,
                                XV_HEIGHT, 500,
                                XV_WIDTH, 700,
                                NULL);

  canvas_menu_panel = (Panel)xv_create(graf_frame, PANEL, XV_HEIGHT, 55, NULL);
```



```
  act_plot_menu = (Menu) xv_create(NULL, MENU,
                                   NULL);

  (void) xv_create(canvas_menu_panel, PANEL_BUTTON,
                   PANEL_LABEL_STRING, "Activate Plot",
                   PANEL_ITEM_MENU, act_plot_menu,
                   PANEL_NOTIFY_PROC, create_act_plot_menu_proc,
                   PANEL_CLIENT_DATA, win_num,
                   NULL);

  clr_plot_menu = (Menu) xv_create(NULL, MENU,
                                   MENU_GEN_PIN_WINDOW,
                                   graf_frame, "Clear Plots",
                                   MENU_CLIENT_DATA, win_num,
                                   MENU_ACTION_ITEM,
                                   "Clear All Plots", clr_all_plots_proc,
                                   MENU_ACTION_ITEM,
                                   "Select Plots...", clr_plots_proc,
                                   NULL);

  (void) xv_create(canvas_menu_panel, PANEL_BUTTON,
                   PANEL_LABEL_STRING, "Clear Plot(s)",
                   PANEL_ITEM_MENU, clr_plot_menu,
                   PANEL_NOTIFY_PROC, clr_plots_notify_proc,
                   PANEL_CLIENT_DATA, win_num,
                   NULL);

  (void) xv_create(canvas_menu_panel, PANEL_BUTTON,
                   PANEL_LABEL_STRING, "Zoom",
                   PANEL_NOTIFY_PROC, zoom_plot_proc,
                   PANEL_CLIENT_DATA, win_num,
                   NULL);

  mouse_menu = (Menu) xv_create(NULL, MENU,
                                MENU_GEN_PIN_WINDOW,
                                main_frame, "Mouse",
                                MENU_ACTION_ITEM,
                                "Line Plot", line_plot_proc,
                                MENU_ACTION_ITEM,
                                "Vertical Line", mouse_mu_proc,
                                MENU_ACTION_ITEM,
                                "Distance", dist_proc,
                                NULL);

  (void) xv_create(canvas_menu_panel, PANEL_BUTTON,
                   PANEL_LABEL_STRING, "Mouse",
                   PANEL_ITEM_MENU, mouse_menu,
                   MENU_CLIENT_DATA, win_num,
                   NULL);

 (void) xv_create(canvas_menu_panel, PANEL_BUTTON,
                   PANEL_LABEL_STRING, "Refresh",
                   PANEL_NOTIFY_PROC, refresh_win_proc,
                   PANEL_CLIENT_DATA, win_num,
                   NULL);

  (void) xv_create(canvas_menu_panel, PANEL_BUTTON,
                   PANEL_LABEL_STRING, "Kill Window",
                   PANEL_NOTIFY_PROC, kill_graf_proc,
                   PANEL_CLIENT_DATA, win_num,
                   NULL);


  canvas_info_panel=(Panel)xv_create(graf_frame, PANEL, XV_HEIGHT, 30, NULL);
```



```
  X = (Panel_item)xv_create(canvas_menu_panel, PANEL_MESSAGE,
                            PANEL_NEXT_ROW, -1,
                            PANEL_LABEL_STRING, "X: ",
                            PANEL_VALUE_DISPLAY_LENGTH, 8,
                            NULL);

  Y = (Panel_item)xv_create(canvas_menu_panel, PANEL_MESSAGE,
                            PANEL_LABEL_STRING, "Y: ",
                            XV_X, xv_col(canvas_menu_panel, 15),
                            PANEL_VALUE_DISPLAY_LENGTH, 8,
                            NULL);

  ROW = (Panel_item)xv_create(canvas_menu_panel, PANEL_MESSAGE,
                              PANEL_LABEL_STRING, "ROW#: ",
                              XV_X, xv_col(canvas_menu_panel, 30),
                              PANEL_VALUE_DISPLAY_LENGTH, 8,
                              NULL);

  canvas = (Canvas)xv_create(graf_frame, CANVAS,
                             /* CANVAS_RESIZE_PROC, resize_proc, */
                             CANVAS_RETAINED, TRUE,
                             CANVAS_REPAINT_PROC, redraw_proc,
                             WIN_BELOW, canvas_menu_panel,
                             CANVAS_AUTO_SHRINK, TRUE,
                             CANVAS_AUTO_EXPAND, TRUE,
                             CANVAS_X_PAINT_WINDOW, TRUE,
                             NULL);

  xv_set(canvas_paint_window(canvas),
         WIN_CURSOR, xhair_cursor, NULL);

  xv_set(canvas_paint_window(canvas),
         WIN_EVENT_PROC, canvas_event_proc,
         WIN_CONSUME_EVENTS, WIN_NO_EVENTS, WIN_MOUSE_BUTTONS, LOC_DRAG, LOC_MOVE, NULL,
         NULL);

  xv_set(canvas, XV_KEY_DATA, CAN_X, X, NULL);
  xv_set(canvas, XV_KEY_DATA, CAN_Y, Y, NULL);
  xv_set(canvas, XV_KEY_DATA, CAN_ROW, ROW, NULL);
  xv_set(canvas, XV_KEY_DATA, CAN_NUM, win_num, NULL);
  xv_set(canvas, XV_KEY_DATA, GRAF_FRAME, graf_frame, NULL);

  window_fit(graf_frame);

  can_info = (canvasinfo *) malloc(sizeof(canvasinfo));
  if (can_info == NULL)
    {
      print_msg("Memory allocation error. Window cannot be created.");
      return;
    }

  old_active_window = active_window;
  active_window = win_num;
  can_info->canvas_num = active_window;
  can_info->total_plots = -1;
  can_info->active_plot = -1;

  for (i=0; i<10; i++)
    can_info->alive_plots[i] = 0 ;

  can_info->canvas = canvas;
  xvwin = (Xv_Window)canvas_paint_window(canvas);
  can_info->xvwin = xvwin;
```

```
    win = (Window) xv_get(xvwin, XV_XID);
    can_info->win = win;

    display_active_window(active_window+1);
    display_active_plot(0);

    wininfo.windows[active_window] = 1;
    wininfo.canvases[active_window] = can_info;

    setup_canvas();
    xv_set(graf_frame, XV_SHOW, TRUE, NULL);

/*  sprintf(msg, "(create_canvas) Window #%d is active.\n", active_window+1);
    print_msg(msg);
    sprintf(msg, "(create_canvas)total windows = %d\n", total_windows+1);
    print_msg(msg);

    sprintf(msg, "(create_canvas) Window #%d is old active window.\n", old_active_windo
    print_msg(msg); */

}


get_new_window_num()
{
  int i;

  for (i = 0; i <10; i++)
    {
      if (wininfo.windows[i] == 0)
        return i;
    }
  /* reached plots limit.. need to delete one or more plots from window */
  return(-1);

}



setup_canvas()
{
  canvasinfo *can_info;
  Canvas canvas;
  int start_xaxis, start_yaxis, end_xaxis, end_yaxis;
  int canvas_width, canvas_height;
  int start_x, start_y, end_x, end_y;

  can_info = wininfo.canvases[active_window];
  canvas = can_info->canvas;

  canvas_width = (int) xv_get(canvas, XV_WIDTH);
  canvas_height = (int) xv_get(canvas, XV_HEIGHT);

  /* xaxis is from 0.15 to 0.85 canvas width */
  start_xaxis = (int)canvas_width*0.15;
  end_xaxis = canvas_width - (int)canvas_width*0.15;

/* yaxis is from 0.2 to 0.9 canvas height;
     make room for title at the top (4 lines),
     make room for xaxis labels at the bottom (4 lines)
     and also 2 pixels of space between lines */
  start_yaxis = canvas_height - 5*tickfontheight - 10;
  end_yaxis = titlefontheight*4 + 10;
```



```
  /* plotting area */
  start_x = start_xaxis + canvas_width/20;
  end_x = end_xaxis - canvas_width/20;

  /* plotting area */
  start_y = start_yaxis - canvas_height/20;
  end_y = end_yaxis;

  can_info->start_x = start_x;
  can_info->end_x = end_x;
  can_info->start_y = start_y;
  can_info->end_y = end_y;
  can_info->start_xaxis = start_xaxis;
  can_info->end_xaxis = end_xaxis;
  can_info->start_yaxis = start_yaxis;
  can_info->end_yaxis = end_yaxis;

}
```



```
#include "global.h"
#include <string.h>
#include <sys/file.h>


extern int action;
extern char msg[MSG_LENGTH];
extern char data_file[20], new_file[20];
extern FILE *data, *fopen(), *new;
extern int doit_des, doit_f_open;
extern char pathname[10][80], default_path[80], metapath[80];
extern int name_action;

extern doit_proc();
extern double simp_rate_state_mod();
extern power1(), power2(), normal(), chisqr(), acchisqr(), genexp(), ExpLin(), gensi
clow(), rcph(), Poly4(); /* in func.c */
extern check_row(), mu_check(), act_col(), null_col();
extern tasc(), stdasc(), getaschead();
extern o_slope();

int simp_func_action;


/* temporary.. until i figure out where to put these vars */
int simp_xch[MAX_COL], simp_ych;
int col1, col2, col_out;
int temp_int;
int first, last;
int intval2;
int h, i, j, k, l;
int adj_init_disp;
int simp_xch[MAX_COL], simp_ych; /* used by scm */
static int order; /* must use static, there's order() function in simplex1.c */


float val2;
float dx_increment;
float trend[2];
float test1, test2;
float temp_float;
float init_h;
int temp_col;


double a_max, a_min, dc_max, dc_min, err;
double sl, dx, dmu; /* used by scm */
double *disp_ptr, *mu_ptr; /* used by scm */
static double dvar[MAX_COL];
double convery_tol, lamda, wc;        /* used by do_qi -inversion */

char desire[128];
char t_string[300], t_string_2[300];
char opr, type;
char mu_fit_mess_1[512];       /* help message for scm, cm, rsm */
char mu_fit_mess_2[512];       /* help message for scm, cm, rsm */
char name_arg[128], unit_arg[128];
char *junkp;



/************************** slope ************************/
do_slope(arg)
```



```
    char arg[128];
{
  nocom(arg);

  if (sscanf(arg,"%s %d %d %d %d %d %d %d %s %s", desire, &col1, &col2, &col_out,
st, &temp_int, name_arg, unit_arg) != 9)
    {
      mea();
      action = MAIN;
      top();
      return;
    }

  if(check_row(&first,&last,col1) != 0)
    {
      cre();
      action = MAIN;
      top();
      return;
    }

  if(temp_int > (last - first +1))
    {
      sprintf(msg, "Window can't be bigger than interval for smoothing.\n");
      print_msg(msg);
      action = MAIN;
      top();
      return;
    }

  if(col1 >= max_col || col_out >= max_col || col2 >= max_col)
    {
      coe();
      action = MAIN;
      top();
      return;
    }

  if (name(&col_out,col1, name_arg, unit_arg) == 0)
    {
      ne();
      top();
      action = MAIN;
      return;
    }

  o_slope(col1,col2,col_out,first,last,temp_int); /*in special.c*/

  sprintf(msg, "SLOPE: DONE\n");
  print_msg(msg);

  action = MAIN;
  top();
}


/************************** slope ************************/
/*
do_slope(arg)
    char arg[128];
{
  nocom(arg);

  if (sscanf(arg,"%s %d %d %d", desire, &col1, &col2, &col_out, &temp_int) != 5)
```

```
    {
      nea();
      action = MAIN;
      top();
      return;
    }

  if(col1 >= max_col || col2 >= max_col || col_out >= max_col)
    {
      coe();
      action = MAIN;
      top();
      return;
    }

  slope(&col1,&col2,&col_out,&temp_int) ;

  sprintf(msg, "SLOPE: DONE\n");
  print_msg(msg);

  action = MAIN;
  top();
}
*/


/************************** trig ************************/
do_trig(arg)
     char arg[128];
{
  nocom(arg);

  if (sscanf(arg, "%s %d %d %s %s %s",
          desire, &col1, &col_out, t_string, name_arg, unit_arg) != 6)
    {
      nea();
      action = MAIN;
      top();
      return;
    }

  if(col1 >= max_col || col_out >= max_col)
    {
      coe();
      action = MAIN;
      top();
      return;
    }

  if ( name(&col_out,col1, name_arg, unit_arg) == 0)
    {
      ne();
      top();
      action = MAIN;
      return;
    }

  switch(t_string[0])
    {
    case 's':
      sprintf(msg,"Computing sin...\n");
      print_msg(msg);
      sinvec(&col1,&col_out) ;
```



```
      break;
    case 'c':
      sprintf(msg,"Computing cos...\n");
      print_msg(msg);
      cosvec(&col1,&col_out) ;
      break;
    case 't':
      sprintf(msg,"Computing tan...\n");
      print_msg(msg);
      tanvec(&col1,&col_out) ;
      break;
    case 'a':
      switch(t_string[1])
        {
        case 's':
          sprintf(msg,"Computing asin...\n");
          print_msg(msg);
          asinvec(&col1,&col_out) ;
          break;
        case 'c':
          sprintf(msg,"Computing acos...\n");
          print_msg(msg);
          acosvec(&col1,&col_out) ;
          break;
        case 't':
          sprintf(msg,"Computing atan...\n");
          print_msg(msg);
          atanvec(&col1,&col_out) ;
          break;
        default:
          sprintf(msg,"Func has to be either: sin, cos, tan, asin, acos, atan
\n");
          print_msg(msg);
          break;
        }
    }

  sprintf(msg, "TRIG: DONE\n");
  print_msg(msg);

  action = MAIN;
  top();

}


/************************** rcph ************************/
do_rcph(arg)
     char arg[128];
{
  nocom(arg);

  if (sscanf(arg, "%s %d %d %lf %lf %s %s",
          desire, &col1, &col_out, &dvar[0], &dvar[1], name_arg, unit_arg) != 7)
    {
      nea();
      action = MAIN;
      top();
      return;
    }

  if(col1 >= max_col || col_out >= max_col)
    {
      coe();
```



```
      action = MAIN;
      top();
      return;
    }

  if (name(&col_out,col1, name_arg, unit_arg) == 0)
    {
      ne();
      top();
      action = MAIN;
      return;
    }

  for(i=0; i<head.ch[col1].nelem; ++i)
    darray[col_out][i] = rcph((double)darray[col1][i],dvar[0],dvar[1]);

  sprintf(msg, "RCPH: DONE\n");
  print_msg(msg);

  action = MAIN;
  top();
}


/************************** col_power ************************/
do_col_power(arg)
     char arg[128];
{
  nocom(arg);

  if (sscanf(arg, "%s %d %d %d %s %s",
          desire, &col2, &col1, &col_out) != 6)
    {
      nea();
      action = MAIN;
      top();
      return;
    }

  if(col1 >= max_col || col_out >= max_col)
    {
      coe();
      action = MAIN;
      top();
      return;
    }

  if (name(&col_out,col1, name_arg, unit_arg) == 0)
    {
      ne();
      top();
      action =MAIN;
      return;
    }

  powcvec(&col1,&col_out,&col2) ;

  sprintf(msg, "COL_POWER: DONE\n");
  print_msg(msg);
  action = MAIN;
  top();
}
```



```
/************************** power ************************/
do_power(arg)
     char arg[128];
{
  nocom(arg);

  if (sscanf(arg, "%s %f %d %d %s %s",
          desire, &val2, &col1, &col_out, name_arg, unit_arg) != 6)
    {
      nea();
      action = MAIN;
      top();
      return;
    }

  if(col1 >= max_col || col_out >= max_col)
    {
      coe();
      action = MAIN;
      top();
      return;
    }

  if (name(&col_out,col1, name_arg, unit_arg) == 0)
    {
      ne();
      top();
      action = MAIN;
      return;
    }

  powvec(&col1,&col_out,&val2) ;

  sprintf(msg, "POWER: DONE\n");
  print_msg(msg);

  action = MAIN;
  top();
}

/************************** recip ************************/
do_recip(arg)
     char arg[128];
{
  nocom(arg);

  if (sscanf(arg, "%s %d %d %s %s", desire, &col1, &col_out, name_arg, unit_
    {
      nea();
      action = MAIN;
      top();
      return;
    }

  if(col1 >= max_col || col_out >= max_col)
    {
      coe();
      action = MAIN;
      top();
      return;
    }

  if (name(&col_out,col1, name_arg, unit_arg) == 0)
    {
```

```
        ne();
        top();
        action = MAIN;
        return;
      }

    recipvec(&col1,&col_out) ;

    sprintf(msg, "RECIP: DONE\n");
    print_msg(msg);

    action = MAIN;
    top();
}

/************************** log ***********************/
do_log(arg)
     char arg[128];
{
   nocom(arg);

    if (sscanf(arg, "%s %d %d %s %s",
             desire, &col1, &col_out, name_arg, unit_arg) != 5)
      {
        nea();
        action = MAIN;
        top();
        return;
      }

    if(col1 >= max_col || col_out >= max_col)
      {
        coe();
        action = MAIN;
        top();
        return;
      }


    logvec(&col1,&col_out) ;

    sprintf(msg, "LOG: DONE\n");
    print_msg(msg);

    action = MAIN;
    top();
}

/************************** explin ***********************/
do_ExpLin(arg)
     char arg[128];
{
   nocom(arg);

    if (sscanf(arg, "%s %d %d %lf %lf %lf %s %s",
             desire, &col1, &col_out, &dvar[0], &dvar[1], &dvar[2], name_arg, unit_arg
8)
      {
        nea();
        action = MAIN;
        top();
        return;
      }
```



```
    if(col1 >= max_col || col_out >= max_col)
      {
        coe();
        action = MAIN;
        top();
        return;
      }

    if (name(&col_out,col1, name_arg, unit_arg) == 0)
      {
        ne();
        action = MAIN;
        top();
        return;
      }

    for(i=0; i<head.ch[col1].nelem; ++i)
      darray[col_out][i] = ExpLin((double)darray[col1][i],dvar[0],dvar[1],dvar[2]);
    sprintf(msg, "EXPLIN: DONE\n");
    print_msg(msg);

    action = MAIN;
    top();
}


/************************** poly4 ***********************/
do_Poly4(arg)
     char arg[128];
{
   nocom(arg);

    if (sscanf(arg, "%s %d %d %lf %lf %lf %lf %lf %s %s",
             desire, &col1, &col_out, &dvar[0], &dvar[1], &dvar[2], &dvar[3], &dvar[4
me_arg, unit_arg) != 10)
      {
        nea();
        action = MAIN;
        top();
        return;
      }

    if(col1 >= max_col || col_out >= max_col)
      {
        coe();
        action = MAIN;
        top();
        return;
      }

    if (name(&col_out,col1, name_arg, unit_arg) == 0)
      {
        ne();
        action = MAIN;
        top();
        return;
      }

    for(i=0; i<head.ch[col1].nelem; ++i)
      darray[col_out][i] = Poly4((double)darray[col1][i],dvar[0],dvar[1],dvar[2],dvar[3],dva
r[4]);

    sprintf(msg, "POLY4: DONE\n");
    print_msg(msg);
```



```
    action = MAIN;
    top();
}


/************************** gensin ***********************/
do_gensin(arg)
     char arg[128];
{
   nocom(arg);

    if (sscanf(arg, "%s %d %d %lf %lf %lf %lf %s %s",
             desire, &col1, &col_out, &dvar[0], &dvar[1], &dvar[2], &dvar[3], name_arg
it_arg) != 9)
      {
        nea();
        action = MAIN;
        top();
        return;
      }

    if(col1 >= max_col || col_out >= max_col)
      {
        coe();
        action = MAIN;
        top();
        return;
      }

    if (name(&col_out,col1, name_arg, unit_arg) == 0)
      {
        ne();
        action = MAIN;
        top();
        return;
      }

    for(i=0; i<head.ch[col1].nelem; ++i)
      darray[col_out][i] = gensin((double)darray[col1][i],dvar[0],dvar[1],dvar[2],dvar[3]);

    sprintf(msg, "GENSIN: DONE\n");
    print_msg(msg);

    action = MAIN;
    top();
}


/************************** genexp ***********************/
do_genexp(arg)
     char arg[128];
{
   nocom(arg);

    if (sscanf(arg, "%s %d %d %lf %lf %lf %lf %s %s",
             desire, &col1, &col_out, &dvar[0], &dvar[1], &dvar[2], &dvar[3], name_arg
it_arg) != 9)
      {
        nea();
        action = MAIN;
        top();
        return;
      }
```



```
    if(col1 >= max_col || col_out >= max_col)
      {
        coe();
        action = MAIN;
        top();
        return;
      }

    if (name(&col_out,col1, name_arg, unit_arg) == 0)
      {
        ne();
        action = MAIN;
        top();
        return;
      }

    for(i=0; i<head.ch[col1].nelem; ++i)
      darray[col_out][i] = genexp((double)darray[col1][i],dvar[0],dvar[1],dvar[2],dvar[3]);

    sprintf(msg, "GENEXP: DONE\n");
    print_msg(msg);

    action = MAIN;
    top();
}


/************************** rclow ***********************/
do_rclow(arg)
     char arg[128];
{
   nocom(arg);

    if (sscanf(arg, "%s %d %d %lf %lf %s %s",
             desire, &col1, &col_out, &dvar[0], &dvar[1], name_arg, unit_arg) != 7)
      {
        nea();
        action = MAIN;
        top();
        return;
      }

    if(col1 >= max_col || col_out >= max_col)
      {
        coe();
        action = MAIN;
        top();
        return;
      }

    if (name(&col_out,col1, name_arg, unit_arg) == 0)
      {
        ne();
        action = MAIN;
        top();
        return;
      }

    for(i=0; i<head.ch[col1].nelem; ++i)
      darray[col_out][i] = rclow((double)darray[col1][i],dvar[0],dvar[1]);

    sprintf(msg, "RCLOW: DONE\n");
    print_msg(msg);
```

```
    action = MAIN;
    top();
}


/************************** scchisqr ************************/
do_scchisqr(arg)
     char  arg[128];
{
   nocom(arg);

  if  (sscanf(arg,  "%s  %d  %d  %lf  %lf  %lf  %s  %s",
          desire,  &col1,  &col_out,  &dvar[0],  &dvar[1],  &dvar[2],  name_arg,  unit_arg
 8)
    {
      nea();
      action = MAIN;
      top();
      return;
    }

  if(col1  >=  max_col  ||  col_out  >=  max_col)
    {
      coe();
      action  =  MAIN;
      top();
      return;
    }

  if  (name(&col_out,col1,  name_arg,  unit_arg)  ==  0)
    {
      ne();
      action  =  MAIN;
      top();
      return;
    }

  for(i=0;  i<head.ch[col1].nelem;  ++i)
    darray[col_out][i]  =  (float)scchisqr((double)darray[col1][i],dvar[0],dvar[1],dvar[2]);


  sprintf(msg,  "SCCHISQR:  DONE\n");
  print_msg(msg);

  action  =  MAIN;
  top();
}


/************************** chisqr ************************/
do_chisqr(arg)
     char  arg[128];
{
   nocom(arg);

  if  (sscanf(arg,  "%s  %d  %d  %lf  %s  %s",
          desire,  &col1,  &col_out,  &dvar[0],  name_arg,  unit_arg)  !=  6)
    {
      nea();
      action  =  MAIN;
      top();
      return;
    }
```



```
  if(col1  >=  max_col  ||  col_out  >=  max_col)
    {
      coe();
      action  =  MAIN;
      top();
      return;
    }

  if  (name(&col_out,col1,  name_arg,  unit_arg)  ==  0)
    {
      ne();
      action  =  MAIN;
      top();
      return;
    }

  for(i=0;  i<head.ch[col1].nelem;  ++i)
    darray[col_out][i]  =  (float)chisqr((double)darray[col1][i],dvar[0]);

  sprintf(msg,  "CHISQR:  DONE\n");
  print_msg(msg);

  action  =  MAIN;
  top();
}


/************************** normal ************************/
do_normal(arg)
     char  arg[128];
{
   nocom(arg);

  if  (sscanf(arg,  "%s  %d  %d  %lf  %lf  %s  %s",
          desire,  &col1,  &col_out,  &dvar[0],  &dvar[1],  name_arg,  unit_arg)  !=
    {
      nea();
      action  =  MAIN;
      top();
      return;
    }

  if(col1  >=  max_col  ||  col_out  >=  max_col)
    {
      coe();
      action  =  MAIN;
      top();
      return;
    }

  if  (name(&col_out,col1,  name_arg,  unit_arg)  ==  0)
    {
      ne();
      action  =  MAIN;
      top();
      return;
    }

  for(i=0;  i<head.ch[col1].nelem;  ++i)
    darray[col_out][i]  =  normal((double)darray[col1][i],dvar[0],dvar[1]);

  sprintf(msg,  "NORMAL:  DONE\n");
  print_msg(msg);
```



```
    action  =  MAIN;
    top();
}


/************************** power2 ************************/
do_Power2(arg)
     char  arg[128];
{
   nocom(arg);

  if  (sscanf(arg,  "%s  %d  %d  %lf  %lf  %lf  %s  %s",
          desire,  &col1,  &col_out,  &dvar[0],  &dvar[1],  &dvar[2],  name_arg,  unit_arg
 8)
    {
      nea();
      action  =  MAIN;
      top();
      return;
    }

  if(col1  >=  max_col  ||  col_out  >=  max_col)
    {
      coe();
      action  =  MAIN;
      top();
      return;
    }

  if  (name(&col_out,col1,  name_arg,  unit_arg)  ==  0)
    {
      ne();
      action  =  MAIN;
      top();
      return;
    }

  for(i=0;  i<head.ch[col1].nelem;  ++i)
    darray[col_out][i]  =  power2((double)darray[col1][i],dvar[0],dvar[1],dvar[2]);

  sprintf(msg,  "POWER2:  DONE\n");
  print_msg(msg);

  action  =  MAIN;
  top();
}


/************************** power1 ************************/
do_Power1(arg)
     char  arg[128];
{
   nocom(arg);

  if  (sscanf(arg,  "%s  %d  %d  %lf  %lf  %s  %s",
          desire,  &col1,  &col_out,  &dvar[0],  &dvar[1],  name_arg,  unit_arg)  !=  7)
    {
      nea();
      action  =  MAIN;
      top();
      return;
    }
```



```
  if(col1  >=  max_col  ||  col_out  >=  max_col)
    {
      coe();
      action  =  MAIN;
      top();
      return;
    }

  if  (name(&col_out,col1,  name_arg,  unit_arg)  ==  0)
    {
      ne();
      action  =  MAIN;
      top();
      return;
    }

  for(i=0;  i<head.ch[col1].nelem;  ++i)
    darray[col_out][i]  =  power1((double)darray[col1][i],dvar[0],dvar[1]);

  sprintf(msg,  "POWER1:  DONE\n");
  print_msg(msg);

  action  =  MAIN;
  top();
}


/************************** ln ************************/
do_ln(arg)
     char  arg[128];
{
   nocom(arg);
  if  (sscanf(arg,  "%s  %d  %d  %s  %s",
          desire,  &col1,  &col_out,  name_arg,  unit_arg)  !=  5)
    {
      nea();
      action  =  MAIN;
      top();
      return;
    }

  if(col1  >=  max_col  ||  col_out  >=  max_col)
    {
      coe();
      action  =  MAIN;
      top();
      return;
    }

  if  (name(&col_out,col1,  name_arg,  unit_arg)  ==  0)
    {
      ne();
      action  =  MAIN;
      top();
      return;
    }

  lnvec(&col1,&col_out)  ;

  sprintf(msg,  "LN:  DONE\n");
  print_msg(msg);

  action  =  MAIN;
  top();
```

```
}

/************************** exp ************************/
do_exp(arg)
    char  arg[128];
{
 nocom(arg);
 if  (sscanf(arg,  "%s  %d  %d  %s  %s",
          desire,  &col1,  &col_out,  name_arg,  unit_arg)  != 5)
  {
    nea();
    action  =  MAIN;
    top();
    return;
  }

 if(col1  >=  max_col  ||  col_out  >=  max_col)
  {
    coe();
    action  =  MAIN;
    top();
    return;
  }

 if  (name(&col_out,col1,  name_arg,  unit_arg)  ==  0)
  {
    ne();
    action  =  MAIN;
    top();
    return;
  }

 expvec(&col1,&col_out)  ;

 sprintf(msg,  "EXP:  DONE\n");
 print_msg(msg);

 action  =  MAIN;
 top();
}


/************************** deriv ************************/
do_deriv(arg)
    char  arg[128];
{
  /*finite  difference  derivative  using  central  differences,
    taylor  expansion  about  f(x),  and  assuming  the  data  is  unequally  spaced*/
 nocom(arg);
 if  (sscanf(arg,  "%s  %d  %d  %d  %d  %d  %s  %s",
          desire,  &col1,  &col2,  &col_out,  &i,  &j,  name_arg,  unit_arg)  != 8)
  {
    nea();
    action  =  MAIN;
    top();
    return;
  }

 if(col1  >=  max_col  ||  col2  >=  max_col  ||  col_out  >=  max_col)
  {
    coe();
    action  =  MAIN;
    top();
```



```
    return;
  }

 if(i>j  ||  i<0  ||  i>head.nrec-1  ||  j>head.nrec-1)
  {
    sprintf(msg,  "Undefined  row  interval.\n");
    print_msg(msg);
    action  =  MAIN;
    top();
    return;
  }

 if  (name(&col_out,col1,  name_arg,  unit_arg)  ==  0)
  {
    ne();
    action  =  MAIN;
    top();
    return;
  }

 deriv(&col1,&col2,&col_out,i,j);

 sprintf(msg,  "DERIV:  DONE\n");
 print_msg(msg);

 action  =  MAIN;
 top();
}


/************************** vc ************************/
do_vc(arg)
    char  arg[128];
{
  /*  corrrect  porosity/volume  strain  (during  loading/unloading)  for  the  effect  of
g  pressure  on  the  material  at  the  edges  of  the  layer.  User  provides  a  compress
/V/dP)  and  volume  which  is  appropriate  to  the  "edge
material"  undergoing  pseudo  elastic  volume  change  during  loading/unloading  -a  good
or  the  volume  of  material
affected  by  Pc  during  load/unload  is  an  anular  -elliptical  area-  of  width  =  lay
s  *  thickness*/
  /*  use  tau  instead  of  Pc,  because  we're  sure  to  have  that,  Pc  may  have  been

 nocom(arg);
 if  (sscanf(arg,  "%s  %d  %d  %d  %d  %d  %f  %s  %s",
          desire,  &col1,  &col2,  &col_out,  &i,  &j,  &temp_float,  name_arg,  unit_arg
)
  {
    nea();
    action  =  MAIN;
    top();
    return;
  }

 if(col1  >=  max_col  ||  col2  >=  max_col  ||  col_out  >=  max_col)
  {
    coe();
    action  =  MAIN;
    top();
    return;
  }

 if(i>j  ||  i<0  ||  i>head.nrec-1  ||  j>head.nrec-1)
  {
```



```
    sprintf(msg,  "Undefined  row  interval.\n");
    print_msg(msg);
    action  =  MAIN;
    top();
    return;
  }

 if  (name(&col_out,col1,  name_arg,  unit_arg)  ==  0)
  {
    ne();
    action  =  MAIN;
    top();
    return;
  }

 vol_corr(col1,col2,col_out,i,j,temp_float);

 sprintf(msg,  "VC:  DONE\n");
 print_msg(msg);

 action  =  MAIN;
 top();
}


/************************** rgt ************************/
do_rgt(arg)
    char  arg[128];
{
  /*  correct  horizontal  displacement  measurement  during  direct  shear  test  for  "gec
binning"  */
  /*  correction  is:  del_h  =  h  dx/ZL  ;  where  h  is  thickness,  dx  is  slip  increm
s  length  of  the  sliding  block  parallel  to  slip  */

 nocom(arg);
 if  (sscanf(arg,  "%s  %f  %d  %d  %d  %s  %s",  desire,  &temp_float,  &col1,  &col2,  &cc
e_arg,  unit_arg)  != 7)
  {
    nea();
    action  =  MAIN;
    top();
    return;
  }

 if(col1  >=  max_col  ||  col2  >=  max_col  ||  col_out  >=  max_col)
  {
    coe();
    action  =  MAIN;
    top();
    return;
  }

 if(col1  ==  col_out)
  {
    sprintf(msg,  "Sorry,  routine  is  too  dumb  to  allow  input  col.  to  equal  outpu
Choose  another  column  for  the  output\n");
    action  =  MAIN;
    top();
    return;
  }

 if  (name(&col_out,col2,  name_arg,  unit_arg)  ==  0)
  {
    ne();
```



```
    action  =  MAIN;
    top();
    return;
  }

 geom_thin(col1,col2,col_out,temp_float);

 sprintf(msg,  "RGT:  DONE\n");
 print_msg(msg);

 action  =  MAIN;
 top();
}


/************************** cgt ************************/
do_cgt(arg)
    char  arg[128];
{
  /*  calculate  geometric  thinning  */
  /*  calculation  is:  del_h  =  h  dx/2L  ;  where  h  is  input  (not  real)  thickness,
  increment  and  L  is  length  of  the  sliding  block  parallel  to  slip  */

 nocom(arg);
 if  (sscanf(arg,  "%s  %f  %f  %d  %d  %s  %s",  desire,  &temp_float,  &init_h,  &col1,
ame_arg,  unit_arg)  != 7)
  {
    nea();
    action  =  MAIN;
    top();
    return;
  }

 if(col1  >=  max_col  ||  col_out  >=  max_col)
  {
    coe();
    action  =  MAIN;
    top();
    return;
  }

 if  (name(&col_out,col1,  name_arg,  unit_arg)  ==  0)
  {
    ne();
    action  =  MAIN;
    top();
    return;
  }

 calc_geom_thin(col1,col_out,temp_float,init_h);

 sprintf(msg,  "CGT:  DONE\n");
 print_msg(msg);

 action  =  MAIN;
 top();
}


/************************** ec ************************/
do_ec(arg)
    char  arg[128];
{
 nocom(arg);
```

```
  if (sscanf(arg, "%s %d %d %d %d %d %f %s %s", desire, &col1, &col2, &col_out,
emp_float, name_arg, unit_arg) != 9)
    {
     nea();
     action = MAIN;
     top();
     return;
    }

  if(col1 >= max_col || col2 >= max_col || col_out >= max_col)
    {
     coe();
     action = MAIN;
     top();
     return;
    }

 if(check_row(&i,&j,col1) != 0)
   {
    cre();
    action = MAIN;
    top();
    return;
   }

 if (name(&col_out,col1, name_arg, unit_arg) == 0)
   {
    ne();
    action = MAIN;
    top();
    return;
   }

 elastic_corr(col1,col2,col_out,i,j,temp_float);            /*in array.c*/

 sprintf(msg, "EC: DONE\n");
 print_msg(msg);

 action = MAIN;
 top();
}


/************************** cs ***********************/
do_cs(arg)
    char arg[128];
{
 nocom(arg);
 if (sscanf(arg, "%s %d %d %d %d %d %s %s", desire, &col1, &col2, &col_out, &i
arg, unit_arg) != 8)
   {
    nea();
    action = MAIN;
    top();
    return;
   }

 if(col1 >= max_col || col2 >= max_col || col_out >= max_col)
   {
    coe();
    action = MAIN;
    top();
    return;
   }
```



```
 if(check_row(&i,&j,col1) != 0)
   {
    cre();
    action = MAIN;
    top();
    return;
   }

 if (name(&col_out,col1, name_arg, unit_arg) == 0)
   {
    ne();
    action = MAIN;
    top();
    return;
   }

 shear_strain(col1,col2,col_out,i,j);  /*in array.c*/

 sprintf(msg, "CS: DONE\n");
 print_msg(msg);

 action = MAIN;
 top();
}



/************************** polyfit ***********************/
do_polyfit(arg)
    char arg[128];
{
 double *px, *py, *pw, *pjml, *pj, *error, ortval();

 nocom(arg);

 if (sscanf(arg, "%s %d %d %d %d %d %d %s %d %s %s", desire, &col1, &co
er, &i, &j, t_string, &temp_int, name_arg, unit_arg) != 11)
   {
    nea();
    action = MAIN;
    top();
    return;
   }

 if(col1 >= max_col || col2 >= max_col || col_out >= max_col)
   {
    coe();
    action = MAIN;
    top();
    return;
   }

 if(check_row(&i,&j,col1) != 0)
   {
    cre();
    action = MAIN;
    top();
    return;
   }

 if(strncmp(desire,"polyfit_i",9) == 0)
   {
    if(t_string[0] == 'd')
```



```
       {
        stats(&col1,&i,&j) ;
        dx_increment= (col_stat.max-col_stat.min)/(j-i+1);
       }
     else
       {
        dx_increment = atof(t_string);
        t_string[0] = 'y';   /*so we know to extend below*/
       }
   }
 else
   {
    if (t_string[0] == 'a' )
      {
       sprintf(msg, "Applying fit to entire col.\n");
       print_msg(msg);
      }
   }

 if(temp_int>head.nrec-1)
   {
    sprintf(msg, "Undefined row interval.\n");
    print_msg(msg);
    action = MAIN;
    top();
    return;
   }

 if (name(&col_out,col1, name_arg, unit_arg) == 0)
   {
    ne();
    action = MAIN;
    top();
    return;
   }

 l = (j-i)+1;
 px= (double *) calloc(l,sizeof(double));
 py= (double *) calloc(l,sizeof(double));
 pw= (double *) calloc(l,sizeof(double));
 pjml= (double *) calloc(l,sizeof(double));
 pj= (double *) calloc(l,sizeof(double));
 error= (double *) calloc(l,sizeof(double));

 for(k=0;k<l;++k)
   {
    px[k] = darray[col1][i+k];
    py[k] = darray[col2][i+k];
    pw[k] = 1.0;
   }

 ortpol(px,py,pw,l,pjml,pj,error,order);  /*fit*/

 /* write y values for a the x's in a given row or if interpolating (polyfit_
e y for some given dx */

 if(strncmp(desire,"polyfit_i",9) == 0)
   {
    if(t_string[0] == 'y') /*extend the fit past the "fitted" interval*/
      {
       for(k=0;k<=(temp_int-i);++k)
         {
          darray[col_out][i+k] = ortval(darray[col1][i]+(k*dx_increment));
         }
```



```
       }
     else                          /*using the default increment take */
                                   /*the fit only to the end of the */
                                   /*fitted interval               */
      {
       for(k=0;(i+k)<=j;++k)
         {
          darray[col_out][i+k] = ortval(darray[col1][i]+(k*dx_increment));
         }
       fprintf(stderr,"the dx increment used was %f\n",dx_increment);
      }
   }
 else
   {
    if(t_string[0] == 'y') /*extend the fit past the "fitted" interval*/
      {
       for(k=0;k<=(temp_int-i);++k)
         {
          darray[col_out][i+k] = ortval(darray[col1][i+k]);
         }
      }
    else if(t_string[0] == 'a') /* apply fit to entire col */
      {
       for(k=0;k<head.ch[col1].nelem;++k)
         {
          darray[col_out][k] = ortval(darray[col1][k]);
         }
      }
    else
      {
       for(k=0;(i+k)<=j;++k)
         {
          darray[col_out][i+k] = ortval(darray[col1][i+k]);
         }
      }
   }
 cfree(px);
 cfree(py);
 cfree(pw);
 cfree(pjml);
 cfree(pj);
 cfree(error);

 sprintf(msg, "POLYFIT: DONE\n");
 print_msg(msg);

 action = MAIN;
 top();
}



/************************** sort ***********************/
do_sort(arg)
    char arg[128];
{
 float ascend;
 float *good_points;

 nocom(arg);

 if (sscanf(arg, "%s %d %d %d", desire, &col1,&first,&last) != 4)
   {
    nea();
```

```
    action = MAIN;
    top();
    return;
  }

ascend=1.0;
if(col1 <0)
  {
    col1 *= -1;
    ascend = -1.0;
  }

if(col1 >= max_col)
  {
    coe();
    action = MAIN;
    top();
    return;
  }

if(check_row(&first,&last,col1) != 0)
  {
    cre();
    action = MAIN;
    top();
    return;
  }

good_points = darray[0];      /*use col 0 of look table, store ints as floats*/
good_points[0] = first;              /* first point is "in order" by def. */
for(i=first, j=0; i<last; )
  {
    k=1;
    while((ascend*darray[col1][i+k] <= ascend*darray[col1][i]) && (i+k<=last))
      k++;          /*find next point that's not out-of-order*/
    i += k;
    if(i <= last)
      good_points[++j] = (float)i;    /*skip points that are out of order*/
  }

temp_int = last-first-j;                    /* number of points to ax */

for(i=last;i<head.ch[col1].nelem;i++)
  good_points[++j] = (float)i;              /*skip points out of order*/

for(h=1;h<max_col;++h)        /*loop over cols*/
  {
    for(k=first, j=0; k<head.ch[h].nelem-temp_int; k++)
      darray[h][k] = darray[h][(int)(good_points[j++])];      /*put next good row hea
    head.ch[h].nelem -= temp_int;                    /*reduce nelems*/
  }

head.nrec -= temp_int;

sprintf(msg, "SORT: DONE\n");
print_msg(msg);

action = MAIN;
top();

}

/************************** interpolate ************************/
```



```
do_interpolate(arg)
     char arg[128];
{
  char name_arg1[128], unit_arg1[128];

  nocom(arg);

  if (sscanf(arg, "%s %d %d %d %d %d %d %s %s %s %s %s",
          desire, &col1, &col2, &temp_int, &col_out, &first, &last, t_string, name
  unit_arg, name_arg1, unit_arg1) != 12)
    {
      nea();
      action = MAIN;
      top();
      return;
    }

  if(col1 >= max_col || col2 >= max_col || col_out >= max_col || temp_int >= ma
    {
      coe();
      action = MAIN;
      top();
      return;
    }

  if(check_row(&first,&last,col1) != 0)
    {
      cre();
      action = MAIN;
      top();
      return;
    }

  if(t_string[0] == 'd')
    {
      stats(&col1, &first, &last) ;
      dx_increment= (col_stat.max-col_stat.min)/(last-first+1);
      sprintf(msg, "Interpolation interval is %f\n",dx_increment);
      print_msg(msg);
    }
  else
    dx_increment = atof(t_string);

  if (name(&temp_int, col1, name_arg, unit_arg) == 0)
    {
      ne();
      action = MAIN;
      top();
      return;
    }

  if (name(&col_out, col1, name_arg1, unit_arg1) == 0)
    {
      ne();
      action = MAIN;
      top();
      return;
    }

  i=first;
  k=first+1;          /*leave first point "as is"*/
  darray[col_out][first] = darray[col2][first];        /**new_y**/
  darray[temp_int][first] = darray[col1][first];              /**new_x**/
```



```
  while(k<=last)
    {
      if(darray[temp_int][i]+dx_increment > darray[col1][k])   /*find relevant interval
or dx*/
        ++k;
      else
        {
          darray[col_out][i+1] = darray[col2][k-1] + ( (darray[col2][k]-darray[col2][k-1
 * (darray[temp_int][i]+dx_increment-darray[col1][k-1]) / (darray[col1][k]-darray[col1][k
1]) );
          darray[temp_int][i+1] = darray[temp_int][i]+dx_increment;
          ++i;
        }
    }

  i = i-first+1;
  sprintf(msg, "\t%d points written to cols %d and %d\n",i,temp_int,col_out);
  print_msg(msg);


  /*allow interpolate_r or interpolater*/
  if(desire[(strlen((char *)desire)-1)] == 'r')
    {
      sprintf(msg, "Doing auto row remove from %d to %d --end of interpolated val
d of interpolation interval.\n",first+i,last);
      print_msg(msg);

      /* set up tmp doit file */
      strcpy(pathname[doit_f_open+1], default_path);
      strcat(pathname[doit_f_open+1],"temp_junk_doit_file");
      com_file[doit_f_open+1] = fopen(pathname[doit_f_open+1],"w");
      fprintf(com_file[doit_f_open+1],"begin\nr_row %d %d\nend\n", first+i, last);
      fclose(com_file[doit_f_open+1]);

      doit_des = open(pathname[doit_f_open+1],O_RDONLY) ;

      /* call doit_proc(); set action to MAIN first so that it will not go back
later if doit fails */
      action = MAIN;
      doit_proc("temp_junk_doit_file");
    }

  sprintf(msg, "INTERPOLATE: DONE\n");
  print_msg(msg);

  action = MAIN;
  top();
}


/************************** math ************************/
do_math(arg)
     char arg[128];
{
  nocom(arg);

  if (sscanf(arg, "%s %d %c %f %c %d %s %s", desire, &col1, &opr, &val2, &type,
name_arg, unit_arg) != 8)
    {
      nea();
      action = MAIN;
      top();
      return;
    };
```



```
  if(col1 >= max_col || col_out >= max_col)
    {
      sprintf(msg, "Not enough array space\nReallocate array for more columns\n");
      print_msg(msg);
      action = MAIN;
      top();
      return;
    }

  if((type == ':') && ((int)val2 >= max_col))
    {
      sprintf(msg, "Not enough array space\nReallocate array for more columns\n");
      print_msg(msg);
      action = MAIN;
      top();
      return;
    }

  i = 0;
  if(type == ':')
    j = (head.ch[col1].nelem > head.ch[(int)val2].nelem) ? head.ch[col1].nelem : hea
int)val2].nelem ;
  else
    j = head.ch[col1].nelem;

  math_final(name_arg, unit_arg);
}

do_mathint(arg)
     char arg[128];
{
  nocom(arg);

  if (sscanf(arg, "%s %d %c %f %c %d %d %d %s %s",
          desire, &col1, &opr, &val2, &type, &col_out, &i, &j, name_arg, unit_a
0)
    {
      nea();
      action = MAIN;
      top();
      return;
    }

  if(i>j || i<0 || i>head.nrec-1 || j>head.nrec-1)
    {
      sprintf(msg, "Undefined row interval\n");
      print_msg(msg);
      action = MAIN;
      top();
      return;
    }

  if(col1 >= max_col || col_out >= max_col)
    {
      sprintf(msg, "Not enough array space\nReallocate array for more columns\n");
      print_msg(msg);
      action = MAIN;
      top();
      return;
    }

  if((type == ':') && ((int)val2 >= max_col))
    {
```

```
        sprintf(msg, "Not enough array space\nReallocate array for more columns\n");
        print_msg(msg);
        action = MAIN;
        top();
        return;
      }
    math_final(name_arg, unit_arg);
  }

  math_final(name_arg, unit_arg)
  {
    if(type == ':' && (head.ch[col1].nelem < head.ch[(int)val2].nelem) )
      {
        if (name(&col_out,(int)val2, name_arg, unit_arg) == 0)
          {
            ne();
            action = MAIN;
            top();
            return;
          }
      }

    else
      if (name(&col_out,col1, name_arg, unit_arg) == 0)
        {
          ne();
          action = MAIN;
          top();
          return;
        }

    switch(opr)
      {
      case '+':
        intval2 = (int)val2 ;
        add(&col1,&intval2,&val2,&col_out,&type,i,j) ;
        break;
      case '-':
        intval2 = (int)val2 ;
        sub(&col1,&intval2,&val2,&col_out,&type,i,j) ;
        break;
      case '*':
        intval2 = (int)val2 ;
        prod(&col1,&intval2,&val2,&col_out,&type,i,j) ;
        break;
      case '/':
        intval2 = (int)val2 ;
        div(&col1,&intval2,&val2,&col_out,&type,i,j) ;
        break ;
      default :
        sprintf(msg, "Illegal operation\n") ;
        print_msg(msg);
      }

    sprintf(msg, "MATH: DONE\n");
    print_msg(msg);

    action = MAIN;
    top();
  }


  /************************** r_spike ***********************/
  do_r_spike(arg)
```



```
      char arg[128];
  {
    nocom(arg);

    if (sscanf(arg, "%s %d %d", desire, &col1, &first) != 3)
      {
        nea();
        top();
        action = MAIN;
        return;
      }

    if( col1 >= max_col)
      {
        coe();
        top();
        action = MAIN;
        return;
      }
    if (first == 0 || first >= head.ch[col1].nelem)
      {
        sprintf(msg, "Spike can't be > n_recs or 0!\n");
        print_msg(msg);
        top();
        action = MAIN;
        return;
      }
    darray[col1][first] = darray[col1][first-1] ;

    sprintf(msg, "R_SPIKE: DONE\n");
    print_msg(msg);

    action = MAIN;
    top();
  }

  /************************** positive ***********************/
  do_positive(arg)
      char arg[128];
  {
    nocom(arg);

    if (sscanf(arg, "%s %d %d %s %s", desire, &col1, &col_out, name_arg, unit_a
      {
        nea();
        top();
        action = MAIN;
        return;
      }

    if( col1 >= max_col || col_out >= max_col)
      {
        coe();
        top();
        action = MAIN;
        return;
      }

    if (name(&col_out,col1, name_arg, unit_arg) == 0)
      {
        ne();
        action = MAIN;
        top();
        return;
```



```
      }

    for(i=0; i<head.ch[col_out].nelem; ++i)
      {
        if(darray[col1][i] > 0.0) darray[col_out][i] = 1.0 ;
        else darray[col_out][i] = 0.0 ;
      }

    sprintf(msg, "POSITIVE: DONE\n");
    print_msg(msg);

    action = MAIN;
    top();
  }


  /************************** compress ***********************/
  do_compress(arg)
      char arg[128];
  {
    nocom(arg);

    if (sscanf(arg, "%s %d %d %s %s", desire, &col1, &col_out, name_arg, unit_arg)
      {
        nea();
        top();
        action = MAIN;
        return;
      }

    if( col1 >= max_col || col_out >= max_col)
      {
        coe();
        top();
        action = MAIN;
        return;
      }

    if (col1 == col_out)
      {
        sprintf(msg, "Can't use same column\n");
        print_msg(msg);
        top();
        action = MAIN;
        return;
      }

    if (name(&col_out,col1, name_arg, unit_arg) == 0)
      {
        ne();
        action = MAIN;
        top();
        return;
      }

    j=0;
    for(i=0; i<head.ch[col_out].nelem; ++i)
      {
        if(darray[col1][i] != 0.0)
          {
            darray[col_out][j] = darray[col1][i] ;
            j+=1;
          }
      }
```



```
    head.ch[col_out].nelem = j;

    sprintf(msg, "COMPRESS: DONE\n");
    print_msg(msg);

    action = MAIN;
    top();
  }


  /************************** r_mean ***********************/
  /* r_mean, r_min, r_max call sub() in array.c */

  do_r_mean(arg)
      char arg[128];
  {
    nocom(arg);

    if (sscanf(arg, "%s %d %d %s %s", desire, &col1, &col_out, name_arg, unit_arg
      {
        nea();
        top();
        action = MAIN;
        return;
      }

    if( col1 >= max_col || col_out >= max_col)
      {
        coe();
        top();
        action = MAIN;
        return;
      }

    i = 0 ;
    j = head.ch[col1].nelem -1;
    stats(&col1,&i,&j) ;
    type = '=' ;
    sub(&col1,&col1,&(col_stat.mean),&col_out,&type,i,j) ;

    if (name(&col_out,col_out, name_arg, unit_arg) == 0)
      {
        ne();
        action = MAIN;
        top();
        return;
      }

    sprintf(msg, "R_MEAN: DONE\n");
    print_msg(msg);

    action = MAIN;
    top();

  }


  /************************** r_min ***********************/
  do_r_min(arg)
      char arg[128];
  {
    nocom(arg);

    if (sscanf(arg, "%s %d %d %s %s", desire, &col1, &col_out, name_arg, unit_arg)
```

```
    {
      nea();
      top();
      action = MAIN;
      return;
    }

  if( col1 >= max_col || col_out >= max_col)
    {
      coe();
      top();
      action = MAIN;
      return;
    }
  i = 0 ;
  j = head.ch[col1].nelem -1;
  stats(&col1,&i,&j) ;
  type = 'x' ;
  sub(&col1,&col1,&(col_stat.min),&col_out,&type,i,j) ;

  if (name(&col_out,col_out, name_arg, unit_arg) == 0)
    {
      ne();
      action = MAIN;
      top();
      return;
    }

  sprintf(msg, "Z_MIN: DONE\n");
  print_msg(msg);

  action = MAIN;
  top();

}


/************************** z_max ************************/
do_z_max(arg)
     char arg[128];
{
  nocom(arg);

  if (sscanf(arg, "%s %d %d %s %s", desire, &col1, &col_out, name_arg, unit_arg)
    {
      nea();
      top();
      action = MAIN;
      return;
    }

  if( col1 >= max_col || col_out >= max_col)
    {
      coe();
      top();
      action = MAIN;
      return;
    }
  i = 0 ;
  j = head.ch[col1].nelem -1;
  stats(&col1,&i,&j) ;
  type = 'x' ;
  sub(&col1,&col1,&(col_stat.max),&col_out,&type,i,j) ;
```



```
  if (name(&col_out,col_out, name_arg, unit_arg) == 0)
    {
      ne();
      action = MAIN;
      top();
      return;
    }

  sprintf(msg, "Z_MAX: DONE\n");
  print_msg(msg);

  action = MAIN;
  top();

}


/************************** trend ************************/
do_trend(arg)
     char arg[128];
{
  nocom(arg);

  if (sscanf(arg, "%s %d %d %d %d", desire, &col1, &col2, &i, &j) != 5)
    {
      nea();
      top();
      action = MAIN;
      return;
    }

  if (col1 >= max_col || col2 >= max_col)
    {
      coe();
      top();
      action = MAIN;
      return;
    }

  if ( i > head.nrec || i > j )
    {
      sprintf(msg, "Row %d not active.\n", i) ;
      print_msg(msg);
      top();
      action = MAIN;
      return;
    }

  if ( j > head.nrec )
    {
      j = head.nrec -1 ;
      sprintf(msg, "Total number of records is %d!\n",head.nrec) ;
    }
  do_final_trend();
}

do_trend_a(arg)
     char arg[128];
{
  nocom(arg);

  if (sscanf(arg, "%s %d %d", desire, &col1, &col2) != 3)
    {
      nea();
```



```
      top();
      action = MAIN;
      return;
    }

  if (col1 >= max_col || col2 >= max_col)
    {
      sprintf(msg, "ERROR: Column not allocated.\n") ;
      print_msg(msg);
      top();
      action = MAIN;
      return;
    }

  i = 0;
  j = (head.ch[col1].nelem < head.ch[col2].nelem) ? head.ch[col1].nelem -1 : head.ch
.nelem -1;

  do_final_trend();
}


do_final_trend()
{
  line(&col1,&col2,&i,&j,trend);

  sprintf(msg, "\ntrend: X = %s , Y = %s ; records %d to %d\n",head.ch[col1].nam
col2].name,i,j);
  print_msg(msg);

  sprintf(msg, "trend: slope = %g , intercept = %g\n\n",trend[0],trend[1]);
  print_msg(msg);

  sprintf(msg, "TREND: DONE\n");
  print_msg(msg);

  action = MAIN;
  top();
}


/************************** r_trend ************************/
do_r_trend_input(arg)
     char arg[128];
{
  nocom(arg);

  if (sscanf(arg, "%s %s %s %d %d %d %f %f %s %s", desire, t_string, t_string_
ol2, &col_out, &trend[0], &trend[1], name_arg, unit_arg) != 10)
    {
      nea();
      top();
      action = MAIN;
      return;
    }

  if(col1 >= max_col || col_out >= max_col || col2 >= max_col)
    {
      coe();
      action = MAIN;
      top();
      return;
    }
```



```
  first = 0;

  do_r_trend_final();

  sprintf(msg, "R_TREND: DONE\n");
  print_msg(msg);

  action = MAIN;
  top();
}


do_r_trend_comp(arg)
     char arg[128];
{
  nocom(arg);

  if (sscanf(arg, "%s %s %s %d %d %d %d %d %s %s", desire, t_string, t_string
ol2, &col_out, &h, &j, name_arg, unit_arg) != 10)
    {
      nea();
      top();
      action = MAIN;
      return;
    }

  if(col1 >= max_col || col_out >= max_col || col2 >= max_col)
    {
      coe();
      action = MAIN;
      top();
      return;
    }

  if(check_row(&h,&j,col1) != 0)
    {
      cre();
      action = MAIN;
      top();
      return;
    }

  first = h;
/* line(&col1,&col2,&h,&j,trend); */
  sprintf(msg, "slope = %g , intercept = %g\n",trend[0],trend[1]);
  print_msg(msg);

  do_r_trend_final();

  sprintf(msg, "R_TREND: DONE\n");
  print_msg(msg);

  action = MAIN;
  top();
}


do_r_trend_final()
{

  temp_col = 0 ;
  for(i=0; i<head.nrec; ++i) darray[0][i] = 0.0 ;
  /* use h and j as limits for array functions */
  h = 0 ;
```

```
    j = head.nrec - 1 ;
    type = '=' ;

    if (t_string[0] == 'X' || t_string[0] == 'x')
      {
       sprintf(msg, "Detrending x values\n");
       print_msg(msg);
       sub(&col2,&temp_col,&(trend[1]),&temp_col,&type,h,j) ;
       div(&temp_col,&temp_col,&(trend[0]),&temp_col,&type,h,j) ;
       type = ':' ;
       sub(&col1,&temp_col,&val2,&col_out,&type,h,j) ;
      }
    else
      {
       sprintf(msg, "Detrending y values\n");
       print_msg(msg);
       /* sub(&col1,&temp_col,&(darray[col1][h]),&temp_col,&type,h,j) ; */
       prod(&col1,&temp_col,&(trend[0]),&temp_col,&type,h,j) ;
       add(&temp_col,&temp_col,&(trend[1]),&temp_col,&type,h,j) ;
       type = ':' ;
       sub(&col2,&temp_col,&val2,&col_out,&type,h,j) ;
      }

    i = 0 ;
    j = head.nrec - 1 ;
    stats(&col_out,&i,&j) ;

   if (name(&col_out,col2, name_arg, unit_arg) == 0)
     {
      ne();
      action = MAIN;
      top();
      return;
     }

   sprintf(msg, "stats for new column # %d\n",col_out);
   print_msg(msg);
   sprintf(msg, "mean = %f , max = %f , min = %f , stddev = %f\n",col_stat.mean
x,col_stat.min,col_stat.stddev) ;
   print_msg(msg);

   /* re-zero temporary column */
   for(i=0; i<head.nrec; ++i) darray[0][i] = 0.0 ;

   /*allow r_trend_o or r_trendo */
   if(desire[(strlen((char *)desire)-1)] == 'o')
     {
      sprintf(msg, "Doing auto offset of col %d to = col %d at row %d\n",col_out
t);
      print_msg(msg);

      strcpy(pathname[doit_f_open+1], default_path);
      strcat(pathname[doit_f_open+1],"temp_junk_doit_file");
      com_file[doit_f_open+1] = fopen(pathname[doit_f_open+1],"w");
      fprintf(com_file[doit_f_open+1],"begin\noffset\n%d,%d,%d,%d\nend\n", col_out, first,
 col2, first);
      fclose(com_file[doit_f_open+1]);

      doit_des = open(pathname[doit_f_open+1],O_RDONLY) ;
      action = MAIN;
      doit_proc("temp_junk_doit_file");
     }

   sprintf(msg, "R_TREND: DONE\n");
```



```
    print_msg(msg);

    action = MAIN;
    top();

  }


  /************************** findint ***********************/
  do_findint(arg)
      char arg[128];
  {
   nocom(arg);

   if (sscanf(arg, "%s %d %d %d", desire, &col1, &i, &j) != 4)
     {
      nea();
      top();
      action = MAIN;
      return;
     }

   if ( i > head.nrec || i > j )
     {
      sprintf(msg, "Rows not active.\n") ;
      print_msg(msg);
      top();
      action = MAIN;
      return;
     }

   if ( j > head.nrec )
     {
      j = head.nrec ;
      sprintf(msg, "Total number of records is %d!\n",head.nrec) ;
      print_msg(msg);
     }

   test1 = 1 ;
   for( k = i; k < j; ++k)
     {
      test2 = test1 ;
      test1 = darray[col1][k] - darray[col1][k+1] ;
      if((test2*test1) < 0.0)
        {
         sprintf(msg, "findint: record %d\n",k) ;
         print_msg(msg);
        }
     }

   sprintf(msg, "FINDINT: DONE\n");
   print_msg(msg);

   top();
   action = MAIN;
  }


  /************************** summation ***********************/
  /* calls summation() in array.c */
  do_summation(arg)
      char arg[128];
  {
   nocom(arg);
```



```
   if (sscanf(arg, "%s %d %d %s %s" , desire, &col1, &col_out, name_arg, unit_arg)
     {
      nea();
      top();
      action = MAIN;
      return;
     }

   if (col1 >= max_col || col_out >= max_col)
     {
      coe();
      action = MAIN;
      top();
      return;
     }

   if (name(&col_out,col1, name_arg, unit_arg) == 0)
     {
      ne();
      action = MAIN;
      top();
      return;
     }

   summation(&col1, &col_out);

   sprintf(msg, "SUMMATION: DONE\n");
   print_msg(msg);

   action = MAIN;
   top();
  }


  /************************** curv ***********************/
  do_curv(arg)
      char arg[128];
  {
   nocom(arg);

   if (sscanf(arg, "%s %d %d %f %s %s", desire, &col1, &col_out, &temp_float, nam
t_arg) != 6)
     {
      nea();
      top();
      action = MAIN;
      return;
     }

   if (col1 >= max_col || col_out >= max_col)
     {
      coe();
      action = MAIN;
      top();
      return;
     }

   if (name(&col_out,col1, name_arg, unit_arg) == 0)
     {
      ne();
      action = MAIN;
      top();
      return;
```



```
     }

   for(i=2; i<head.ch[col1].nelem-2; ++i)
     {
      darray[col_out][i] = -30.0*darray[col1][i] + 16.0*(darray[col1][i+1]+darray[col1][i
1]) - (darray[col1][i+2]+darray[col1][i-2]) ;
      darray[col_out][i] /= 12.0*pow(temp_float,2.0);
     }

   sprintf(msg, "CURV: DONE\n");
   print_msg(msg);

   action = MAIN;
   top();
  }


  /************************** peak ***********************/
  do_peak(arg)
      char arg[128];
  {
   nocom(arg);

   if (sscanf(arg, "%s %d %d %s %s", desire, &col1, &col_out, name_arg, unit_a
     {
      nea();
      top();
      action = MAIN;
      return;
     }

   if (col1 >= max_col || col_out >= max_col)
     {
      coe();
      action = MAIN;
      top();
      return;
     }

   if (name(&col_out,col1, name_arg, unit_arg) == 0)
     {
      ne();
      action = MAIN;
      top();
      return;
     }

   for(i=0; i<head.ch[col_out].nelem; ++i)
    darray[col_out][i] = 0.0 ;

   j = 0;
   for(i=2; i<head.ch[col1].nelem-2; ++i)
     {
      if( darray[col1][i] > darray[col1][i+1] && darray[col1][i] > darray[col1][i-
        {
         if( darray[col1][i] > darray[col1][i+2] && darray[col1][i] > darray[col1,..
           {
            darray[col_out][i] = darray[col1][i] ;
            j += 1;
           }
        }
     }
   sprintf(msg, "PEAK: %d peaks found\n",j);
   print_msg(msg);
```

```
  sprintf(msg,  "PEAK:  DONE\n");
  print_msg(msg);

  action  =  MAIN;
  top();
}


/***************************** decimat ***************************/
do_decimat(arg)
     char  arg[128];
{
  nocom(arg);

  if  (sscanf(arg,  "%s  %d  %d  %d  %d  %d  %s  %s",  desire,  &col1,  &col_out,  &k,  &firs
ame_arg,  unit_arg)  !=  8)
    {
      nea();
      top();
      action  =  MAIN;
      return;
    }

  if  (col1  >=  max_col  ||  col_out  >=  max_col)
    {
      coe();
      action  =  MAIN;
      top();
      return;
    }

  if(check_row(&first,&last,col1)  !=  0)
    {
      cre();
      action  =  MAIN;
      top();
      return;
    }

  if  (name(&col_out,col1,  name_arg,  unit_arg)  ==  0)
    {
      ne();
      action  =  MAIN;
      top();
      return;
    }

  for(i=first,  j=first;  i  <=  last;  i+=k,  j++)
    {
      darray[col_out][j]  =  darray[col1][i]  ;
    }

  first=j;      /*  first  row  to  remove  */
  if(desire[(strlen((char  *)desire)-1)]  ==  'r')
    {
      head.ch[col_out].nelem  =  first;
    }

  sprintf(msg,  "DECIMAT:  DONE\n");
  print_msg(msg);

  action  =  MAIN;
  top();
```



```
}


/***************************** pdf ***************************/
do_pdf(arg)
     char  arg[128];
{
  int  max,  min;

  nocom(arg);

  if  (sscanf(arg,  "%s  %d  %d  %d  %d  %d  %d",  desire,  &col1,  &col_out,  &col2,  &k,  &
!=  7)
    {
      nea();
      top();
      action  =  MAIN;
      return;
    }

  if  (col1  >=  max_col  ||  col_out  >=  max_col  ||  col2  >=  max_col)
    {
      coe();
      action  =  MAIN;
      top();
      return;
    }
  do_pdf1();
  i  =  0;
  j  =  head.nrec-1;
  col_stat.max  =  max;
  col_stat.min  =  min;
  do_pdf2();
}


do_pdfauto(arg)
     char  arg[128];
{
  nocom(arg);

  if  (sscanf(arg,  "%s  %d  %d  %d  %d",  desire,  &col1,  &col_out,  &col2,  &k)  !=  5)
    {
      nea();
      action  =  MAIN;
      top();
      return;
    }

  if  (col1  >=  max_col  ||  col_out  >=  max_col  ||  col2  >=  max_col)
    {
      coe();
      action  =  MAIN;
      top();
      return;
    }
  do_pdf1();
  i  =  0;
  j  =  head.nrec-1;
  stats(&col1,  &i,  &j);
  do_pdf2();
}

do_pdf1()
```



```
{
  if(check_col(col_out)  ==  0)  head.nchan  +=  1;
  if(check_col(col2)  ==  0)  head.nchan  +=  1;
  strcpy(&(head.ch[col_out].name[0]),"prob_dens");
  strcpy(&(head.ch[col2].name[0]),"bin");
  strcpy(&(head.ch[col_out].units[0]),"prob_dens");
  strcpy(&(head.ch[col2].units[0]),&(head.ch[col1].units[0]));
  head.ch[col_out].gain  =  1.0  ;
  head.ch[col2].gain  =  1.0  ;
  if(head.nrec<k)  head.nrec=k  ;
  head.ch[col_out].nelem  =  k  ;
  head.ch[col2].nelem  =  k  ;
}

do_pdf2()
{
  for(j=0;  j<k;  ++j)    /*initialize  and  set  up  dependent  axis*/
    {
      darray[col2][j]  =  col_stat.min  +  ((float)j+0.5)/(float)(k-1)*(col_stat.max  -  col_
t.min);
      darray[col_out][j]  =  0.0;
    }
  for(i=0;  i<head.ch[col1].nelem;  ++i)  /*bin  data*/
    {
      temp_float  =  (float)(k-1)*(darray[col1][i]-col_stat.min)/(col_stat.max-col_stat.min)
;
      darray[col_out][(int)temp_float]  +=  1.0;
    }
  for(j=0;  j<k;  ++j)    /*scale  for  probability  density*/
    {
      darray[col_out][j]  /=  head.ch[col1].nelem*((col_stat.max-col_stat.min)/((float)k-1.0
));
    }

  sprintf(msg,  "PDF:  DONE\n");
  print_msg(msg);

  action  =  MAIN;
  top();

}


/***************************** r_row ***************************/
do_r_row_col(arg)
     char  arg[128];
{
  nocom(arg);

  if  (sscanf(arg,  "%s  %d  %d  %d",  desire,  &col1,  &first,  &last)  !=  4)
    {
      nea();
      action  =  MAIN;
      top();
      return;
    }

  do_r_row_final(col1+1,  col1,  first,  last);
  action  =  MAIN;
  top();
}

do_r_row(arg)
     char  arg[128];
```



```
{
  if  (sscanf(arg,  "%s  %d  %d",  desire,  &first,  &last)  !=  3)
    {
      nea();
      action  =  MAIN;
      top();
      return;
    }

  do_r_row_final(max_col,  1,  first,  last);

  action  =  MAIN;
  top();
}

do_r_row_final(l,  col1,  first,  last)
     int  l,  col1,  first,  last;
{
  j  =  first;
  k  =  last;

  if  (  k  >=  head.nrec  )
    {
      k  =  head.nrec  -  1  ;
      sprintf(msg,  "Total  number  of  records  is  %d!\n",head.nrec)  ;
      print_msg(msg);
      sprintf(msg,  "Rows  %d  through  %d  removed.\n",j,head.nrec-1)  ;
      print_msg(msg);
    }

  if(check_row(&j,&k,1)  !=  0)  /*  use  col  1  */
    {
      cre();
      action  =  MAIN;
      top();
      return;
    }

  temp_int  =  (k  -  j  +  1)  ;
  for(  h  =  col1;  h  <  l;  ++h)    /*changed  so  that  rows  could  be  removed*/
    {                                  /*from  just  one  col  within  the  table*/
      j  =  first  ;
      for(  i  =  k;  i  <  head.nrec;  ++i)
        {
          if(head.ch[h].nelem  >  0)
            {
              darray[h][j]  =  darray[h][j+temp_int]  ;
            }
          j  +=  1  ;
        }
          if(head.ch[h].nelem  >=  k  +  1)
            {
              head.ch[h].nelem  -=  temp_int  ;
            }
          else
            {
              if(head.ch[h].nelem  >  first  +  1)  head.ch[h].nelem  =  first  ;
            }
    }

  head.nrec  =  head.ch[1].nelem;
  for(  i  =  2;  i  <  max_col;  ++i)
    head.nrec  =  (head.nrec  <  head.ch[i].nelem)  ?  head.ch[i].nelem  :  head.nrec  ;
```

```
    sprintf(msg, "R_ROW: DONE\n");
    print_msg(msg);

    action =MAIN;
    top();

  }


  /***************************** comment ***********************/
  do_comment(arg)
       char arg[128];
  {
    char *com;

    nocom(arg);

    if (sscanf(arg, "%s %d %s", desire, &col1, msg) != 3)
      {
        nea();
        action = MAIN;
        top();
        return;
      }

    if (col1 > max_col)
      {
        coe();
        action = MAIN;
        top();
        return;
      }

    com = (char *)strip(arg, 2);

    if (check_col(col1) != 0)
      {
        sprintf(msg, "Old comment reads: %s\n", &(head.ch[col1].comment[0]));
        print_msg(msg);
        strncpy(&(head.ch[col1].comment[0]), com, 50);
        head.ch[col1].comment[strlen(&(head.ch[col1].comment[0]))] = '\0';
        sprintf(msg, "Current comment reads: %s\n", &(head.ch[col1].comment[0]));
        print_msg(msg);
        sprintf(msg, "COMMENT: DONE\n");
        print_msg(msg);

      }
    else
      {
        sprintf(msg, "Sorry, you cannot comment on unnamed columns.\n");
        print_msg(msg);
      }

    action = MAIN;
    top();
  }

  /***************************** name ***************************/
  /* calls name() in look_funcs.c */

  do_name(arg)
       char arg[128];
  {
```



```
    nocom(arg);

    if (sscanf(arg, "%s %d %s %s", desire, &col_out, name_arg, unit_arg) != 4)
      {
        nea();
        action = MAIN;
        top();
        return;
      }

    if (col_out > max_col)
      {
        coe();
        action = MAIN;
        top();
        return;

      }

    if (name(&col_out, col_out, name_arg, unit_arg) == 0)
      {
        na();
        action = MAIN;
        top();
        return;
      }

    sprintf(msg, "NAME: DONE\n");
    print_msg(msg);

    action = MAIN;
    top();
  }


  /***************************** r_col ************************/
  /* calls null_col() in look_funcs.c */
  do_r_col(col)
       int col;
  {
    if (col > max_col)
      {
        coe();
      }
    else
      {
        null_col(col);
        head.nchan -= 1;
      }

    sprintf(msg, "R_COL: DONE\n");
    print_msg(msg);

    action = MAIN;
    top();
  }


  /***************************** zero ************************/
  /* calls zero() in array.c */
  do_zero_all(arg)
       char arg[128];
  {
    int rec;
```



```
    if (sscanf(arg, "%s %d", desire, &rec) != 2)
      {
        nea();
        action = MAIN;
        top();
        return;
      }

    if (rec >= head.nrec)
      {
        sprintf(msg,"No way! nrec = %d\n",head.nrec);
        print_msg(msg);
      }
    else
      for(i=1; i <= max_col-1; ++i)
        {
          zero(&i,&rec) ;
        }

    sprintf(msg, "ZERO_ALL: DONE\n");
    print_msg(msg);

    action = MAIN;
    top();
  }

  do_zero(arg)
       char arg[128];
  {
    int rec;

    nocom(arg);

    if (sscanf(arg, "%s %d %d", desire, &col1, &rec) != 3)
      {
        nea();
        action = MAIN;
        top();
        return;
      }

    if (col1 >= max_col || rec >= head.nrec)
      {
        sprintf(msg, "Not in array space.\n");
        print_msg(msg);
      }
    else
      zero(&col1,&rec) ;

    sprintf(msg, "ZERO: DONE\n");
    print_msg(msg);

    action = MAIN;
    top();
  }


  /***************************** offset_int *********************/
  /* calls offset_int() in array.c */

  do_offset_int(arg)
       char arg[128];
  {
```



```
    int rec1, rec2;

    nocom(arg);

    if (sscanf(arg, "%s %d %d %d %s", desire, &col1, &rec1, &rec2, t_string) != 5)
      {
        nea();
        action = MAIN;
        top();
        return;
      }

    if (rec1 >= head.ch[col1].nelem || rec2 >= head.ch[col1].nelem )
      {
        sprintf(msg, "That column only has %d recs.\n",head.ch[col1].nelem);
        print_msg(msg);
        action = MAIN;
        top();
        return;
      }

    offset_int(&rec1, &col1, &rec2, t_string);

    sprintf(msg, "OFFSET_INT: DONE\n");
    print_msg(msg);

    action = MAIN;
    top();
  }


  /***************************** offset ************************/
  /* calls offset() in array.c */

  do_offset(arg)
       char arg[128];
  {
    int rec1, rec2;

    nocom(arg);

    if (sscanf(arg, "%s %d %d %d %d", desire, &col1, &rec1, &col2, &rec2) != 5,
      {
        nea();
        action = MAIN;
        top();
        return;
      }

    if (rec1 >= head.ch[col1].nelem || rec2 >= head.ch[col2].nelem )
      {
        sprintf(msg, "ERROR: Check the # of elements in col1 and col2.\n");
        print_msg(msg);
        action = MAIN;
        top();
        return;
      }
    offset(&rec1,&col1,&rec2,&col2);

    sprintf(msg, "OFFSET: DONE\n");
    print_msg(msg);

    action = MAIN;
    top();
```

```
}


/********************************** simplex ***************************/
/* calls simp_func() in simplex1.c */

simplex_info()
{
  sprintf(msg,"A listing of functions can be found in /lamont/lamont/boitnott/LCO/Sir
ctions\n");
    print_msg(msg);
}

do_simplex(arg)
     char arg[128];
{
  int xcol, ycol;

  nocom(arg);

  if (sscanf(arg, "%s %s %d %d %d %d", desire, t_string, &first, &last, &xcol, &
)
    {
      nea();
      action = MAIN;
      top();
      return;
    }

  if(strcmp(t_string,"Power1") == 0)
    {
      sprintf(msg,"\n\tfitting Y = pow(X[0],A) + B\n");
      print_msg(msg);
      l = 2 ; /* number of free parameters to optimize */
      h = 1 ; /* number of independent variables */
    }
  else if(strcmp(t_string,"Power2") == 0)
    {
      sprintf(msg,"\n\tfitting Y = A * pow(X[0],B) + C\n");
      print_msg(msg);
      l = 3 ; /* number of free parameters to optimize */
      h = 1 ; /* number of independent variables */
    }
  else if(strcmp(t_string,"normal") == 0)
    {
      sprintf(msg,"\n\tfitting normal distribution, A=mean, B=stddev\n");
      print_msg(msg);
      l = 2 ; /* number of free parameters to optimize */
      h = 1 ; /* number of independent variables */
    }
  else if(strcmp(t_string,"chisqr") == 0)
    {
      sprintf(msg,"\n\tfitting chi-squared distribution, A= #degree of freedom\n");
      print_msg(msg);
      l = 1 ; /* number of free parameters to optimize */
      h = 1 ; /* number of independent variables */
    }
  else if(strcmp(t_string,"scchisqr") == 0)
    {
      sprintf(msg,"\n\tfitting scaled chi-squared distribution, A=stddev, B=ndf, C=offs
n");
      print_msg(msg);
      l = 3 ; /* number of free parameters to optimize */
      h = 1 ; /* number of independent variables */
```



```
    }
  else if(strcmp(t_string,"genexp") == 0)
    {
      sprintf(msg,"\n\tfitting y = A * exp(Bx+C) + D \n");
      print_msg(msg);
      l = 4 ; /* number of free parameters to optimize */
      h = 1 ; /* number of independent variables */
    }
  else if(strcmp(t_string,"Poly4") == 0)
    {
      sprintf(msg,"\n\tfitting y = A + B*x + C*pow(x,2) + D*pow(x,3) + E*pow(x,4)
      print_msg(msg);
      l = 5 ; /* number of free parameters to optimize */
      h = 1 ; /* number of independent variables */
    }
  else if(strcmp(t_string,"gensin") == 0)
    {
      sprintf(msg,"\n\tfitting y = A * sin(Bx+C) + D \n ");
      print_msg(msg);
      l = 4 ; /* number of free parameters to optimize */
      h = 1 ; /* number of independent variables */
    }
  else if(strcmp(t_string,"ExpLin") == 0)
    {
      sprintf(msg,"\n\tfitting y = A * (1.0 - exp(B*x)) + C*x \n ");
      print_msg(msg);
      l = 3 ; /* number of free parameters to optimize */
      h = 1 ; /* number of independent variables */
    }
  else if(strcmp(t_string,"rclow") == 0)
    {
      sprintf(msg,"\n\tfitting y = (1/(x*B))*sqrt(1+x*x*B*B*A*A) \n");
      print_msg(msg);
      l = 2 ; /* number of free parameters to optimize */
      h = 1 ; /* number of independent variables */
    }
  else if(strcmp(t_string,"rcph") == 0)
    {
      sprintf(msg,"\n\tfitting y = -1.0*atan((1.0/(x*A*B))) \n");
      print_msg(msg);
      l = 2 ; /* number of free parameters to optimize */
      h = 1 ; /* number of independent variables */
    }
  else
    {
      sprintf(msg,"Illegal function: %s \n", t_string);
      print_msg(msg);
      action = MAIN;
      top();
      return;
    }

/* for(k=0; k<h; ++k)
    {
      if(command==stdin && h>1) fprintf(stderr,"enter column for X[%d] -> ",k);
      if(command==stdin && h==1) fprintf(stderr,"enter column for X -> ");
      fscanf(command,"%d",&simp_xch[k]);

    }*/
/* h is always 1, therefore, no need the loop.. which is complicated in X */
  simp_xch[0] = xcol;

  if(check_col(simp_xch[0]) == 0 || head.ch[simp_xch[k]].nelem < last+1)
    {
```



```
      sprintf(msg,"X-col is not in array space: %d\n", xcol);
      print_msg(msg);
      action = MAIN;
      top();
      return;
    }
  simp_ych = ycol;
  if(check_col(simp_ych) == 0 || head.ch[simp_ych].nelem < last+1)
    {
      sprintf(msg,"Y-col is not in array space: %d\n", ycol);
      print_msg(msg);
      action = MAIN;
      top();
      return;
    }


/*

   simp_func(command,1,simp_xch,&simp_ych,t_string,first,last, temp_int);

  sprintf(msg, "SIMPLEX: DONE\n");
  print_msg(msg);

  action = MAIN;
  top();
  */


  /* set simp_func_action so that it knows from where it's called from
     set action to simp func
     */

  simp_func_action = SIMPLEX;
  action = SIMP_FUNC;
  return;

}


/********************************** stdasc ***************************/
/* calls stdasc() in filtersm.c */

do_stdasc(arg)
     char arg[128];
{
  char fi[20], cr[20];
  int i;

  nocom(arg);

  if (sscanf(arg, "%s %s %s %s", desire, data_file, cr, fi) != 4)
    {
      nea();
      action = MAIN;
      top();
      return;
    }

  if ((data = fopen(data_file, "r")) == NULL)
    {
      sprintf(msg, "Can't open data file %s", data_file);
      print_msg(msg);
    }
```



```
  else
    {
      if (stdasc(data, cr, fi) != 1)
        {
          sprintf(msg, "Input error! stdasc command aborted.");
          print_msg(msg);
        }
      fclose(data);
      for (i = (head.nchan+1); i<17; ++i)
        null_col(i);
    }

  sprintf(msg, "STDASC: DONE\n");
  print_msg(msg);

  action = MAIN;
  top();
}


/********************************** tasc ***************************/
/* calls tasc() in filtersm.c */

do_tasc(arg)
     char arg[128];
{
  char fi[20];
  int i, f_flag;

  nocom(arg);

  if (sscanf(arg, "%s %s %s", desire, data_file, fi) != 3)
    {
      nea();
      action = MAIN;
      top();
      return;
    }

  if ((data = fopen(data_file, "r")) == NULL)
    {
      sprintf(msg, "Can't open data file %s.\n", data_file);
      print_msg(msg);
    }
  else
    {
      if (fi[0] == 'f') f_flag = 1;
      else f_flag = 0;

      if (tasc(data, f_flag) != 1)
        {
          sprintf(msg, "Input error! tasc command aborted.\n");
          print_msg(msg);
        }
      fclose(data);
      sprintf(msg, "Warning: assumed active columns are between 1 and head.nchan
head.nchan);
      print_msg(msg);

      for (i = (head.nchan+1); i<MAX_COL; ++i)
        null_col(i);
    }

  sprintf(msg, "TASC: DONE\n");
```

```
    print_msg(msg);

    action  =  MAIN;
    top();
  }


  /********************************** head ********************************/
  /*  calls  aschead_scrn()  or  aschead_file()  in  lookio.c  */

  do_head(arg)
       char  arg[128];
  {
    char  new_file[128];

    strcpy(new_file,  arg);

    if  ((strncmp(new_file,  "S",  1)  ==  0  ||  strncmp(new_file,  "s",  1)  ==  0)
        &&  (strlen(new_file)  ==  1))
     {
       aschead_scrn(&head);
       action  =MAIN;
       top();
       return;
     }

    else
     {
       if  ((new  =  fopen(new_file,  "a"))  ==  NULL)
         {
           sprintf(msg,  "Can't  open  file:  %s.\n",  new_file);
           print_msg(msg);
           action  =MAIN;
           top();
           return;
         }
       else
         {
           if  ((int)ftell(new)  >  0)
             {
               if  (write_show_warning()  !=  1)
                 {
                   sprintf(msg,  "Head  command  aborted!\n");
                   print_msg(msg);
                   action  =MAIN;
                   top();
                   return;
                 }
               else
                 {
                   if  ((new  =  fopen(new_file,  "w"))  ==  NULL)
                     {
                       sprintf(msg,  "Can't  open  file:  %s.\n",  new_file);
                       print_msg(msg);
                       action  =MAIN;
                       top();
                       return;
                     }
                   else
                     {
                       sprintf(msg,  "Overwriting  %s.\n",  new_file);
                       print_msg(msg);
                     }
                 }
```



```
                }
              aschead_file(&head,  new);
              fclose(new);

              sprintf(msg,  "HEAD:  DONE\n");
              print_msg(msg);

              action  =  MAIN;
              top();
              return;
            }
       }
    }

    /*****************************  getaschead  ****************************/
    /*  calls  getaschead()  in  lookio.c  */

  do_getaschead(arg)
       char  arg[128];
    {
      if  ((data  =  fopen(arg,  "r"))  ==  NULL)
        {
          sprintf(msg,  "Can't  open  header  file:  %s.\n",  arg);
          print_msg(msg);
        }
      else
        {
          if  (getaschead(&head,  data)  !=  1)
            {
              sprintf(msg,  "Input  error.  getaschead  command  aborted.\n");
              print_msg(msg);
            }
          fclose(data);
          sprintf(msg,  "GETASCHEAD:  DONE\n");
          print_msg(msg);
        }
      action  =  MAIN;
      top();
    }


    /********************************** examin ****************************/
    /*  calls  examin()  in  filtersm.c  */

    do_examin(arg)
         char  arg[128];
    {
      if  ((data  =  fopen(arg,  "r"))  ==  NULL)
        {
          sprintf(msg,  "Can't  open  data  file:  %s.\n",  arg);
          print_msg(msg);
        }
      else
        {
          examin(data);
          fclose(data);
        }

      sprintf(msg,  "EXAMIN:  DONE\n");
      print_msg(msg);

      action  =  MAIN;
      top();
    }
```



```
  /********************************** rsm ********************************/

  print_rsm_help_info()
  {
  /*  if(desire[(strlen((char  *)desire)-1)]  ==  'h')  */

    sprintf(msg,  "> This function computes the Dieterich-Ruina rate/state variable fri
  model  at  specified  displacements.\n");
    print_msg(msg);

    sprintf(msg,  "> Friction is calculated at displacement values given by measuremer
    displacement  column:\n");
    print_msg(msg);

    sprintf(msg,  "> See  cm_h  or  cm2_h  for  a  similar  calc  -but  which  generates  thei
  lacement  values.\n");
    print_msg(msg);

    sprintf(msg,  "> Input  is:  the  column  that  contains  displacement,  the  row  #  of
  ty  step,\n");
    print_msg(msg);

    sprintf(msg,  "> the  row  #  to  which  the  calculation  should  be  extended,  and  th
  onstitutive  parameters.\n");
    print_msg(msg);

    sprintf(msg,  "> k  should  be  given  in  the  same  units  as  normal  stress  and  dc.'
    print_msg(msg);

    sprintf(msg,  "> The  pre-step  mu  value  is  extended  back  100  rows  in  the  model
  ");
    print_msg(msg);
  }


  do_rsm(arg)
       char  arg[128];
  {

    if  (ascanf(arg,  "%s  %d  %d  %d  %d  %d  %lf  %lf  %lf  %lf  %lf  %lf  %lf  %lf  %lf  %lf  %l
              &rs_param.disp_col,  &rs_param.mu_col,  &rs_param.vs_row,
              &rs_param.last_row,  &rs_param.mu_fit_col,
              &rs_param.stiff,  &rs_param.sig_n,  &rs_param.vo,  &rs_param.vf,
              &rs_param.muo,  &rs_param.muf,  &rs_param.a,  &rs_param.bl,
              &rs_param.dc1,  &rs_param.dc2)  !=  16)
      {
        nea();
        action  =  MAIN;
        top();
        return;
      }

    sprintf(msg,  "Enter:  disp.  col.,  mu  col,  row  #  of  vel.  step  and  end  of  data
  l  for  model_mu,  stiffness  (k),  Sigma_n,\n\tv_initial  (v_o),  v_final,  mu_o,  mu_f,  a,
  c1,  and  Dc2\n");
    print_msg(msg);

    adj_init_disp  =  mu_check(&rs_param.muo,1);  /*  returns  false  if  passed  -33  */
    mu_check(&rs_param.muf,0);                    /*  this  is  in  look_funcs.c  */

    rs_param.amb  =  (rs_param.muf-rs_param.muo)/log(rs_param.vf/rs_param.vo);
```



```
    if(rs_param.dc2  <  0.0)
      {
        rs_param.b2  =  0.0;
        rs_param.dc2  =  1e10;  /*  I  think  leaving  it  neg.  will
                                    lead  to  matherr()  being  called  */
        rs_param.b1  =  rs_param.a  -  rs_param.amb;
        l  =  2;
      }
    else
      {
        rs_param.b2  =  -(rs_param.amb-rs_param.a+rs_param.b1);
        l  =  4;
      }

    if(rs_param.disp_col  >=  max_col  ||  rs_param.mu_col  >=  max_col  ||  rs_param.mu_fit_co
    max_col)
      {
        cos();
        action  =  MAIN;
        top();
        return;
      }

    if(rs_param.last_row  <  rs_param.vs_row  ||  rs_param.last_row  >  head.ch[rs_param.d
  .nelem)
      {
        sprintf(msg,  "vs_step  begin  (%d)  >  end  (%d)  or  end  >  nelems\n",  rs_param.vs_
  param.last_row);
        print_msg(msg);
        action  =  MAIN;
        top();
        return;
      }

    /*naming  BS  here  so  we  don't  call  name()  */
    if(strncmp(head.ch[rs_param.mu_fit_col].name,"no_val",6)  ==  0)
      head.nchan  +=  1  ;
    head.ch[rs_param.mu_fit_col].nelem  =  head.ch[rs_param.disp_col].nelem  ;

    /*  name  for  model  fric.  col  is  "rsm_mu".  Units  are  "."  */
    strcpy(head.ch[rs_param.mu_fit_col].name,  "rsm_mu");
    strcpy(head.ch[rs_param.mu_fit_col].units,  ".");

    rs_param.added_pts  =  rs_param.weight_pts  =  0;

    /*  allocate  space  for  arrays  used  by  simplex  and  fq  */
    rs_param.disp_data  =  (double*)calloc((unsigned)(rs_param.last_row-rs_param.vs_row+13
  izeof(double))  ;
    rs_param.mu_data  =  (double*)calloc((unsigned)(rs_param.last_row-rs_param.vs_row+13),
  izeof(double))  ;
    rs_param.model_mu  =  (double*)calloc((unsigned)(rs_param.last_row-rs_param.vs_row+13),
  izeof(double))  ;

    disp_ptr  =  rs_param.disp_data;
    mu_ptr  =  rs_param.mu_data;

    if(adj_init_disp)
      {
        /*  so  as  to  be  consistent  with  scm....  */
        /*  adjust  disp.  of  first  pt  to  account  for  mu0  (possibly)  not  =  mu[vs_
        sl  =  (darray[rs_param.mu_col][rs_param.vs_row+1]-darray[rs_param.mu_col][rs_para
  _row]]  /  (darray[rs_param.disp_col][rs_param.vs_row+1]-darray[rs_param.disp_col][rs_pa
  vs_row]]  ;
        dx  =  (rs_param.muo-darray[rs_param.mu_col][rs_param.vs_row])/sl  +  darray[rs_pa
```

```
sp_col][rs_param.vs_row];
      if(dx  >  darray[rs_param.disp_col][rs_param.vs_row+1])
        {
          sprintf(msg, "Problem with getting init. disp. correct for input init. m
          print_msg(msg);
          sprintf(msg, "Sorry, but you'll have to fix something or change init. m
loser to mu of vs_row\n");
          print_msg(msg);
          action = MAIN;
          top();
          return;
        }
      else
        *disp_ptr = dx; /* install as vs_row */
    }
  else
    *disp_ptr = darray[rs_param.disp_col][rs_param.vs_row];

  *mu_ptr = rs_param.muo; /* install as vs_row */

  /* disp, mu pt is now on a line <> vs pt and next pt */
  /* load data into temp arrays */
  for(i=1, j=rs_param.vs_row+1; j <= rs_param.last_row; i++, j++)
    {
      *(disp_ptr+i) = darray[rs_param.disp_col][j];
      *(mu_ptr+i) = darray[rs_param.mu_col][j] ;
    }

  /* so that calc stops properly*/
  *(disp_ptr+i) = darray[rs_param.disp_col][rs_param.last_row]+1e9 ;

  /* in fq.c */
  simp_rate_state_mod();

  if(l == 2 )
    {
      sprintf(msg, "a-b = %g\ta=%g\tb=%g\tdc=%g\n", rs_param.amb, rs_param.a, rs_param.
rs_param.dc1);
      print_msg(msg);
    }
  else
    {
      sprintf(msg, "a-b = %g\ta=%g\tb1=%g\tb2=%g\tdc1=%g\tdc2=%g\n", rs_param.amb, rs_p
m.a, rs_param.b1, rs_param.b2, rs_param.dc1, rs_param.dc2);
      print_msg(msg);
    }

  for ( i= (rs_param.vs_row > 15) ? (rs_param.vs_row-15) : 0; i <= rs_param.vs_r
    darray[rs_param.mu_fit_col][i] = rs_param.muo;

  /* install model data */
  mu_ptr = rs_param.model_mu;

  for(i=rs_param.vs_row+1; i <=rs_param.last_row; ++i)
    darray[rs_param.mu_fit_col][i] = *mu_ptr++ ;


  cfree(rs_param.model_mu);
  cfree(rs_param.mu_data);
  cfree(rs_param.disp_data);

  sprintf(msg, "RSM: DONE\n");
  print_msg(msg);
```



```
  action = MAIN;
  top();

}


/****************************** cm ******************************/
print_cm_help_info()
{
  sprintf(msg, "> This function computes the Dieterich-Ruina rate/state variable fri
model.\n");
  print_msg(msg);

  sprintf(msg, "> See rsm_h for a similar calc -but which uses the measured disp
n");
  print_msg(msg);

  sprintf(msg, "> It needs: the column that contains displacement, the row # of
ty step,\n");
  print_msg(msg);

  sprintf(msg, "> the row # to which the calculation should be extended, and th
onstitutive parameters.\n");
  print_msg(msg);

  sprintf(msg, "> k should be given in the same units as normal stress and dc.
  print_msg(msg);

  sprintf(msg, "> The model data occupies 1000 rows more than the original data
le) -although 50%% of the data is written within the first 20%% of the total
\n");
  print_msg(msg);

  sprintf(msg, "> The pre-step mu value is extended back 100 rows in the model
");
  print_msg(msg);

  sprintf(msg, mu_fit_mess_1);
  print_msg(msg);
}


do_cm(arg)
    char arg[128];
{

  if (sscanf(arg, "%s %d %d %d %d %d %d %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf
            desire,
            &col_out,&rs_param.disp_col,&rs_param.mu_col, &rs_param.vs_row,
            &rs_param.last_row,&temp_col, &rs_param.stiff, &rs_param.sig_n,
            &rs_param.vo, &rs_param.vf, &rs_param.muo, &rs_param.muf,
            &rs_param.a, &rs_param.b1, &rs_param.dc1, &rs_param.dc2) != 17 )
    {
      nea();
      action = MAIN;
      top();
      return;
    }

  mu_check(&rs_param.muo,1);
  mu_check(&rs_param.muf,0);

  rs_param.amb = (rs_param.muf-rs_param.muo)/log(rs_param.vf/rs_param.vo);
  if(rs_param.dc2 < 0)
```



```
    {
      rs_param.b1 = -(rs_param.amb-rs_param.a);
      rs_param.b2 = 0;
      rs_param.dc2 = 1e10;
    }
  else
    rs_param.b2 = -(rs_param.amb-rs_param.a+rs_param.b1);

  if(rs_param.mu_col >= max_col || col_out >= max_col || temp_col >= max_col)
    {
      coe();
      action = MAIN;
      top();
      return;
    }

  if(rs_param.last_row < rs_param.vs_row || rs_param.last_row > head.ch[rs_param.mu_col
elem)
    {
      sprintf(msg, "vs_step begin (%d) > end (%d) or end > nelems\n", rs_param.vs_
param.last_row);
      print_msg(msg);
      action = MAIN;
      top();
      return;
    }

  /*nameing BS here so we don't call name() */

/*  if(strncmp(head.ch[col_out].name,"no_val",6) == 0)
    head.nchan += 1 ;
  if(strncmp(head.ch[temp_col].name,"no_val",6) == 0)
    head.nchan += 1 ;
  head.ch[col_out].nelem = head.ch[temp_col].nelem = head.ch[rs_param.mu_col].nelem ;

*/
  /* name for model disp. col is "mod_disp". units are same as orig. disp. col
  strcpy(head.ch[col_out].name, "mod_disp");
  strcpy(head.ch[col_out].units, head.ch[rs_param.mu_col].units );

  /* name for model fric. col is "mod_mu". Units are "." */
  strcpy(head.ch[temp_col].name, "mod_mu");
  strcpy(head.ch[temp_col].units, "." );

  /* function is in fq.c */
  rate_state_mod(rs_param.disp_col,rs_param.vs_row,rs_param.last_row,col_out,temp_col,rs_p
aram.stiff,rs_param.sig_n,rs_param.muo,rs_param.muf,rs_param.vo,rs_param.vf,rs_param.a,rs_
param.b1,rs_param.b2,rs_param.dc1,rs_param.dc2);

  if( rs_param.b2 == 0 && rs_param.dc2 == 1e10) /* 1 state variable model */
    {
      sprintf(msg, "a-b = %g\ta=%g\tb=%g\tdc=%g\n", rs_param.amb, rs_param.a, rs_param.
rs_param.dc1);
      print_msg(msg);
    }

  else
    {
      sprintf(msg, "a-b = %g\ta=%g\tb1=%g\tb2=%g\tdc1=%g\tdc2=%g\n", rs_param.amb, rs_p
m.a, rs_param.b1, rs_param.b2, rs_param.dc1, rs_param.dc2);
      print_msg(msg);
    }

  sprintf(msg, "CM: DONE\n");
```



```
  print_msg(msg);

  action = MAIN;
  top();

}


/****************************** mem ******************************/

do_mem(arg)
    char arg[128];
{
  int n_freq, n_poles, type, junk;
  float *mem_dat, *poles, pnp, fdt, facm, facp, ww, tempfloat;


  nocom(arg);

  if (sscanf(arg,"%d %d %d %d %d %d %d %s %s", desire, &col1, &col2, &col_out,
st, &temp_int, name_arg, unit_arg) != 9);

  if (sscanf(arg, "%s %d , %d , %d , %d , %d, %d , %d , %c, %c, %c
col2,&col_out,&temp_col,&first,&last,&n_freq,&n_poles,&type,&junk) != 11 )
    {
      nea();
      action = MAIN;
      top();
      return;
    }

  if(col1 >= max_col || col_out >= max_col || col2 >= max_col || temp_col >=
    {
      coe();
      action = MAIN;
      top();
      return;
    }

  /*nameing BS here so we don't call name() */
  if(strncmp(head.ch[col_out].name,"no_val",6) == 0)
    head.nchan += 1 ;
  if(strncmp(head.ch[temp_col].name,"no_val",6) == 0)
    head.nchan += 1 ;
  head.ch[col_out].nelem = head.ch[temp_col].nelem = head.ch[col1].nelem ;

  if(type == '1' || type == 'n') ;
  else
    {
      sprintf(msg, "defaulting to linear frequency distribution\n");
      print_msg(msg);
      type = 'n';
    }

  strcpy(t_string, "1/");
  strcat(t_string, head.ch[col1].units);
  t_string[13] = '\0';

  /* name(&col_out,col1); name(&temp_col,col2); */

  strcpy(head.ch[col_out].name, "frequency");
  strcpy(head.ch[col_out].units, t_string);

  strncpy(t_string, head.ch[col2].units, 10); /*Only 10 in case it's long*/
```

```
    t_string[10] = '\0';  /*make sure it's null terminated*/
    strcat(t_string, "**2" );
    t_string[13] = '\0';
    strcpy(head.ch[temp_col].name, "power");
    strcpy(head.ch[temp_col].units, t_string);

    test1 = darray[col1][first+1] - darray[col1][first];
    test2 = darray[col1][first+2] - darray[col1][first+1];
    if( test1 - test2 > SMALL)
      {
       sprintf(msg, "Warning: it doesn't look like the data are equally spaced, ch
d\n",col1);
       print_msg(msg);
      }

    sprintf(msg, "Nyquist freq. is %f 1/%s\t",1/(2*test1),head.ch[col1].units);
    print_msg(msg);
    /*   sprintf(msg, "increment = %f first=%d last=%d\n",test1,first,last);  */

    temp_int = last-first+1;                    /*n_data = last-first+1 */

    mem_dat = (float *)vector(1, temp_int);       /* allocate array space */
    poles = (float *)vector(1,n_poles);

    if(junk=='w')                /*apply welsh taper*/
      {
       sprintf(msg, "\t applying Welch taper to data\n");
       print_msg(msg);
       facm = temp_int-0.5;      facp = 1.0/(temp_int + 0.5);
       for(j=first, k=1, i=0; j <=last; ++j, ++k, ++i)  /*get data, apply welsh wi
         {
           ww = WELCH_WINDOW(i,facm,facp);
           mem_dat[k] = ww*darray[col2][j];
         }
      }
    else
      {
       sprintf(msg, "\t no taper applied to data\n");
       print_msg(msg);
       for(j=first, k=1, i=0; j <=last; ++j, ++k, ++i)  /*get data, apply welsh wi
         mem_dat[k] = darray[col2][j];
      }
    sprintf(msg, "\t calculating power spectrum\n");
    print_msg(msg);
    memcof(mem_dat,temp_int,n_poles,&pnp,poles); /*get coefficients*/


    first--;     /*so the points are in the right place, with j starting at 1*/
    /*select frequency distribution*/
    if(type == 'n')                      /*linear frequency dist*/
      {
       for(j=1;j<=n_freq;++j)
         {
           /*start at lowest freq, go to nyquist=1/2dt*/
           /* dt=sampling interval */
           fdt = (float)j / (float)(n_freq*2.000);
           darray[col_out][first+j] = fdt/test1; /*test1 = samp. int.*/
           darray[temp_col][first+j]=evlmem(fdt,poles,n_poles,pnp);
         }
      }
    else                         /*log freq distribution*/
      {
       tempfloat = log10(0.5);          /*evaluate power over 4 decades*/
       for(j=1;j<=n_freq;++j) {
```



```
         test2= tempfloat - 4.0 + 4.0*(float)j/(float)n_freq;
         fdt= (float)pow((double)10.0,(double)test2);
         darray[col_out][first+j] = fdt/test1;
         darray[temp_col][first+j]=evlmem(fdt,poles,n_poles,pnp);
        }
      }

    /* free array space */
    free_vector(mem_dat,1,temp_int );
    free_vector(poles,1,n_poles);

    sprintf(msg, "MEM: DONE\n");
    print_msg(msg);

    action = MAIN;
    top();

  }



  /********************************** median smooth ************************/

  do_median_smooth(arg)
       char arg[128];
  {
    nocom(arg);

    if (sscanf(arg,"%s %d %d %d %d %d %s %s", desire, &col1, &col_out, &first  i
int, name_arg, unit_arg) != 8)
      {
       nea();
       action = MAIN;
       top();
       return;
      }

    if(check_row(&first,&last,col1) != 0)
      {
       cre();
       action = MAIN;
       top();
       return;
      }

    if(temp_int > (last - first +1))
      {
       sprintf(msg, "Window can't be bigger than interval for smoothing.\n");
       print_msg(msg);
       action = MAIN;
       top();
       return;
      }

    if(col1 >= max_col || col_out >= max_col)
      {
       coe();
       action = MAIN;
       top();
       return;
      }

    if (name(&col_out, col1, name_arg, unit_arg) == 0)
      {
```



```
       ne();
       top();
       action = MAIN;
       return;
      }

    median_smooth(col1,col_out,first,last,temp_int);  /*in special.c*/

    sprintf(msg, "MEDIAN_SMOOTH: DONE\n");
    print_msg(msg);

    top();
    action = MAIN;
  }



  /****************************** smooth ******************************/

  do_smooth(arg)
       char arg[128];
  {
    nocom(arg);

    if (sscanf(arg,"%s %d %d %d %d %d %s %s", desire, &col1, &col_out, &first, &l
int, name_arg, unit_arg) != 8 )
      {
       nea();
       action = MAIN;
       top();
       return;
      }

    if(check_row(&first,&last,col1) != 0)
      {
       cre();
       action = MAIN;
       top();
       return;
      }

    if(temp_int > (last - first +1))
      {
       sprintf(msg, "Window can't be bigger than interval for smoothing\n");
       print_msg(msg);
       action = MAIN;
       top();
       return;
      }

    if(col1 >= max_col || col_out >= max_col)
      {
       coe();
       action = MAIN;
       top();
       return;
      }

    if (name(&col_out, col1, name_arg, unit_arg) == 0)
      {
       ne();
       top();
       action = MAIN;
       return;
```



```
      }

    smooth(col1,col_out,first,last,temp_int);  /*in special.c*/

    sprintf(msg, "SMOOTH: DONE\n");
    print_msg(msg);

    top();
    action = MAIN;
  }


  /****************************** typeall ********************************/
  /* calls vision() and p_vision() in look_funcs.c */
  /* msg_vision() and msg_p_vision() are equivalent funcs of the above which prints
  the msg_window instead of to a file. (also in look_funcs.c) */


  do_typeall(arg)
       char arg[128];
  {
    nocom(arg);

    if (sscanf(arg, "%s %s", desire, new_file) != 2)
      {
       nea();
       top();
       action = MAIN;
       return;
      }

    i = 0 ;
    j = head.nrec - 1 ;
    h = 1 ;
    k = max_col - 1 ;

    do_type_final();

    sprintf(msg, "TYPEALL: DONE\n");
    print_msg(msg);
    action = MAIN;
    top();
  }


  do_type(arg)
       char arg[128];
  {
    nocom(arg);

    if (sscanf(arg, "%s %d %d %d %d %d %s", desire, &i, &j, &h, &k, new_file) != 6)
      {
       nea();
       top();
       action = MAIN;
       return;
      }

    if ( j >= head.nrec )
      {
       j = head.nrec - 1;
       sprintf(msg, "Total number of records is %d!\n", head.nrec);
       print_msg(msg);
      }
```

```
  if ( b < 1 || b > k )
    {
      sprintf(msg, "First col is < 1 or > last.\n");
      print_msg(msg);
      top();
      action = MAIN;
      return;
    }

  check_row(&i, &j, b);

  if ( k > (max_col-1) )
    {
      sprintf(msg, "Total number of columns is %d!\t Setting col %d to %d\n", max
, max_col-1);
      print_msg(msg);
      k = max_col - 1;
    }

  do_type_final();

  sprintf(msg, "TYPE: DONE\n");
  print_msg(msg);
  action = MAIN;
  top();
}

do_type_final()
{
  if ((strncmp(new_file,"S",1) == 0 || strncmp(new_file,"s",1) == 0) &&
      (strlen(new_file) == 1) )
    {
      if (strncmp(desire,"type_p", 6) == 0)
        msg_p_vision(&i,&j,&h,&k) ;
      else
        msg_vision(&i,&j,&h,&k) ;
    }
  else
    {
      if ((new = fopen(new_file, "a")) == NULL )
        {
          sprintf(msg, "Can't open file %s.\n", new_file);
          print_msg(msg);
          top();
          action = MAIN;
          return;
        }
      else
        {
          if ((int)ftell(new) > 0)
            {
              if (write_show_warning() != 1)
                {
                  sprintf(msg, "Write aborted!\n");
                  print_msg(msg);
                  action =MAIN;
                  top();
                  return;
                }
              else
                {
                  if ((new = fopen(new_file, "w")) == NULL)
                    {
```



```
                      sprintf(msg, "Can't open file: %s.\n", new_file);
                      print_msg(msg);
                      action =MAIN;
                      top();
                      return;
                    }
                  else
                    {
                      sprintf(msg, "Overwriting %s.\n", new_file);
                      print_msg(msg);
                    }
                }
            }
        }
      if (strncmp(desire,"type_p",6) == 0)
        p_vision(new,&i,&j,&h,&k) ;
      else
        vision(new,&i,&j,&h,&k) ;
      fclose(new) ;
    }
}


/****************************** stat ******************************/

do_stats(arg)
    char arg[128];
{
  double hmean, gmean, rms;

  nocom(arg);

  if (sscanf(arg,"%s %d %d %d", desire, &col1, &i, &j) != 4)
    {
      nea();
      action = MAIN;
      top();
      return;
    }

 if(col1 >= max_col)
    {
      coe();
      action = MAIN;
      top();
      return;
    }

  if(strncmp(desire,"stat_a",6) == 0)
    {
      i=0;
      j=head.ch[col1].nelem -1;
      stats(&col1,&i,&j) ;
    }
  else
    {
      if(check_row(&i,&j,col1) != 0)
        {
          cre();
          action = MAIN;
          top();
          return;
        }
```



```
      stats(&col1,&i,&j) ;
    }

  sprintf(msg, "%s: mean = %e , max = %e , min = %e , stddev = %e\n",head.ch[
l_stat.mean,col_stat.max,col_stat.min,col_stat.stddev);
  print_msg(msg);

  h = 0;
  hmean = 0.0;
  gmean = 0.0;
  rms = 0.0;
  for(l=i; l<=j; ++l)
    {
      rms += (double)darray[col1][l]*(double)darray[col1][l];

      if(darray[col1][l] > 0.0)
        {
          hmean += (double)1.0 / (double)darray[col1][l] ;
          gmean += log((double)darray[col1][l]);
        }
      else
        h++;
    }

  rms /= (double)(j-i+1);
  rms = sqrt((double)rms);
  hmean /= (double)(j-i+1) ;
  hmean = 1.0 / hmean ;
  gmean /= (double)(j-i+1) ;
  gmean = exp((double)gmean);

  sprintf(msg, "rms = %E , gmean = %E , hmean = %E\n",rms,gmean,hmean);
  print_msg(msg);
  sprintf(msg, "%d zeros or negatives ignored for gmean and hmean\n",h);
  print_msg(msg);

  sprintf(msg, "STAT: DONE\n");
  print_msg(msg);
  action = MAIN;
  top();
}



/************************************ qi ******************************/
/* calls exec_qi() in */

do_qi(arg)
    char arg[256];
{
  nocom(arg);

  sprintf(msg, "Linearized inversion of one or two state variable friction model.\r
  gives a detailed description of this function) \n");
  print_msg(msg);
  sprintf(msg, "\nNeed disp. col, mu col, model_mu col, beginning row for fit, r
l. step, end row, weight_row, lin_term(c), converg_tol, lamda, wc, stiffness, v_ini
v_final, mu_init, a, b1, Dc1, b2 and Dc2 for input.\n ");
  print_msg(msg);
  sprintf(msg, "Input: d_col, mu_col, mod_col, first_row, vs_row, end_row, weight_row
tol, lamda, wc, k, v_o, vf, mu_o, a, b1, Dc1, b2, Dc2 \n") ;
  print_msg(msg);
```



```
  /*defaults, this flag is set to 0 at run time*/
  rs_param.op_file_flag &= 0x08; /* leave eights bit, everything else off*/
  rs_param.op_file_flag |= 0x04;            /*fours bit on by default -used for i
ctive control of iterations*/
  rs_param.law = 'r';
  wc = lamda = rs_param.lin_term = 0.0;
  rs_param.end_weight_row=0;
  rs_param.n_vsteps = 1;


  if (sscanf(arg, "%s %d %d %d %d %d %d %d %lf %lf %lf %lf %lf %lf %lf %lf
  %lf",
             desire,
             &rs_param.disp_col, &rs_param.mu_col, &rs_param.mu_fit_col,
             &rs_param.first_row, &rs_param.vs_row, &rs_param.last_row,
             &rs_param.weight_row, &rs_param.lin_term, &converg_tol,
             &lamda, &wc, &rs_param.stiff, &rs_param.vo, &rs_param.vf,
             &rs_param.muo, &rs_param.a, &rs_param.b1, &rs_param.dc1,
             &rs_param.b2, &rs_param.dc2) != 21)
    {
      nea();
      sprintf(msg, "Error reading rs info and initial guesses. Too many or too f
ars. Correct number is 20\n");
      print_msg(msg);
      action = MAIN;
      top();
      return;
    }


  junkp = strchr(desire, '.');
  if( junkp != NULL)            /*pointer to . -or null char if no .*/
    {
      while(*(++junkp) != '\0')
        switch(*junkp)
          {
          case 'd':                     /*Use Dieterich, slowness law*/
          case 'D':
            rs_param.law = 'd';
            break;
          case 'p':                     /*Use Perrin-Rice, quadratic law*/
          case 'P':
            rs_param.law = 'p';
            break;
          case 'j':                     /*Use Rice law*/
          case 'J':
            rs_param.law = 'j';
            break;
          case 'v':                     /*qi.v or qi_mvs.v indicates verbose mode fo
file*/
            rs_param.op_file_flag |= 0x01;          /*ones bit for verbose more*/
            break;
          case 'n':                     /*twos bit means create rather than append
*/
            rs_param.op_file_flag |= 0x02;
            break;
          case 'i':                      /*turn off fours bit*/
            rs_param.op_file_flag &= ~0x04;
            break;
            /*really want to toggle this*/
          case 'f':                      /*eights bit means write data table*/
            rs_param.op_file_flag |= 0x08;
```

```
            break;
          case 's':                    /*sixteens bit means use separate file for ea
ep*/
            rs_param.op_file_flag |= 0x10;          /*by appending vs_row to name*/
            break;
          }
    }


  if(strncmp(desire,"qi_mvs",6) == 0 )
    {
      sprintf(msg, "Multiple velocity steps.\nWeighting is from vs_row to next vs_r
weight vector -or between two rows if wc <0\n");
      print_msg(msg);
      sprintf(msg, "Enter the number of additional velocity steps, ");
      print_msg(msg);
      sprintf(msg, "a velocity for each step starting with the step after the st
nal' velocity (v1 v2 v3...) and ");
      print_msg(msg);
      sprintf(msg, "a row number for each velocity change (row_v1_to_v2 row row..)
      print_msg(msg);

      set_left_footer("");
      set_cmd_prompt("QI_MVS Input: ");
      action = QI_MVS;
      return;
    }

  if(wc < 0)
    {
      sprintf(msg, "Weight data between row %d and end_weight_row by a factor of
end_weight_row.\n ", rs_param.weight_row, -wc);
      print_msg(msg);
      set_left_footer("");
      set_cmd_prompt("End_weight_row: ");
      action = QI_WC;
      return;
  }

  do_qi_final();
}


do_qi_mvs(arg)
     char arg[256];
{

  nocom(arg);

  sscanf(arg, "%d", &rs_param.n_vsteps);
  rs_param.n_vsteps++;

  for(i=1;i<rs_param.n_vsteps;i++)
    sscanf(arg, " %lf", &rs_param.vel_list[i]);

  for(i=1;i<rs_param.n_vsteps;i++)
    sscanf(arg, " %d", &rs_param.vs_row_list[i]);

  do_qi_final();
}


do_qi_final()
{
```



```
  rs_param.vel_list[0]=rs_param.vf;
  rs_param.vs_row_list[0]=rs_param.vs_row;
  rs_param.vs_row_list[rs_param.n_vsteps]=rs_param.last_row;

  /* check cols and rows */
  if(rs_param.disp_col >= max_col || rs_param.mu_col >= max_col || rs_param.mu_f
  max_col)
    {
      coe();
      action = MAIN;
      top();
      return;
    }

  if( rs_param.last_row < rs_param.weight_row || rs_param.last_row < rs_param.vs
_param.last_row < rs_param.first_row || rs_param.weight_row < rs_param.first_row
ram.vs_row < rs_param.first_row || rs_param.last_row > head.ch[rs_param.disp_col].nelem
    {
      sprintf(msg, "vs_step begin (%d) > end (%d) or end > nelems or weight_row
row (%d)\n", rs_param.first_row, rs_param.last_row, rs_param.weight_row, rs_param.
      print_msg(msg);
      action = MAIN;
      top();
      return;
  }


  /*set up space for disp, mu, mu_fit*/
  rs_param.disp_data = (double *)calloc((unsigned)(rs_param.last_row-rs_param.first_
, sizeof(double)) ;
  rs_param.mu_data = (double *)calloc((unsigned)(rs_param.last_row-rs_param.first_ro
, sizeof(double)) ;
  rs_param.model_mu = (double *)calloc((unsigned)(rs_param.last_row-rs_param.first_row+2)
, sizeof(double)) ;

  /* copy data from look table into arrays*/
  for(j=1, i=rs_param.first_row; i <= rs_param.last_row; ++i, ++j)
    {
      /*start at 1 for "fortran" c of NR*/
      rs_param.disp_data[j]=(double)darray[rs_param.disp_col][i];
      rs_param.mu_data[j]=(double)darray[rs_param.mu_col][i];
    }


  if(lamda < 0)
    {
      sprintf(msg, "\nCalculating rs model for given parameters --no inversion bein
\n\n");
      print_msg(msg);
    }
  else
    {
      sprintf(msg, "Calling qi...%s recs: %d to %d\n\n", head.title, rs_param.f
s_param.last_row);
      print_msg(msg);
    }

  if(exac_qi(converg_tol, lamda, wc, &rs_param) == -1)
    {
      cfree(rs_param.model_mu);
      cfree(rs_param.mu_data);
      cfree(rs_param.disp_data);
      sprintf(msg, "problem in qi inversion\n");
      print_msg(msg);
```



```
      action = MAIN;
      top();
      return;
    }


  /*nameing BS bare so we don't call name()*/
  if(strncmp(head.ch[rs_param.mu_fit_col].name,"no_val",6) == 0)
    head.nchan += 1 ;
  head.ch[rs_param.mu_fit_col].nelem = head.ch[rs_param.disp_col].nelem ;

  /* name for model fric. col is "mod_mu". Units are "." */
  if(rs_param.dc2<0)
    strcpy(head.ch[rs_param.mu_fit_col].name, "mod_mu1_");
  else
    strcpy(head.ch[rs_param.mu_fit_col].name, "mod_mu2_");

  strcat(head.ch[rs_param.mu_fit_col].name,&rs_param.law);
  strcpy(head.ch[rs_param.mu_fit_col].units, "." );

  /* install model data */
  for(j=1, i=rs_param.first_row; i <= rs_param.last_row; ++i, ++j)
    darray[rs_param.mu_fit_col][i]=(float)rs_param.model_mu[j];

  cfree(rs_param.model_mu);
  cfree(rs_param.mu_data);
  cfree(rs_param.disp_data);

  sprintf(msg, "QI: DONE\n");
  print_msg(msg);
  action = MAIN;
  top();
}



do_qi_help()
{

  sprintf(msg, "> An iterative, non-linear inverse routine using the\n\tLevenberg-Ma
t method (variable damping) and svd to solve the inverse problem.\n");
  print_msg(msg);

  sprintf(msg, "> Default is to invert for the Ruina (slip law) rate and state
w,\n");
  print_msg(msg);

  sprintf(msg, "> \tother laws can also be used (see options below) \n");
  print_msg(msg);

  sprintf(msg, "> tol is the convergence tolerance (1-chisq/prev_chisq), default is
");
  print_msg(msg);

  sprintf(msg, "> lamda is the Levenberg-Marquardt factor (0.1 is a good start)\n"
  print_msg(msg);

sprintf(msg, "> wc is a weighting factor for the weight vector w[i] = (ndata/i)"
print_msg(msg);

  sprintf(msg, "> for weighting: i is relative to weight_row (this will normally
)\n");
print_msg(msg);
```



```
  sprintf(msg, "%f \n", mu_fit_mess_2);
  print_msg(msg);

  sprintf(msg, "> A linear term can be added to the friction law to accoun
weakening\n\t\t: mu = mu_* + a ln(V/V*) + b phi + (c)*dx, where c has units
Set c=0 for no lin_term\n");
  print_msg(msg);

  sprintf(msg, "> This routine needs: the columns that contain disp. and fri
# of the velocity step,\n");
  print_msg(msg);

  sprintf(msg, "> the row # to which the fit should be extended, and initial g
be various constitutive parameters.\n");
  print_msg(msg);

  sprintf(msg, "> Give k in the units of 1/dc (e.g., friction/dc).\n");
  print_msg(msg);

  sprintf(msg, "> The model output has the same number of rows as the input da
  print_msg(msg);

  sprintf(msg, ">If lamda is < 0, a rate state calc. is done for the init
ut in mod_mu_col\n");
  print_msg(msg);

  sprintf(msg, ">If wc is < 0 weighting is not done via the standard function
nstead data between weight_row and end_weight_row are weighted by a factor of wc
other data.\n");
  print_msg(msg);

  sprintf(msg, ">Multiple velocity steps may be computed (set lamda<0) or im
he command 'qi_mvs'\n");
  print_msg(msg);

  sprintf(msg, ">The following options are available by appending '#' to the comm
e # is:\n");
  print_msg(msg);

  sprintf(msg, "\t.i turn off interactive mode (on by default) which queries
to continue iterating in some situations.\n");
  print_msg(msg);

  sprintf(msg, "\t.f final mode: create data table. Once turned on this stays on
is used again\n");
  print_msg(msg);

  sprintf(msg, "\t.n create new log file rather than append to existing fil
troyed).\n");
  print_msg(msg);

  sprintf(msg, "\t.s write a separate log file for each step (by appending vs_ro
ame).\n");
  print_msg(msg);

  sprintf(msg, "\t.v write a slightly more verbose log file.\n");
  print_msg(msg);

  sprintf(msg, "\t.d invert for the Dieterich-Ruina, slowness law\n");
  print_msg(msg);

  sprintf(msg, "\t.p invert for the Perrin-Rice, quadratic law\n");
  print_msg(msg);
```

```
  sprintf(msg,  "\t.j  invert  for  the  Rice  law\n");
  print_msg(msg);

  sprintf(msg,  ">multiple  options  are  allowed,  use  (e.g.)  .nvi\n");
  print_msg(msg);

}



/************************************ scm ***************************/
/*  calls  simp_func()  in  simplex1.c,  simp_weight()  function  is  moved  from  look_funcs
  this  section  */


do_scm(arg)
     char  arg[256];
{
  nocom(arg);

  if  (sscanf(arg,"%s  %d  %d  %d  %d  %d  %lf  %lf  %lf  %lf  %lf  %lf  %lf  %lf  %lf",
            desire,
            &rs_param.disp_col,  &rs_param.mu_col,  &rs_param.vs_row,
            &rs_param.last_row,  &rs_param.mu_fit_col,
            &rs_param.stiff,  &rs_param.sig_n,  &rs_param.vo,  &rs_param.vf,
            &rs_param.muo,
            &rs_param.muf,  &rs_param.a,  &rs_param.b1,  &rs_param.dc1,
            &rs_param.dc2)  !=  16)
    {
      nea();
      action  =  MAIN;
      top();
      return;
    }

  adj_init_disp  =  mu_check(&rs_param.muo,1);  /*  returns  false  if  passed  -33  */
  mu_check(&rs_param.muf,0);        /*  this  is  in  look_funcs.c  */
  rs_param.amb  =  (rs_param.muf-rs_param.muo)/log(rs_param.vf/rs_param.vo);
  if(rs_param.dc2  <  0.0)
    {
      rs_param.b2  =  0.0;
      rs_param.dc2  =  1e10;            /*  I  think  leaving  it  neg.  will  lead  to  m
being  called  */
      rs_param.b1  =  rs_param.a  -  rs_param.amb;
      1  =  2;
    }
  else
    {
      rs_param.b2  =  -(rs_param.amb-rs_param.a+rs_param.b1);
      1  =  4;
    }

  if(rs_param.disp_col  >=  max_col  ||  rs_param.mu_col  >=  max_col  ||  rs_param.mu_fit_co
  max_col)
    {
      coe();
      top();
      action  =  MAIN;
      return;
    }

  if(rs_param.last_row  <  rs_param.vs_row  ||  rs_param.last_row  >  head.ch[rs_param.disp_c
.nelem)
```



```
    {
      sprintf(msg,"vs_step  begin  (%d)  >  end  (%d)  or  end  >  nelems.\n",  rs_param.vs_
param.last_row);
      print_msg(msg);
      top();
      action  =  MAIN;
      return;
    }

  /*nameing  BS  here  so  we  don't  call  name()  */
  if(strncmp(head.ch[rs_param.mu_fit_col].name,"no_val",6)  ==  0)
    head.nchan  +=  1  ;
  head.ch[rs_param.mu_fit_col].nelem  =  head.ch[rs_param.disp_col].nelem  ;

  /*  name  for  model  fric.  col  is  "simp_mu".  Units  are  "."  */
  if(1==2)
    strcpy(head.ch[rs_param.mu_fit_col].name,  "simp_1_mu");
  else
    strcpy(head.ch[rs_param.mu_fit_col].name,  "simp_2_mu");

  strcpy(head.ch[rs_param.mu_fit_col].units,  ".");

  /*  set  weighting.  rs_param.weight_control  is  neg.  if  */
  /*pre-peak  overestimate  is  favored  over  underestimate  and  postpeak*/
  /*  underestimate  is  favored  over  underestimate  */

  /*simp_weight(command,1,&temp_int);  */


  if(1==2)
    {
      sprintf(msg,  "\nEnter  the  simplex  parameters:  max_iterations,  peak  row  #,  row
eight  to,  parameter  error  (a,  dc,  and  total  error)  ,  and  first  step  size  (a  a
      print_msg(msg);

      sprintf(msg,  ">>  Enter  -1  for  weight  row  to  get  less  than  std  weighting  o
no  weighting\n");
      print_msg(msg);

      sprintf(msg,  ">>  Enter  -1*(peak  row)  to  favor  overestimate  of  mu  between  th
and  the  peak  row\n");
      print_msg(msg);

      sprintf(msg,  ">>  If  you  want  (very  tight)  defaults  for  the  error  and  step
a_err  to  <  0\n");
      print_msg(msg);

      sprintf(msg,  "\tInput:  max_iter,  peak_row,  wt_row,  a_err,  dc_err,  total_err,  a
dc_step\n");
      print_msg(msg);

      set_left_footer("Type  the  arguments  for  simplex  weight");
      set_cmd_prompt("Input:  ");
      action  =  SCM_SIMP_WEIGHT_L2;
      return;
    }
  else
    {
      sprintf(msg,  "Enter  the  simplex  parameters:  max_interations,  peak  row  #,  row
ight  to,  parameter  error  (a,  b1,  dc1,  dc2,  and  total  error)  ,  and  first  step
dc1,  and  dc2)\n");
      print_msg(msg);

      sprintf(msg,  ">>  Enter  -1  for  weight  row  to  get  less  than  std  weighting  c
```



```
no  weighting\n");
      print_msg(msg);

      sprintf(msg,  ">>  Enter  -1*(peak  row)  to  favor  overestimate  of  mu  between  th
and  the  peak  row\n");
      print_msg(msg);

      sprintf(msg,  ">>  If  you  want  (very  tight)  defaults  for  the  error  and  step
a_err  to  <  0\n");
      print_msg(msg);

      sprintf(msg,  "\tInput:  max_iter,  peak_row,  wt_row,  a_err,  b1_err,  dc1_err,  dc2_
total_err,  a_step,  b1_step,  dc1_step,  dc2_step\n");
      print_msg(msg);

      set_left_footer("Type  the  arguments  for  simplex  weight");
      set_cmd_prompt("Input:  ");
      action  =  SCM_SIMP_WEIGHT;
      return;
    }
}


do_simp_weight_l2(arg)
     char  arg[128];
{
  /*  int  l,  *temp_int;  */
  int    i  ,  j  ;

  nocom(arg);

  if  (sscanf(arg,  "%d  %d  %d  %lf  %lf  %lf  %lf  %lf",  temp_int,  &rs_param.peak_row,  &
weight_pts,  &rs_param.a_er,  &rs_param.dc1_er,  &rs_param.total_er,&rs_param.a_step,&rs_par
m.dc1_step)  !=  8)
    {
      nea();
      top();
      action  =  MAIN;
      return;
    }

  if(  rs_param.a_er  <  0)
    {
      if(rs_param.a_er  ==  -22)
        {
          rs_param.a_er  =  1.00000e-4;  /*  less  stringent  values  */
          rs_param.dc1_er  =  1.00000e-3;
          rs_param.total_er  =  1.00000e-4;
        }

      else
        {
          rs_param.a_er  =  1.00000e-8;  /*  install  default  values  */
          rs_param.dc1_er  =  1.00000e-8;
          rs_param.total_er  =  1.00000e-9;
        }

      rs_param.a_step  =  0.010;
      (rs_param.dc1  >  1.2)  ?  rs_param.dc1_step  =  -1.0  :  rs_param.dc1_step  -0.5;
    }

  do_simp_weight_final();

}
```



```
do_simp_weight(arg)
     char  arg[128];
{
  nocom(arg);

  if  (sscanf(arg,  "%d  %d  %d  %lf  %lf  %lf  %lf  %lf  %lf  %lf  %lf  %lf",
            temp_int,  &rs_param.peak_row,  &rs_param.weight_pts,
            &rs_param.a_er,  &rs_param.b1_er,
            &rs_param.dc1_er,  &rs_param.dc2_er,  &rs_param.total_er,
            &rs_param.a_step,  &rs_param.b1_step,
            &rs_param.dc1_step,  &rs_param.dc2_step)  !=  12)
    {
      nea();
      top();
      action  =  MAIN;
      return;
    }

  if(  rs_param.a_er  <  0)
    {
      if(rs_param.a_er  ==  -22)
        {
          rs_param.a_er  =  1.00000e-4;  /*  less  stringent  values  */
          rs_param.b1_er  =  1.00000e-4;
          rs_param.dc1_er  =  1.00000e-3;
          rs_param.dc2_er  =  1.00000e-3;
          rs_param.total_er  =  1.00000e-4;
        }
      else
        {
          rs_param.a_er  =  1.00000e-8;  /*  install  default  values  */
          rs_param.b1_er  =  1.00000e-8;
          rs_param.dc1_er  =  1.00000e-8;
          rs_param.dc2_er  =  1.00000e-8;
          rs_param.total_er  =  1.00000e-9;
        }

      rs_param.a_step  =  0.01;
      rs_param.b1_step  =  0.01;
      rs_param.dc1_step  =  -0.5;
      rs_param.dc2_step  =  -5.0;
    }

  do_simp_weight_final();
}


do_simp_weight_final()
{

  /*  gives  weight  of  5x  for  100  pts*/
  /*  peak  is  weighted  6x  weight  and  interpolated  pts  are  weighted  0.75x  weight
  rs_param.weight  =  (rs_param.last_row-rs_param.vs_row)/20.00;

  /*  reduce  weighting  optionally  or  for  big  data  sets  */
  if(  (rs_param.last_row-rs_param.vs_row  >  300)  ||  (rs_param.weight_pts  <  0)  )
    {
      rs_param.weight  /=  (rs_param.last_row-rs_param.vs_row)/100.0;
      sprintf(msg,  "weight  limited  optionally  or  because  data  set  >  300..%c\n",BEI
      print_msg(msg);

      if(rs_param.weight_pts  <  0)
```

```
        rs_param.weight_pts  *= -1;
      }

    if(rs_param.weight  <  2)
      rs_param.weight  =  2.00;

    if(rs_param.weight_pts  ==  0)          /*  no  weighting  */
      rs_param.weight=1.00;

    if(rs_param.peak_row  <  0)
      {
        rs_param.peak_row  *= -1;
        rs_param.weight_control  =  -1;
      }
    else
      rs_param.weight_control  =  1;

    do_scm_next();

}



do_scm_next()
{

  /*  allocate  space  for  arrays  used  by  simplex  and  fq  */
  rs_param.disp_data  =  (double  *)calloc((unsigned)(rs_param.last_row-rs_param.vs_row+13),
sizeof(double))  ;
  rs_param.mu_data  =  (double  *)calloc((unsigned)(rs_param.last_row-rs_param.vs_row+13),
sizeof(double))  ;
  rs_param.model_mu  =  (double  *)calloc((unsigned)(rs_param.last_row-rs_param.vs_row+13),
sizeof(double))  ;

  disp_ptr  =  rs_param.disp_data;
  mu_ptr  =  rs_param.mu_data;

  if(adj_init_disp)
    {
      /*  adjust  disp.  of  first  pt  to  account  for  mu0  (possibly)  not  =  mu[vs_row]

     sl  =  (darray[rs_param.mu_col][rs_param.vs_row+1]-darray[rs_param.mu_col][rs_param.vs
_row])  /
          (darray[rs_param.disp_col][rs_param.vs_row+1]-darray[rs_param.disp_col][rs_param.v
s_row])  ;

      dx  =  (rs_param.mu0-darray[rs_param.mu_col][rs_param.vs_row])/sl  +  darray[rs_param.d
sp_col][rs_param.vs_row];

      if  (dx  >  darray[rs_param.disp_col][rs_param.vs_row+1])
        {
          sprintf(msg,"Problem  with  getting  init.  disp.  correct  for  input  init.  mu.'
          print_msg(msg);
          sprintf(msg,"Sorry,  but  you'll  have  to  fix  something  or  change  init.  mu
oser  to  mu  of  vs_row.\n");
          print_msg(msg);
          top();
          action  =  MAIN;
          return;
        }
      else
        *disp_ptr++  =  dx;       /*  install  as  vs_row  */
    }
  else
```



```
      *disp_ptr++=darray[rs_param.disp_col][rs_param.vs_row];


      *mu_ptr++  =  rs_param.mu0;     /*  install  as  vs_row  */

      /*  disp,  mu  pt  is  now  on  a  line  <>  vs  pt  and  next  pt  */

      rs_param.added_pts  =  0;              /*  default  value  */

      if(rs_param.peak_row-rs_param.vs_row  <  4  )
        {
          /*  take  avg.  slope  -weighted  a  bit  more  toward  peak  pt  */
          sl  =  0.0;
          for(j  =  rs_param.vs_row+1;  j  <=  rs_param.peak_row;  ++j)
            sl  +=  (darray[rs_param.mu_col][j]-  *rs_param.mu_data)/(darray[rs_param.disp_
[
}-  *rs_param.disp_data);

          /*  weight  slope  to  peak  */
          sl  +=  (darray[rs_param.mu_col][rs_param.peak_row]-  *rs_param.mu_data)/(darray[rs_pa
am.disp_col][rs_param.peak_row]-  *rs_param.disp_data);
          sl  /=  (double)  (rs_param.peak_row-rs_param.vs_row  +1);

          dx  =  (darray[rs_param.disp_col][rs_param.peak_row]-  *rs_param.disp_data)  /  5

          /*  16/6/93  changed  this  from  10  points  between  peak  and  vs_row  to  5  point
          for(i=1;  i  <=  5;  ++i)  /*  interpolated  points  */
            {
              *disp_ptr++  =  *rs_param.disp_data  +  dx*(double)i;
              *mu_ptr++  =  *rs_param.mu_data  +  sl*dx*(double)i;
            }
          rs_param.added_pts  =  (int)(floor(5.0/(rs_param.peak_row  -  rs_param.vs_row)));

        }
      else
        {
          for(j=rs_param.vs_row+1;  j  <=rs_param.peak_row;  ++j)
            {
              *disp_ptr++  =  darray[rs_param.disp_col][j]  ;
              *mu_ptr++  =  darray[rs_param.mu_col][j]  ;
            }
        }

      for(j=rs_param.peak_row+1;  j  <=rs_param.last_row;  ++j)
        {
          *disp_ptr++  =  darray[rs_param.disp_col][j]  ;
          *mu_ptr++  =  darray[rs_param.mu_col][j]  ;
        }

      /*  so  that  calc  stops  properly*/
      *disp_ptr  =  darray[rs_param.disp_col][rs_param.last_row]+1e9  ;

      /*  to  check  interp  pts..  */
      /*    disp_ptr  =  rs_param.disp_data;
            mu_ptr  =  rs_param.mu_data;
            for(i=rs_param.vs_row;  i  <  rs_param.last_row+12;  ++i)
            {
            darray[rs_param.disp_col][i]  =  *disp_ptr++;
            darray[rs_param.mu_col][i]  =  *mu_ptr++;
            }
            break;          */

      strcpy(t_string,  "rs_fit");                  /*  for  simplex1  */
```



```
    if(rs_param.a  <  0)            /*calculate  error  matrix*/
      {
        sprintf(msg,"Calculate  the  chi-squarred  error  between  a  model  and  data  (lsv  c
Output  is  to  a  file  named  cse_exp_name\n\tEnter  the  range  for  a  and  dc:  ta_min,
nc,  dc_min,  dc_max,  dc_inc");
        print_msg(msg);

        set_left_footer("Type  the  arguments");
        set_cmd_prompt("Input:  ");

        /*  get  args  for  scm_1
            call  do_scm_1()
            DONE
            */

        action  =  SCM_1;
        return;
      }


    else  /*  just  do  the  simp  calc  like  I  used  to  */
      {
        /*  temp_int  =  max_iterations  */
        sprintf(msg,  "calling  simp_func...  %s  recs:  %d  to  %d\n",  head.title,  rs_param.
,  rs_param.last_row);
        print_msg(msg);

        /*
            set  simp_func_action  so  it  knows  where  to  go  to  after  that
            (because  simp_func  can  also  be  called  from  simplex  cmd)
          call  simp_func()
          then  go  to  do_scm_2()
          DONE
          */

/*
        set_left_footer("Type  the  arguments");
        set_cmd_prompt("Input:  ");
*/

        simp_func_action  =  SCM;
        action  =  SIMP_FUNC;
        return;

        /*  simp_func(command,1,simp_xch,&simp_ych,t_string,first,last,  temp_int);
         */

      }
}



do_scm_1(arg)
    char  arg[128];
{
  if  (sscanf(arg,"%lf  ,  %lf  ,  %lf  ,  %lf  ,  %lf  ,  %lf",&a_min,  &a_max,  &rs_param.a
_min,  &dc_max,  &rs_param.dcl_step)  !=  6)
    {
      nea();
      top();
      action  =  MAIN;
      return;
    }
```



```
    /*open  file  for  data:  x  y  z  vectors*/
    strcpy(t_string,"cse_");
    strcat(t_string,head.title);
    new  =  fopen(t_string,  "w");

    rs_param.b2  =  0.0;            /*lsv  model*/
    rs_param.dc2  =  1e10;
    l  =  2;

    /*put  x  and  y  vectors  as  first  two  rows*/
    for(rs_param.a=a_min;  rs_param.a<=a_max;  rs_param.a  +=rs_param.a_step)
      fprintf(new,"%f\t",rs_param.a);
    fprintf(new,"\n");
    for(rs_param.dc1=dc_min;  rs_param.dc1<=dc_max;  rs_param.dc1  +=rs_param.dc1_step)
      fprintf(new,"%f\t",rs_param.dc1);
    fprintf(new,"\n");

    for(rs_param.a=a_min;  rs_param.a<=a_max;  rs_param.a  +=rs_param.a_step)
      {
        rs_param.b1  =  rs_param.a  -  rs_param.amb;
        for(rs_param.dc1=dc_min;  rs_param.dc1<=dc_max;  rs_param.dc1  +=rs_param.dc1_step)
          {
            err=simp_rate_state_mod();
            fprintf(new,"%g\t",err);      /*put  in  matrix  form  for  contour  plot*/
            /*fprintf(new,"%f\t%f\t%g\n",rs_param.a,rs_param.dc1,err);*/
            sprintf(msg,  "loops  over  'a'  remaining  %d,  loops  over  'dc'  remaining
,(int)((a_max-rs_param.a)/rs_param.a_step)+1,  (int)((dc_max-rs_param.dc1)/rs_param.dc1_ste
p)+1);
          }
        fprintf(new,"\n");
      }
    fclose(new);
    sprintf(msg,  "\n");

    cfree(rs_param.model_mu);
    cfree(rs_param.mu_data);
    cfree(rs_param.disp_data);

    sprintf(msg,  "SCM:  DONE\n");
    print_msg(msg);

    action  =  MAIN;
    top();
}



do_scm_2()
{
  if(l  ==  2  )
    {
      sprintf(msg,  "a-b  =  %g\t  Best  simplex  fit  (vs_row=%d):  a=%g\tb=%g\tdc=%g\n",  r
m.amb,  rs_param.vs_row,  rs_param.a,  rs_param.b1,  rs_param.dc1);
      print_msg(msg);
    }
  else
    {
      sprintf(msg,  "a-b  =  %g\t  Best  simplex  fit  (vs_row=%d):  a=%g\tb1=%g\tb2=%g\tdc1=
c2=%g\n",  rs_param.amb,  rs_param.vs_row,  rs_param.a,  rs_param.b1,  rs_param.b2,  rs_paran
1,  rs_param.dc2);
      print_msg(msg);
    }

  for  (  i=  (rs_param.vs_row  >  15)  ?  (rs_param.vs_row-15)  :  0;  i  <=  rs_param.vs_r
```

```
    darray[rs_param.mu_fit_col][i] = rs_param.muo;;

  /* install model data */
  mu_ptr = rs_param.model_mu;

  if(rs_param.added_pts != 0)
    {
      disp_ptr = rs_param.disp_data+1;  /* recall vs_row is in array*/

      dx = 0.0;
      for(i=0; i < 10; ++i);
      {
        dx += fabs( *(disp_ptr+1) - *disp_ptr++)/50.00 ;
      }
      dx /= 10.00;
      if(dx < 0.0200) dx = 0.0200;

      disp_ptr = rs_param.disp_data+1;

      for(i=rs_param.vs_row+1, j = 0; j < 5; ++j)
        {
          /* do it this way, since if pts <> vs_row and peak are now evenly sp
t simply jump ahead x pts to find the mu value corresponding to x at a given
          if( (darray[rs_param.disp_col][i] - *(disp_ptr++)) < dx)
            darray[rs_param.mu_fit_col][i++] = *mu_ptr++ ;
          else
            mu_ptr++;
        }

      if( i != rs_param.peak_row+1]
        {
          sprintf(msg, "Problem installing data from simp. array %c%c\n",BELL,BELL);
          print_msg(msg);
        }

      for(i=rs_param.peak_row+1; i <=rs_param.last_row; ++i)
        darray[rs_param.mu_fit_col][i] = *mu_ptr++ ;
    }

  else
    for(i=rs_param.vs_row+1; i <=rs_param.last_row; ++i)
      darray[rs_param.mu_fit_col][i] = *mu_ptr++ ;

  cfree(rs_param.model_mu);
  cfree(rs_param.mu_data);
  cfree(rs_param.disp_data);

  sprintf(msg, "SCM: DONE\n");
  print_msg(msg);

  action = MAIN;
  top();
}

do_scm_help()
{
  sprintf(msg,"> This function does a simplex fit of the Dieterich-Ruina rate/state
le friction model to data.\n");
  print_msg(msg);
  sprintf(msg,"> It needs: the columns that contain disp. and friction, the row o
elocity step,\n");
  print_msg(msg);
  sprintf(msg,"> the row # to which the fit should be extended, and initial gues
e various constitutive parameters.\n");
```



```
  print_msg(msg);
  sprintf(msg,"> Give k in the same units as normal stress and dc.\n");
  print_msg(msg);
  sprintf(msg,"> The pre-step mu value is extended back 15 rows in the model out
  print_msg(msg);
  sprintf(msg, mu_fit_mess_1);
  print_msg(msg);
  /*sprintf(msg,"> To get a one state variable fit set Dc2 to < 0. ");
    print_msg(msg);
    sprintf(msg,">      In this case b1 will be set automatically ");
    print_msg(msg);
    sprintf(msg,"> On picking the velocity step row: Pick v_step row as first row
step\n>\tThe displacement of that point is adjusted to account for mu_o (possibly)
u(v_step row)");
    print_msg(msg);
    sprintf(msg,"> On mu: enter -22 to get the average mu value within a 10point
ound vs_row or end_row\n\tenter -33 to get the mu value 1pt before either row o
y other negative number to get the mu value at vs_row or end_row");
    print_msg(msg); */
}

do_scm_info()
{
  sprintf(msg,"Simplex fit of one or two state variable friction model. (scm_h or
_h gives a detailed description of this function) \n");
  print_msg(msg);

  sprintf(msg,"> To get a one state variable fit set Dc2 to < 0. b1 will be
lly \n");
  print_msg(msg);

  sprintf(msg,"\nInput: disp. col., mu col, row # of vel. step and end of data
ol for model_mu, stiffness (k), Sigma_n, v_initial (v_o), v_final, mu_o, mu_f, a,
, and Dc2, tdisp_col, mu_col, vs_row, end_row, n_mu_col, k, Sigma_n, v_o, vf, mu
, mu_f, a, b1, Dc1, Dc2: ") ;
  print_msg(msg);
}
```



```
#include "global.h"
#include <string.h>
#include <sys/file.h>
#include <xview/frame.h>
#include <xview/canvas.h>
#include <X11/Xlib.h>
#include <xview/xv_xrect.h>
#include <xview/panel.h>
#include <xview/xview.h>

char pathname[10][80];
char default_path[80];
char metapath[80];
char data_file[20];
char new_file[20];
char headline[20];

FILE *data, *fopen(), *new;
int reed();

int doit_des, doit_f_open, meta_fd, errno;
int plot_error, oldplot;
extern char plot_cmd[128], trig_cmd[128];;

extern int active_window;
extern int old_active_window;
extern int total_windows;
extern int action;
extern char msg[MSG_LENGTH];

extern void redraw_proc();
extern Frame main_frame;
extern Panel_item cmd_panel_item;


new_win_proc()
{
  total_windows++;
  create_canvas();
  sprintf(msg, "Total windows = %d\n", total_windows+1);
  print_msg(msg);
}


set_active_window(win_num)
    int win_num;
{
  if (wininfo.windows[win_num] != 1)
    {
      sprintf(msg, "Window #%d does not exist.\n", win_num+1);
      print_msg(msg);
      action = MAIN;
      top();
      return;
    }

  if (active_window != win_num)
    {
      if (win_num == old_active_window)
        {
          old_active_window = active_window;
          active_window = win_num;
        }
```



```
      else
        {
          old_active_window = active_window;
          active_window = win_num;
        }
    }

  display_active_window(active_window+1);
  display_active_plot(wininfo.canvases[active_window]->active_plot+1);
}


set_active_plot(i)
    int i;
{
  canvasinfo *can_info;
  Canvas canvas;
  Xv_Window canvas_xv_window;

  can_info = wininfo.canvases[active_window];
  canvas = can_info->canvas;
  canvas_xv_window = canvas_paint_window(canvas);

  can_info->active_plot = i;
  redraw_proc(canvas, canvas_xv_window, xv_get(canvas, XV_DISPLAY),
              xv_get(canvas_xv_window, XV_XID), NULL);
}



del_plot_proc(pn)
    int pn;
{
  canvasinfo *can_info;
  int i;

  pn--;

  can_info = wininfo.canvases[active_window];

  if (can_info->alive_plots[pn] == 0)
    {
      sprintf(msg, "Plot does not exist.\n");
      print_msg(msg);
      top();
      action = MAIN;
      return;
    }

  can_info->total_plots--;
  can_info->alive_plots[pn] = 0;
  free(can_info->plots[pn]);
  if ((can_info->active_plot == pn) && (can_info->active_plot != 0))
    can_info->active_plot--;

  clear_canvas_proc(can_info->canvas);
  can_info->total_plots = -1;

  /* redraw can_info->canvas */
  for (i=0; i<10; i++)
      {
        if (can_info->alive_plots[i] == 1)
          {
            can_info->active_plot = i;
```

```
            can_info->total_plots++;
            redraw_proc(can_info->canvas,
                        can_info->xvwin,
                        xv_get(can_info->canvas, XV_DISPLAY),
                        can_info->win, NULL);
          }
        }

  action = MAIN;
  top();
}


kill_win_proc(wn)
    int wn;
{
  int i;
  Frame graf_frame;
  canvasinfo *can_info;

  wn--;

  if (wininfo.windows[wn] == 0)
    {
      sprintf(msg, "Window #%d does not exist.\n", wn+1);
      print_msg(msg);
      action = MAIN;
      top();
      return;
    }

  sprintf(msg, "destroying window #%d\n", wn+1);
  print_msg(msg);

  can_info = wininfo.canvases[wn];

  /* free all the plots in this window */
  for (i=0; i<10; i++)
    {
      if(can_info->alive_plots[i])
        free(can_info->plots[i]);
    }

  /* kill the xwindow */
  graf_frame = xv_get(can_info->canvas, XV_KEY_DATA, GRAF_FRAME);
  xv_destroy_safe(graf_frame);

  /* free the canvasinfo for this window */
  free(can_info);

  if (active_window == wn)
    {
      active_window = old_active_window;
      old_active_window = get_old_active_window(active_window);
    }

  if (old_active_window == wn)
    {
      old_active_window = get_old_active_window(old_active_window);
    }

  total_windows--;
  if (total_windows == -1)
    {
```



```
      active_window = -1;
      old_active_window = -1;
    }

  display_active_window(active_window+1);
  /* if no window is opened, active_plot = NONE */
  if (active_window != -1)
    display_active_plot(wininfo.canvases[active_window]->active_plot+1);
  else
    display_active_plot(-1);

  wininfo.windows[wn] = 0;

  sprintf(msg, "Total windows = %d\n", total_windows+1);
  print_msg(msg);

  action = MAIN;
  top();

}


get_old_active_window(wn)
    int wn;
{
  int i;

  if (wn > 0)
    {
      for (i=wn-1; i>=0; i--)
        {
          if (wininfo.windows[i] == 1)
            return i;
        }
    }

  for (i=wn+1; i<10; i++)
    {
      if (wininfo.windows[i] == 1)
        return i;
    }
  return -1;
}


get_new_plot_num(alive_plots)
    int alive_plots[10];
{
  int i;

  for (i = 0; i < 10; i++)
    {
      if (alive_plots[i] == 0)
        return i;
    }
  /* reached plots limit.. need to delete one or more plots from window */
  return(-1);
}


plotting_error(pn)
    int pn;
{
  canvasinfo *can_info;
```



```
  int i;

  can_info = wininfo.canvases[active_window];
  can_info->alive_plots[pn] = 0;
  can_info->total_plots--;
  free(can_info->plots[pn]);
  for(i=0; i<10; i++)
    {
      if (can_info->alive_plots[i] == 1)
        {
          can_info->active_plot = i;
          return;
        }
    }
  can_info->active_plot = -1;
}


do_plot(cmd)
    char cmd[128];
{
  char desire[128];
  int i, j;
  canvasinfo *can_info;
  plotarray *data;
  int start_x, start_y, end_x, end_y;
  int begin, end;
  int xaxis, yaxis;
  float xmax, xmin, ymax, ymin;
  int plot_num;
  float scale_x, scale_y;
  Canvas canvas;
  Xv_Window canvas_xv_window, xvwin;
  Display *dpy;
  Window win;


  plot_error = 0;

  nocom(cmd);

  if (total_windows == -1) new_win_proc();

  can_info = wininfo.canvases[active_window];

  if (can_info->total_plots >= 9)
    {
      sprintf(msg, "Reached limit of 10 plots per window! Ignored\n");
      print_msg(msg);
      top();
      action = MAIN;
      return;
    }

  if (strncmp(plot_cmd, "plotover", 8) == 0)
    {
      if (can_info->total_plots == -1)
        {
          sprintf(msg, "No plot exist.\n");
          print_msg(msg);
          top();
          action = MAIN;
          return;
        }
```



```
    }

  can_info->total_plots += 1;
  /* printf("(plot proc)total_plots: %d\n", can_info->total_plots+1); */

  /* needed for plotover */
  oldplot = can_info->active_plot;

  data = (plotarray *) malloc(sizeof(plotarray));
  if (data == NULL)
    {
      can_info->total_plots -= 1;
      sprintf(msg,"Memory allocation error. Plot cannot be drawn.\n");
      print_msg(msg);
      top();
      action = MAIN;
      return;
    }
  plot_num = get_new_plot_num(can_info->alive_plots);
  can_info->alive_plots[plot_num] = 1;
  can_info->active_plot = plot_num;
  can_info->plots[plot_num] = data;
  /* printf("plot_num: %d active_plot: %d\n", plot_num+1, can_info->active_plot+1   *,

  if (strcmp(plot_cmd, "pa") == 0)
    data->label_type = 1;
  else
    data->label_type = 0;

  if (strcmp(plot_cmd, "plotover") == 0)
    {
      if (sscanf(cmd, "%s %d %d", desire, &xaxis, &yaxis) != 3)
        {
          nea();
          plotting_error(plot_num);
          top();
          action = MAIN;
          return;
        }

      else
        {
          /* copy values from previous plot */
          xmax = can_info->plots[oldplot]->xmax;
          xmin = can_info->plots[oldplot]->xmin;
          ymax = can_info->plots[oldplot]->ymax;
          ymin = can_info->plots[oldplot]->ymin;

          begin = can_info->plots[oldplot]->begin;
          end = can_info->plots[oldplot]->end;

          if (xaxis >= max_col || yaxis >= max_col)
            {
              sprintf(msg, "Out of range. Col not allocated.\n");
              print_msg(msg);
              plotting_error(plot_num);
              top();
              action = MAIN;
              return;
            }
        }
    }

  else
```

```
{
  if (strcmp(plot_cmd, "plotall") == 0)
    if (sscanf(cmd, "%s %d %d", desire, &xaxis, &yaxis) != 3)
      {
        nea();
        plotting_error(plot_num);
        top();
        action = MAIN;
        return;
      }
    else
      {
        begin = 0;
        end = head.nrec - 1;
      }

  else if (strcmp(plot_cmd, "plotsr") == 0)
    {
      if (sscanf(cmd, "%s %d %d", desire, &xaxis, &yaxis) != 3)
        {
          nea();
          plotting_error(plot_num);
          top();
          action = MAIN;
          return;
        }

      begin = can_info->plots[oldplot]->begin;
      end = can_info->plots[oldplot]->end;
    }

  else if (strcmp(plot_cmd, "plotscale") == 0)
    {
      if (sscanf(cmd, "%s %d %d %d %d %f %f %f %f", desire,
                 &xaxis, &yaxis, &begin, &end,
                 &xmax, &xmin,
                 &ymax, &ymin) != 9)
        {
          nea();
          plotting_error(plot_num);
          top();
          action = MAIN;
          return;
        }
    }

  else /* plotauto, plotlog, plotsame, pa */
    {
      if (sscanf(cmd, "%s %d %d %d %d", desire, &xaxis, &yaxis, &begin, &end)
        {
          nea();
          plotting_error(plot_num);
          top();
          action = MAIN;
          return;
        }
    }

  if (xaxis >= max_col || yaxis >= max_col)
    {
      sprintf(msg, "Out of range. Col not allocated.\n");
      print_msg(msg);
      plotting_error(plot_num);
      top();
```



```
      action = MAIN;
      return;
    }

  if (strcmp(plot_cmd, "plotscale") == 0 || strcmp(plot_cmd, "plotauto") == 0 |
p(plot_cmd, "plotlog") == 0 || strcmp(plot_cmd, "plotsame") == 0)
    {
      if (end > head.nrec-1)
        {
          sprintf(msg, "nrec = %d, truncating interval to (%d, %d)\n", head.nrec
n, head.nrec-1);
          print_msg(msg);
          end = head.nrec - 1;
        }
      if (check_row(&begin, &end, xaxis) != 0)
        {
          plot_error = 1;
          begin = 0;
        }
    }

  /* end reading arguments */

  if (strncmp(desire, "plotauto", 8) == 0 || strncmp(desire, "plotall", 7) ==
ncmp(desire, "plotsr", 6) == 0 || strncmp(desire, "pa", 2) == 0)
    {
      stats(&xaxis, &begin, &end);
      xmax = col_stat.max;
      xmin = col_stat.min;

      stats(&yaxis, &begin, &end);
      ymax = col_stat.max;
      ymin = col_stat.min;
    }

  else if (strncmp(desire, "plotlog", 7) == 0)
    {
      stats(&xaxis, &begin, &end);
      if (col_stat.min <= 0)
        {
          sprintf(msg, "Negative numbers within range. Can't do log_plot.\n");
          print_msg(msg);
          plot_error = 1;
          plotting_error(plot_num);
          top();
          action = MAIN;
          return;
        }
      else
        {
          xmax = ceil(log10((double)col_stat.max));
          xmin = floor(log10((double)col_stat.min));
        }
      stats(&yaxis, &begin, &end);
      if (col_stat.min <= 0)
        {
          sprintf(msg, "Negative numbers within range. Can't do log_plot.\n");
          print_msg(msg);
          plot_error = 1;
          plotting_error(plot_num);
          top();
          action = MAIN;
          return;
        }
```



```
      else
        {
          ymax = ceil(log10((double)col_stat.max));
          ymin = floor(log10((double)col_stat.min));
        }
    }
  else if (strncmp(desire, "plotsame", 8) == 0)
    {
      xmax = can_info->plots[oldplot]->xmax;
      xmin = can_info->plots[oldplot]->xmin;
      ymax = can_info->plots[oldplot]->ymax;
      ymin = can_info->plots[oldplot]->ymin;
    }

  if ((end < begin) || (xmax <= xmin) || (ymax <= ymin))
    {
      /*     printf("begin:%d end:%d max.xaxis:%.3f min.xaxis:%.3f max.yaxis:%.3f r
.yaxis:%.3f\n", begin, end, xmax, xmin, ymax, ymin); */
      sprintf(msg, "Limits problem: MAX <= MIN ???\n");
      print_msg(msg);

      plotting_error(plot_num);
      plot_error = 1;
      top();
      action = MAIN;
      return;
    }
}

/* sets all the variables in plot_array data structure */
data->begin = begin;
data->end = end;
data->col_x = xaxis;
data->col_y = yaxis;
data->xmax = xmax;
data->xmin = xmin;
data->ymax = ymax;
data->ymin = ymin;
data->nrows_x = end - begin +1;
data->nrows_y = end - begin +1;
data->mouse = 0;
data->x1 = 0;
data->x2 = 0;
data->y1 = 0;
data->y2 = 0;
data->p1 = 0;
data->p2 = 0;
data->zp1 = 0;
data->zp2 = 0;

data->xarray = (float *)malloc(sizeof(float)*data->nrows_x);
data->yarray = (float *)malloc(sizeof(float)*data->nrows_x);


/* copy data from darray to xarray and yarray (plot data arrays) */
j = 0;
if (strncmp(desire, "plotlog", 7) == 0)
  {
    for (i = begin; i <= end; ++i)
      {
        data->xarray[j] = log10((double)darray[xaxis][i]);
        data->yarray[j] = log10((double)darray[yaxis][i]);
        j++;
      }
```



```
  }
else
  {
    for (i = begin; i <= end; ++i)
      {
        data->xarray[j] = darray[xaxis][i];
        data->yarray[j] = darray[yaxis][i];
        j++;
      }
  }


/*
   sprintf(msg, "Active window: %d Canvas num: %d Plot num: %d\n", active_windo
, can_info->canvas_num+1, can_info->active_plot+1);
   print_msg(msg);
   */

canvas = can_info->canvas;
canvas_xv_window = can_info->xvwin;
win = can_info->win;

/* draw the plot */
redraw_proc(canvas, canvas_xv_window, (Display *)xv_get(canvas, XV_DISPLAY),
        (Window)xv_get(canvas_xv_window, XV_XID), NULL);

top();
action = MAIN;
}



write_proc(arg)
   char arg[128];
{
  char new_file[128];

  strcpy(new_file, arg);

  if ((new = fopen(new_file, "a")) == NULL)
    {
      sprintf(msg, "Can't open file: %s.\n", new_file);
      print_msg(msg);
      action =MAIN;
      top();
      return;
    }
  else
    {
      /* file already exist */
      if ((int)ftell(new) > 0)
        {
          /* ask if want to overwrite old file */
          if (write_show_warning() != 1)
            {
              /* don't overwrite */
              sprintf(msg, "Write aborted!\n");
              print_msg(msg);
              action =MAIN;
              top();
              return;
            }
          else
```

```
        {
          if ((new = fopen(new_file, "w")) == NULL)
            {
              sprintf(msg, "Can't open file: %s.\n", new_file);
              print_msg(msg);
              action =MAIN;
              top();
              return;
            }
          else
            {
              sprintf(msg, "Overwriting %s.\n", new_file);
              print_msg(msg);
            }
        }

      if (act_col() > bead.nchan)
        {
          sprintf(msg, "Active columns = %d\n", act_col);
          print_msg(msg);
          sprintf(msg, "Please compact data array to use columns 1 through head.nch
n", bead.nchan);
          print_msg(msg);
          sprintf(msg, "Reallocation is not necessary.\n");
          print_msg(msg);
          action =MAIN;
          top();
          return;
        }

      rite(new);
      fclose(new);
      action =MAIN;
      top();
    }
}


read_proc(cmd)
    char cmd[128];
{
  char dummy[128];

  sscanf(cmd, "%s %s", dummy, data_file);

  if ((data = fopen(data_file, "r")) == NULL)
    {
      sprintf(msg, "Can't open data file: %s.\n", data_file);
      print_msg(msg);
      action = MAIN;
      top();
      return;
    }
  else
    {
      sprintf(msg, "Reading %s...\n", data_file);
      print_msg(msg);

      /* reed is in filtersm.c */
      if (reed(data, ((strncmp(dummy, "append", 6) == 0) ? TRUE : FALSE)) != 1)
        {
          fclose(data);
          action = MAIN;
```



```
          top();
          return;
        }
      fclose(data);
      sprintf(msg, "Reading %s done.\n", data_file);
      print_msg(msg);
      action = MAIN;
      top();
    }
}


doit_proc(arg)
    char arg[128];
{
  /* reads a doit file and passes the commands to command_handler() directly
  char cmd[128], ch;
  int i;


  if ((doit_f_open+1) > 9)
    {
      sprintf(msg, "Sorry but you can only nest up to 10 doit files.\n");
      print_msg(msg);
      action=MAIN;
      top();
      return;
    }

  /* need to set action to MAIN here so that the commands in doit file c:  is
  action = MAIN;


  strcpy(pathname[doit_f_open+1], default_path);
  strcpy(data_file, arg);
  strcat(pathname[doit_f_open+1], data_file);

  /* printf("doit file: %s\n", pathname[doit_f_open+1]); */

  doit_des = open(pathname[doit_f_open+1], O_RDONLY);

  if (doit_des <= 0)
    {
      sprintf(msg, "Open failed- can't open data file: %s\n", pathname[doit_f_o  ]
      print_msg(msg);
      action = MAIN;
      top();
      return;
    }
  else
    {
      if ((temp_com_file = fdopen(doit_des, "r")) == NULL)
        {
          sprintf(msg, "fdopen failed- can't open data file: %s.\n", pathname[doit_f_
1]);
          print_msg(msg);
          action = MAIN;
          top();
          return;
        }
      else
        {
          if (fscanf(temp_com_file, "%s", cmd) != 1)
            {
```



```
              sprintf(msg,"Cannot read file.\n");
              print_msg(msg);
              fclose(temp_com_file);
              action = MAIN;
              top();
              return;
            }
          printf("1st Command: %s\n", cmd);
          if (strncmp(cmd, "begin", 5) != 0)
            {
              sprintf(msg, "doit file must begin with the string: begin\n");
              print_msg(msg);
              fclose(temp_com_file);
              action = MAIN;
              top();
              return;
            }

          /*
             loop thru the file until EOF;
             read the file line by line, put the string (whole line) in cmd
             pass cmd to command_handler()
             */

          doit_f_open++;
          com_file[doit_f_open] = temp_com_file;

          i = 0;
          while(fscanf(com_file[doit_f_open], "%c", &cmd[i]) != EOF)
            {
              if (cmd[i] == '\n')
                break;
              i++;
            }

          i=0;
          while(fscanf(com_file[doit_f_open], "%c", &cmd[i]) != EOF)
            {
              if (cmd[i] != '\n')
                {
                  i++;
                  continue;
                }
              else
                {
                  cmd[i] = '\0';
                  i = 0;

                  sprintf(msg, "Command: %s\n", cmd);
                  print_msg(msg);

                  if (strncmp(cmd, "end", 3) == 0)
                    {
                      sprintf(msg, "Closing doit file %s.\n",pathname[doit_f_open]);
                      print_msg(msg);

                      fclose(com_file[doit_f_open]);
                      close(doit_des);
                      if (--doit_f_open < 0)
                        doit_f_open = 0;
                      break;
                    }

                  else if (strncmp(cmd, "#", 1) == 0)
                    {
```



```
                      continue;
                    }

                  else
                    {
                      command_handler(cmd);
                    }
                }
            }

          action = MAIN;
          top();
        }
    }
}


set_path_proc(arg)
    char arg[128];
{
  /* change: if path is inaccessible, set path to current path instead of defaul

  /* printf("path: %s def_path: %s\n", arg, default_path); */

  if (access(arg, 4) != 0)
    {
      sprintf(msg, "Inaccessible path: %s.\n", arg);
      print_msg(msg);
    }

  else
    {
      strcpy(default_path, arg);
      if (default_path[strlen(default_path)-1] != '/')
        strcat(default_path, "/");
    }
  sprintf(msg, "Default path is %s\n", default_path);
  print_msg(msg);

  top();
  action = MAIN;
}



all_final_proc(cmd)
    char cmd[128];
{
  char dummy1[128], dummy2[128];
  int i, j;

  if (sscanf(cmd, "%s %s %d %d", dummy1, dummy2, &i, &j) != 4)
    {
      print_msg("Input not recognized.\n");
      action = MAIN;
      top();
      return;
    }

  if ((strcmp(dummy2, "yes") != 0) || (strcmp(dummy2, "y") != 0))
    {
      action = MAIN;
```

```
      top();
      return;
    }

  if  (j==0  ||  j>18  ||  i==0  ||  i>MAX_ROW)
    {
      sprintf(msg,  "Illegal  allocation:  0  <  NROW  <  %d  <  NCOL  <  18\n",  MAX_ROW);
      print_msg(msg);
      action  =  MAIN;
      top();
      return;
    }

  /*  allocate(i,  j);*/
  action  =  MAIN;
  top();

}
```



```
#include  <math.h>
#include  <X11/Xlib.h>
#include  <xview/xview.h>
#include  <xview/canvas.h>
#include  <xview/xv_xrect.h>
#include  <xview/panel.h>

#include  "global.h"

extern  int  active_window;
extern  int  old_active_window;
extern  int  total_windows;
extern  int  action;
extern  char  msg[MSG_LENGTH];

extern  char  plot_cmd[128];

extern  int  tickfontheight,  tickfontwidth;
extern  int  titlefontheight,  titlefontwidth;
extern  GC  gctitle,  gctick;
extern  GC  gcbg;

extern  Frame  main_frame;


void  redraw_proc(canvas,  paint_window,  dpy,  win,  area)
      Canvas  canvas;
      Xv_Window  paint_window;
      Display  *dpy;
      Window  win;
      Xv_xrectlist  *area;
{
  int    i,  j;
  float  x1,  x2,  y1,  y2;
  canvasinfo  *can_info;
  int  plot,  plots;
  plotarray  *data;
  float  xmax,  xmin,  ymax,  ymin;
  int  start_xaxis,  start_yaxis,  end_xaxis,  end_yaxis;
  int  start_x,  start_y,  end_x,  end_y;
  float  scale_x,  scale_y;


  can_info  =  wininfo.canvases[active_window];
  plot  =  can_info->active_plot;
  plots  =  can_info->total_plots;
  /*  printf("total  plots:  %d  plot  num:  %d\n",  plots+1,  plot+1);  */
  if  (plots  ==  -1)  return;

  display_active_plot(plot+1);

  data  =  can_info->plots[plot];

  xmin  =  data->xmin;
  ymin  =  data->ymin;
  xmax  =  data->xmax;
  ymax  =  data->ymax;

  start_x  =  can_info->start_x;
  end_x  =  can_info->end_x;
  start_y  =  can_info->start_y;
  end_y  =  can_info->end_y;
```



```
  scale_x  =  (float)(end_x  -  start_x)/(xmax  -  xmin);
  scale_y  =  (float)(start_y  -  end_y)/(ymax  -  ymin);

  data->scale_x  =  scale_x;
  data->scale_y  =  scale_y;

  /*  print  the  labels  */
  if  (data->label_type)
    label_type1();
  else
    label_type0();

  /*  plot  each  point  */
  for  (i=0;  i  <  data->nrows_x  -1;  i++)
    {
      x1  =  data->xarray[i]  -  xmin;
      x2  =  data->xarray[i+1]  -  xmin;
      y1  =  data->yarray[i]  -  ymin;
      y2  =  data->yarray[i+1]  -  ymin;

      XDrawLine(dpy,  win,  gctitle,
                start_x  +  (int)(x1*scale_x),
                start_y  -  (int)(y1*scale_y),
                start_x  +  (int)(x2*scale_x),
                start_y  -  (int)(y2*scale_y));
    }
}


label_type0()
{
  canvasinfo  *can_info;
  int  plot,  plots;
  plotarray  *data;
  int  tickx,  ticky;
  double  ten  =  10.000;
  char  string[128];
  int  stringlen;
  float  xmax,  xmin,  ymax,  ymin;
  int  start_xaxis,  start_yaxis,  end_xaxis,  end_yaxis;
  int  start_x,  start_y,  end_x,  end_y;
  Display  *dpy;
  Window  win;
  int  width,  height;


  can_info  =  wininfo.canvases[active_window];
  plot  =  can_info->active_plot;
  data  =  can_info->plots[plot];
  win  =  can_info->win;
  dpy  =  (Display  *)xv_get(main_frame,  XV_DISPLAY);

  width  =  (int)xv_get(can_info->canvas,  XV_WIDTH);
  height  =  (int)xv_get(can_info->canvas,  XV_HEIGHT);

  start_xaxis  =  can_info->start_xaxis;
  start_yaxis  =  can_info->start_yaxis;
  end_xaxis  =  can_info->end_xaxis;
  end_yaxis  =  can_info->end_yaxis;

  xmin  =  data->xmin;
  ymin  =  data->ymin;
  xmax  =  data->xmax;
  ymax  =  data->ymax;
```



```
  start_x  =  can_info->start_x;
  end_x  =  can_info->end_x;
  start_y  =  can_info->start_y;
  end_y  =  can_info->end_y;

  if  (can_info->total_plots  !=  0)
    {
      XFillRectangle(dpy,  win,  gcbg,  0,  0,  width,  end_yaxis-1);
      XFillRectangle(dpy,  win,  gcbg,
                     0,  end_y-1,
                     start_x-1,  start_y-end_y+2);
      XFillRectangle(dpy,  win,  gcbg,
                     end_x+1,  end_y-1,
                     width-end_x,  start_y-end_y+2);
      XFillRectangle(dpy,  win,  gcbg,  0,  start_y+1,  width,  height-start_y);
    }

  /*  x-axis  (bottom)  */
  XDrawLine(dpy,  win,  gctick,
            start_xaxis,  start_yaxis,
            end_xaxis,  start_yaxis);
  /*  left  y-axis  */
  XDrawLine(dpy,  win,  gctick,
            start_xaxis,  start_yaxis,
            start_xaxis,  end_yaxis);
  /*  right  y-axis  */
  XDrawLine(dpy,  win,  gctick,
            end_xaxis,  start_yaxis,
            end_xaxis,  end_yaxis);


  /*  sizes  for  the  tick  marks  */
  ticky=(int)width*0.02/2;
  tickx=(int)height*0.02/2;

  /*  tick  and  label  for  ymin  */
  XDrawLine(dpy,  win,  gctick,
            start_xaxis,  start_y,
            start_xaxis+tickx,  start_y);
  XDrawLine(dpy,  win,  gctick,
            end_xaxis,  start_y,
            end_xaxis-tickx,  start_y);
  sprintf(string,  "%.5g",  (strncmp(plot_cmd,  "plotlog",  7)==0)  ?
          pow(ten,  (double)ymin)  :  ymin);
  stringlen  =  strlen(string);
  XDrawString(dpy,  win,  gctick,
              start_xaxis  -  tickfontwidth*(strlen(string)+1),
              start_y  +  tickfontheight/2,
              string,  stringlen);

  /*  tick  and  label  for  ymax  */
  XDrawLine(dpy,  win,  gctick,
            start_xaxis,  end_y,
            start_xaxis+tickx,  end_y);
  XDrawLine(dpy,  win,  gctick,
            end_xaxis,  end_y,
            end_xaxis-tickx,  end_y);
  sprintf(string,  "%.5g",  (strncmp(plot_cmd,  "plotlog",  7)==0)  ?
          pow(ten,  (double)ymax)  :  ymax);
  stringlen  =  strlen(string);
  XDrawString(dpy,  win,  gctick,
              start_xaxis  -  tickfontwidth*(strlen(string)+1),
              end_y  +  tickfontheight/2,
```

```
            string, stringlen);

    /* tick and label for y mid */
    XDrawLine(dpy, win, gctick,
            start_xaxis, end_y + (start_y - end_y)/2,
            start_xaxis+ticky, end_y + (start_y - end_y)/2);
    XDrawLine(dpy, win, gctick,
            end_xaxis, end_y + (start_y - end_y)/2,
            end_xaxis-ticky, end_y + (start_y - end_y)/2);
    sprintf(string, "%.5g", (strncmp(plot_cmd, "plotlog", 7)==0) ?
           pow(ten, (double)(ymax-ymin)/2) : (ymax+ymin)/2);
    stringlen = strlen(string);
    XDrawString(dpy, win, gctick,
             start_xaxis - tickfontwidth*(strlen(string)+1),
             end_y + (start_y - end_y)/2 + tickfontheight/2,
             string, stringlen);

   /* min x axis */
   XDrawLine(dpy, win, gctick,
            start_x, start_yaxis,
            start_x, start_yaxis - ticky);
    sprintf(string, "%.5g", (strncmp(plot_cmd, "plotlog", 7)==0) ?
           pow(ten, (double)xmin) : xmin);
    stringlen = strlen(string);
    XDrawString(dpy, win, gctick,
             start_x - tickfontwidth*stringlen/2,
             start_yaxis + tickfontheight + 2,
             string, stringlen);

    /* max x axis */
   XDrawLine(dpy, win, gctick,
            end_x, start_yaxis,
            end_x, start_yaxis - ticky);
    sprintf(string, "%.5g", (strncmp(plot_cmd, "plotlog", 7)==0) ?
           pow(ten, (double)xmax) : xmax);
    stringlen = strlen(string);
    XDrawString(dpy, win, gctick,
             end_x - tickfontwidth*stringlen/2,
             start_yaxis + tickfontheight + 2,
             string, stringlen);

    /* mid x axis */
    XDrawLine(dpy, win, gctick,
            start_x + (end_x-start_x)/2, start_yaxis,
            start_x + (end_x-start_x)/2, start_yaxis - ticky);
    sprintf(string, "%.5g", (strncmp(plot_cmd, "plotlog", 7)==0) ?
           pow(ten, (double)(xmax-xmin)/2) : (xmax+xmin)/2);
    stringlen = strlen(string);
    XDrawString(dpy, win, gctick,
             start_x + (end_x-start_x)/2 - tickfontwidth*stringlen/2,
             start_yaxis + tickfontheight + 2,
             string, stringlen);

    sprintf(string, "%d. %s vs %s",
           plot+1,
           head.ch[data->col_x].name,
           head.ch[data->col_y].name);

    XDrawString(dpy, win, gctitle,
             start_xaxis, titlefontheight+2,
             string, strlen(string));

}
```



```
label_type1()
{
  canvasinfo *can_info;
  int plot, plots;
  plotarray *data;
  double ten = 10.000;
  char string[128];
  int stringlen;
  float xmax, xmin, ymax, ymin;
  float scale_x, scale_y;
  int start_xaxis, start_yaxis, end_xaxis, end_yaxis;
  int start_x, start_y, end_x, end_y;
  Display *dpy;
  Window win;
  int width, height;
  int a;
  float big_tickx, big_ticky;
  int tickx, ticky;
  int   i, j;
  int labelx, labely;
  double diffx, diffy;
  double decx, decy;
  float stop_xmin, stop_xmax, stop_ymin, stop_ymax;
  int where_x, where_y;


  can_info = wininfo.canvases[active_window];
  plot = can_info->active_plot;
  data = can_info->plots[plot];
  win = can_info->win;
  dpy = (Display *)xv_get(main_frame, XV_DISPLAY);

  width = (int)xv_get(can_info->canvas, XV_WIDTH);
  height = (int)xv_get(can_info->canvas, XV_HEIGHT);

  start_xaxis = can_info->start_xaxis;
  start_yaxis = can_info->start_yaxis;
  end_xaxis = can_info->end_xaxis;
  end_yaxis = can_info->end_yaxis;

  xmin = data->xmin;
  ymin = data->ymin;
  xmax = data->xmax;
  ymax = data->ymax;

  start_x = can_info->start_x;
  end_x = can_info->end_x;
  start_y = can_info->start_y;
  end_y = can_info->end_y;

  scale_x = data->scale_x;
  scale_y = data->scale_y;

  if (can_info->total_plots != 0)
    {
      /* clear title */
      XFillRectangle(dpy, win, gcbg, 0, 0, width, end_yaxis-1);
      /* clear left y axis */
      XFillRectangle(dpy, win, gcbg,
                     0, end_y-1,
                     start_x-1, start_y-end_y+2);
      /* clear right y axis */
      XFillRectangle(dpy, win, gcbg,
```



```
                     end_x+1, end_y-1,
                     width-end_x, start_y-end_y+2);
      /* clear x axis */
      XFillRectangle(dpy, win, gcbg, 0, start_y+1, width, height-start_y);
    }

  /* x-axis (bottom) */
  XDrawLine(dpy, win, gctick,
           start_xaxis, start_yaxis,
           end_xaxis, start_yaxis);
  /* left y-axis */
  XDrawLine(dpy, win, gctick,
           start_xaxis, start_yaxis,
           start_xaxis, end_yaxis);
  /* right y-axis */
  XDrawLine(dpy, win, gctick,
           end_xaxis, start_yaxis,
           end_xaxis, end_yaxis);


  /* sizes for the tick marks */
  ticky=(int)width*0.02;
  tickx=(int)height*0.02;

  diffx = xmax - xmin;
  diffy = ymax - ymin;

  stop_xmax = xmax + diffx/20;
  stop_ymax = ymax + diffy/20;

  stop_xmin = xmin - diffx/20;
  stop_ymin = ymin - diffy/20;

  decx=(double)floor(log10(diffx));
  decy=(double)floor(log10(diffy));

  big_tickx= ceil(xmin*pow(ten,-decx)) * pow(ten, decx);
  big_ticky= ceil(ymin*pow(ten,-decy)) * pow(ten, decy);

  /* y axis big tick marks */
  for (a=0;big_ticky<=stop_ymax;a++)
    {
      where_y = start_y-(int)((big_ticky-ymin)*scale_y);

      XDrawLine(dpy, win, gctick,
               start_xaxis, where_y,
               start_xaxis+ticky, where_y);

      XDrawLine(dpy, win, gctick,
               end_xaxis, where_y,
               end_xaxis-ticky, where_y);

      sprintf(string, "%.5g", (strncmp(plot_cmd, "plotlog", 7)==0) ?
             pow(ten, (double)big_ticky) : big_ticky);
      stringlen = strlen(string);

      XDrawString(dpy, win, gctick,
               start_xaxis - tickfontwidth*(strlen(string)+1),
               start_y-(int)((big_ticky-ymin)*scale_y)+tickfontheight/2,
               string, stringlen);

      big_ticky += pow(ten, decy);

    }
```



```
  /* y axis small tick marks */
  if (a<2)
    {
      big_ticky -= 1.5 * pow(ten, decy);
      if (big_ticky < stop_ymin) big_ticky += pow(ten, decy);

      while (big_ticky < stop_ymax)
        {
          where_y = start_y-(int)((big_ticky-ymin)*scale_y);

          XDrawLine(dpy, win, gctick,
                   start_xaxis, where_y,
                   start_xaxis+ticky, where_y);

          XDrawLine(dpy, win, gctick,
                   end_xaxis, where_y,
                   end_xaxis-ticky, where_y);

          sprintf(string, "%.5g", (strncmp(plot_cmd, "plotlog", 7)==0) ?
                 pow(ten, (double)big_ticky) : big_ticky);
          stringlen = strlen(string);

          XDrawString(dpy, win, gctick,
                   start_xaxis - tickfontwidth*(strlen(string)+1),
                   start_y-(int)((big_ticky-ymin)*scale_y)+tickfontheight/2,
                   string, stringlen);

          big_ticky += pow(ten, decy);
        }
    }



  /* x axis big tick marks */
  for (a=0;big_tickx<=stop_xmax;a++)
    {
      where_x= start_x + (int)(big_tickx-xmin)*scale_x;

      XDrawLine(dpy, win, gctick,
               where_x, start_yaxis,
               where_x, start_yaxis - tickx);

      sprintf(string, "%.5g", (strncmp(plot_cmd, "plotlog", 7)==0) ?
             pow(ten, (double)big_tickx) : big_tickx);
      strioglen = strlen(string);

      XDrawString(dpy, win, gctick,
                where_x - tickfontwidth*stringlen/2,
                start_yaxis + tickfontheight + 2,
                string, stringlen);

      big_tickx += pow(ten, decx);
    }

  /* x axis small tick marks */
  if (a<2)
    {
      big_tickx -= 1.5 * pow(ten, decx);

      if (big_tickx < stop_xmin) big_tickx += pow(ten, decx);

      while (big_tickx < stop_xmax)
        {
```

```
          where_x=start_x + (int)(big_tickx-xmin)*scale_x;

          XDrawLine(dpy, win, gctick,
                    where_x, start_yaxis,
                    where_x, start_yaxis - tickx);

          sprintf(string, "%.5g", (strncmp(plot_cmd, "plotlog", 7)==0) ?
                  pow(ten, (double)big_tickx) : big_tickx);
          stringlen = strlen(string);

          XDrawString(dpy, win, gctick,
                      where_x - tickfontwidth*stringlen/2,
                      start_yaxis + tickfontheight + 2,
                      string, stringlen);

          big_tickx += pow(ten, decx);
        }
    }

  sprintf(string, "%d. %s vs %s",
          plot+1,
          head.ch[data->col_x].name,
          head.ch[data->col_y].name);

  XDrawString(dpy, win, gctitle,
              start_xaxis, titlefontheight+2,
              string, strlen(string));

}



clear_canvas_proc(canvas)
     Canvas canvas;
{
  Display *dpy;
  Window win;

  dpy = (Display *)xv_get(canvas, XV_DISPLAY);
  win = (Window)xv_get(canvas_paint_window(canvas), XV_XID);
  XClearWindow(dpy, win);

}



void redraw_all_proc(canvas, xvwindow, dpy, win, area)
     Canvas canvas;
     Xv_Window xvwindow;
     Display *dpy;
     Window win;
     Xv_xrectlist *area;
{
  int can_num, i;
  canvasinfo *can_info;
  int active_plot;

  can_num = xv_get(canvas, XV_KEY_DATA, CAN_NUM);
  set_active_window(can_num);

  can_info = wininfo.canvases[active_window];
  active_plot = can_info->active_plot;
  display_active_plot(active_plot+1);
```



```
  if (can_info->total_plots == -1) return;

  XClearWindow(dpy, win);

  can_info->total_plots = -1;
  for (i=0; i<10; i++)
    {
      if (can_info->alive_plots[i] == 1)
        {
          can_info->active_plot = i;
          can_info->total_plots++;
          redraw_proc(canvas, xvwindow, dpy, win, NULL);
        }
    }
}
```



```
#include "global.h"
#include <string.h>
#include <sys/file.h>
#include <xview/notice.h>
#include <xview/frame.h>
#include <xview/canvas.h>
#include <X11/Xlib.h>
#include <xview/xv_xrect.h>
#include <xview/panel.h>

char tmp_cmd[128];
char pathname[10][80];
char default_path[80];
char metapath[80];
char data_file[20];
char new_file[20];
char headline[20];

FILE *data, *fopen(), *new;
int read();

char plot_cmd[16], trig_cmd[16], qi_cmd[16];

extern int action;
extern char msg[MSG_LENGTH];

int nargs;
char *args_left;

extern char mu_fit_mess_1[512];


command_handler(input)
     char input[128];
{
   char cmd[128];
  char arg1[128], arg2[128], arg3[128], arg4[128];
  int int1, int2, int3, int4, int5, int6, int7, int8, int9, int10;
  float farg1, farg2, farg3, farg4, farg5, farg6, farg7, farg8, farg9, farg10;


  if (action != MAIN)
    {
      process_action(input);
      return;
    }

  sscanf(input, "%s", cmd);

  nargs = token_count(input);


  if (strcmp(cmd, "kp") == 0 ||
      strcmp(cmd, "kill_plot") == 0)
    {
      if (sscanf(input, "%s %d", arg1, &int1) == 2)
        {
          del_plot_proc(int1);
          action = MAIN;
        }
      else
        {
          set_left_footer("Type the plot number to delete");
          set_cmd_prompt("Plot Number: ");
```



```
          action = KILL_PLOT;
        }
    }

  else if (strcmp(cmd, "kill_window") == 0 ||
           strcmp(cmd, "kw") == 0)
    {
      if (sscanf(input, "%s %d", arg1, &int1) == 2)
        {
          kill_win_proc(int1);
          action = MAIN;
        }
      else
        {
          set_left_footer("Type the window number to kill");
          set_cmd_prompt("Window Number: ");
          action = KILL_WIN;
        }
    }

  else if (strcmp(cmd, "new_window") == 0 ||
           strcmp(cmd, "nw") == 0)
    {
      if (sscanf(input, "%s", arg1) == 1)
        {
          new_win_proc();
          action = MAIN;
        }
    }

  else if (strcmp(cmd, "aw") == 0 ||
           strcmp(cmd, "activate_window") == 0)
    {
      if (sscanf(input, "%s %d", arg1, &int1) == 2)
        {
          set_active_window(int1-1);
          action = MAIN;
        }
      else
        {
          set_left_footer("Which window to be set active?");
          set_cmd_prompt("Window Number: ");
          action = SET_ACTIVE_WINDOW;
        }
    }

  else if (strcmp(cmd, "ap") == 0 ||
           strcmp(cmd, "activate_plot") == 0)
    {
      if (sscanf(input, "%s %d", arg1, &int1) == 2)
        {
          set_active_plot(int1-1);
          action = MAIN;
        }
      else
        {
          set_left_footer("Which plot to be set active?");
          set_cmd_prompt("Plot Number: ");
          action = SET_ACTIVE_PLOT;
        }
    }


  else if (strcmp(cmd, "q") == 0 ||
```

```
        strncmp(cmd, "quit", 4) == 0)
    {
      /* may need to free all the memory allocations */
      quit_xlook();
    }

  else if (strncmp(cmd, "set_path", 8) == 0)
    {
      if (sscanf(input, "%s %s", arg1, arg2) == 2)
        {
          set_path_proc(arg2);
          action = MAIN;
        }
      else
        {
          set_cmd_prompt("Pathname: ");
          set_left_footer("Type the pathname");
          action = SET_PATH;
        }
    }

  else if (strncmp(cmd, "doit", 4) == 0)
    {
      if (sscanf(input, "%s %s", arg1, arg2) == 2)
        {
          doit_proc(arg2);
          action = MAIN;
        }
      else
        {
          set_cmd_prompt("Filename: ");
          set_left_footer("Type the doit filename");
          action=DOIT;
        }
    }

  else if (strncmp(cmd, "all", 3) == 0)
    {
      if (nargs == 4)
        {
          all_final_proc(input);
          action = MAIN;
          set_left_footer("Type a command");
          set_cmd_prompt("Command: ");
        }

      else
        all_show_warning_proc(input);
    }

  else if (strncmp(cmd, "read", 4) == 0 || strncmp(cmd, "append", 6) == 0)
    {
      if (nargs == 2)
        {
          read_proc(input);
          action = MAIN;
        }
      else
        {
          if (strncmp(cmd, "read", 4) == 0)
```



```
            {
              set_left_footer("Type the data file to read");
              action = READ;
            }
          else
            {
              set_left_footer("Type the file to append");
              action = APPEND;
            }
          set_cmd_prompt("Filename: ");
          return;
        }
    }

  else if (strncmp(cmd, "write", 5) == 0)
    {
      if (sscanf(input, "%s %s", arg1, arg2) == 2)
        {
          write_proc(arg2);
          action = MAIN;
        }
      else
        {
          set_left_footer("Type the filename to write");
          set_cmd_prompt("Filename: ");
          action = WRITE;
        }
    }

  else if (strncmp(cmd, "plotover", 8) == 0 ||
           strncmp(cmd, "plotall", 7) == 0 ||
           strncmp(cmd, "plotsr", 6) == 0)
    {
      strcpy(plot_cmd, cmd);

      if (nargs == 3)
        {
          do_plot(input);
          action = MAIN;
        }
      else
        {
          if (nargs > 1)
            stripper(input, 1);

          set_left_footer("Type the x-axis and y-axis");
          set_cmd_prompt("X-axis, Y-axis: ");
          action = PLOT_GET_XY;
        }
    }

  else if (strncmp(cmd, "plotauto", 8) == 0 ||
           strncmp(cmd, "pa", 2) == 0 ||
           strncmp(cmd, "plotlog", 7) == 0 ||
           strncmp(cmd, "plotsame", 8) == 0)
    {
      strcpy(plot_cmd, cmd);

      if (nargs == 5)
        {
          do_plot(input);
```



```
          action = MAIN;
        }
      else
        {
          if (nargs > 1)
            stripper(input, 1);

          set_left_footer("Type the x-axis and y-axis");
          set_cmd_prompt("X-axis, Y-axis: ");
          action = PLOT_GET_BE;
        }
    }

  else if (strncmp(cmd, "plotscale", 9) == 0)
    {
      strcpy(plot_cmd, cmd);

      if (nargs == 9)
        {
          do_plot(input);
          action = MAIN;
        }
      else
        {
          if (nargs > 1)
            stripper(input, 1);

          set_left_footer("Type the x-axis and y-axis");
          set_cmd_prompt("X-axis, Y-axis: ");
          action = PLOT_GET_BE;
        }
    }

  else if (strncmp(cmd, "examin", 6) == 0)
    {
      if (sscanf(input, "%s %s", arg1, arg2) == 2)
        {
          do_examin(arg2);
          action = MAIN;
        }
      else
        {
          set_cmd_prompt("Filename: ");
          set_left_footer("Type the input filename");    action = EXAMIN_GET_FILENAME;
        }
    }

  else if (strncmp(cmd, "getaschead", 10) == 0)
    {
      if (sscanf(input, "%s %s", arg1, arg2) == 2)
        {
          do_getaschead(arg2);
          action = MAIN;
        }
      else
        {
          set_cmd_prompt("Filename: ");
          set_left_footer("Type the input filename");
          action = GETASCHEAD_GET_FILENAME;
        }
    }
```



```
  else if (strncmp(cmd, "head", 4) == 0)
    {
      if (sscanf(input, "%s %s", arg1, arg2) == 2)
        {
          do_head(arg2);
          action = MAIN;
        }
      else
        {
          set_cmd_prompt("Filename: ");
          set_left_footer("Type the output filename or S for screen output.");
          action = HEAD_GET_FILENAME;
        }
    }

  else if (strncmp(cmd, "tasc", 4) == 0)
    {
      if (nargs == 3)
        {
          do_tasc(input);
          action = MAIN;
        }
      else
        {
          if (nargs > 1)
            stripper(input, 1);

          set_cmd_prompt("Filename: ");
          set_left_footer("Type the input filename");
          action = TASC_GET_FILENAME;
        }
    }

  else if (strncmp(cmd, "stdasc", 6) == 0)
    {
      if (nargs == 4)
        {
          do_stdasc(input);
          action = MAIN;
        }
      else
        {
          if (nargs > 1)
            stripper(input, 1);

          set_cmd_prompt("Filename: ");
          set_left_footer("Type the input filename");
          action = STDASC_GET_FILENAME;
        }
    }

  else if (strncmp(cmd, "simplex", 7) == 0)
    {
      simplex_info();
      if (nargs == 6)
        {
          do_simplex(input);
        }
      else
```

```
        {
          if (nargs > 1)
           stripper(input, 1);

          set_cmd_prompt("Function: ");
          set_left_footer("Type the function to use");
          action = SIMPLEX_GET_FN;
        }
    }


  else if (strncmp(cmd, "offset_int", 10) == 0)
    {
     if (nargs == 5)
       {
        do_offset_int(input);
       }
     else
       {
         sprintf(msg, "Offset COL (from REC2 to the end) by the difference betw
nd REC1");
        print_msg(msg);
        if (nargs > 1)
         stripper(input, 1);

        set_cmd_prompt("Col, Rec1, Rec2: ");
        set_left_footer("Type the column (COL), record1 (REC1) and record2 (REC2)");
        action = OFFSET_INT;
       }
    }


  else if (strncmp(cmd, "offset", 6) == 0)
    {
     if (nargs == 5)
       {
        do_offset(input);
        action = MAIN;
       }
     else
       {
         sprintf(msg, "Offset COL1 by the difference between COL2, REC2 and COL1,
");
        print_msg(msg);
        if (nargs > 1)
         stripper(input, 1);

        set_cmd_prompt("Col1, Rec1, Col2, Rec2: ");
        set_left_footer("Type COL1, REC1, COL2 and REC2");
        action = OFFSET;
       }
    }


  else if (strncmp(cmd, "zero", 4) == 0)
    {
      if (nargs == 3)
       {
        do_zero(input);
        action = MAIN;
       }
      else
       {
         if (nargs > 1)
```



```
          stripper(input, 1);

        set_cmd_prompt("Col, Rec: ");
        set_left_footer("Type the column and record number");
        action = ZERO;
       }
    }


  else if (strncmp(cmd, "r_row_col", 9) == 0 ||
           strncmp(cmd, "rrc", 3) == 0)
    {
      if (nargs == 4)
       {
        do_r_row_col(input);
        action = MAIN;
       }
      else
       {
         if (nargs > 1)
          stripper(input, 1);

        set_cmd_prompt("Col: ");
        set_left_footer("Which column to remove?");
        action = R_ROW_COL;
       }
    }


  else if (strncmp(cmd, "r_row", 5) == 0 || strncmp(cmd, "rr", 2) == 0)
    {
      if (nargs == 3)
       {
        do_r_row(input);
        action = MAIN;
       }
      else
       {
         if (nargs > 1)
          stripper(input, 1);

        set_cmd_prompt("First, Last: ");
        set_left_footer("Type the first and last row to remove");
        action = R_ROW_F_L;
       }
    }


  else if (strncmp(cmd, "r_col", 5) == 0)
    {
     sprintf(msg, "# of allocated columns = %d\n", max_col);
     print_msg(msg);
     if (nargs == 2)
       {
        do_r_col(int1);
        action = MAIN;
       }
      else
       {
         if (nargs > 1)
          stripper(input, 1);

         set_cmd_prompt("Col: ");
         set_left_footer("Remove which column?");
```



```
        action = R_COL;
       }
    }


  else if (strncmp(cmd, "name", 4) == 0)
    {
     sprintf(msg, "# of allocated columns = %d\n", max_col);
     print_msg(msg);
     if (nargs == 4)
       {
        do_name(input);
       }
     else
       {
         if (nargs > 1)
          stripper(input, 1);

         set_left_footer("Type the column number to name, the name and the unit");
         set_cmd_prompt("Col, Name, Unit: ");
         action = NAME;
       }
    }


  else if (strncmp(cmd, "comment", 7) == 0)
    {
      if (nargs == 3)
       {
        do_comment(input);
        action = MAIN;
       }
      else
       {
         if (nargs > 1)
          stripper(input, 1);

         set_cmd_prompt("Col: ");
         set_left_footer("Comment which column?");
         action = COMMENT_COL;
       }
    }


  else if (strncmp(cmd, "pdfauto", 7) == 0)
    {
      if (nargs == 5)
       {
         do_pdfauto(input);
         action = MAIN;
       }
      else
       {
         if (nargs > 1)
          stripper(input, 1);

         set_left_footer("Type the Col, NewCol Prob, NewCol Bin, #Bins");
         set_cmd_prompt("Col, NCP, NCB, #Bins: ");
         action = PDFAUTO;
       }
    }

  else if (strncmp(cmd, "pdf", 3) == 0)
    {
```



```
      if (nargs == 7)
       {
         do_pdf(input);
         action = MAIN;
       }
      else
       {
         if (nargs > 1)
          stripper(input, 1);

         set_left_footer("Type the Col, NewCol Prob, NewCol Bin, #Bins, Max, Min"
         set_cmd_prompt("Col, NCP, NCB, #Bins, Max, Min: ");
         action = PDF;
       }
    }

  else if (strncmp(cmd, "decimat", 7) == 0)
    {
      if (nargs == 8)
       {
         do_decimat(input);
       }
      else
       {
         if (nargs > 1)
          stripper(input, 1);

         set_left_footer("Type the Col, New Col and Increment");
         set_cmd_prompt("Col, New_Col, Inc: ");

         if (strncmp(cmd, "decimat_r", 9) == 0)
          action = DECIMAT_R;
         else
          action = DECIMAT;
       }
    }

  else if (strncmp(cmd, "peak", 4) == 0)
    {
      if (nargs == 5)
       {
         do_peak(input);
       }
      else
       {
         if (nargs > 1)
          stripper(input, 1);

         set_left_footer("Type the Col and New Col");
         set_cmd_prompt("Col, New_Col: ");
         action = PEAK;
       }
    }

  else if (strncmp(cmd, "curv", 4) == 0)
    {
      if (nargs == 6)
       {
         do_curv(input);
       }
      else
       {
         if (nargs > 1)
          stripper(input, 1);
```

```
        set_left_footer("Type the column number, new column number and dx");
        set_cmd_prompt("Col, New_Col, dx: ");
        action = CURV;
      }
  }

else if (strncmp(cmd, "summation", 9) == 0)
  {
    if (nargs == 5)
      {
        do_summation(input);
      }
    else
      {
        if (nargs > 1)
          stripper(input, 1);

        set_left_footer("Type the Col and New Col");
        set_cmd_prompt("Col, New_Col: ");
        action = SUMMATION;
      }
  }

else if (strncmp(cmd, "findint", 7) == 0)
  {
    if (nargs == 4)
      {
        do_findint(input);
        action = MAIN;
      }
    else
      {
        if (nargs > 1)
          stripper(input, 1);

        set_left_footer("Type the Column, First and Last record number");
        set_cmd_prompt("Col, First, Last: ");
        action = FINDINT;
      }
  }

else if (strncmp(cmd, "trend_a", 7) == 0)
  {
    if (nargs == 3)
      {
        do_trend_a(input);
        action = MAIN;
      }
    else
      {
        if (nargs > 1)
          stripper(input, 1);

        set_left_footer("Type X-column and Y-column");
        set_cmd_prompt("X-col, Y-col: ");
        action = TREND_A;
      }
  }

else if (strncmp(cmd, "trend", 5) == 0)
  {
    if (nargs == 5)
      {
```



```
        do_trend(input);
        action = MAIN;
      }
    else
      {
        if (nargs > 1)
          stripper(input, 1);

        set_left_footer("Type X-column and Y-column");
        set_cmd_prompt("X-col, Y-col: ");
        action = TREND_XY;
      }
  }


else if (strncmp(cmd, "r_trend", 7) == 0)
  {
    if (nargs == 10)
      {
        do_r_trend_input(input);
      }
    else if (nargs == 8)
      {
        do_r_trend_comp(input);
      }
    else
      {
        if (nargs > 1)
          stripper(input, 1);

        set_left_footer("Detrend X or Y values?");
        set_cmd_prompt("X/Y: ");
        action = R_TREND;
      }
  }

else if (strncmp(cmd, "z_max", 5) == 0)
  {
    if (nargs == 5)
      {
        do_z_max(input);
      }
    else
      {
        if (nargs > 1)
          stripper(input, 1);

        set_left_footer("Type Col and New Col");
        set_cmd_prompt("Col, New_Col: ");
        action = Z_MAX;
      }
  }

else if (strncmp(cmd, "z_min", 5) == 0)
  {
    if (nargs == 5)
      {
        do_z_min(input);
      }
    else
      {
        if (nargs > 1)
          stripper(input, 1);
```



```
        set_left_footer("Type COL and NEW_COL");
        set_cmd_prompt("Col, New_Col: ");
        action = Z_MIN;
      }
  }


else if (strncmp(cmd, "r_mean", 6) == 0)
  {
    if (nargs == 5)
      {
        do_r_mean(input);
      }
    else
      {
        if (nargs > 1)
          stripper(input, 1);

        set_left_footer("Type COL and NEW_COL");
        set_cmd_prompt("Col, New_Col: ");
        action = R_MEAN;
      }
  }

else if (strncmp(cmd, "compress", 8) == 0)
  {
    if (nargs == 5)
      {
        do_compress(input);
      }
    else
      {
        if (nargs > 1)
          stripper(input, 1);

        set_left_footer("Type COL and NEW_COL");
        set_cmd_prompt("Col, New_Col: ");
        action = COMPRESS;
      }
  }

else if (strncmp(cmd, "positive", 8) == 0)
  {
    if (nargs == 5)
      {
        do_positive(input);
      }
    else
      {
        if (nargs > 1)
          stripper(input, 1);

        set_left_footer("Type COL and NEW_COL");
        set_cmd_prompt("Col, New_Col: ");
        action = POSITIVE;
      }
  }

else if (strncmp(cmd, "r_spike", 7) == 0)
  {
    if (nargs == 3)
      {
        do_r_spike(input);
      }
```



```
    else
      {
        if (nargs > 1)
          stripper(input, 1);

        set_left_footer("Type the Col and Row number to remove the spike");
        set_cmd_prompt("Col, Row#: ");
        action = R_SPIKE;
      }
  }

else if (strncmp(cmd, "mathint", 7) == 0 ||
         strncmp(cmd, "math_int", 8) == 0)
  {
    if (nargs == 10)
      {
        do_mathint(input);
      }
    else
      {
        if (nargs > 1)
          stripper(input, 1);

        set_left_footer("Type the math command: X-col $ Y-col # New_col");
        set_cmd_prompt("X-col, $, Y-col, #, New_Col: ");
        action = MATHINT;
      }
  }

else if (strncmp(cmd, "math", 4) == 0)
  {
    if (nargs == 8)
      {
        do_math(input);
      }
    else
      {
        if (nargs > 1)
          stripper(input, 1);

        set_left_footer("Type the math command: X-col $ Y-col # New_col");
        set_cmd_prompt("X-col, $, Y-col, #, New_Col: ");
        action = MATH;
      }
  }

else if (strncmp(cmd, "interpolate", 11) == 0)
  {
    if (nargs == 12)
      {
        do_interpolate(input);
      }
    else
      {
        if (cmd[strlen((char *)cmd)-1] == 'h')
          {
            sprintf(msg, "Mid point interpolation for y.\nRows from the end of it
ted data can be removed using interpolate_r.\n");
            print_msg(msg);
          }

        if (nargs > 1)
          stripper(input, 1);
```

```
        set_left_footer("Type the x-col, y-col, new x-col and new y-col");
        set_cmd_prompt("X-col, Y-col, New_X-col, New_Y-col: ");

        if (cmd[strlen((char *)cmd)-1] == 'r')
          action = INTERPOLATE_R;
        else
          action = INTERPOLATE;
      }
    }

  else if (strncmp(cmd, "sort", 4) == 0 || strncmp(cmd, "order", 5) == 0)
    {
      if (nargs == 4)
        {
          do_sort(input);
        }
      else
        {
          if (nargs > 1)
            stripper(input, 1);

          set_left_footer("Type Col, Start_row and End_row");
          set_cmd_prompt("Col, Start, End: ");
          action = SORT;
        }
    }

  else if (strncmp(cmd, "polyfit", 7) == 0)
    {
      if (nargs == 11)
        {
          do_polyfit(input);
        }

      else
        {
          sprintf(msg, "Polynomial fit of order n.\n");
          print_msg(msg);

          if (nargs > 1)
            stripper(input, 1);

          set_left_footer("Type the X-col, Y-col, New_col, Order");
          set_cmd_prompt("X-col, Y-col, New_col, Order: ");

          if (strncmp(cmd, "polyfit_i", 9) == 0)
            action = POLYFIT_I;
          else
            action = POLYFIT;
        }
    }

  else if (strncmp(cmd, "cs", 2) == 0 || strncmp(cmd, "strain"), 6 == 0)
    {
      if (nargs == 8)
        {
          do_cs(input);
        }

      else
        {
          sprintf(msg, "Incremental shear strain calculation.\n");
          print_msg(msg);
```



```
          if (nargs > 1)
            stripper(input, 1);

          set_left_footer("Type the axial_disp col, layer_thick col and output col");
          set_cmd_prompt("Axial_disp, Layer_thick, Output Cols: ");
          action = CS;
        }
      }

    else if (strncmp(cmd, "ec", 2) == 0)
      {
        if (nargs == 9)
          {
            do_ec(input);
          }
        else
          {
            sprintf(msg, "Correct axial displacement for elastic distortion within pis
d sample column.\n");
            print_msg(msg);

            if (nargs > 1)
              stripper(input, 1);

            set_left_footer("Type the axial_disp col, shear_stress col and new col");
            set_cmd_prompt("Axial_disp, Shear_stress, New Cols: ");
            action = EC;
          }
      }

    else if (strncmp(cmd, "cgt", 3) == 0 || strncmp(cmd, "calc_geometric_thinning",
)
      {
        if (nargs == 7)
          {
            do_cgt(input);
          }
        else
          {
            if (cmd[strlen((char *)cmd)-1] == 'h')
              {
                sprintf(msg,"Calculate geometric thinning\n calculation is: (1) del_h =
dx/2L ; where h is input thickness, dx is slip increment and L is length of
ock parallel to slip \n");
                print_msg(msg);
                sprintf(msg, "(1) Assumes an h appropriate for "one" layer (not both
hus the calculation is for one layer\n");
                print_msg(msg);
                sprintf(msg, "Also, make sure your equation is dimensionally correct!
ee also rgt\n");
                print_msg(msg);
              }

            if (nargs > 1)
              stripper(input, 1);

            set_left_footer("Type the Length, initial_h, Disp_col and New_col");
            set_cmd_prompt("L, h, Disp_col, New_col: ");
            action = CGT;
          }
      }

    else if ((strncmp(cmd, "rgt", 3) == 0) || (strncmp(cmd, "geometric_thinning", 1
      {
```



```
        if (nargs == 7)
          {
            do_rgt(input);
          }
        else
          {
            if(cmd[strlen((char *)cmd)-1] == 'h')
              {
                sprintf(msg, " Correct horizontal displacement measurement during direct
ar test for geometric thinning\n correction is: (1) del_h = h dx/2L ; where h
ss, dx is slip increment and L is length of the sliding block parallel to sli
                print_msg(msg);
                sprintf(msg, "(1) Assumes an h appropriate for "one" layer (not both
hus the H. disp. measurements being corrected should be for one layer\nAlso, mak
r equation is dimensionally correct! --see also cgt\n");
                print_msg(msg);
              }

            if (nargs > 1)
              stripper(input, 1);

            set_left_footer("Type the Length, Disp_col, Gouge_thickness col and New_col'
            set_cmd_prompt("L, Disp_col, Thick_col, New_col: ");
            action = RGT;
          }
      }

    else if ((strncmp(cmd, "vc", 2) == 0) || (strncmp(cmd, "vol_cor", 7) == 0))
      {
        if (nargs == 9)
          {
            do_vc(input);
          }
        else
          {
            if(cmd[strlen((char *)cmd)-1] == 'h')
              {
                sprintf(msg, "Corrrect porosity/volume strain (during loading/unloading)
the effect of confining pressure on the material at the edges of the layer.\n'
                print_msg(msg);
                sprintf(msg, "User provides a compressibility (dv/V/dP) * the ratio o
ffected volume (eg. at the edges) to the total volume.\n");
                print_msg(msg);
                sprintf(msg, "A first guess for the volume of material affected by l
load/unload is an annular -elliptical area- of width = layer thickness * thickn
                print_msg(msg);
              }

            if (nargs > 1)
              stripper(input, 1);

            set_left_footer("Type the Vol strain, Shear stress col and New col");
            set_cmd_prompt("Vol_strain, Shear_stress_col, New_col: ");
            action = VC;
          }
      }

    else if ((strncmp(cmd, "deriv", 5) == 0))
      {
        if (nargs == 8)
          {
            do_deriv(input);
          }
        else
```



```
          {
            if (nargs > 1)
              stripper(input, 1);

            set_left_footer("Type the X col, Y col and New col for dy/dx derivativ
            set_cmd_prompt("X-col, Y-col, New-col: ");
            action = DERIV;
          }
      }

    else if ((strncmp(cmd, "exp", 3) == 0))
      {
        if (nargs == 5)
          {
            do_exp(input);
          }
        else
          {
            if (nargs > 1)
              stripper(input, 1);

            set_left_footer("Type the X col and New col");
            set_cmd_prompt("X-col, New-col: ");
            action = EXP;
          }
      }

    else if ((strncmp(cmd, "ln", 2) == 0))
      {
        if (nargs == 5)
          {
            do_ln(input);
          }
        else
          {
            if (nargs > 1)
              stripper(input, 1);

            set_left_footer("Type the X col and New col");
            set_cmd_prompt("X-col, New-col: ");
            action = LN;
          }
      }

    else if ((strncmp(cmd, "Power1", 6) == 0))
      {
        if (nargs == 7)
          {
            do_Power1(input);
          }
        else
          {
            if (nargs > 1)
              stripper(input, 1);
            set_left_footer("Type the X col, New col, A and B");
            set_cmd_prompt("X-col, New-col, A, B: ");
            action = POWER1;
          }
      }

    else if ((strncmp(cmd, "Power2", 6) == 0))
      {
        if (nargs == 8)
          {
```

```
        do_Power2(input);
       }
      else
       {
        if (nargs > 1)
         stripper(input, 1);

        set_left_footer("Type the X col, New col, A, B, C");
        set_cmd_prompt("X-col, New-col, A, B, C: ");
        action = POWER2;
       }
    }

  else if ((strncmp(cmd, "normal", 6) == 0))
    {
     if (nargs == 7)
       {
        do_normal(input);
       }
     else
       {
        if (nargs > 1)
         stripper(input, 1);

        set_left_footer("Type the X col, New col, mean an stddev");
        set_cmd_prompt("X-col, New-col, Mean, Stddev: ");
        action = NORMAL;
       }
    }

  else if ((strncmp(cmd, "chisqr", 6) == 0))
    {
     if (nargs == 6)
       {
        do_chisqr(input);
       }
     else
       {
        if (nargs > 1)
         stripper(input, 1);

        set_left_footer("Type the X col, New col and ndf");
        set_cmd_prompt("X-col, New-col, ndf: ");
        action = CHISQR;
       }
    }

  else if ((strncmp(cmd, "scchisqr", 8) == 0))
    {
     if (nargs == 8)
       {
        do_chisqr(input);
       }
     else
       {
        if (nargs > 1)
         stripper(input, 1);

        set_left_footer("Type the X col, New col, sigma, ndf and offset");
        set_cmd_prompt("X-col, New-col, sigma, ndf, offset: ");
        action = SCCHISQR;
       }
    }
```



```
  else if ((strncmp(cmd, "rclow", 5) == 0))
    {
     if (nargs == 7)
       {
        do_rclow(input);
       }
     else
       {
        if (nargs > 1)
         stripper(input, 1);

        set_left_footer("Type the X col, New col, sigma, A and B");
        set_cmd_prompt("X-col, New-col, A, B: ");
        action = RCLOW;
       }
    }

  else if ((strncmp(cmd, "genexp", 6) == 0))
    {
     if (nargs == 9)
       {
        do_genexp(input);
       }
     else
       {
        if (nargs > 1)
         stripper(input, 1);

        set_left_footer("Type the X col, New col, A, B, C and D");
        set_cmd_prompt("X-col, New-col, A, B, C, D: ");
        action = GENEXP;
       }
    }

  else if ((strncmp(cmd, "gensin", 6) == 0))
    {
     if (nargs == 9)
       {
        do_gensin(input);
       }
     else
       {
        if (nargs > 1)
         stripper(input, 1);

        set_left_footer("Type the X col, New col, A, B, C and D");
        set_cmd_prompt("X-col, New-col, A, B, C, D: ");
        action = GENSIN;
       }
    }

  else if ((strncmp(cmd, "Poly4", 5) == 0))
    {
     if (nargs == 10)
       {
        do_Poly4(input);
       }
     else
       {
        if (nargs > 1)
         stripper(input, 1);
        set_left_footer("Type the X col, New col, A, B, C, D and E");
        set_cmd_prompt("X-col, New-col, A, B, C, D, E: ");
        action = POLY4;
```



```
       }
    }

  else if ((strncmp(cmd, "ExpLin", 6) == 0))
    {
     if (nargs == 8)
       {
        do_ExpLin(input);
       }
     else
       {
        if (nargs > 1)
         stripper(input, 1);

        set_left_footer("Type the X col, New col, A, B and C");
        set_cmd_prompt("X-col, New-col, A, B, C: ");
        action = EXPLIN;
       }
    }

  else if ((strncmp(cmd, "log", 3) == 0))
    {
     if (nargs == 5)
       {
        do_log(input);
       }
     else
       {
        if (nargs > 1)
         stripper(input, 1);

        set_left_footer("Type the X col and New col");
        set_cmd_prompt("X-col, New-col: ");
        action = LOG;
       }
    }

  else if ((strncmp(cmd, "recip", 5) == 0))
    {
     if (nargs == 5)
       {
        do_recip(input);
       }
     else
       {
        if (nargs > 1)
         stripper(input, 1);

        set_left_footer("Type the X col and New col");
        set_cmd_prompt("X-col, New-col: ");
        action = RECIP;
       }
    }

  else if ((strncmp(cmd, "power", 5) == 0))
    {
     if (nargs == 6)
       {
        do_power(input);
       }
     else
       {
        if (nargs > 1)
         stripper(input, 1);
```



```
        set_left_footer("Type the Power, X col and New col");
        set_cmd_prompt("Power, X-col, New-col: ");
        action = POWER;
       }
    }

  else if ((strncmp(cmd, "col_power", 9) == 0))
    {
     if (nargs == 6)
       {
        do_col_power(input);
       }
     else
       {
        if (nargs > 1)
         stripper(input, 1);

        set_left_footer("Type the Power col, X col and New col");
        set_cmd_prompt("Power-col, X-col, New-col: ");
        action = COL_POWER;
       }
    }


  else if ((strncmp(cmd, "rcph", 4) == 0))
    {
     if (nargs == 7)
       {
        do_rcph(input);
       }
     else
       {
        if (nargs > 1)
         stripper(input, 1);

        set_left_footer("Type the X col, New col, A and B");
        set_cmd_prompt("X-col, New-col, A, B: ");
        action = RCPH;
       }
    }


  else if (strncmp(cmd,"trig", 4) == 0 ||
           strncmp(cmd,"sin", 3) == 0 ||
           strncmp(cmd,"cos", 3) == 0 ||
           strncmp(cmd,"tan", 3) == 0 ||
           strncmp(cmd,"asin", 4) == 0 ||
           strncmp(cmd,"acos", 4) == 0 ||
           strncmp(cmd,"atan", 4) == 0 )
    {
     strcpy(trig_cmd, cmd);

     if (nargs == 6)
       {
        do_col_power(input);
       }
     else
       {
        if (nargs > 1)
           stripper(input, 1);

        set_left_footer("Type the Column, New col and Function");
        set_cmd_prompt("Col, New-col, Func: ");
```

```
        action = TRIG;
      }
    }

/*
  remove this from the command list. o_slope is renamed to slope.

  else if ((strncmp(cmd, "slope", 5) == 0))
    {
      if (nargs == 5)
        {
          do_slope(input);
        }
      else
        {
          if (nargs > 1)
            stripper(input, 1);

          set_left_footer("Type the X col, Y Col, New Col and Interval");
          set_cmd_prompt("X-col, Y-Col, New-col, Interval: ");
          action = SLOPE;
        }
    }
*/

  else if ((strncmp(cmd, "slope", 5) == 0))
    {
      /* used to be called o_slope */
      if (nargs == 9)
        {
          do_slope(input);
        }
      else
        {
          if (nargs > 1)
            stripper(input, 1);

          set_left_footer("Type the X col, Y Col, New Col, Start Row, End Row and
ize");
          set_cmd_prompt("X-col, Y-Col, New-col, Start, End, Win-size: ");
          action = SLOPE;
        }
    }

  else if ((strncmp(cmd, "rsm", 3) == 0))
    {
      if (cmd[strlen((char *)cmd)-1] == 'h')
        print_rsm_help_info();

      sprintf(msg, "One or two state variable friction model calculated at displac
pecified by disp col.\n");
      print_msg(msg);
      sprintf(msg, "rsm_h gives a datailed description of this function. See also
);
      print_msg(msg);
      sprintf(msg, mu_fit_mess_1);
      print_msg(msg);

      if (nargs == 16)
        {
          do_rsm(input);
        }
      else
        {
```



```
          if (nargs > 1)
            stripper(input, 1);

          sprintf(msg, "Enter: disp. col., mu col, row # of vel. step and end of
odel, col for model_mu, stiffness (k), Sigma_n, v_initial (v_o), v_final, mu_o, mu
b1, Dc1, and Dc2\n");
          print_msg(msg);
          set_left_footer("");
          set_cmd_prompt("Cmds: ");
          action = RSM;
        }
      }

    else if ((strncmp(cmd, "cm", 2) == 0))
      {
        if (cmd[strlen((char *)cmd)-1] == 'h')
          print_cm_help_info();

        if (nargs == 17)
          {
            do_cm(input);
          }
        else
          {
            if (nargs > 1)
              stripper(input, 1);

            sprintf(msg, "Forward modelling of rate/state variable friction. (cm_h give
etailed description of this function) \n");
            print_msg(msg);
            sprintf(msg, "Enter: model_disp_col, (data) disp. col., (data) mu col, vel
_row, end_row, model_mu_col, \n\tstiffness (k), Sigma_n,v_initial (v_o), v_final, mu_
_f, a, b1, Dc1, and Dc2\n");
            print_msg(msg);
            sprintf(msg, "\tmodel_disp_col, disp_col, mu_col, vs_row, end_row, model_mu_c
k, Sigma_n, v_o, vf, mu_o, mu_f, a, b1, Dc1, Dc2 \n ") ;
            print_msg(msg);

            set_left_footer("");
            set_cmd_prompt("Inputs: ");
            action = CM;
          }
      }


    else if ((strncmp(cmd, "mem", 3) == 0))
      {
        if (cmd[strlen((char *)cmd)-1] == 'h')
          {
            sprintf(msg, "Command line interpretation: \n ycol = signal for which yo
ower spectra.\n xcol = spacing of signal (i.e., time, dist)\n 2 cols are output
d power.\n the spectra is calculated between records 'first_row' and 'last row'\n
is the # of frequencies (beginning at the nyquist) at which to calculate power.
les is the # of poles to use. The frequencies can either be distributed linear
\n logarithmically (l=> linear on a log plot), a welsh taper may be applied to
)\n\n");
            print_msg(msg);
          }

        if (nargs == 11)
          {
            do_mem(input);
          }
        else
```



```
          {
            if (nargs > 1)
              stripper(input, 1);

            sprintf(msg, "Power spectra estimate using maximum entropy method, n=linea
. series, l=log, w=welsh taper, n=no taper (mem_h for help)\n");
            print_msg(msg);
            sprintf(msg, "Inputs for mem: xcol, ycol, freq_col, power_col, first_row,
ow, $freqs, $poles, l/n, w/n\n") ;
            print_msg(msg);
            set_left_footer("");
            set_cmd_prompt("Inputs: ");
            action = MEM;
          }
      }


    else if ((strncmp(cmd, "median_smooth", 13) == 0) ||
             (strncmp(cmd, "ms", 2) == 0))
      {
        if (cmd[strlen((char *)cmd)-1] == 'h')
          {
            sprintf(msg, "Calculate the running average "median" value of COL within
g window between start_row and end_row.\n\t"the length of 'window' is forced to
);
            print_msg(msg);
          }

        if (nargs == 8)
          {
            do_median_smooth(input);
          }
        else
          {
            if (nargs > 1)
              stripper(input, 1);

            sprintf(msg, "Inputs for median smooth: Col, New_Col , Start_Row, End_Row,
w_Size\n") ;
            print_msg(msg);
            set_left_footer("");
            set_cmd_prompt("Inputs: ");
            action = MEDIAN_SMOOTH;
          }
      }


    else if ((strncmp(cmd, "smooth", 6) == 0))
      {
        if (nargs == 8)
          {
            do_smooth(input);
          }
        else
          {
            if (nargs > 1)
              stripper(input, 1);

            sprintf(msg, "Inputs for smooth: Col, New_Col , Start_Row, End_Row, Windo
n") ;
            print_msg(msg);
            set_left_footer("");
            set_cmd_prompt("Inputs: ");
            action = SMOOTH;
```



```
          }
      }

    else if ((strncmp(cmd, "typeall", 7) == 0) ||
             (strncmp(cmd, "type_all", 8) == 0))
      {
        if (nargs == 2)
          {
            do_typeall(input);
          }
        else
          {
            if (nargs > 1)
              stripper(input, 1);

            set_left_footer("Input the filename or type 'S' for screen output");
            set_cmd_prompt("Input: ");
            action = TYPEALL;
          }
      }

    else if ((strncmp(cmd, "type", 4) == 0))
      {
        if (nargs == 6)
          {
            do_type(input);
          }
        else
          {
            if (nargs > 1)
              stripper(input, 1);

            set_left_footer("Input the starting and ending record numbers");
            set_cmd_prompt("Start End: ");
            action = TYPE;
          }
      }

    else if ((strncmp(cmd, "stat", 4) == 0))
      {
        if (nargs == 4)
          {
            do_stats(input);
          }
        else
          {
            if (nargs > 1)
              stripper(input, 1);

            set_left_footer("Type the starting and ending record numbers");
            set_cmd_prompt("Col Start End: ");
            action = STAT;
          }
      }

    else if ((strncmp(cmd, "qi", 2) == 0))
      {
        if (nargs >= 21)
          {
            stripper(input, 21);
            do_qi(input);
          }

        else
```

```
        {
          strcpy(qi_cmd, cmd);

          if(cmd[(strlen((char *)cmd)-1)] == 'h')
            do_qi_help();

          sprintf(msg, "Linearized inversion of one or two state variable friction
n\t(qi_h gives a detailed description of this function) \n");
          print_msg(msg);
          sprintf(msg, "\nNeed disp. col, mu col, model_mu col, beginning row for
  # of vel. step, end row, weight_row, lin_term(c), converg_tol, lamda, wc, stiffn
nitial, v_final, mu_init, a, b1, Dc1, b2 and Dc2 for input.\n ");
          print_msg(msg);
          sprintf(msg, "Input: d_col, mu_col, mod_col, first_row, vs_row, end_row, we
row, c, tol, lamda, wc, k, v_o, vf, mu_o, a, b1, Dc1, b2, Dc2 \n") ;
          print_msg(msg);
          set_left_footer("");
          set_cmd_prompt("QI Input: ");
          action = QI;
        }
    }


  else if (strncmp(cmd, "scm", 3) == 0)
    {
      if (nargs > 16)
        {
          stripper(input, 16);
          do_scm(input);
        }
      else
        {
          if ((strlen(cmd) >= 5) && (strncmp(cmd, "scm_h", 5) == 0))
            do_scm_help();
          do_scm_info();
          set_cmd_prompt("Input: ");
          set_left_footer("Type all the arguments for scm");
          action = SCM_GET_ARGS;
        }
    }


}



process_action(arg)
    char arg[128];
{
  int nargs;

  nargs = token_count(arg);

  switch(action)
    {

    case SIMP_FUNC_GET_MAX_ITER:
      if (nargs > 1) stripper(arg, 1);
      sprintf(tmp_cmd, "%s ", arg);
      sprintf(msg, "Type the initial guesses for the %d free parameters", n_param);
      set_left_footer(msg);
      set_cmd_prompt("Initial guesses: ");
      action = SIMP_FUNC_GET_INIT_GUESSES;
      break;
```



```
    case SIMP_FUNC_GET_INIT_GUESSES:
      if (nargs > n_param) stripper(arg, n_param);
      strcat(tmp_cmd, arg);
      strcat(tmp_cmd, " ");
      sprintf(msg, "Type the initial step size for the %d free parameters",
      set_left_footer(msg);
      set_cmd_prompt("Initial step size: ");
      action = SIMP_FUNC_GET_INIT_STEP_SIZE;
      break;

    case SIMP_FUNC_GET_INIT_STEP_SIZE:
      if (nargs > n_param) stripper(arg, n_param);
      strcat(tmp_cmd, arg);
      do_get_initial_values(tmp_cmd);
      break;


    case SIMP_FUNC:
      do_simp_func(arg);
      break;

    case SCM_GET_ARGS:
      sprintf(tmp_cmd, "scm %s", arg);
      do_scm(tmp_cmd);
      break;

    case SCM_SIMP_WEIGHT_L2:
      do_simp_weight_l2(arg);
      break;

    case SCM_SIMP_WEIGHT:
      do_simp_weight(arg);
      break;

    case SCM_1:
      do_scm_1(arg);
      break;

    case QI_MVS:
      do_qi_mvs(arg);
      break;

    case QI_WC:
      sscanf(arg, "%d", &rs_param.end_weight_row);
      do_qi_final();
      break;

    case QI:
      sprintf(tmp_cmd, "%s %s", qi_cmd, arg);
      do_qi(tmp_cmd);
      break;

    case STAT:
      sprintf(tmp_cmd, "stat %s", arg);
      do_stats(tmp_cmd);
      break;

    case TYPE:
      if (nargs == 5)
        {
          sprintf(tmp_cmd, "type %s", arg);
          do_type(tmp_cmd);
        }
```



```
      else
        {
          if (nargs > 2) stripper(arg, 2);
          sprintf(tmp_cmd, "type %s ", arg);
          set_left_footer("Input the first and last column");
          set_cmd_prompt("First Last: ");
          action = TYPE_FL;
        }
      break;

    case TYPE_FL:
      if (nargs == 3)
        {
          strcat(tmp_cmd, arg);
          do_type(tmp_cmd);
        }
      else
        {
          strcat(tmp_cmd, arg);
          strcat(tmp_cmd, " ");
          set_left_footer("Input the filename or type 'S' for screen output");
          set_cmd_prompt("Input: ");
          action = TYPE_S;
        }
      break;

    case TYPE_S:
      strcat(tmp_cmd, arg);
      do_type(tmp_cmd);
      break;

    case TYPEALL:
      sprintf(tmp_cmd, "typeall %s", arg);
      do_typeall(tmp_cmd);
      break;

    case SMOOTH:
      if (nargs == 7)
        {
          sprintf(tmp_cmd, "smooth %s", arg);
          do_smooth(tmp_cmd);
        }
      else
        {
          if (nargs > 5) stripper(arg, 5);
          sprintf(tmp_cmd, "smooth %s ", arg);
          name_col(getcol(tmp_cmd, 3));
          action = SMOOTH_NAME;
        }
      break;

    case SMOOTH_NAME:
      strcat(tmp_cmd, arg);
      do_smooth(tmp_cmd);
      break;

    case MEDIAN_SMOOTH:
      if (nargs == 7)
        {
          sprintf(tmp_cmd, "median_smooth %s", arg);
          do_median_smooth(tmp_cmd);
        }
      else
        {
```



```
          if (nargs > 5) stripper(arg, 5);
          sprintf(tmp_cmd, "median_smooth %s ", arg);
          name_col(getcol(tmp_cmd, 3));
          action = MEDIAN_SMOOTH_NAME;
        }
      break;

    case MEDIAN_SMOOTH_NAME:
      strcat(tmp_cmd, arg);
      do_median_smooth(tmp_cmd);
      break;

    case MEM:
      sprintf(tmp_cmd, "mem %s", arg);
      do_mem(tmp_cmd);
      break;

    case CM:
      sprintf(tmp_cmd, "cm %s", arg);
      do_cm(tmp_cmd);
      break;

    case RSM:
      sprintf(tmp_cmd, "rsm %s", arg);
      do_rsm(tmp_cmd);
      break;

    case SLOPE:
      if (nargs == 8)
        {
          sprintf(tmp_cmd, "slope %s ", arg);
          do_slope(tmp_cmd);
        }
      else
        {
          if (nargs > 6) stripper(arg, 6);
          sprintf(tmp_cmd, "slope %s ", arg);
          name_col(getcol(tmp_cmd, 4));
          action = SLOPE_NAME;
        }
      break;

    case SLOPE_NAME:
      strcat(tmp_cmd, arg);
      do_slope(tmp_cmd);
      break;

    case TRIG:
      if (nargs == 5)
        {
          sprintf(tmp_cmd, "%s %s ", trig_cmd, arg);
          do_trig(tmp_cmd);
        }
      else
        {
          if (nargs > 3) stripper(arg, 3);
          sprintf(tmp_cmd, "%s %s ", trig_cmd, arg);
          name_col(getcol(tmp_cmd, 3));
          action = TRIG_NAME;
        }
      break;

    case TRIG_NAME:
      strcat(tmp_cmd, arg);
```

```
    do_trig(tmp_cmd);
    break;

  case  RCPH:
    if  (nargs  ==  6)
      {
        sprintf(tmp_cmd,  "rcph  %s  ",  arg);
        do_rcph(tmp_cmd);
      }
    else
      {
        if  (nargs  >  4)
          stripper(arg,  4);
        sprintf(tmp_cmd,  "rcph  %s  ",  arg);
        name_col(getcol(tmp_cmd,  3));
        action  =  RCPH_NAME;
      }
    break;

  case  RCPH_NAME:
    strcat(tmp_cmd,  arg);
    do_rcph(tmp_cmd);
    break;

  case  COL_POWER:
    sprintf(tmp_cmd,  "col_power  %s",  arg);
    do_col_power(tmp_cmd);
    break;

  case  POWER:
    if  (nargs  ==  6)
      {
        sprintf(tmp_cmd,  "rcph  %s  ",  arg);
        do_power(tmp_cmd);
      }
    else
      {
        if  (nargs  >  4)
          stripper(arg,  4);
        sprintf(tmp_cmd,  "power  %s  ",  arg);
        name_col(getcol(tmp_cmd,  4));
        action  =  POWER_NAME;
      }
    break;

  case  POWER_NAME:
    strcat(tmp_cmd,  arg);
    do_power(tmp_cmd);
    break;

  case  RECIP:
    if  (nargs  ==  4)
      {
        sprintf(tmp_cmd,  "recip  %s  ",  arg);
        do_recip(tmp_cmd);
      }
    else
      {
        if  (nargs  >  2)
          stripper(arg,  2);
        sprintf(tmp_cmd,  "recip  %s  ",  arg);
        name_col(getcol(tmp_cmd,  3));
        action  =  RECIP_NAME;
      }
```



```
    break;

  case  RECIP_NAME:
    strcat(tmp_cmd,  arg);
    do_recip(tmp_cmd);
    break;

  case  LOG:
    if  (nargs  ==  4)
      {
        sprintf(tmp_cmd,  "log  %s  ",  arg);
        do_log(tmp_cmd);
      }
    else
      {
        if  (nargs  >  2)
          stripper(arg,  2);
        sprintf(tmp_cmd,  "log  %s  ",  arg);
        name_col(getcol(tmp_cmd,  3));
        action  =  LOG_NAME;
      }

    break;

  case  LOG_NAME:
    strcat(tmp_cmd,  arg);
    do_log(tmp_cmd);
    break;

  case  EXPLIN:
    if  (nargs  ==  7)
      {
        sprintf(tmp_cmd,  "ExpLin  %s  ",  arg);
        do_ExpLin(tmp_cmd);
      }
    else
      {
        if  (nargs  >  5)
          stripper(arg,  5);
        sprintf(tmp_cmd,  "ExpLin  %s  ",  arg);
        name_col(getcol(tmp_cmd,  3));
        action  =  EXPLIN_NAME;
      }

    break;

  case  EXPLIN_NAME:
    strcat(tmp_cmd,  arg);
    do_ExpLin(tmp_cmd);
    break;

  case  POLY4:
    if  (nargs  ==  9)
      {
        sprintf(tmp_cmd,  "Poly4  %s  ",  arg);
        do_Poly4(tmp_cmd);
      }
    else
      {
        if  (nargs  >  7)
          stripper(arg,  7);
        sprintf(tmp_cmd,  "Poly4  %s  ",  arg);
        name_col(getcol(tmp_cmd,  3));
        action  =  POLY4_NAME;
```



```
      }

    break;

  case  POLY4_NAME:
    strcat(tmp_cmd,  arg);
    do_Poly4(tmp_cmd);
    break;

  case  GENSIN:
    if  (nargs  ==  8)
      {
        sprintf(tmp_cmd,  "gensin  %s  ",  arg);
        do_gensin(tmp_cmd);
      }
    else
      {
        if  (nargs  >  6)
          stripper(arg,  6);
        sprintf(tmp_cmd,  "gensin  %s  ",  arg);
    name_col(getcol(tmp_cmd,  3));
    action  =  GENSIN_NAME;
      }

    break;

  case  GENSIN_NAME:
    strcat(tmp_cmd,  arg);
    do_gensin(tmp_cmd);
    break;

  case  GENEXP:
     if  (nargs  ==  8)
      {
        sprintf(tmp_cmd,  "genexp  %s  ",  arg);
        do_genexp(tmp_cmd);
      }
    else
      {
        if  (nargs  >  6)
          stripper(arg,  6);
        sprintf(tmp_cmd,  "genexp  %s  ",  arg);
        name_col(getcol(tmp_cmd,  3));
        action  =  GENEXP_NAME;
      }

    break;

  case  GENEXP_NAME:
    strcat(tmp_cmd,  arg);
    do_genexp(tmp_cmd);
    break;

  case  RCLOW:
    if  (nargs  ==  6)
      {
        sprintf(tmp_cmd,  "rclow  %s  ",  arg);
        do_rclow(tmp_cmd);
      }
    else
      {
        if  (nargs  >  4)
          stripper(arg,  4);
        sprintf(tmp_cmd,  "rclow  %s  ",  arg);
```



```
        name_col(getcol(tmp_cmd,  3));
        action  =  RCLOW_NAME;
      }

    break;

  case  RCLOW_NAME:
    strcat(tmp_cmd,  arg);
    do_rclow(tmp_cmd);
    break;

  case  SCCHISQR:
    if  (nargs  ==  7)
      {
        sprintf(tmp_cmd,  "scchisqr  %s  ",  arg);
        do_scchisqr(tmp_cmd);
      }
    else
      {
        if  (nargs  >  5)
          stripper(arg,  5);
        sprintf(tmp_cmd,  "scchisqr  %s  ",  arg);
        name_col(getcol(tmp_cmd,  3));
        action  =  SCCHISQR_NAME;
      }

    break;

  case  SCCHISQR_NAME:
    strcat(tmp_cmd,  arg);
    do_scchisqr(tmp_cmd);
    break;

  case  CHISQR:
    if  (nargs  ==  5)
      {
        sprintf(tmp_cmd,  "chisqr  %s  ",  arg);
        do_chisqr(tmp_cmd);
      }
    else
      {
        if  (nargs  >  3)
          stripper(arg,  3);
        sprintf(tmp_cmd,  "chisqr  %s  ",  arg);
        name_col(getcol(tmp_cmd,  3));
        action  =  CHISQR_NAME;
      }

    break;

  case  CHISQR_NAME:
    strcat(tmp_cmd,  arg);
    do_chisqr(tmp_cmd);
    break;

  case  NORMAL:
    if  (nargs  ==  6)
      {
        sprintf(tmp_cmd,  "normal  %s  ",  arg);
        do_normal(tmp_cmd);
      }
    else
      {
        if  (nargs  >  4)
```

```
            stripper(arg, 4);
          sprintf(tmp_cmd, "normal %s ", arg);
          name_col(getcol(tmp_cmd, 3));
          action = NORMAL_NAME;
        }

      break;

    case NORMAL_NAME:
      strcat(tmp_cmd, arg);
      do_normal(tmp_cmd);
      break;

    case POWER2:
      if (nargs == 7)
        {
          sprintf(tmp_cmd, "Power2 %s ", arg);
          do_Power2(tmp_cmd);
        }
      else
        {
          if (nargs > 5)
            stripper(arg, 5);
          sprintf(tmp_cmd, "Power2 %s ", arg);
          name_col(getcol(tmp_cmd, 3));
          action = POWER2_NAME;
        }

      break;

    case POWER2_NAME:
      strcat(tmp_cmd, arg);
      do_Power2(tmp_cmd);
      break;

    case POWER1:
      if (nargs == 6)
        {
          sprintf(tmp_cmd, "Power1 %s ", arg);
          do_Power1(tmp_cmd);
        }
      else
        {
          if (nargs > 4)
            stripper(arg, 4);
          sprintf(tmp_cmd, "Power1 %s ", arg);
          name_col(getcol(tmp_cmd, 3));
          action = POWER1_NAME;
        }

      break;

    case POWER1_NAME:
      strcat(tmp_cmd, arg);
      do_Power1(tmp_cmd);
      break;

    case LN:
      if (nargs == 4)
        {
          sprintf(tmp_cmd, "ln %s ", arg);
          do_ln(tmp_cmd);
        }
      else
```



```
        {
          if (nargs > 2)
            stripper(arg, 2);
          sprintf(tmp_cmd, "ln %s ", arg);
          name_col(getcol(tmp_cmd, 3));
          action = LN_NAME;
        }

      break;

    case LN_NAME:
      strcat(tmp_cmd, arg);
      do_ln(tmp_cmd);
      break;

    case EXP:
      if (nargs == 4)
        {
          sprintf(tmp_cmd, "exp %s ", arg);
          do_exp(tmp_cmd);
        }
      else
        {
          if (nargs > 2)
            stripper(arg, 2);
          sprintf(tmp_cmd, "exp %s ", arg);
          name_col(getcol(tmp_cmd, 3));
          action = EXP_NAME;
        }

      break;

    case EXP_NAME:
      strcat(tmp_cmd, arg);
      do_exp(tmp_cmd);
      break;

    case DERIV:
      if (nargs == 7)
        {
          sprintf(tmp_cmd, "deriv %s, ", arg);
          do_deriv(tmp_cmd);
        }
      else
        {
          if (nargs > 5)
            {
              stripper(arg, 5);
              sprintf(tmp_cmd, "deriv %s, ", arg);
              name_col(getcol(tmp_cmd, 5));
              action = DERIV_NAME;
            }
          else
            {
              if (nargs > 3) stripper(arg, 3);
              sprintf(tmp_cmd, "deriv %s, ", arg);
              set_left_footer("Type the first and last record for the interval"
              set_cmd_prompt("First, Last: ");
              action = DERIV_GET_INT;
            }
        }
      break;

    case DERIV_GET_INT:
```



```
      if (nargs == 4)
        {
          strcat(tmp_cmd, arg);
          do_deriv(tmp_cmd);
        }
      else
        {
          if (nargs > 2) stripper(arg, 2);
          strcat(tmp_cmd, arg);
          strcat(tmp_cmd, " ");
          name_col(getcol(tmp_cmd, 4));
          action = DERIV_NAME;
        }
      break;

    case DERIV_NAME:
      strcat(tmp_cmd, arg);
      do_deriv(tmp_cmd);
      break;

    case VC:
      if (nargs == 8)
        {
          sprintf(tmp_cmd, "vc %s", arg);
          do_vc(tmp_cmd);
        }
      else
        {
          if (nargs > 3) stripper(arg, 3);
          sprintf(tmp_cmd, "vc %s, ", arg);
          action = VC_GET_INT;
          set_left_footer("Type the first and last record for the interval");
          set_cmd_prompt("First, Last: ");
        }

      break;

    case VC_GET_INT:
      if (nargs > 2) stripper(arg, 2);
      strcat(tmp_cmd, arg);
      strcat(tmp_cmd, ", ");
      action = VC_GET_F;
      sprintf(msg, "(S*Va/V) (supply the compressibility ([dv/V]/dP) and the ratio o
volume of material effected to the total volume -in the same units as tau \n:
      print_msg(msg);
      set_left_footer("Type the compressability");
      set_cmd_prompt("[dv/V]/dP: ");
      break;

    case VC_GET_F:
      if (nargs == 3)
        {
          strcat(tmp_cmd, arg);
          do_vc(tmp_cmd);
        }
      else
        {
          if (nargs > 1) stripper(arg, 1);
          strcat(tmp_cmd, arg);
          strcat(tmp_cmd, " ");
          name_col(getcol(tmp_cmd, 4));
          action = VC_NAME;
        }
      break;
```



```
    case VC_NAME:
      strcat(tmp_cmd, arg);
      do_vc(tmp_cmd);
      break;

    case RGT:
      if (nargs == 6)
        {
          sprintf(tmp_cmd, "rgt %s ", arg);
          do_rgt(tmp_cmd);
        }
      else
        {
          if (nargs > 4)
            stripper(arg, 4);
          sprintf(tmp_cmd, "rgt %s ", arg);
          name_col(getcol(tmp_cmd, 5));
          action = RGT_NAME;
        }

      break;

    case RGT_NAME:
      strcat(tmp_cmd, arg);
      do_rgt(tmp_cmd);
      break;

    case CGT:
      if (nargs == 6)
        {
          sprintf(tmp_cmd, "cgt %s ", arg);
          do_cgt(tmp_cmd);
        }
      else
        {
          if (nargs > 4)
            stripper(arg, 4);
          sprintf(tmp_cmd, "cgt %s ", arg);
          name_col(getcol(tmp_cmd, 5));
          action = CGT_NAME;
        }

      break;

    case CGT_NAME:
      strcat(tmp_cmd, arg);
      do_cgt(tmp_cmd);
      break;

    case EC:
      if (nargs == 8)
        {
          sprintf(tmp_cmd, "ec %s", arg);
          do_ec(tmp_cmd);
        }
      else if (nargs >= 6)
        {
          if (nargs > 6) stripper(arg, 6);
          sprintf(tmp_cmd, "ec %s, ", arg);
          name_col(getcol(tmp_cmd, 4));
          action = EC_NAME;
        }
      else if (nargs == 5)
```

```
    {
      sprintf(tmp_cmd, "ec %s, ", arg);
      set_left_footer("Type the ratio (E/L) in the same unitsas tau and ax_diaj
      set_cmd_prompt("E/L: ");
      action = EC_GET_F;
    }
  else
    {
      if (nargs > 3) stripper(arg, 3);
      sprintf(tmp_cmd, "ec %s, ", arg);
      action = EC_GET_INT;
      set_left_footer("Type the first and last record for the interval");
      set_cmd_prompt("First, Last: ");
    }
  break;

case EC_GET_INT:
  if (nargs == 5)
    {
      strcat(tmp_cmd, arg);
      do_ec(tmp_cmd);
    }
  else if (nargs >= 3)
    {
      if (nargs > 3) stripper(arg, 3);
      strcat(tmp_cmd, arg);
      name_col(getcol(tmp_cmd, 4));
      action = EC_NAME;
    }
  else
    {
      strcat(tmp_cmd, arg);
      action = EC_GET_F;
      set_left_footer("Type the ratio (E/L) in the same unitsas tau and ax_diaj
      set_cmd_prompt("E/L: ");
    }
  break;

case EC_GET_F:
  if (nargs == 3)
    {
      strcat(tmp_cmd, arg);
      do_ec(tmp_cmd);
    }
  else
    {
      if (nargs > 1) stripper(arg, 1);
      strcat(tmp_cmd, ", ");
      strcat(tmp_cmd, arg);
      strcat(tmp_cmd, ", ");
      name_col(getcol(tmp_cmd, 4));
      action = EC_NAME;
    }
  break;

case EC_NAME:
  strcat(tmp_cmd, arg);
  do_ec(tmp_cmd);
  break;

case CS:
  if (nargs > 3) stripper(arg, 3);
  sprintf(tmp_cmd, "cs %s, ", arg);
  action = CS_GET_INT;
```



```
  set_left_footer("Type the first and last record for the interval");
  set_cmd_prompt("First, Last: ");
  break;

case CS_GET_INT:
  if (nargs > 2) stripper(arg, 2);
  strcat(tmp_cmd, arg);
  strcat(tmp_cmd, " ");
  name_col(getcol(tmp_cmd, 4));
  action = CS_NAME;
  break;

case CS_NAME:
  strcat(tmp_cmd, arg);
  do_cs(tmp_cmd);
  break;

case POLYFIT:
  if (nargs > 4) stripper(arg, 4);
  sprintf(tmp_cmd, "polyfit %s, ", arg);
  action = POLYFIT_GET_INT;
  set_left_footer("Type the first and last record for the interval");

  set_cmd_prompt("First, Last: ");
  break;

case POLYFIT_GET_INT:
  if (nargs > 2) stripper(arg, 2);
  strcat(tmp_cmd, arg);
  action = POLYFIT_GET_INC;
  set_left_footer("Extend the fit past the interval (y/n) OR apply to entire
");
  set_cmd_prompt("Y/N/A: ");
  break;

case POLYFIT_GET_INC:
  if (arg[0] == 'y')
    {
      if (nargs > 1) stripper(arg, 1);
      strcat(tmp_cmd, ", ");
      strcat(tmp_cmd, arg);
      set_left_footer("Extend to what row number?");
      set_cmd_prompt("Row #: ");
      action = POLYFIT_I_GET_ROW;
    }
  else
    {
      strcat(tmp_cmd, ", ");
      strcat(tmp_cmd, arg);
      strcat(tmp_cmd, " 0");
      do_polyfit(tmp_cmd);
    }
  break;

case POLYFIT_I:
  if (nargs > 4) stripper(arg, 4);
  sprintf(tmp_cmd, "polyfit_i %s, ", arg);
  action = POLYFIT_I_GET_INT;
  set_left_footer("Type the first and last record for the interval");
  set_cmd_prompt("First, Last: ");
  break;

case POLYFIT_I_GET_INT:
  if (nargs > 2) stripper(arg, 2);
```



```
  strcat(tmp_cmd, arg);
  action = POLYFIT_I_GET_INC;
  set_left_footer("Type the dx increment or 'd' for default (max_x-min_x/#of ro
  set_cmd_prompt("dx increment: ");
  break;

case POLYFIT_I_GET_INC:
  if (arg[0] == 'd')
    {
      strcat(tmp_cmd, ", ");
      strcat(tmp_cmd, arg);
      strcat(tmp_cmd, ", 0");
      do_polyfit(tmp_cmd);
    }
  else
    {
      if (nargs > 1) stripper(arg, 1);
      strcat(tmp_cmd, ", ");
      strcat(tmp_cmd, arg);
      set_left_footer("Extend to what row number?");
      set_cmd_prompt("Row #: ");
      action = POLYFIT_I_GET_ROW;
    }
  break;

case POLYFIT_I_GET_ROW:
  strcat(tmp_cmd, " ");
  strcat(tmp_cmd, arg);
  do_polyfit(tmp_cmd);
  break;

case SORT:
  sprintf(tmp_cmd, "sort %s", arg);
  do_sort(tmp_cmd);
  break;

case INTERPOLATE_R:
  if (nargs > 4) stripper(arg, 4);
  sprintf(tmp_cmd, "interpolater %s ", arg);
  action = INTERPOLATE_GET_F_L;
  set_left_footer("Type the first and last record for the interval");
  set_cmd_prompt("First, Last: ");
  break;

case INTERPOLATE:
  if (nargs > 4) stripper(arg, 4);
  sprintf(tmp_cmd, "interpolate %s ", arg);
  action = INTERPOLATE_GET_F_L;
  set_left_footer("Type the first and last record for the interval");
  set_cmd_prompt("First, Last: ");
  break;

case INTERPOLATE_GET_F_L:
  if (nargs > 2) stripper(arg, 2);
  strcat(tmp_cmd, arg);
  strcat(tmp_cmd, " ");
  action = INTERPOLATE_GET_INC;
  set_left_footer("Type the dx increment or 'd' to get a default");
  set_cmd_prompt("Increment: ");
  break;

case INTERPOLATE_GET_INC:
  if (nargs > 1) stripper(arg, 1);
  strcat(tmp_cmd, arg);
```



```
  strcat(tmp_cmd, " ");
  name_col(getcol(tmp_cmd, 4));
  action = INTERPOLATE_NAME1;
  break;

case INTERPOLATE_NAME1:
  strcat(tmp_cmd, arg);
  strcat(tmp_cmd, " ");
  name_col(getcol(tmp_cmd, 5));
  action = INTERPOLATE_NAME2;
  break;

case INTERPOLATE_NAME2:
  strcat(tmp_cmd, arg);
  do_interpolate(tmp_cmd);
  break;

case MATHINT :
  if (nargs > 5) stripper(arg, 5);
  sprintf(tmp_cmd, "mathint %s ", arg);
  action = MATHINT_F_L;
  set_left_footer("Type the first and last record for the interval");
  set_cmd_prompt("First, Last: ");
  break;

case MATHINT_F_L:
  if (nargs > 2) stripper(arg, 2);
  strcat(tmp_cmd, arg);
  strcat(tmp_cmd, ", ");
  name_col(getcol(tmp_cmd, 6));
  action = MATHINT_NAME;
  break;

case MATHINT_NAME:
  strcat(tmp_cmd, arg);
  do_mathint(tmp_cmd);
  break;

case MATH:
  if (nargs == 7)
    {
      sprintf(tmp_cmd, "math %s ", arg);
      do_math(tmp_cmd);
    }
  else
    {
      if (nargs > 5)
        stripper(arg, 5);
      sprintf(tmp_cmd, "math %s ", arg);
      name_col(getcol(tmp_cmd, 6));
      action = MATH_NAME;
    }

  break;

case MATH_NAME:
  strcat(tmp_cmd, arg);
  do_math(tmp_cmd);
  break;

case POSITIVE:
  if (nargs == 4)
  {
    sprintf(tmp_cmd, "positive %s ", arg);
```

```
        do_positive(tmp_cmd);
      }
    else
      {
        if (nargs > 2)
          stripper(arg, 2);
        sprintf(tmp_cmd, "positive %s ", arg);
        name_col(getcol(tmp_cmd, 3));
        action = POSITIVE_NAME;
      }

    break;

  case POSITIVE_NAME:
    strcat(tmp_cmd, arg);
    do_positive(tmp_cmd);
    break;

  case COMPRESS:
     if (nargs == 4)
      {
        sprintf(tmp_cmd, "compress %s ", arg);
        do_compress(tmp_cmd);
      }
    else
      {
        if (nargs > 2)
          stripper(arg, 2);
        sprintf(tmp_cmd, "compress %s ", arg);
        name_col(getcol(tmp_cmd, 3));
        action = COMPRESS_NAME;
      }

    break;

  case COMPRESS_NAME:
    strcat(tmp_cmd, arg);
    do_compress(tmp_cmd);
    break;

  case R_MEAN:
     if (nargs == 4)
      {
        sprintf(tmp_cmd, "r_mean %s ", arg);
        do_r_mean(tmp_cmd);
      }
    else
      {
        if (nargs > 2)
          stripper(arg, 2);
        sprintf(tmp_cmd, "r_mean %s ", arg);
        name_col(getcol(tmp_cmd, 3));
        action = R_MEAN_NAME;
      }

    break;

  case R_MEAN_NAME:
    strcat(tmp_cmd, arg);
    do_r_mean(tmp_cmd);
    break;

  case R_SPIKE:
    sprintf(tmp_cmd, "r_spike %s", arg);
```



```
    do_r_spike(tmp_cmd);
    break;

  case Z_MAX:
  if (nargs == 4)
      {
        sprintf(tmp_cmd, "z_max %s ", arg);
        do_z_max(tmp_cmd);
      }
    else
      {
        if (nargs > 2)
          stripper(arg, 2);
        sprintf(tmp_cmd, "z_max %s ", arg);
        name_col(getcol(tmp_cmd, 3));
        action = Z_MAX_NAME;
      }

    break;

  case Z_MAX_NAME:
    strcat(tmp_cmd, arg);
    do_z_max(tmp_cmd);
    break;

  case Z_MIN:
    if (nargs == 4)
      {
        sprintf(tmp_cmd, "z_min %s ", arg);
        do_z_min(tmp_cmd);
      }
    else
      {
        if (nargs > 2)
          stripper(arg, 2);
        sprintf(tmp_cmd, "z_min %s ", arg);
        name_col(getcol(tmp_cmd, 3));
        action = Z_MIN_NAME;
      }

    break;

  case Z_MIN_NAME:
    strcat(tmp_cmd, arg);
    do_z_min(tmp_cmd);
    break;



    /* r_trend -> r_trend_type -> r_trend_input -> r_trend_input_name */
    /* r_trend -> r_trend_type -> r_trend_comp -> r_trend_comp_fl -> r_trend   p
/
  case R_TREND:
    sprintf(tmp_cmd, "r_trend %s ", arg);
    set_left_footer("Input slope and intercept or compute from least squares?");
    set_cmd_prompt("Input/Comp: ");
    action = R_TREND_TYPE;
    break;

  case R_TREND_TYPE:
    strcat(tmp_cmd, arg);
    strcat(tmp_cmd, " ");
    if (strncmp(arg, "input", 5) == 0)
      {
```



```
        set_left_footer("Input the X col, Y col, New col, Slope and Y-intercept")
        set_cmd_prompt("X-col Y-col New-col Slope Y-int: ");
        action = R_TREND_INPUT;
      }
    else
      {
        set_left_footer("Input the X col, Y col and New col");

        set_cmd_prompt("X-col Y-col New-col: ");
        action = R_TREND_COMP;
      }
    break;

  case R_TREND_INPUT:
    strcat(tmp_cmd, arg);
    strcat(tmp_cmd, " ");
    name_col(getcol(tmp_cmd, 6));
    action = R_TREND_INPUT_NAME;
    break;

  case R_TREND_INPUT_NAME:
    strcat(tmp_cmd, arg);
    do_r_trend_input(tmp_cmd);
    break;

  case R_TREND_COMP:
    strcat(tmp_cmd, arg);
    strcat(tmp_cmd, " ");
    set_left_footer("Input the interval for linear trend analysis");
    set_cmd_prompt("First Last: ");
    action = R_TREND_COMP_FL;
    break;

  case R_TREND_COMP_FL:
    strcat(tmp_cmd, arg);
    strcat(tmp_cmd, " ");
    name_col(getcol(tmp_cmd, 6));
    action = R_TREND_COMP_NAME;
    break;

  case R_TREND_COMP_NAME:
    strcat(tmp_cmd, arg);
    do_r_trend_comp(tmp_cmd);
    break;

  case TREND_XY:
    if (nargs > 2) stripper(arg, 2);
    sprintf(tmp_cmd, "trend %s, ", arg);
    action = TREND_F_L;
    set_left_footer("Type the first and last record number");
    set_cmd_prompt("First, Last: ");
    break;

  case TREND_F_L:
    strcat(tmp_cmd, arg);
    do_trend(tmp_cmd);
    break;

  case TREND_A:
    sprintf(tmp_cmd, "trend_a %s", arg);
    do_trend_a(tmp_cmd);
    break;

  case FINDINT:
```



```
    sprintf(tmp_cmd, "findint %s", arg);
    do_findint(tmp_cmd);
    break;

  case SUMMATION:
     if (nargs == 4)
      {
        sprintf(tmp_cmd, "summation %s ", arg);
        do_summation(tmp_cmd);
      }
    else
      {
        if (nargs > 2)
          stripper(arg, 2);
        sprintf(tmp_cmd, "summation %s ", arg);
        name_col(getcol(tmp_cmd, 3));
        action = SUMMATION_NAME;
      }

    break;

  case SUMMATION_NAME:
    strcat(tmp_cmd, arg);
    do_summation(tmp_cmd);
    break;

  case CURV:
    if (nargs == 4)
      {
        sprintf(tmp_cmd, "curv %s ", arg);
        do_curv(tmp_cmd);
      }
    else
      {
        if (nargs > 2)
          stripper(arg, 2);
        sprintf(tmp_cmd, "curv %s ", arg);
        name_col(getcol(tmp_cmd, 3));
        action = CURV_NAME;
      }

    break;

  case CURV_NAME:
    strcat(tmp_cmd, arg);
    do_curv(tmp_cmd);
    break;

  case PEAK:
     if (nargs == 4)
      {
        sprintf(tmp_cmd, "peak %s ", arg);
        do_peak(tmp_cmd);
      }
    else
      {
        if (nargs > 2)
          stripper(arg, 2);
        sprintf(tmp_cmd, "peak %s ", arg);
        name_col(getcol(tmp_cmd, 3));
        action = PEAK_NAME;
      }

    break;
```

```
case PEAK_NAME:
  strcat(tmp_cmd, arg);
  do_peak(tmp_cmd);
  break;

case DECIMAT:
  if (nargs > 3) stripper(arg, 3);
  sprintf(tmp_cmd, "decimat %s, ", arg);
  set_left_footer("Interval for decimation? (first_row, last_row)");
  set_cmd_prompt("First, Last: ");
  action = DECIMAT_F_L;
  break;

case DECIMAT_R:
  if (nargs > 3) stripper(arg, 3);
  sprintf(tmp_cmd, "decimat_r %s, ", arg);
  set_left_footer("Interval for decimation? (first_row, last_row)");
  set_cmd_prompt("First, Last: ");
  action = DECIMAT_F_L;
  break;

case DECIMAT_F_L:
  if (nargs > 2) stripper(arg, 2);
  strcat(tmp_cmd, arg);
  strcat(tmp_cmd, " ");
  name_col(getcol(tmp_cmd, 3));
  action = DECIMAT_NAME;
  break;

case DECIMAT_NAME:
  strcat(tmp_cmd, arg);
  do_decimat(tmp_cmd);
  break;

case PDP:
  sprintf(tmp_cmd, "pdf %s", arg);
  do_pdf(tmp_cmd);
  action = MAIN;
  break;

case PDPAUTO:
  sprintf(tmp_cmd, "pdfauto %s", arg);
  do_pdfauto(tmp_cmd);
  action = MAIN;
  break;

case R_ROW_COL:
  if (nargs > 1) stripper(arg, 1);
  sprintf(tmp_cmd, "r_row_col, %s, ", arg);
  set_left_footer("Type the first and last row to remove");
  set_cmd_prompt("First, Last: ");
  action = R_ROW_COL_F_L;
  break;

case R_ROW_COL_F_L:
  strcat(tmp_cmd, arg);
  do_r_row_col(tmp_cmd);
  action = MAIN;
  break;

case R_ROW_F_L:
  sprintf(tmp_cmd, "r_row %s", arg);
  do_r_row(tmp_cmd);
```



```
  action = MAIN;
  break;

case COMMENT_COL:
  if (nargs > 2)
    {
      sprintf(tmp_cmd, "comment %s", arg);
      do_comment(tmp_cmd);
      action = MAIN;
    }
  else
    {
      sprintf(tmp_cmd, "comment %s ", arg);
      set_left_footer("Type comment (up to 50 chars)");
      set_cmd_prompt("Comment: ");
      action = COMMENT_STR;
    }
  break;

case COMMENT_STR:
  strcat(tmp_cmd, arg);
  do_comment(tmp_cmd);
  break;

case NAME:
  sprintf(tmp_cmd, "cmd %s", arg);
  do_name(tmp_cmd);
  break;

/*
case NAME_GET_NAME:
  printf("NAME_GET_NAME\n");
  if (nargs > 1)
    stripper(arg, 1);
  name_got_name(arg);
  break;

case NAME_GET_UNIT:
  printf("NAME_GET_UNIT\n");
  name_got_unit(arg);
  break;
*/

case R_COL:
  do_r_col(atoi(arg));
  break;

case ZERO_ALL:
  sprintf(tmp_cmd, "zero_all %s", arg);
  do_zero_all(tmp_cmd);
  break;

case ZERO:
  sprintf(tmp_cmd, "zero %s", arg);
  do_zero(tmp_cmd);
  break;

case OFFSET:
  sprintf(tmp_cmd, "offset %s", arg);
  do_offset(tmp_cmd);
  break;

case OFFSET_INT:
  if (nargs == 4)
```



```
    {
      sprintf(tmp_cmd, "offset_int %s", arg);
      do_offset_int(tmp_cmd);
    }
  else
    {
    /* if (nargs > 3)
         stripper(arg, 3); */
      sprintf(tmp_cmd, "offset_int %s, ", arg);
      set_left_footer("Set COL values between REC1 and REC2 equal to REC1?");
      set_cmd_prompt("y/n: ");
      action = OFFSET_INT_GET_YN;
    }

  break;

case OFFSET_INT_GET_YN:
  strcat(tmp_cmd, arg);
  do_offset_int(tmp_cmd);
  break;

case SIMPLEX_GET_FN:
  if (nargs > 1)
    stripper(arg, 1);
  sprintf(tmp_cmd, "simplex %s, ", arg);
  set_left_footer("Enter the first and last records for fit");
  set_cmd_prompt("First, Last: ");
  action = SIMPLEX_GET_F_L;
  break;

case SIMPLEX_GET_F_L:
  if (nargs > 2)
    stripper(arg, 2);
  strcat(tmp_cmd, arg);
  set_left_footer("Enter the X-col and Y-col");
  set_cmd_prompt("X-col, Y-col: ");
  action = SIMPLEX_GET_COLS;
  break;

case SIMPLEX_GET_COLS:
  strcat(tmp_cmd, ", ");
  strcat(tmp_cmd, arg);
  do_simplex(tmp_cmd);
  break;

case EXAMIN_GET_FILENAME:
  do_examin(arg);
  action = MAIN;
  break;

case HEAD_GET_FILENAME:
  do_head(arg);
  action = MAIN;
  break;

case GETASCHEAD_GET_FILENAME:
  do_head(arg);
  action = MAIN;
  break;

case TASC_GET_FILENAME:
  if (nargs == 2)
    {
      sprintf(tmp_cmd, "tasc %s, ", arg);
```



```
      do_tasc(tmp_cmd);
    }
  else
    {
      if (nargs > 1)
        stripper(arg, 1);
      sprintf(tmp_cmd, "tasc %s, ", arg);
      set_left_footer("Integer or float data?");
      set_cmd_prompt("integer/float: ");
      action = TASC_GET_FI;
    }
  break;

case TASC_GET_FI:
  strcat(tmp_cmd, arg);
  do_tasc(tmp_cmd);
  break;

case STDASC_GET_FILENAME:
  if (nargs > 1)
    stripper(arg, 1);
  sprintf(tmp_cmd, "stdasc %s, ", arg);
  set_left_footer("Read complex or real data?");
  set_cmd_prompt("complex/real: ");
  action = STDASC_GET_CR;
  break;

case STDASC_GET_CR:
  if (nargs > 1)
    stripper(arg, 1);
  strcat(tmp_cmd, arg);
  set_left_footer("Use float or scaled integer format?");
  set_cmd_prompt("float/int: ");
  action = STDASC_GET_FI;
  break;

case STDASC_GET_FI:
  strcat(tmp_cmd, ", ");
  strcat(tmp_cmd, arg);
  do_stdasc(tmp_cmd);
  break;

case DOIT:
  doit_proc(arg);
  break;

case READ:
  sprintf(tmp_cmd, "read %s", arg);
  read_proc(tmp_cmd);
  break;

case APPEND:
  sprintf(tmp_cmd, "append %s", arg);
  read_proc(tmp_cmd);
  break;

case WRITE:
  write_proc(arg);
  break;

case ALL_NEED_ROW_COL:
  sprintf(tmp_cmd, "all yes, %s", arg);
  all_final_proc(tmp_cmd);
  action = MAIN;
```

```
      break;

    case  SET_PATH:
      set_path_proc(arg);
      action = MAIN;
      break;

    case  PLOT_GET_KY:
      sprintf(tmp_cmd, "%s %s", plot_cmd, arg);
      do_plot(tmp_cmd);
      break;

    case  PLOT_GET_BE:
      if (nargs > 2) stripper(arg, 2);
      sprintf(tmp_cmd, "%s %s, ", plot_cmd, arg);
      set_left_footer("Type the first and last records to plot.");
      set_cmd_prompt("Begin, End: ");
      if (strncmp(plot_cmd, "plotscale", 9) == 0)
        action = PLOT_SCALE;
      else
        action = PLOT_OTHERS;
      break;

    case  PLOT_SCALE:
      if (nargs > 2) stripper(arg, 2);
      strcat(tmp_cmd, arg);
      strcat(tmp_cmd, ", ");
      set_left_footer("Type the limits X_max, X_min, Y_max, Y_min");
      set_cmd_prompt("X_max, X_min, Y_max, Y_min: ");
      action = PLOT_OTHERS;
      break;

    case  PLOT_OTHERS:
      strcat(tmp_cmd, arg);
      do_plot(tmp_cmd);
      break;

    case  SET_ACTIVE_PLOT:
      set_active_plot(atoi(arg)-1);
      action = MAIN;
      top();
      break;

    case  SET_ACTIVE_WINDOW:
      set_active_window(atoi(arg)-1);
      action = MAIN;
      top();
      break;

    case  KILL_WIN:
      kill_win_proc(atoi(arg));
      action = MAIN;
      break;

    case  KILL_PLOT:
      del_plot_proc(atoi(arg));
      action = MAIN;
      break;

    default:
      action = MAIN;
      top();
      break;
```



```
    }
  }
```



```
/* functions RECALL, EXAMIN, RITE, REED, TASC, and OLDREAD for program LOOK */
#include "global.h"

extern  void  print_msg();
extern  char  msg[MSG_LENGTH];
extern  int  action;


/****************************** examin ******************************/
/* done */
void  examin(dfile)
    FILE  *dfile  ;
{
  int  i  ;
  struct  header  temphead  ;


  fread(&(temphead.title[0]),1,20,dfile)  ;
  fread(&(temphead.nrec),4,1,dfile)  ;
  fread(&(temphead.nchan),4,1,dfile)  ;
  fread(&(temphead.swp),4,1,dfile)  ;
  fread(&(temphead.dtime),4,1,dfile)  ;
  for( i = 1; i < MAX_COL; ++i )
    {
      fread(&(temphead.ch[i].name[0]),1,13,dfile)  ;
      fread(&(temphead.ch[i].units[0]),1,13,dfile)  ;
      fread(&(temphead.ch[i].gain),4,1,dfile)  ;
      fread(&(temphead.ch[i].comment[0]),1,50,dfile)  ;
      fread(&(temphead.ch[i].nelem),4,1,dfile)  ;
    }

  sprintf(msg,"%s\n",temphead.title)  ;
  print_msg(msg);

  sprintf(msg,"nrec = %5d, nchan = %5d, swp = %5d, dtime = %14.7e\n",temphead.nrec
d.nchan,temphead.swp,temphead.dtime);
  print_msg(msg);

  for( i = 1; i < MAX_COL; ++i )
    {
      sprintf(msg,"%2d %13s %13s %14.7e %5d\n",i,&(temphead.ch[i].name[0]),&(temphead.ch
].units[0]),temphead.ch[i].gain,temphead.ch[i].nelem)  ;
      print_msg(msg);
      sprintf(msg,"%50s\n",&(temphead.ch[i].comment[0]))  ;
      print_msg(msg);
    }
}

/****************************** rite ******************************/
/* done */
void  rite(dfile)
    FILE  *dfile;
{
  int  i  ;

  fwrite(&(head.title[0]),1,20,dfile)  ;
  fwrite(&(head.nrec),4,1,dfile)  ;
  fwrite(&(head.nchan),4,1,dfile)  ;
  fwrite(&(head.swp),4,1,dfile)  ;
  fwrite(&(head.dtime),4,1,dfile)  ;
  for( i = 1; i < MAX_COL; ++i )
    {
      fwrite(&(head.ch[i].name[0]),1,13,dfile)  ;
      fwrite(&(head.ch[i].units[0]),1,13,dfile)  ;
```



```
      fwrite(&(head.ch[i].gain),4,1,dfile)  ;
      fwrite(&(head.ch[i].comment[0]),1,50,dfile)  ;
      fwrite(&(head.ch[i].nelem),4,1,dfile)  ;
    }

  for( i = 1; i < MAX_COL; ++i )
    {
      if( (strncmp(&(head.ch[i].name[0]),"no_val",6) != 0))
        {
          fwrite(darray[i],4,head.ch[i].nelem,dfile)  ;
        }
    }
}


/****************************** reed ******************************/
int  reed(dfile,append)
    FILE       *dfile  ;
    int        append;
{
  int  i  ;
  int  rec , chan, file1_nrec  ;
  int  file1_nelem[MAX_COL];
  char  title[20]  ;


  i = 2;
  file1_nrec = 0;
  fread(&(title[0]),1,20,dfile)  ;
  fread(&(rec),4,1,dfile)  ;
  fread(&(chan),4,1,dfile)  ;

  if(append == TRUE)
    {
      file1_nrec = head.nrec;
      chan = (chan > head.nchan) ? chan : head.nchan;
    }

  if( (file1_nrec+rec+1>max_row || chan+3>max_col ) && (allocate((file1_nrec+rec
+3)) != 1))
    {
      sprintf(msg,"Problem with allocation.\nALLOCATION\tnrec = %d\tnchan = %d\n ,_ec
);
      print_msg(msg);
      return 0 ;
    }

  head.nrec = file1_nrec+rec  ;
  head.nchan = chan  ;

  if(append == TRUE)
    {
      fseek(dfile,8, 1);
      for( i = 1; i < MAX_COL; ++i )
        {
          fseek(dfile, 80, 1);
          file1_nelem[i] = head.ch[i].nelem;    /* save old */
          fread(&(head.ch[i].nelem),4,1,dfile)  ;
        }
    }
  else                  /* use the first file for title, gain etc */
    {
      strcpy(&(head.title[0]),&(title[0]))  ;
```

```
    fread(&(head.swp),4,1,dfile)  ;
    fread(&(head.dtime),4,1,dfile)  ;
    for(  i = 1;  i  <  MAX_COL;  ++i  )
      {
        fread(&(head.ch[i].name[0]),1,13,dfile)  ;
        fread(&(head.ch[i].units[0]),1,13,dfile)  ;
        fread(&(head.ch[i].gain),4,1,dfile)  ;
        fread(&(head.ch[i].comment[0]),1,50,dfile)  ;
        fread(&(head.ch[i].nelem),4,1,dfile)  ;
        filei_nelem[i]  =  0;
      }
  }

 for(  i  =  1;  i  <  MAX_COL;  ++i  )
  {
   if(  (strncmp(&(head.ch[i].name[0]),"no_val",6)  !=  0))
    {
     if(head.ch[i].nelem  <=  0)
      {
        sprintf(msg,"WARNING:  problem  with  header.\n  Active  column  has  no  elem
n");
        print_msg(msg);
        sprintf(msg,"Assumed  nelem  =  nrec\nCONTINUE  READ\n");
        print_msg(msg);
        head.ch[i].nelem  =  rec  ;
      }
     fread(&darray[i][filei_nelem[i]],4,head.ch[i].nelem,dfile)  ;
     head.ch[i].nelem  +=  filei_nelem[i];
    }
  }
 return  1  ;
}

/*  ----------------------------  STDASC  ----------------------------  */
/*  done  */
int  stdasc(pfile,  cr,  fi)
     FILE  *pfile  ;
     char  cr[20],  fi[20];
{
 int  i  ,  j  ,  temp_int;
 char  headline[20];
 float  samp;
 float  tmax,  tmin;
 /*  char  *strcat()  ,  *strcmp()  ;*/
 int  chan  ,  rec  ;


 while(strncmp(headline,"ivar",4)  !=  0)
  {
   fscanf(pfile,"%s\n",headline)  ;
  }
 fscanf(pfile,"%6d\n",&(chan))  ;
 while(strncmp(headline,"ndata",5)  !=  0)
  {
   fscanf(pfile,"%s,\n",headline)  ;
  }
 fscanf(pfile,"%6d\n",&(rec))  ;
 if(  rec>=max_row  ||  chan>=max_col  )
  {
    sprintf(msg,"INSUFFICIENT  ALLOCATION\nnrec  =  %d\nnchan  =  %d\n",rec,chan);
    print_msg(msg);
    return  0  ;
  }
 else
```



```
  {
    head.nrec  =  rec  ;
    head.nchan  =  chan  ;
  }
 while(strncmp(headline,"samp",4)  !=  0)
  {
   fscanf(pfile,"%s,\n",headline)  ;
  }
 fscanf(pfile,"%f\n",&samp)  ;
 while(strncmp(headline,"tmin",4)  !=  0)
  {
   fscanf(pfile,"%s,\n",headline)  ;
  }
 fscanf(pfile,"%f\n",&tmin)  ;
 while(strncmp(headline,"tmax",4)  !=  0)
  {
   fscanf(pfile,"%s,\n",headline)  ;
  }
 fscanf(pfile,"%f\n",&tmax)  ;
 while(strncmp(headline,"title",5)  !=  0)
  {
   fscanf(pfile,"%s,\n",headline)  ;
  }
 fscanf(pfile,"%s,\n",&(head.title[0]))  ;
 while(strncmp(headline,"24,",3)  !=  0)
  {
   fscanf(pfile,"%s,\n",headline)  ;
  }
 fscanf(pfile,"%f\n",&(head.ch[2].gain))  ;
 while(strncmp(headline,"data",4)  !=  0)
  {
   fscanf(pfile,"%s\n",headline)  ;
  }
 strcpy(headline,  cr);
 if(strncmp(headline,"comp",4)  ==  0)
  {
    strcpy(&(head.ch[1].name[0]),"ind_var")  ;
    strcpy(&(head.ch[1].units[0]),"no_info")  ;
    strcpy(&(head.ch[2].name[0]),"real")  ;
    strcpy(&(head.ch[2].units[0]),"no_info")  ;
    strcpy(&(head.ch[3].name[0]),"complex")  ;
    strcpy(&(head.ch[3].units[0]),"no_info")  ;
    head.nchan  =  3  ;
    head.nrec  /=  2  ;
  }
 else
  {
    fprintf(stderr,"reading  data  as  real  numbers\n");
    strcpy(&(head.ch[1].name[0]),"independ")  ;
    strcpy(&(head.ch[1].units[0]),"no_info")  ;
    strcpy(&(head.ch[2].name[0]),"depend")  ;
    strcpy(&(head.ch[2].units[0]),"no_info")  ;
    head.nchan  =  2  ;
/*  printf("float  or  scaled  interger  format??\n");
    fscanf(stdin,"%s",headline);  */
    strcpy(headline,  fi);
  }

 /*    READ  DATA  */
 for  (  i  =  0;  i  <  head.nrec;  ++i)
  {
    for  (  j  =  1;  j  <  3;  ++j)
     {
       if  (  j  ==  1  )
```



```
          {
            darray[j][i]  =  samp  ;
          }
        else
          {
            if  (  head.nchan  ==  2  )
              {
                if(strncmp(headline,  "float",  5)  ==  0)
                  {
                    fscanf(pfile,"  %f,\n",&(darray[j][i]));
                  }
                else
                  {
                    fscanf(pfile,"  %f\n",&(darray[j][i]));
                    darray[j][i]  *=  head.ch[2].gain;
                  }
              }
            if  (  head.nchan  ==  3  )
              {
                fscanf(pfile,"  %f,\n",&(darray[j][i]));
                darray[j][i]  *=  head.ch[2].gain;
                fscanf(pfile,"  %f,\n",&(darray[j+1][i]));
                darray[j+1][i]  *=  head.ch[2].gain;
              }
          }
      }
  }
 return  1  ;
}


/*------------------------------  TASC  ----------------------------------*/
/*  done  */
int  tasc(pfile,  float_flag)
     FILE  *pfile  ;
     int  float_flag;
{

 int  i  ,  j  ,  temp_int  ,  rec  ,  chan;
 char  junk  ;
 float  xtra[5]  ;
 char  end_head[5]  ,  title[20];
 /*  char  *strcat()  ,  *strcmp()  ;*/
 float  temp_float;

 fscanf(pfile,"%s\n",&(title[0]))  ;
 fscanf(pfile,"%5d%5d\n",&(chan),&(rec))  ;
 if  (chan  >=  max_col  ||  rec  >=  max_row)
  {
    sprintf(msg,  "INSUFFICIENT  ALLOCATION\nnrec  =  %d\nnchan  =  %d\n",rec,chan+1);
    print_msg(msg);
    return  0  ;
  }
 else
  {
    head.nrec  =  rec  ;
    head.nchan  =  chan  ;
    strcpy(&(head.title[0]),&title[0])  ;
  }
 fscanf(pfile,"%5d  %e\n",&(head.swp),&(head.dtime))  ;
 for  (  i  =  2;  i  <=  head.nchan+1;  ++i)
  {
    fscanf(pfile,"%e",&(head.ch[i].gain))  ;
    fscanf(pfile,"%s  %s\n",&(head.ch[i].name[0]),&(head.ch[i].units[0]))  ;
```



```
    strcpy(&(head.ch[i].comment[0]),"none")  ;
    head.ch[i].nelem  =  head.nrec  ;
  }
 i  =  1  ;
 strcpy(&(head.ch[i].name[0]),"deltatime")  ;
 strcpy(&(head.ch[i].units[0]),"seconds")  ;
 strcpy(&(head.ch[i].comment[0]),"none")  ;
 head.ch[i].gain  =  1.0  ;
 head.ch[i].nelem  =  head.nrec  ;
 head.nchan  +=  1  ;

 for  (  i  =  0;  i  <  5;  ++i)
  {
    fscanf(pfile,"%s\n",&(head.extra[i]))  ;
  }
 fscanf(pfile,"%s\n",&(end_head[0]))  ;

 if  (strncmp(&(end_head[0])  ,  "EOH",  4  )  ==  0)
  {
    sprintf(msg,  "Header  read  successfully.\n");
    print_msg(msg);
  }
 else
  {
    sprintf(msg,  "Problem  reading  header  info  -  sorry!\n");
    print_msg(msg);
    return  0  ;
  }

 /*    READ  DATA  */

 if(float_flag  ==  0)
   sprintf(msg,  "Reading  data  as  int.\n");
 else
   sprintf(msg,  "Reading  data  as  float.\n");
 print_msg(msg);

 for  (  i  =  0;  i  <  head.nrec;  ++i)
  {
    for  (  j  =  1;  j  <=  head.nchan;  ++j)
     {
       /*0  =  int     1  =  float       */
       if(float_flag  ==  0)
         {
           fscanf(pfile,"%10d",&temp_int)  ;

           if  (  j  ==  1  )
             {
               darray[j][i]  =  temp_int  *  head.dtime;
             }
           else
             {
               darray[j][i]  =  temp_int  *  head.ch[j].gain  ;
             }
         }

       else
         {
           fscanf(pfile,"%e",&temp_float)  ;

           if  (  j  ==  1  )
             {
               darray[j][i]  =  temp_float  *  head.dtime;
             }
```

```
            else
              {
                darray[j][i] = temp_float * head.cb[j].gain ;
              }
          }

      }
     fscanf(pfile,"%c",&junk);
     if ( junk == '\n' || junk == ' ' || junk == '\t' ) ;
     else
       {
         sprintf(msg, "Error reading data on line %d!\n",i) ;
         print_msg(msg);
         i = head.nrec ;
       }

  }
 fclose(pfile);
 return 1 ;
}
/*****************************************/
```



```
#include        "global.h"
#include        <math.h>
#include        <stdio.h>

#define         SR_2  1.41421356
#define         BIGNUM  1e20

extern  char  msg[MSG_LENGTH];



rate_state_mod(disp_col, vstep_row, vstep_end, mod_disp_col, mod_mu_col, k, sig_n, m
f, vo, vf, A, B1, B2, dc1, dc2)
     int        disp_col, vstep_row, vstep_end, mod_disp_col, mod_mu_col;
     double     k, sig_n, muo, muf, vo, vf, A, B1, B2, dc1, dc2;
{
  int   i,j,n_data,end, init_pts, init_write_pt;
  int   isnan();
  double        psi1, psi2, h, H, K, K1, K2, K3, K4, L;
  double        x, x_inc, fin_disp_inc, init_disp_inc, init_disp, mu, v;
  double M1, M2, M3, M4, Vo_dc1, Vo_dc2, w_psi1, w_mu, time, alpha;
  double        J1, J2, J3, J4, w_psi2;
  double  wmu_muf_on_A, B1_on_A, B2_on_A, omB1_on_A, omB2_on_A, K_s_vf;
  double  exp();


  /*    if(k > 0.05)
        init_write_pt = 1;*/    /* write every other pt */
  /*    else */
  init_write_pt =2;     /* write every 3rd pt */

  /*set up so that Vo is velocity *after* step. MUo is mu *after* step */
  mu = muo;
  /* get disp increment so we know how many points to write*/
  fin_disp_inc = (darray[disp_col][vstep_end] - darray[disp_col][vstep_row]) / (5 0*(
_end - vstep_row) );
  /* write initial 20 % of the data closely spaced */
  init_pts = vstep_row + (int) (0.2*(vstep_end - vstep_row));

  /*init_disp_inc = fin_disp_inc / 25.00;
    init_disp = darray[disp_col][vstep_row] + (darray[disp_col][vstep_end] - darray[d
ol][vstep_row]) * 0.02000;
    fin_disp_inc /= 0.5102; */

  /* the model mu output contains 100 rows (if possible) of the initial m    a
lish a "baseline" */

  for ( i= (vstep_row > 100) ? (vstep_row-100) : 0; i <= vstep_row; ++i)
    {
     darray[mod_mu_col][i] = muo;
     darray[mod_disp_col][i] = darray[disp_col][i];;
    }
  /* increase so that model reaches sufficient disp for comparsion*/
  vstep_end = (vstep_end + 1000 < max_row) ? (vstep_end+1000) : max_row-1;
  /*                                                                    */
  /*                                                                    */
  /*dpsi/dt = -vf/dc1 exp((muo - muf - Bpsi)/A) [psi(1-B/A) + (muo - muf)/B*
  /*                                                                    */
  /*dmu/dt = k/sigma_n [vf (1 - exp((muo - muf - Bpsi)/A)                 *
  /*                                                                    */
  /*                                                                    */
  /*                                                                    */

  time = 0.0;           /*x is slip distance */
```



```
K_s_vf = (k / sig_n) * vf;
h = 0.010000; /* h = time step*/
/* normalize by velocity so that we have more points*/
/* in the fast areas and less when the slip vel is low*/
H = h / vo;
psi1 = psi2 = -log(vo/vf);

Vo_dc1 = -vf/dc1;
Vo_dc2 = -vf/dc2;

x = darray[disp_col][vstep_row];
x_inc = 0;

B1_on_A = B1/A;
B2_on_A = B2/A;

omB1_on_A = 1.000 - B1_on_A;
omB2_on_A = 1.000 - B2_on_A;

j = 0;                                  /* loop counter */
while( i <= vstep_end)
  {
    alpha = exp((mu - muf - B2*psi2 - B1*psi1)/A);
    wmu_muf_on_A = (mu - muf)/A;
    J1 = H * (Vo_dc1 * alpha * (psi1 * omB1_on_A - B2_on_A*psi2 + wmu_muf_on_
    K1 = H * (Vo_dc2 * alpha * (psi2 * omB2_on_A - B1_on_A*psi1 + wmu_muf_on_
    M1 = H * K_s_vf * (1.00000 - alpha) ;

    w_psi1 = psi1 + J1*0.500000;
    w_psi2 = psi2 + K1*0.500000;
    w_mu = mu + M1*0.500000;
    alpha = exp((w_mu - muf - B2*w_psi2 - B1*w_psi1)/A);
    wmu_muf_on_A = (w_mu - muf)/A;
    J2 = H*( Vo_dc1 * alpha * (w_psi1 * omB1_on_A - B2_on_A*w_psi2 + wmu_muf_
    K2 = H*( Vo_dc2 * alpha * (w_psi2 * omB2_on_A - B1_on_A*w_psi1 + wmu_muf_
    M2 = H* K_s_vf * (1.00000 - alpha) ;


    w_psi1 = psi1 + J2*0.500000000;
    w_psi2 = psi2 + K2*0.500000000;
    w_mu = mu + M2*0.500000000;
    alpha = exp((w_mu - muf - B2*w_psi2 - B1*w_psi1)/A);
    wmu_muf_on_A = (w_mu - muf)/A;
    J3 = H*( Vo_dc1 * alpha * (w_psi1 * omB1_on_A - B2_on_A*w_psi2 + wmu_muf_
    K3 = H*( Vo_dc2 * alpha * (w_psi2 * omB2_on_A - B1_on_A*w_psi1 + wmu_muf_
    M3 = H* K_s_vf * (1.00000 - alpha) ;


    w_psi1 = psi1 + J3;
    w_psi2 = psi2 + K3;
    w_mu = mu + M3;
    alpha = exp((w_mu - muf - B2*w_psi2 - B1*w_psi1)/A);
    wmu_muf_on_A = (w_mu - muf)/A;
    J4 = H*( Vo_dc1 * alpha * (w_psi1 * omB1_on_A - B2_on_A*w_psi2 + wmu_muf_
    K4 = H*( Vo_dc2 * alpha * (w_psi2 * omB2_on_A - B1_on_A*w_psi1 + wmu_muf_
    M4 = H* K_s_vf * (1.00000 - alpha) ;



    psi1 += (J1 + 2.0000*J2 + 2.0000*J3 + J4) * 0.166666666666667;  /*psi1[n+1]
] + .*/
    psi2 += (K1 + 2.0000*K2 + 2.0000*K3 + K4) * 0.166666666666667;  /*psi2[n+1]
] + .*/
    mu += (M1 + 2.0000*M2 + 2.0000*M3 + M4) * 0.166666666666667;
```



```
      v = vf * exp( (mu - muf - B1*psi1 -B2*psi2)/A );
      x_inc += vf * H;

      H = h / v;
      time += H;

     /* write out a reasonable # of points -instead of the huge # we'd
ery point */
     if( (j++ == init_write_pt && i < init_pts) || x_inc > fin_disp_inc )
       {
         j = 0;
         x += x_inc;
         x_inc = 0;
         darray[mod_disp_col][i] = x;
         darray[mod_mu_col][i] = mu;
         ++i;
         if( isnan(mu) || isnan(x) )
           {
             sprintf(msg, "Something bombed in calculation. Check parameter values
y reducing a or k \n");
             print_msg(msg);
             return(-1);
           }
       }
    }

  sprintf(msg, "%d points written before the vel. step and %d points writte af
step_row > 100) ? 100 : vstep_row, i-vstep_row-1);
  print_msg(msg);

}
/*****************************************************************/


double simp_rate_state_mod()

{

  /* declaring the doubles as register doesn't make any diff. to calc time */
  /*   at least when compiling with optimization... */


  char inc_pt = FALSE, adjust_H = TRUE;
  int  i, j, k, total_added_pts, row_num, calc_bombed = 0;
  int  isnan();
  double        *disp_ptr, *mu_ptr, *model_mu_ptr;
  double        psi1, psi2, h, H, X, K1, K2, K3, K4;
  double        close, not_close, mu, v, x;
  double M1, M2, M3, M4, Vo_dc1, Vo_dc2, w_psi1, w_mu, alpha;
  double        J1, J2, J3, J4, w_psi2;
  double B1_on_A, B2_on_A, wmu_muf_on_A, omB1_on_A, omB2_on_A, K_s_vf;
  double        current_disp;  /* used to handle out of order pts. -tdkr no
  double        final_disp;
  double        error, exp();
  double old_v, old_x, old_mu, old_psi1, old_psi2, old_H;

  total_added_pts = rs_param.added_pts*(rs_param.peak_row-rs_param.vs_row);

  /* first pt in array is vs_row */
  disp_ptr      = rs_param.disp_data;
  mu_ptr                = rs_param.mu_data;
```

```
model_mu_ptr  =  rs_param.model_mu;

error  =  0.00;

/*  look at x increment near v. step -assume this is the smallest inc */
/*    don't let close be smaller than 0.020 */

close  =  0.000;

for(i=0;  i  <  10;  ++i)
  {
    close  +=  fabs(  *(disp_ptr+1)  -  *disp_ptr++)/50.00  ;
  }
close  /=  10.00;

if(close  <  0.0200)  close  =  0.0200;
not_close  =  close*3.000;

/*  for a vo of 10 mic/s and H_init = 0.001  x_inc will be 0.01 micron ;  Sa
every  micron,  then */
/*    this should be small enough so that it's no problem getting  model  mu  val
right  disp.  */

/*                                                                    */
/*                                                                    */
/*dpsi/dt  =  -vf/dc1  exp[(muo  -  muf  -  Bpsi)/A]  [psi(1-B/A)  +  (muo  -  muf)/A]
/*                                                                    */
/*dmu/dt  =  k/sigma_n  [vf  (1  -  exp[(muo  -  muf  -  Bpsi)/A]                    */
/*                                                                    */
/*                                                                    */
/*                                                                    */

/*set  up  so  that  Vo  is  velocity  *after*  step.  MUo  is  mu  *after*  step  */
mu  =  rs_param.muo;
/*A  =  rs_param.a;  */
/*B1  =  rs_param.b1;  */
/*B2  =  rs_param.b2;  */

K_s_vf  =  (rs_param.stiff  /  rs_param.sig_n)  *  rs_param.vf;

if(close  >  0.1)
  h  =  0.01;   /*  use  bigger  time  step  if  disp.  spacing  is  large  */
else
  h  =  0.0050000;      /*  h  =  time  step*/
/*  normalize  by  velocity  so  that  we  have  more  points*/
/*  in  the  fast  areas  and  less  when  the  slip  vel  is  low*/
H  =  h  /  rs_param.vo;
psi1  =  psi2  =  -log(rs_param.vo/rs_param.vf);

Vo_dc1  =  -rs_param.vf/rs_param.dc1;
Vo_dc2  =  -rs_param.vf/rs_param.dc2;

B1_on_A  =  rs_param.b1/rs_param.a;
B2_on_A  =  rs_param.b2/rs_param.a;

omB1_on_A  =  1.000  -  B1_on_A;
omB2_on_A  =  1.000  -  B2_on_A;
                                        /*x  is  slip  distance  */
disp_ptr  =  rs_param.disp_data;          /*  re-align  */

x  =  current_disp  =  *disp_ptr++;
mu_ptr++;                          /*  first  pt  for  compar.  is  vs_row  +1*/

(rs_param.added_pts  ==  0)  ?  (row_num  =  rs_param.vs_row+1)  :  (row_num  =  rs_param.
```



```
    j  =  1;
    k  =  0;                                    /*  counter  for  sections  of  added  poir

    final_disp  =  *(disp_ptr  +  (rs_param.last_row-rs_param.vs_row-1+total_added_pts));
    while(  row_num  <=  rs_param.last_row  )
      {
        alpha  =  exp((mu  -  rs_param.muf  -  rs_param.b2*psi2  -  rs_param.b1*psi1)/rs_param
        wmu_muf_on_A  =  (mu  -  rs_param.muf)/rs_param.a;
        J1  =  H*(Vo_dc1  *  alpha  *  (psi1  *  omB1_on_A  -  B2_on_A*psi2  +  wmu_muf_on_A)
        K1  =  H*(Vo_dc2  *  alpha  *  (psi2  *  omB2_on_A  -  B1_on_A*psi1  +  wmu_muf_on_A)
        M1  =  H*  K_s_vf  *  (1.00  -  alpha)  ;

        w_psi1  =  psi1  +  J1*0.50000000000;
        w_psi2  =  psi2  +  K1*0.50000000000;
        w_mu  =  mu  +  M1*0.50000000000;
        alpha  =  exp((w_mu  -  rs_param.muf  -  rs_param.b2*w_psi2  -  rs_param.b1*w_psi1)/rs
.a);
        wmu_muf_on_A  =  (w_mu  -  rs_param.muf)/rs_param.a;
        J2  =  H*(  Vo_dc1  *  alpha  *  (w_psi1  *  omB1_on_A  -  B2_on_A*w_psi2  +  wmu_muf_
        K2  =  H*(  Vo_dc2  *  alpha  *  (w_psi2  *  omB2_on_A  -  B1_on_A*w_psi1  +  wmu_muf_
        M2  =  H*  K_s_vf  *  (1.00000  -  alpha)  ;


        w_psi1  =  psi1  +  J2*0.500000000000;
        w_psi2  =  psi2  +  K2*0.500000000000;
        w_mu  =  mu  +  M2*0.500000000000;
        alpha  =  exp((w_mu  -  rs_param.muf  -  rs_param.b2*w_psi2  -  rs_param.b1*w_psi1)/rs
.a);
        wmu_muf_on_A  =  (w_mu  -  rs_param.muf)/rs_param.a;
        J3  =  H*(  Vo_dc1  *  alpha  *  (w_psi1  *  omB1_on_A  -  B2_on_A*w_psi2  +  wmu_muf_
        K3  =  H*(  Vo_dc2  *  alpha  *  (w_psi2  *  omB2_on_A  -  B1_on_A*w_psi1  +  wmu_muf_
        M3  =  H*  K_s_vf  *  (1.00000  -  alpha)  ;


        w_psi1  =  psi1  +  J3;
        w_psi2  =  psi2  +  K3;
        w_mu  =  mu  +  M3;
        alpha  =  exp((w_mu  -  rs_param.muf  -  rs_param.b2*w_psi2  -  rs_param.b1*w_psi1)/rs
.a);
        wmu_muf_on_A  =  (w_mu  -  rs_param.muf)/rs_param.a;
        J4  =  H*(  Vo_dc1  *  alpha  *  (w_psi1  *  omB1_on_A  -  B2_on_A*w_psi2  +  wmu_muf_
        K4  =  H*(  Vo_dc2  *  alpha  *  (w_psi2  *  omB2_on_A  -  B1_on_A*w_psi1  +  wmu_muf_
        M4  =  H*  K_s_vf  *  (1.00000  -  alpha)  ;


        old_psi1  =  psi1;          /*  save  last  vals  */
        old_psi2  =  psi2;
        old_mu  =  mu;
        old_H  =  H;
        old_v  =  v;
        old_x  =  x;

        psi1  +=  (J1  +  2.00000*J2  +  2.00000*J3  +  J4)  *  0.16666666666666666666667;
+1]  =  psi1[n]  +  ./
        psi2  +=  (K1  +  2.00000*K2  +  2.00000*K3  +  K4)  *  0.16666666666666666666667;
+1]  =  psi2[n]  +  ./
        mu  +=  (M1  +  2.00000*M2  +  2.00000*M3  +  M4)  *  0.16666666666666666666667;



        v  =  rs_param.vf  *  exp(  (mu  -  rs_param.muf  -  rs_param.b1*psi1  -rs_param.b2*p
aram.a  );
        x  +=  rs_param.vf  *  H;
```



```
        H  =  h  /  v;

        if(  isnan(mu)  ||  isnan(v)  ||  (v  >  BIGNUM)  )
          {
            if(++calc_bombed  ==  5)              /*  give  up  if  problem  persists  */
              {
                sprintf(msg,  "calc  bombed  *out*  on:  a=%g,  b1=%g,  b2=%g,  dc1=%g,  dc2=%g
plex  continuing\n",rs_param.a,rs_param.b1,rs_param.b2,rs_param.dc1,rs_param.dc2);
                print_msg(msg);

                return(BIGNUM);          /*  inform  simplex  of  problem*/
              }
            else
              {
                psi1  =  old_psi1;  /*  install  old  vals  */
                psi2  =  old_psi2;
                mu  =  old_mu;
                H  =  old_H/2.00;       /*  reduce  H  -stablize  calc?  */
                v  =  old_v;
                x  =  old_x;
              }
          }
        else
          {
            calc_bombed  =  0;            /*  reset  */


            if(  x  >=  *disp_ptr  )   /*  gone  past  x  point  */
              {
                if(adjust_H)                    /*  adjust_H  starts  =  TRUE  */
                {             /*  adjust  H  to  get  mu  at  right  disp*/
                  psi1  =  old_psi1;  /*  start  calc  from  old  vals  */
                  psi2  =  old_psi2;
                  mu  =  old_mu;
                  v  =  old_v;
                  x  =  old_x;
                  H  =  fabs(x-*disp_ptr)/rs_param.vf;
                  adjust_H  =  FALSE;
                }
                else            /*  at  correct  disp;  calc  error  etc.*/
                {
                  adjust_H  =  TRUE;      /*  ready  for  next  pt*/

                  if(  fabs(*disp_ptr  -  x)  >  not_close)
                    {
                      sprintf(msg,  "\nProblem  in  rate/state  calc:  row#  %d,  not  compau
u  at  appropriate  x\n",  row_num);
                      print_msg(msg);

                      return(-2);  /*  not  comparing  mu  at  appropriate  x  */
                    }


                  /*  weighting  criteria  set  up  in  look_funcs.c  */
                  if(row_num  <=  rs_param.weight_pts  )          /*  put  extra  weight  o
e  portion  of  initial  data  */
                    {
                      if(  k  <  (rs_param.peak_row-rs_param.vs_row)  &&  rs_param.added_pts
0)
                        {
                          /*heavily  favor  overestimate  for  points  between  vs_row
                            and  peak  row  */
                          if(rs_param.weight_control  <  0)
                            {
```



```
                              /*  overestimate  better  than  underestimate  */
                              if(  fabs(mu  -  rs_param.muo)  >  fabs(*mu_ptr  -  rs_param.
                                error  +=  0.1*rs_param.weight*(*mu_ptr  -  mu)*(*mu_ptr  -
;
                              else
                                error  +=  10.0*rs_param.weight*(*mu_ptr  -  mu)*(*mu_ptr
);
                            }
                          else
                            error  +=  rs_param.weight*(*mu_ptr  -  mu)*(*mu_ptr  -  mu)  ;
                        }
                      else  if(  row_num  ==  rs_param.peak_row  )
                        {
                          if(rs_param.weight_control  <  0)
                            {
                              /*  overestimate  better  than  underestimate  */
                              if(  fabs(mu  -  rs_param.muo)  >  fabs(*mu_ptr  -  rs_param.
                                error  +=  0.200*rs_param.weight*(*mu_ptr  -  mu)*(*mu_ptr
u)  ;
                              else
                                error  +=  50.000*rs_param.weight*(*mu_ptr  -  mu)*(*mu_ptr
mu)  ;
                            }
                          else
                            error  +=  6.0*rs_param.weight*(*mu_ptr  -  mu)*(*mu_ptr  -  m
                        }
                      else                              /*  past  peak  row  or  no  added
s,  but  still  weighting  */
                        {
                          if(rs_param.weight_control  <  0)
                            {
                              if(  row_num  <  rs_param.peak_row  )
                                {
                                  /*  overestimate  better  than  underestimate  */
                                  if(  fabs(mu  -  rs_param.muo)  >  fabs(*mu_ptr  -  rs_p
o)  )
                                    error  +=  rs_param.weight*(*mu_ptr  -  mu)*(*mu_ptr
;
                                  else
                                    error  +=  5.000*rs_param.weight*(*mu_ptr  -  mu)*(*mu
-
                                                                                mu)  ;
                                }
                              else
                                {
                                  /*  underestimate  better  than  overestimate  */
                                  if(  fabs(mu  -  rs_param.muo)  <  fabs(*mu_ptr  -  rs_p
o)  )
                                    error  +=  rs_param.weight*(*mu_ptr  -  mu)*(*mu_ptr
;
                                  else
                                    error  +=  5.000*rs_param.weight*(*mu_ptr  -  mu)*(*mu
-  mu)  ;
                                }
                            }
                          else
                            error  +=  rs_param.weight*(*mu_ptr  -  mu)*(*mu_ptr  -  mu)
                        }
                    }
                  else
                    error  +=  (*mu_ptr  -  mu)*(*mu_ptr  -  mu)  ;

                  inc_pt  =  TRUE;
```

```
                while(inc_pt) /* increm. until next_x > curr. x */
                  {
                    disp_ptr++;                         /* increment pointers */
                    mu_ptr++;
                    *model_mu_ptr++ = mu;                /* save mu value */

  /* for debuging...
  printf("x=%g\tdisp_ptr=%g\tmu=%g\tmu_ptr=%g\trow_num=%d\n",x, *(disp_ptr-1), mu, *(mu_ptr-
  1),row_num); */

                    if( k >= rs_param.peak_row-rs_param.vs_row || rs_param.added_pts
  0 )
                      {
                        k++;
                        row_num++;
                      }
                    else if( j++ == rs_param.added_pts )
                      {
                        row_num++;
                        j = 0; /* start at zero, to allow for real pts <> inte
                        k++;
                      }

                    if(row_num == rs_param.weight_pts )
                      {
                        close *= 2.00;
                        not_close *= 2.00;
                        H *= 2.00; /* increase time step after weighted points *,
                        /* psuedo = to starting with H = 0.05 instead of 0.01 *
                      }
                    if(*disp_ptr <= current_disp)     /* check for noise in disp. t
  . */
                      inc_pt = TRUE;
                    else
                      {
                        inc_pt = FALSE; /* don't increment again */
                        current_disp = *disp_ptr; /* update disp. */
                      }
                  } /* end of increment pt loop */
              } /* end of else for adjust_H test */
          }  /* end of x > *disp_ptr test */
      }      /* end of else for nan test */
  }          /* end of loop for calc */

  if( row_num-1 != rs_param.last_row )
    {
      disp_ptr = rs_param.disp_data + (rs_param.last_row-rs_param.vs_row-1+total_added_pt
  -10);

      /* check for noise in disp. tdxr */
      for(k = 0, j = 0; j < 10; ++j)
        {
          if(*disp_ptr++ > final_disp)
            k = -1;            /* signify pts out of order */
        }

      if( isnan(mu) || isnan(v) )
        return(BIGNUM);                 /* signify calc. out of bounds -num. error *,

      /* got a fair way through calc. if i-1 > j */
      /* guess that problem is noise in disp. tdxr */
      else if(k < 0 && row_num-1 > rs_param.last_row - 10)
        return(error);                  /* no error message, just ignore last pts */
```



```
        else                                /* some other problem */
          {
            sprintf(msg, "\nProblem in rate/state calc: missed points in error calc
  red the wrong ones \n");
            print_msg(msg);
            sprintf(msg, "last row for mu comparison = %d, disp. = %g\n", row_
            print_msg(msg);
            return(-3);             /* missed points in error calc -or compared th
  */
          }
      }
    else
      return(error);                /* cumulative error -for simplex */

  }
```



```
#include "global.h"
#define TWOPI 6.283185308
#define PI 3.141592654
#define SRTWO 1.414213562

double power1(x,A,B)    /* y = pow(x,A) + B */
double x,A,B;
{
        double value;
        value = pow(x,A) + B ;
        return(value);
}

double power2(x,A,B,C)  /* y = A *pow(x,B) + C */
double x,A,B,C;
{
        double value;
        value = A * pow(x,B) + C ;
        return(value);
}

double normal(x,mean,sigma)
double x,mean,sigma;
{
        double value, temp;
        temp = pow((x-mean)/(sigma),2.0) / 2.0 ;
        value = exp(-temp) / sqrt(TWOPI) / sigma ;
        return(value);
}

double chisqr(x,ndf)
double x, ndf;
{
        double value, temp, gam;
        double gamma();
        extern int signgam;
        gam = gamma(ndf/2.0);
        gam = exp(gam);
        temp = pow(2.0,ndf/2.0) * gam ;
        value = pow(x,(ndf-2.0)/2.0) * exp(-x/2.0) / temp ;
        return(value);
}

double scchisqr(x,sigma,ndf,offset)
double x, sigma, ndf, offset;
{
        double value, temp, gam, sigx;
        double gamma();
        sigx = sigma/(sqrt(ndf)*SRTWO); sigx = sqrt(sigx);
        gam = gamma(ndf/2.0); gam = exp(gam);
        temp = pow(sigx,ndf) * pow(2.0,ndf/2.0) * gam ;
        if(offset>=x) value = 0.0;
        else value = pow((x-offset),(ndf-2.0)/2.0) * exp((offset-x)/(2.0*sigx*sigx)) /
p ;
        return(value);
}

double rclow(x,A,B)             /*      y = [1/(x*B)]*sqrt[1+x*x*B*B*A*A]       */
double x,A,B;
{
        double value;
        value = (1.0/(x*B))*sqrt((1.0+pow((x*B*A),2.0)));
        return(value);
}
```



```
double rcpb(x,A,B)              /*      y = -1.0 * atan((1.0/(x*A*B)))  */
double x,A,B;
{
        double value;
        value = -1.0 * atan((1.0/(x*A*B)));
        return(value);
}

double genexp(x,A,B,C,D)                /*      y = A * exp(B*x+C) + D  */
double x,A,B,C,D;
{
        double value;
        value = A * exp(B*x+C) + D;
        return(value);
}

double gensin(x,A,B,C,D)                /*      y = A * sin(B*x+C) + D */
double x,A,B,C,D;
{
        double value;
        value = A * sin(B*x+C) + D;
        return(value);
}

double ExpLin(x,A,B,C)          /*      y = A * (1.0 - exp(B*x)) + C*x */
double x,A,B,C;
{
        double value;
        value = A * (1.0 - exp(B*x)) + C * x;
        return(value);
}

double Poly4(x,A,B,C,D,E)       /* y = A + B*x + C*pow(x,2) + D*pow(x,3) + E*pow(x,4) */
double x,A,B,C,D,E;
{
        double value;
        value = A + B*x + C*pow(x,2.0) + D*pow(x,3.0) + E*pow(x,4.0);
        return value;
}
```

```
#include "global.h"
extern int action;
extern char msg[MSG_LENGTH];
int name_action;


/* all the funcs that used to be at the end of lookv3.c */
/* ---and more...*/

/*-------------------------------------------------------------------------*/
#define OUT 0
#define IN 1

int token_count(buf)
    char buf[];
{

  int state, n, i;

  state=OUT; /* count the number of tokens in buf */
  for(n=0,i=0; buf[i] != '\0'; ++i)
    {
      if(buf[i] == ' ' || buf[i] == '\t' || buf[i] == ',')
        {
          state=OUT;
        }
      else if(state==OUT)
        {
          state=IN;
          ++n;
        }
    }

  return(n);
}
#undef OUT
#undef IN


/* ----------------------------------------------------------------------*/
int allocate(row,col)
    int row , col ;
{
  int i , j ;
  unsigned unbytes ;
  unbytes = (unsigned)(4*row) ;
  if ( col > MAX_COL || col <= 0 || row <= 0) return 0 ;
/* row >= MAX_ROW */
  if ( col >= max_col || row >= max_row)
    {
      for( i=0; i<max_col; ++i)
        {
          darray[i] = (float *)realloc((char *)darray[i],unbytes) ;
          if( row>max_row-1 )
            {
              for (j=max_row; j<row; ++j)
                {
                  darray[i][j] = 0.0 ;
                }
            }
        }
      for( i=max_col; i<col; ++i )
        {
          darray[i] = (float *)calloc((unsigned)row,(unsigned)4) ;
```



```
        }
      }
    if ( row<head.nrec )
      {
        head.nrec = row ;
        for( j=1; j < max_col; ++j )
          {
            if( check_col(j) > 0 ) head.ch[j].nelem = row ;
          }
      }

    max_col = (col > max_col) ? col : max_col ;
    max_row = (row > max_row) ? row : max_row ;

    /* for plotting */
    arrayx = (float *)realloc((char *)arrayx,(unsigned)(max_row * 4)) ;
    arrayy = (float *)realloc((char *)arrayy,(unsigned)(max_row * 4)) ;
    return 1 ;
}

/*-------------------------------------------------------------------------*/

void null_col(col)
    int col ;
{
   strcpy(&(head.ch[col].name[0]),"no_val") ;
   strcpy(&(head.ch[col].units[0]),"no_val") ;
   strcpy(&(head.ch[col].comment[0]),"none") ;
   head.ch[col].nelem = 0 ;
   head.ch[col].gain = 0 ;
}

/*-------------------------------------------------------------------------*/
int mu_check(mu,vstep)
    /*vstep is true if mu is mu_initial and false if mu is mu_final*/
    double    *mu;
    int       vstep;
{

  int i;
  int mu_window_start;
  int mu_row;
  int test;

  test = TRUE;

  if(*mu < 0)
    {
      /*take 10 points before the vstep or 5 points on either side of the last
      mu_window_start = (vstep == TRUE) ? (rs_param.vs_row-9) : (rs_param.last_row-5
      mu_row   = (vstep == TRUE) ? rs_param.vs_row : rs_param.last_row;

      if(*mu == -22) /* avg. of 10 pts */
        {
          *mu = 0.0;
          for(i=mu_window_start; i <= mu_window_start+10; ++i)
            *mu += darray[rs_param.mu_col][i];
          *mu /= 10.0;
        }
      if(*mu == -33) /* val at row < vs_row -> do this to allow for emergent
t effect*/
        {
          *mu = darray[rs_param.mu_col][mu_row-1];
          test=FALSE;
```



```
        }
      else /* just take vs_row val */
        *mu = darray[rs_param.mu_col][mu_row];

      sprintf(msg,"\tauto mu set: mu%c = %f\n", (vstep == TRUE) ? 'o' : 'f', *m
      print_msg(msg);
    }

  return(test);
}


/*-------------------------------------------------------------------------*/
int check_row(first_row, last_row, this_col)
    int *first_row, *last_row, this_col;
{
  int test;

  test = 0;

  if( strncmp(head.ch[this_col].name,"no_val",6) ==0 )
    {
      sprintf(msg, "col (%d) not defined\n",this_col);
      print_msg(msg);
      test++;
    }

  /* use -1 as variable for nrec */
  if(*last_row == -1 || *last_row == head.ch[this_col].nelem)
    *last_row = head.ch[this_col].nelem-1;

  else if(*last_row >= head.ch[this_col].nelem)
    {
      sprintf(msg, "last row (%d) > nelems for col (%d)\n", *last_row, this_col);
      print_msg(msg);
      test++;
    }

  if(*first_row < 0)
    {
      sprintf(msg, "first row (%d) can't be less than 0\n", *first_row);
      print_msg(msg);
      test++;
    }

  if(*last_row < *first_row)
    {
      sprintf(msg, "last row (%d) can't be less than first row (%d)\n", *last_row,
row);
      print_msg(msg);
      test++;
    }
  return test;

}


/*-------------------------------------------------------------------------*/
int check_col(col)
    int col ;
{
  int test ;

  test = 0 ;
```



```
  test += strncmp(head.ch[col].name,"no_val",6) ;
  test += strncmp(head.ch[col].units,"no_val",6) ;
  test += strncmp(head.ch[col].comment,"none",6) ;
  if( head.ch[col].gain != 0 ) test += 1 ;
  if( head.ch[col].gain != 0 ) test += 1 ;
  return test ;
}


/*-------------------------------------------------------------------------*/
act_col()
{
  int numb=0 ;
  int i ;
  for( i=1; i<MAX_COL; ++i)
    {
      if(check_col(i) > 0) numb = i ;
    }
  return numb ;
}


/*-------------------------------------------------------------------------*/
review(file)
    FILE *file ;
{
  int i ;
  if (file == stderr)
    fprintf(file,"ALLOCATION: max_col = %d , max_row = %d\t",max_col,max_row);
  fprintf(file,"number of records = %d\n ",head.nrec) ;
  for( i = 1; i < max_col ; ++i )
    {
      if ( strncmp(&(head.ch[i].name[0]),"no_val",6) != 0)
        {
          fprintf(file," col %ld",i) ;
        }
    }
  fprintf(file,"\n ") ;
  for( i = 1; i < max_col ; ++i )
    {
      if ( strncmp(&(head.ch[i].name[0]),"no_val",6) != 0)
        {
          fprintf(file,"%12s",head.ch[i].name) ;
        }
    }
  fprintf(file,"\n ") ;
  for( i = 1; i < max_col ; ++i )
    {
      if ( strncmp(&(head.ch[i].name[0]),"no_val",6) != 0)
        {
          fprintf(file,"%12s",head.ch[i].units) ;
        }
    }
  fprintf(file,"\n ");
  for( i = 1; i < max_col ; ++i )
    {
      if ( strncmp(&(head.ch[i].name[0]),"no_val",6) != 0)
        {
          fprintf(file,"%7d recs",head.ch[i].nelem) ;
        }
    }
  fprintf(file,"\n") ;
}
```

```
                                WIN_BELOW,  main_frame_menu_panel,
                                NULL);

    cmd_panel_item = (Panel_item) xv_create(main_frame_cmd_panel, PANEL_TEXT,
                                PANEL_LABEL_STRING, "Command:  ",
                                PANEL_VALUE_STORED_LENGTH,  200,
                                PANEL_VALUE_DISPLAY_LENGTH,  80,
                                PANEL_NOTIFY_LEVEL,  PANEL_SPECIFIED,
                                PANEL_NOTIFY_STRING,  "\r",
                                PANEL_NOTIFY_PROC,  get_cmd_proc,
                                NULL);


  /* ---------------------------- FILE INFO WINDOW ---------------------------- */

    fileinfo_window = (Textsw)xv_create(main_frame,  TEXTSW,
                                TEXTSW_IGNORE_LIMIT,  TEXTSW_INFINITY,
                                WIN_ROWS,  7,
                                TEXTSW_BROWSING,  TRUE,
                                TEXTSW_CONFIRM_OVERWRITE,  TRUE,
                                TEXTSW_LINE_BREAK_ACTION,  TEXTSW_WRAP_AT_WORD,
                                NULL);

    /* (void)  xv_create(fileinfo_window,  SCROLLBAR,
                    SCROLLBAR_DIRECTION,  SCROLLBAR_HORIZONTAL,
                    NULL);  */


  /* ------------------------------ INFO PANEL ---------------------------- */

    main_frame_info_panel = (Panel)xv_create(main_frame,  PANEL,  XV_HEIGHT,  35,  NULL);

    active_win_info = (Panel_item)xv_create(main_frame_info_panel,  PANEL_MESSAGE,
                                        PANEL_NEXT_ROW,  -1,
                                        PANEL_LABEL_STRING,  "Active  Window:  ",
                                        NULL);

    active_plot_info = (Panel_item)xv_create(main_frame_info_panel,  PANEL_MESSAGE,
                                        XV_X,  xv_col(main_frame_info_panel,  20),
                                        PANEL_LABEL_STRING,  "Active  Plot:  ",
                                        NULL);


  /* ------------------------------ MESSAGE WINDOW ----------------------------*/

    msgwindow = (Textsw)xv_create(main_frame,  TEXTSW,
                                TEXTSW_IGNORE_LIMIT,  TEXTSW_INFINITY,
                                WIN_ROWS,  20,
                                TEXTSW_BROWSING,  TRUE,
                                TEXTSW_LINE_BREAK_ACTION,  TEXTSW_WRAP_AT_WORD,
                                TEXTSW_MEMORY_MAXIMUM,  200000,
                                NULL);

    /* (void)  xv_create(msgwindow,  SCROLLBAR,
                    SCROLLBAR_DIRECTION,  SCROLLBAR_HORIZONTAL,
                    NULL);  */

    (void)textsw_possibly_normalize(msgwindow,  (Textsw_index)xv_get(msgwindow,  TEXTSW_INSERT
ION_POINT));

    window_fit(msgwindow);
```



```
  /* ---------------- CMD_HISTORY  FRAME  SETUP ------------------ */

    cmd_hist_panel = (Panel)xv_create(cmd_hist_frame,  PANEL,
                                PANEL_LAYOUT,  PANEL_VERTICAL,
                                NULL);

    (void)  xv_create(cmd_hist_panel,  PANEL_BUTTON,
                    PANEL_LABEL_STRING,  "CLOSE",
                    PANEL_NOTIFY_PROC,  close_hist_proc,
                    NULL);

    cmd_hist_panel_list = (Panel_item)  xv_create(cmd_hist_panel,  PANEL_LIST,
                                        PANEL_LIST_DISPLAY_ROWS,  10,
                                        PANEL_LIST_MODE,  PANEL_LIST_READ,
                                        PANEL_LIST_WIDTH,  -1,
                                        PANEL_NOTIFY_PROC,  cmd_hist_proc,
                                        NULL);
    window_fit(cmd_hist_panel);
    window_fit(cmd_hist_frame);

    xhair_cursor = (Xv_Cursor)xv_create(NULL,  CURSOR,
                                        CURSOR_SRC_CHAR,  22,
                                        NULL);

  /*---------------- PLOTTING WINDOW SETUP ---------------------- */
   /*  create_canvas();  */

   /*  create  font  for  title  */
    titlefont=(Xv_Font)xv_find(main_frame,  FONT,
                            /*  FONT_NAME,  "-adobe-courier-medium-r-normal--14-140-75   -m
-90-iso8859-1",          */
                            FONT_FAMILY,  FONT_FAMILY_LUCIDA,
                            FONT_STYLE,  FONT_STYLE_NORMAL,
                            FONT_SIZE,  14,
                            NULL);

    /*  create  font  for  label  */
    tickfont=(Xv_Font)xv_find(main_frame,  FONT,
                            /*FONT_RESCALE_OF,  titlefont,  WIN_SCALE_SMALL,*/
                            FONT_FAMILY,  FONT_FAMILY_LUCIDA,
                            FONT_STYLE,  FONT_STYLE_NORMAL,
                            FONT_SIZE,  12,
                            NULL);

    dpy = (Display *)xv_get(main_frame,  XV_DISPLAY);

    /*  allocate  color  map  for  background  and  foreground  colors  */
    cmap = DefaultColormap(dpy,  DefaultScreen(dpy));
    if  (XAllocNamedColor  (dpy,  cmap,  bgcolor,  &bgc,  &bgc)  ==  0);
    if  (XAllocNamedColor  (dpy,  cmap,  fgcolor,  &fgc,  &fgc)  ==  0);

    /*  create  gc  for  title  font  */
    gcvalues.foreground  =  fgc.pixel;
    gcvalues.background  =  bgc.pixel;
    gcvalues.font  =  (Font)xv_get(titlefont,  XV_XID);
    gcvalues.graphics_exposures  =  False;
    gctitle  =  XCreateGC(dpy,  RootWindow(dpy,  DefaultScreen(dpy)),  GCFont  |  GCGraph   bq
es  |  GCForeground  |  GCBackground,  &gcvalues);

   /*  create  gc  for  tick  font  */
    gcvalues.foreground  =  fgc.pixel;
    gcvalues.background  =  bgc.pixel;
    gcvalues.font  =  (Font)xv_get(tickfont,  XV_XID);
```



```
    gcvalues.graphics_exposures  =  False;
    gctick  =  XCreateGC(dpy,  RootWindow(dpy,  DefaultScreen(dpy)),  GCFont  |  GCGraphicsExpo
s  |  GCForeground  |  GCBackground,  &gcvalues);

    /*  create  gc  for  rubber-band-like  lines  */
    gcvalues.foreground  =  fgc.pixel;
    gcvalues.background  =  bgc.pixel;
    gcvalues.graphics_exposures  =  False;
    gcrubber  =  XCreateGC(dpy,  RootWindow(dpy,  DefaultScreen(dpy)),  GCGraphicsExposures  |
oreground  |  GCBackground,  &gcvalues);
    XSetFunction(dpy,  gcrubber,  GXxor);

    gcvalues.foreground  =  bgc.pixel;
    gcbg  =  XCreateGC(dpy,  RootWindow(dpy,  DefaultScreen(dpy)),  GCForeground,  &gcvalues);


   /*  attach  gcs  to  canvas
    xv_set(canvas,  XV_KEY_DATA,  GC_TICK,  gctick,  NULL);
    xv_set(canvas,  XV_KEY_DATA,  GC_TITLE,  gctitle,  NULL);  */

    titlefontheight=(int)xv_get(titlefont,  FONT_DEFAULT_CHAR_HEIGHT);
    titlefontwidth=(int)xv_get(titlefont,  FONT_DEFAULT_CHAR_WIDTH);

    tickfontheight=(int)xv_get(tickfont,  FONT_DEFAULT_CHAR_HEIGHT);
    tickfontwidth=(int)xv_get(tickfont,  FONT_DEFAULT_CHAR_WIDTH);

  /* ------------------------------------------------------------------------ */

    window_fit(main_frame);

    display_active_window(0);
    display_active_plot(0);

    /*  get  the  maximum  memory  limit  for  the  message  subwindow  */
    textsw_memory_limit  =  (int)xv_get(msgwindow,  TEXTSW_MEMORY_MAXIMUM);


    xv_main_loop(main_frame);

  }


  quit_xlook()
  {
    extern  Frame  clr_plots_frame;

    if  (clr_plots_frame)  xv_destroy_safe(clr_plots_frame);
    xv_destroy_safe(main_frame);
  }




  /*  ------------------------  FILE  MENU  PROCS  ------------------------------  */
  /*
   not  used  anymore.  use  the  quit  button  instead.

  void  exit_file_proc(menu,  item)
       Menu  menu;
       Menu_item  item;
  {
    quit_xlook();
```



```
  }
  */


  void  read_file_proc(menu,  item)
       Menu  menu;
       Menu_item  item;
  {
    set_left_footer("Type  the  data  file  to  read");
    action  =  READ;
    set_cmd_prompt("Filename:  ");
  }

  void  write_file_proc(menu,  item)
       Menu  menu;
       Menu_item  item;
  {
     set_left_footer("Type  the  file  to  write");
    action  =  WRITE;
    set_cmd_prompt("Filename:  ");
  }

  void  append_file_proc(menu,  item)
       Menu  menu;
       Menu_item  item;
  {
    set_left_footer("Type  the  file  to  append");
    action  =  APPEND;
    set_cmd_prompt("Filename:  ");
  }


  /*  ------------------------  WINDOW  MENU  PROCS  ---------------------------  */
  void  act_window_proc(menu,  item)
       Menu  menu;
       Menu_item  item;
  {
    set_left_footer("Which  window  to  be  set  active?");
    set_cmd_prompt("Window  Number:  ");
    action  =  SET_ACTIVE_WINDOW;
  }

  void  kill_window_proc(menu,  item)
       Menu  menu;
       Menu_item  item;
  {
    set_left_footer("Type  the  window  number  to  destroy");
    set_cmd_prompt("Window  Number:  ");
    action  =  KILL_WIN;
  }

  void  new_window_proc(menu,  item)
       Menu  menu;
       Menu_item  item;
  {
    new_win_proc();
  }



  /*  ------------------------  PLOT  COMMANDS  MENU  ---------------------------  */
  void  plotl_proc(menu,  item)
       Menu  menu;
       Menu_item  item;
```

```
{
  void goto_plot1_proc();

  strcpy(plot_cmd, "plotall");
  goto_plot1_proc();
}

void plot2_proc(menu, item)
     Menu menu;
     Menu_item item;
{
 void goto_plot1_proc();

 strcpy(plot_cmd, "plotover");
 goto_plot1_proc();
}

void plot3_proc(menu, item)
     Menu menu;
     Menu_item item;
{
  void goto_plot1_proc();

  strcpy(plot_cmd, "plotsr");
  goto_plot1_proc();
}

void plot4_proc(menu, item)
     Menu menu;
     Menu_item item;
{
  void goto_plot2_proc();

  strcpy(plot_cmd, "plotauto");
  goto_plot2_proc();
}

void plot5_proc(menu, item)
     Menu menu;
     Menu_item item;
{
  void goto_plot2_proc();

  strcpy(plot_cmd, "plotlog");
  goto_plot2_proc();
}

void plot6_proc(menu, item)
     Menu menu;
     Menu_item item;
{
  void goto_plot2_proc();

  strcpy(plot_cmd, "plotsame");
  goto_plot2_proc();
}

void plot7_proc(menu, item)
     Menu menu;
     Menu_item item;
{
  void goto_plot2_proc();

  strcpy(plot_cmd, "plotscale");
```



```
  goto_plot2_proc();
}

void plot8_proc(menu, item)
     Menu menu;
     Menu_item item;
{
  void goto_plot2_proc();

  strcpy(plot_cmd, "ps");
  goto_plot2_proc();
}


void goto_plot1_proc()
{
   set_left_footer("Type the x-axis and y-axis");
   set_cmd_prompt("X-AXIS Y-AXIS: ");
   action = PLOT_GET_XY;
 }

void goto_plot2_proc()
{
   set_left_footer("Type the x-axis and y-axis");
   set_cmd_prompt("X-AXIS Y-AXIS: ");
   action = PLOT_GET_BE;
 }


/* ---------------------- BUTTONS PROCEDURES --------------------------- */

void exit_proc(item, event)
     Panel_item item;
     Event *event;
{
  quit_xlook();
}


void get_cmd_proc(item, event)
     Panel_item item;
     Event *event;
{
  /* command panel callback function:
     passes a copy of the string to command_handler() */

  static int row_num=0;
  extern void command_handler();
  char cmd_text[128];

  strcpy(cmd_text, (char *)xv_get(item, PANEL_VALUE));

  xv_set(cmd_hist_panel_list, PANEL_LIST_INSERT, row_num, PANEL_LIST_STRING, row_num, c
text, NULL);
  row_num++;

  xv_set(item, PANEL_VALUE, "", NULL);

  /* call cmd multiplexor */
  command_handler(cmd_text);
}


void cmd_hist_proc(item, cmd_text, client_data, op, event, row)
```



```
     Panel_item item;
     char *cmd_text;
     Xv_opaque client_data;
     Panel_list_op op;
     Event *event;
     int row;
{
  /* copies every input to the history command window;
   don't exactly know how big this buffer can be */

  xv_set(cmd_panel_item, PANEL_VALUE, cmd_text, NULL);

}


void show_hist_proc(item, event)
     Frame item;
     Event *event;
{
  /* maps the history window to screen */

  Frame cmd_hist_frame = (Frame)xv_get(item, PANEL_CLIENT_DATA);
  if ((int)xv_get(cmd_hist_frame, XV_SHOW)== FALSE)
    xv_set(cmd_hist_frame, XV_SHOW, TRUE, NULL);
}


void close_hist_proc(item, event)
     Frame item;
     Event *event;
{
  /* unmaps the history window from screen */

  Frame graf_frame = (Frame)xv_get(item, PANEL_CLIENT_DATA);
  if ((int)xv_get(cmd_hist_frame, XV_SHOW) == TRUE)
    xv_set(cmd_hist_frame, XV_SHOW, FALSE, NULL);
}



/* --------------------------- CANVAS HANDLER ---------------------------- */

/*
void close_graf_proc(item, event)
     Frame item;
     Event *event;
{
  Frame graf_frame;

  graf_frame = xv_get(wininfo.canvases[active_window]->canvas, XV_KEY_DATA, GRAF_FRAME);
  active_window = old_active_window;
  if (old_active_window != 0)
    old_active_window--;

  if ((int)xv_get(graf_frame, XV_SHOW) == TRUE)
    xv_set(graf_frame, XV_SHOW, FALSE, NULL);
}
*/

void clear_win_proc(item, event)
     Panel_item item;
     Event *event;
{
```



```
  canvasinfo *can_info;
  int can_num, i;

  can_num = xv_get(item, PANEL_CLIENT_DATA);
  set_active_window(can_num);
  can_info = wininfo.canvases[active_window];
  display_active_plot(-1);

  clr_all();
}

clr_all()
{
  canvasinfo *can_info;
  int i;

  can_info = wininfo.canvases[active_window];

  for (i=0; i<10; i++)
    {
      if (can_info->alive_plots[i] == 1)
        {
          free(can_info->plots[i]);
          can_info->alive_plots[i] = 0;
        }
    }
  can_info->active_plot = -1;
  can_info->total_plots = -1;

  clear_canvas_proc(can_info->canvas);
}


void refresh_win_proc(item, event)
     Panel_item item;
     Event *event;
{
  canvasinfo *can_info;
  int can_num, i, active_plot;
  Canvas canvas;

  can_num = xv_get(item, PANEL_CLIENT_DATA);
  set_active_window(can_num);
  can_info = wininfo.canvases[active_window];
  active_plot = can_info->active_plot;
  display_active_plot(active_plot+1);
  canvas = can_info->canvas;

  redraw_all_proc(canvas, can_info->xvwin, xv_get(canvas, XV_DISPLAY), can_info->win,
);
}


void kill_graf_proc(item, event)
     Panel_item item;
     Event *event;
{
  int win;

  win = xv_get(item, PANEL_CLIENT_DATA);

  kill_win_proc(win+1);
}
```

```
void  canvas_event_proc(xvwindow,  event)
    Xv_Window  xvwindow;
    Event  *event;
{
  /*  canvas  event  handler:
     always  get  the  window  number  first  and  set  active  window  to  this  num.
     if  mouse  event,  check  the  current  mode,  then  execute  the  proc.
     */

  int  xloc,  yloc;
  float  yval,  xval;
  int  row_num,  nrows;
  char  xstring[20],  ystring[20],  rowstring[20];
  Canvas  canvas;
  int  can_num;
  canvasinfo  *can_info;
  plotarray  *data;
  int  active_plot;
  int  start_x,  start_y,  end_x,  end_y;
  float  scale_x,  scale_y;
  float  xmin,  ymin;
  int  i;
  extern  void  do_line_plot(),  do_mouse_mu(),  do_dist();


  canvas  =  (Canvas)xv_get(xvwindow,  CANVAS_PAINT_CANVAS_WINDOW);
  can_num  =  xv_get(canvas,  XV_KEY_DATA,  CAN_NUM);
  set_active_window(can_num);

  can_info  =  wininfo.canvases[active_window];
  active_plot  =  can_info->active_plot;
  display_active_plot(active_plot+1);

  switch(event_action(event))
   {

   case  LOC_MOVE:

     if  (can_info->total_plots  <  0)  break;
     data  =  can_info->plots[active_plot];

     xloc  =  event_x(event);
     yloc  =  event_y(event);

     /*  draw  rubber  band  line  for  line  plot  */
     if  (data->mouse  ==  1  &&  data->p1  ==  1)
       /*  &&  (xloc  >  can_info->start_x  &&  xloc  <  can_info->end_x  &&  yloc  <  can_i
rt_y  &&  yloc  >  can_info->end_y))  */
       {
         if  (data->p2  ==  1)
           {
             /*  remove  previous  line  */
             XDrawLine((Display  *)xv_get(canvas,  XV_DISPLAY)  ,  can_info->win,  gcrubber,
ata->x1,  data->y1,  data->x2,  data->y2);
           }
         /*  draw  new  line  */
         XDrawLine((Display  *)xv_get(canvas,  XV_DISPLAY)  ,  can_info->win,  gcrubber,  d
>x1,  data->y1,  xloc,  yloc);
         data->x2  =  xloc;
         data->y2  =  yloc;
         data->p2  =  1;
       }
```



```
       break;


     case  LOC_DRAG:
       break;


     case  ACTION_MENU:  /*  right  button:  done  selecting  parameters.  commit!  */
       if  (!event_is_up(event))  break;
       if  (can_info->total_plots  <  0)  break;

       data  =  can_info->plots[active_plot];
       xloc  =  event_x(event);
       yloc  =  event_y(event);

       /*  default  case  (mouse  ==  0)  */
       if  (data->mouse  ==  0  &&  (xloc  >  can_info->start_x  &&  xloc  <  can_info->end_x
< can_info->start_y  &&  yloc  >  can_info->end_y))
         {
           print_xy(xloc,  yloc);
         }

       if  (data->mouse  ==  4  &&  (xloc  >  can_info->start_x  &&  xloc  <  can_info->end_x
< can_info->start_y  &&  yloc  >  can_info->end_y))
         {
           /*  commit  zoom  */
           zoom();
           data->zp1  =  data->zp2  =  0;
         }

       if  (data->mouse  !=  0)
         {
           sprintf(msg,  "Done.\n");
           print_msg(msg);
           data->x1  =  0;
           data->y1  =  0;
           data->x2  =  0;
           data->y2  =  0;
           data->p1  =  0;
           data->p2  =  0;
           data->zp1  =  0;
           data->zp2  =  0;
           data->mouse  =  0;
         }

       break;


     case  ACTION_ADJUST:  /*  middle  button  */
       if  (!event_is_up(event))  break;
       if  (can_info->total_plots  <  0)  break;

       xloc  =  event_x(event);
       yloc  =  event_y(event);

       /*  if  (xloc  <  can_info->start_x  ||  xloc  >  can_info->end_x  ||  yloc  >  can_in
tart_y  ||  yloc  <  can_info->end_y)  break;  */

       data  =  can_info->plots[active_plot];


       if  (data->mouse  ==  0  &&  (xloc  >  can_info->start_x  &&  xloc  <  can_info->end_x
< can_info->start_y  &&  yloc  >  can_info->end_y))
```



```
         {
           print_xyrow(xloc,  yloc,  1);
           break;
         }

       if  ((data->mouse  ==  1  ||  data->mouse  ==  3  ||  data->mouse  ==  4)  &&  data->p1
         {
           /*  not  enough  points  picked  */
           sprintf(msg,  "Pick  1st  point  with  left  button  first.\n");
           print_msg(msg);
           break;
         }

       if  (data->mouse  ==  1  &&  data->p1  ==  1)
         {
           /*  get  the  2nd  point  */
           sprintf(msg,  "2nd  point  picked\n");
           print_msg(msg);
           data->x2  =  xloc;
           data->y2  =  yloc;
           data->p2  =  1;

           /*  commit  line  plot  */
           draw_crosshair(xloc,  yloc);
           XDrawLine((Display  *)xv_get(canvas,  XV_DISPLAY)  ,  can_info->win,  gcrubber,  d
>x1,  data->y1,  data->x2,  data->y2);
           do_line_plot();

           /*  reset  for  next  line  plot  */
           data->p1  =  0;
           data->p2  =  0;
           data->x1  =  data->x2  =  data->y1  =  data->y2  =  0;
           break;
         }

       if  (data->mouse  ==  2)
         {
           /*  do  nothing  */
           break;
         }

       if  (data->mouse  ==  3  &&  (xloc  >  can_info->start_x  &&  xloc  <  can_info->end_x
< can_info->start_y  &&  yloc  >  can_info->end_y))
         {
           /*  get  the  2nd  point  */
           sprintf(msg,  "2nd  point  picked\n");
           print_msg(msg);
           data->x2  =  xloc;
           data->y2  =  yloc;
           data->p2  =  1;

           /*  commit  distance  */
           draw_crosshair(xloc,  yloc);
           do_dist();
           data->p1  =  0;
           data->p2  =  0;
           data->x1  =  data->x2  =  data->y1  =  data->y2  =  0;
           break;
         }

       if  (data->mouse  ==  4  &&  (xloc  >  can_info->start_x  &&  xloc  <  can_info->end_x
< can_info->start_y  &&  yloc  >  can_info->end_y))
         {
           /*  get  the  2nd  point  */
```



```
           sprintf(msg,  "2nd  point  picked\n");
           print_msg(msg);
           data->x2  =  xloc;
           data->y2  =  yloc;
           data->p2  =  1;

           /*  2nd  corner  for  zoom  */
           zoom_get_pt(xloc,  yloc,  2);
           break;
         }

       break;



     case  ACTION_SELECT:  /*  left  button  */
       if  (!event_is_up(event))  break;
       if  (can_info->total_plots  <  0)  break;

       data  =  can_info->plots[active_plot];

       xloc  =  event_x(event);
       yloc  =  event_y(event);


       /*  if  (xloc  <  can_info->start_x  ||  xloc  >  can_info->end_x  ||  yloc  >  c
y  ||  yloc  <  can_info->end_y)  break;  */

       if  (data->mouse  ==  0  &&  (xloc  >  can_info->start_x  &&  xloc  <  can_info->end_x
< can_info->start_y  &&  yloc  >  can_info->end_y))
         {
           /*  print  row  num  on  info  panel  only  */
           print_xyrow(xloc,  yloc,  0);
           break;
         }


       /*  else,  get  the  first  point  */

       if  (data->mouse  !=  0)
         {

           if  (data->mouse  ==  1  &&  data->p2  ==  1  &&  data->p1  ==  1)
             {
               /*  do  nothing  */
               break;
             }

           data->x1  =  xloc;
           data->y1  =  yloc;
           data->p1  =  1;

           if  (data->mouse  ==  2  &&  (xloc  >  can_info->start_x  &&  xloc  <  can_in
loc  <  can_info->start_y  &&  yloc  >  can_info->end_y))
             {
               do_mouse_mu();
               break;
             }

           if  (data->mouse  ==  4  &&  (xloc  >  can_info->start_x  &&  xloc  <  can_in
loc  <  can_info->start_y  &&  yloc  >  can_info->end_y))
             {
               zoom_get_pt(xloc,  yloc,  1);
               break;
```

```
        }

        draw_crosshair(xloc, yloc);
        /* get the start point */
        sprintf(msg, "1st point picked\n");
        print_msg(msg);

      }
     break;

  /*
    case WIN_REPAINT:
     redraw_all_proc(canvas, xvwindow, (Display *)xv_get(canvas, XV_DISPLAY), can_info->
n, NULL);
     break; */


    case WIN_RESIZE:
      setup_canvas();
      redraw_all_proc(canvas, xvwindow, (Display *)xv_get(canvas, XV_DISPLAY), can_info-
in, NULL);
     /* if (can_info->total_plots == -1) break;
        XClearWindow((Display *)xv_get(canvas, XV_DISPLAY),
        can_info->win);

        can_info->total_plots = -1;
        for (i=0; i<10; i++)
        {
        if (can_info->alive_plots[i] == 1)
        {
        can_info->active_plot = i;
        can_info->total_plots++;
        redraw_proc(canvas, xvwindow, (Display *)xv_get(canvas, XV_DISPLAY), can_info-
in, NULL);
        }
        }
        */
     break; /* if set to return, need to click on the canvas to display */

    default:
      return;
    }
}



/* probably useless.. since already handled by canvas_event_proc()
   void resize_proc(canvas, width, height)
   Canvas canvas;
   int width;
   int height;
{

  Display *dpy = (Display *)xv_get(graf_frame, XV_DISPLAY);
  Xv_Window xvwin = (Xv_Window)canvas_paint_window(canvas);
  Window win = (Window)xv_get(xvwin, XV_XID);

  XClearWindow(dpy, win);
  redraw_proc(canvas, xvwin, dpy, win, NULL);

}
*/
```



```
initialize(argc, argv)
     int argc;
     char *argv[];
{
  int i,j;

  plot_error = -1;

  for (i = 0; i < 12; ++i)
    darray[i] = (float *)calloc((unsigned)10000, (unsigned)4);

  arrayx = (float *)calloc((unsigned)10000, (unsigned)4);
  arrayy = (float *)calloc((unsigned)10000, (unsigned)4);
  max_col = 12;
  max_row = 10000;

  for (i=0; i<17; ++i) null_col[i];

   if (argc > 1)
    {
      if (argc == 2)
        {
          if (access(argv[1], 4) == 0)
            {
              strcpy(default_path, argv[1]);
              fprintf(stderr, "Pathname for DOIT files is %s\n", default_path);
              if (default_path[strlen(default_path)-1] != '/')
                strcat(default_path, "/");
            }
          else
            fprintf(stderr, "Inaccessible pathname \n%s IGNORED\n", argv[1]);
        }
      else
        {
          fprintf(stderr, "Don't understand argument list\n");
          fprintf(stderr, "EXPECTED: look default_path\n");
          exit(-1);
        }
    }
}
```



```
void adlanl(x,n,xmed)
float x[],*xmed;
int n;
{
        int n2,n2p;
        void sort();

        sort(n,x);
        n2p=(n2=n/2)+1;
        *xmed=(n % 2 ? x[n2p] : 0.5*(x[n2]+x[n2p]));
}
```



```
#include <math.h>

float evlmem(fdt,cof,m,pm)
float fdt,cof[],pm;
int m;
{
        int i;
        float sumr=1.0,sumi=0.0;
        double wr=1.0,wi=0.0,wpr,wpi,wtemp,theta;

        theta=6.28318530717959*fdt;
        wpr=cos(theta);
        wpi=sin(theta);
        for (i=1;i<=m;i++) {
                wr=(wtemp=wr)*wpr-wi*wpi;
                wi=wi*wpr+wtemp*wpi;
                sumr -= cof[i]*wr;
                sumi -= cof[i]*wi;
        }
        return pm/(sumr*sumr+sumi*sumi);
}

/*************************************/


static float sqrarg;
#define SQR(a) (sqrarg=(a),sqrarg*sqrarg)

void memcof(data,n,m,pm,cof)
float data[],*pm,cof[];
int n,m;
{
        int k,j,i;
        float p=0.0,*wk1,*wk2,*wkm,*vector();
        void free_vector();

        wk1=vector(1,n);
        wk2=vector(1,n);
        wkm=vector(1,m);
        for (j=1;j<=n;j++) p += SQR(data[j]);
        *pm=p/n;
        wk1[1]=data[1];
        wk2[n-1]=data[n];
        for (j=2;j<=n-1;j++) {
                wk1[j]=data[j];
                wk2[j-1]=data[j];
        }
        for (k=1;k<=m;k++) {
                float num=0.0,denom=0.0;
                for (j=1;j<=(n-k);j++) {
                        num += wk1[j]*wk2[j];
                        denom += SQR(wk1[j])+SQR(wk2[j]);
                }
                cof[k]=2.0*num/denom;
                *pm *= (1.0-SQR(cof[k]));
                if (k != 1)
                        for (i=1;i<=(k-1);i++)
                                cof[i]=wkm[i]-cof[k]*wkm[k-i];
                if (k == m) {
                        free_vector(wkm,1,m);
                        free_vector(wk2,1,n);
                        free_vector(wk1,1,n);
                        return;
                }
```

```
            for  (i=1;i<=k;i++)  wkm[i]=cof[i];
            for  (j=1;j<=(n-k-1);j++)  {
                 wk1[j]  -=  wkm[k]*wk2[j];
                 wk2[j]=wk2[j+1]-wkm[k]*wk1[j+1];
            }
        }
}

#undef  SQR
```



```
#include  <math.h>
#include  <X11/Xlib.h>
#include  <xview/xview.h>
#include  <xview/canvas.h>
#include  <xview/panel.h>
#include  "string.h"

#include  "global.h"

extern  int  active_window;
extern  int  old_active_window;
extern  int  total_windows;
extern  int  action;

/*  extern  void  print_msg();  */
extern  char  msg[MSG_LENGTH];

extern  Frame  main_frame;
Frame  clr_plots_frame;
static  int  chosen[10];
static  Panel_item  p0,  p1,  p2,  p3,  p4,  p5,  p6,  p7,  p8,  p9;
static  Panel_item  message;
static  char  plabel[10][128];


void  clr_plots_notify_proc(item,  event)
     Panel_item  item;
     Event  *event;
{
  set_active_window(xv_get(item,  PANEL_CLIENT_DATA));
}


void  clr_all_plots_proc(menu,  menu_item)
     Menu  menu;
     Menu_item  menu_item;
{
  canvasinfo  *can_info;
  int  i;
  int  can_num;

  can_info  =  wininfo.canvases[active_window];

  for  (i=0;  i<10;  i++)
    {
      if  (can_info->alive_plots[i]  ==  1)
        {
          free  (can_info->plots[i]);
          can_info->alive_plots[i]  =  0;
        }
    }
  can_info->total_plots  =  -1;
  can_info->active_plot  =  -1;

  sprintf(msg,  "Clear  all  plots.\n");
  print_msg(msg);

  clear_canvas_proc(can_info->canvas);
  display_active_window(active_window+1);
  display_active_plot(-1);
}
```



```
void  clr_plots_proc(menu,  menu_item)
     Menu  menu;
     Menu_item  menu_item;
{

  Panel  panel;
  canvasinfo  *can_info;
  int  i;
  char  labels[10][128];
  void  clr_plots_done_proc(),  clr_plots_cancel_proc();
  int  can_num;
  char  msg_label[128];

  /*  can_num  =  xv_get(menu,  MENU_CLIENT_DATA);
  set_active_window(can_num);  */

  can_info  =  wininfo.canvases[active_window];

  sprintf(msg_label,  "Plots  in  window  #%d  to  clear:",  active_window+1);

  for  (i=0;  i<10;  i++)
    {
      if  (can_info->alive_plots[i]  ==  1)
        {
          sprintf(labels[i],  "%s  vs  %s",
                  head.ch[can_info->plots[i]->col_x].name,
                  head.ch[can_info->plots[i]->col_y].name);
        }
      else
        {
          sprintf(labels[i],  "No  Plot");
        }
    }

  if  (!clr_plots_frame)
    {
      /*  can  be  used  by  all  the  windows  */
      clr_plots_frame  =  (Frame)xv_create(main_frame,
                                           FRAME,
                                           FRAME_LABEL,  "Clear  Plots",
                                           NULL);

      panel  =  (Panel)  xv_create(clr_plots_frame,  PANEL,
                                   PANEL_LAYOUT,  PANEL_VERTICAL,
                                   NULL);

      message  =  (Panel_item)  xv_create(panel,  PANEL_MESSAGE,
                                          PANEL_LABEL_BOLD,  TRUE,
                                          PANEL_LABEL_STRING,  msg_label,
                                          XV_X,  xv_col(panel,  3),
                                          NULL);

      p0  =  (Panel_item)  xv_create(panel,  PANEL_CHECK_BOX,
                                     PANEL_CHOICE_STRINGS,  labels[0],  NULL,
                                     PANEL_VALUE_X,  xv_col(panel,  3),
                                     XV_Y,  xv_row(panel,  1),
                                     NULL);
      p1  =  (Panel_item)  xv_create(panel,  PANEL_CHECK_BOX,
                                     PANEL_CHOICE_STRINGS,  labels[1],  NULL,
                                     PANEL_VALUE_X,  xv_col(panel,  3),
                                     XV_Y,  xv_row(panel,  2),
                                     NULL);
      p2  =  (Panel_item)  xv_create(panel,  PANEL_CHECK_BOX,
                                     PANEL_CHOICE_STRINGS,  labels[2],  NULL,
```



```
                                     PANEL_VALUE_X,  xv_col(panel,  3),
                                     XV_Y,  xv_row(panel,  3),
                                     NULL);
      p3  =  (Panel_item)  xv_create(panel,  PANEL_CHECK_BOX,
                                     PANEL_CHOICE_STRINGS,  labels[3],  NULL,
                                     PANEL_VALUE_X,  xv_col(panel,  3),
                                     XV_Y,  xv_row(panel,  4),
                                     NULL);
      p4  =  (Panel_item)  xv_create(panel,  PANEL_CHECK_BOX,
                                     PANEL_CHOICE_STRINGS,  labels[4],  NULL,
                                     PANEL_VALUE_X,  xv_col(panel,  3),
                                     XV_Y,  xv_row(panel,  5),
                                     NULL);
      p5  =  (Panel_item)  xv_create(panel,  PANEL_CHECK_BOX,
                                     PANEL_CHOICE_STRINGS,  labels[5],  NULL,
                                     PANEL_VALUE_X,  xv_col(panel,  3),
                                     XV_Y,  xv_row(panel,  6),
                                     NULL);
      p6  =  (Panel_item)  xv_create(panel,  PANEL_CHECK_BOX,
                                     PANEL_CHOICE_STRINGS,  labels[6],  NULL,
                                     PANEL_VALUE_X,  xv_col(panel,  3),
                                     XV_Y,  xv_row(panel,  7),
                                     NULL);
      p7  =  (Panel_item)  xv_create(panel,  PANEL_CHECK_BOX,
                                     PANEL_CHOICE_STRINGS,  labels[7],  NULL,
                                     PANEL_VALUE_X,  xv_col(panel,  3),
                                     XV_Y,  xv_row(panel,  8),
                                     NULL);
      p8  =  (Panel_item)  xv_create(panel,  PANEL_CHECK_BOX,
                                     PANEL_CHOICE_STRINGS,  labels[8],  NULL,
                                     PANEL_VALUE_X,  xv_col(panel,  3),
                                     XV_Y,  xv_row(panel,  9),
                                     NULL);
      p9  =  (Panel_item)  xv_create(panel,  PANEL_CHECK_BOX,
                                     PANEL_CHOICE_STRINGS,  labels[9],  NULL,
                                     PANEL_VALUE_X,  xv_col(panel,  3),
                                     XV_Y,  xv_row(panel,  10),
                                     NULL);

      (void)  xv_create(panel,  PANEL_BUTTON,
                        PANEL_LABEL_STRING,  "Done",
                        PANEL_NOTIFY_PROC,  clr_plots_done_proc,
                        XV_X,  xv_col(panel,  4),
                        XV_Y,  xv_row(panel,  11),
                      NULL);

      (void)  xv_create(panel,  PANEL_BUTTON,
                        PANEL_LABEL_STRING,  "Cancel",
                        PANEL_NOTIFY_PROC,  clr_plots_cancel_proc,
                        XV_X,  xv_col(panel,  20),
                        XV_Y,  xv_row(panel,  11),
                      NULL);

      window_fit(panel);
      window_fit(clr_plots_frame);
    }

  else
    {
      /*  set  all  the  labels  for  the  items  */
      xv_set(message,  PANEL_LABEL_STRING,  msg_label,  NULL);

      xv_set(p0,  PANEL_CHOICE_STRINGS,  labels[0],  NULL,  NULL);
      xv_set(p1,  PANEL_CHOICE_STRINGS,  labels[1],  NULL,  NULL);
```

```
        xv_set(p2,  PANEL_CHOICE_STRINGS,  labels[2],  NULL,  NULL);
        xv_set(p3,  PANEL_CHOICE_STRINGS,  labels[3],  NULL,  NULL);
        xv_set(p4,  PANEL_CHOICE_STRINGS,  labels[4],  NULL,  NULL);
        xv_set(p5,  PANEL_CHOICE_STRINGS,  labels[5],  NULL,  NULL);
        xv_set(p6,  PANEL_CHOICE_STRINGS,  labels[6],  NULL,  NULL);
        xv_set(p7,  PANEL_CHOICE_STRINGS,  labels[7],  NULL,  NULL);
        xv_set(p8,  PANEL_CHOICE_STRINGS,  labels[8],  NULL,  NULL);
        xv_set(p9,  PANEL_CHOICE_STRINGS,  labels[9],  NULL,  NULL);

      }

    /* set  items  inactive  if  there's  no  corresponding  plot  */
    if  (can_info->alive_plots[0]  ==  0)  xv_set(p0,  PANEL_INACTIVE,  TRUE,  NULL);
    else  xv_set(p0,  PANEL_INACTIVE,  FALSE,  NULL);
    if  (can_info->alive_plots[1]  ==  0)  xv_set(p1,  PANEL_INACTIVE,  TRUE,  NULL);
    else  xv_set(p1,  PANEL_INACTIVE,  FALSE,  NULL);
    if  (can_info->alive_plots[2]  ==  0)  xv_set(p2,  PANEL_INACTIVE,  TRUE,  NULL);
    else  xv_set(p2,  PANEL_INACTIVE,  FALSE,  NULL);
    if  (can_info->alive_plots[3]  ==  0)  xv_set(p3,  PANEL_INACTIVE,  TRUE,  NULL);
    else  xv_set(p3,  PANEL_INACTIVE,  FALSE,  NULL);
    if  (can_info->alive_plots[4]  ==  0)  xv_set(p4,  PANEL_INACTIVE,  TRUE,  NULL);
    else  xv_set(p4,  PANEL_INACTIVE,  FALSE,  NULL);
    if  (can_info->alive_plots[5]  ==  0)  xv_set(p5,  PANEL_INACTIVE,  TRUE,  NULL);
    else  xv_set(p5,  PANEL_INACTIVE,  FALSE,  NULL);
    if  (can_info->alive_plots[6]  ==  0)  xv_set(p6,  PANEL_INACTIVE,  TRUE,  NULL);
    else  xv_set(p6,  PANEL_INACTIVE,  FALSE,  NULL);
    if  (can_info->alive_plots[7]  ==  0)  xv_set(p7,  PANEL_INACTIVE,  TRUE,  NULL);
    else  xv_set(p7,  PANEL_INACTIVE,  FALSE,  NULL);
    if  (can_info->alive_plots[8]  ==  0)  xv_set(p8,  PANEL_INACTIVE,  TRUE,  NULL);
    else  xv_set(p8,  PANEL_INACTIVE,  FALSE,  NULL);
    if  (can_info->alive_plots[9]  ==  0)  xv_set(p9,  PANEL_INACTIVE,  TRUE,  NULL);
    else  xv_set(p9,  PANEL_INACTIVE,  FALSE,  NULL);

    xv_set(clr_plots_frame,  XV_SHOW,  TRUE,  NULL);
}


void  clr_plots_done_proc(item,  event)
    Panel_item  item;
    Event  *event;
{
    int  i;
    Canvas  canvas;
    Xv_Window  canvas_xv_window;
    canvasinfo  *can_info;

    can_info  =  wininfo.canvases[active_window];

    /*  get  all  the  checked  items  */
    chosen[0]  =  (int)xv_get(p0,  PANEL_VALUE);
    chosen[1]  =  (int)xv_get(p1,  PANEL_VALUE);
    chosen[2]  =  (int)xv_get(p2,  PANEL_VALUE);
    chosen[3]  =  (int)xv_get(p3,  PANEL_VALUE);
    chosen[4]  =  (int)xv_get(p4,  PANEL_VALUE);
    chosen[5]  =  (int)xv_get(p5,  PANEL_VALUE);
    chosen[6]  =  (int)xv_get(p6,  PANEL_VALUE);
    chosen[7]  =  (int)xv_get(p7,  PANEL_VALUE);
    chosen[8]  =  (int)xv_get(p8,  PANEL_VALUE);
    chosen[9]  =  (int)xv_get(p9,  PANEL_VALUE);


    /*  reset  all  the  items  to  unchecked  */
    xv_set(p0,  PANEL_VALUE,  0,  NULL);
    xv_set(p1,  PANEL_VALUE,  0,  NULL);
```



```
    xv_set(p2,  PANEL_VALUE,  0,  NULL);
    xv_set(p3,  PANEL_VALUE,  0,  NULL);
    xv_set(p4,  PANEL_VALUE,  0,  NULL);
    xv_set(p5,  PANEL_VALUE,  0,  NULL);
    xv_set(p6,  PANEL_VALUE,  0,  NULL);
    xv_set(p7,  PANEL_VALUE,  0,  NULL);
    xv_set(p8,  PANEL_VALUE,  0,  NULL);
    xv_set(p9,  PANEL_VALUE,  0,  NULL);

    for  (i=0;  i<10;  i++)
      {
        if  (chosen[i])
          {
            if  (can_info->alive_plots[i])
              {
                free  (can_info->plots[i]);
                can_info->alive_plots[i]  =  0;
                can_info->total_plots--;
                sprintf(msg,  "Clear  plot  %d\n",  i+1);
                print_msg(msg);
              }
          }
      }
    xv_set(clr_plots_frame,  XV_SHOW,  FALSE,  NULL);

    canvas  =  can_info->canvas;
    canvas_xv_window  =  canvas_paint_window(canvas);
    clear_canvas_proc(canvas);

    for  (i=0;  i<10;  i++)
      {
        if  (can_info->alive_plots[i])
          {
            can_info->active_plot  =  i;
            redraw_proc(canvas,  canvas_xv_window,  xv_get(canvas,  XV_DISPLAY),
                    xv_get(canvas_xv_window,  XV_XID),  NULL);
          }
      }
}


void  clr_plots_cancel_proc(item,  event)
    Panel_item  item;
    Event  *event;
{
    /*  resets  all  the  items  to  unchecked  state  */
    xv_set(p0,  PANEL_VALUE,  0,  NULL);
    xv_set(p1,  PANEL_VALUE,  0,  NULL);
    xv_set(p2,  PANEL_VALUE,  0,  NULL);
    xv_set(p3,  PANEL_VALUE,  0,  NULL);
    xv_set(p4,  PANEL_VALUE,  0,  NULL);
    xv_set(p5,  PANEL_VALUE,  0,  NULL);
    xv_set(p6,  PANEL_VALUE,  0,  NULL);
    xv_set(p7,  PANEL_VALUE,  0,  NULL);
    xv_set(p8,  PANEL_VALUE,  0,  NULL);
    xv_set(p9,  PANEL_VALUE,  0,  NULL);

    /*  unmaps  the  window  */
    xv_set(clr_plots_frame,  XV_SHOW,  FALSE,  NULL);
}


void  create_act_plot_menu_proc(item,  event)
    Panel_item  item;
```



```
    Event  *event;
{
    int  i;
    canvasinfo  *can_info;
    Menu_item  mi;
    Menu  menu=(Menu)  xv_get(item,  PANEL_ITEM_MENU);
    int  can_num;
    void  get_act_item_proc();

    can_num  =  (int)  xv_get(item,  PANEL_CLIENT_DATA);
    set_active_window(can_num);
    can_info  =  wininfo.canvases[active_window];

    for  (i=10;  i>0;  i--)
      {
        if  (xv_get(menu,  MENU_NTH_ITEM,  i))
          {
            xv_set(menu,  MENU_REMOVE,  i,  NULL);
            xv_destroy(xv_get(menu,  MENU_NTH_ITEM,  i));
          }
      }

    for(i=0;  i<10;  i++)
      {
        if  (can_info->alive_plots[i])
          {
            sprintf(plabel[i],  "%d.  %s  vs  %s",  i+1,
                    head.cb[can_info->plots[i]->col_x].name,
                    head.cb[can_info->plots[i]->col_y].name);

            mi  =  (Menu_item)xv_create(NULL,  MENUITEM,
                                        MENU_STRING,  plabel[i],
                                        MENU_NOTIFY_PROC,  get_act_item_proc,
                                        MENU_RELEASE,
                                        NULL);

            xv_set(menu,  MENU_APPEND_ITEM,  mi,  NULL);
          }
      }
}


void  get_act_item_proc(menu,  item)
    Menu  menu;
    Menu_item  item;
{
    int  i;
    canvasinfo  *can_info;
    Canvas  canvas;
    Xv_Window  canvas_xv_window;

    /*  no  need  to  set  active  window,  since  it  was  already  activated  when  pressing
utton  */

    can_info  =  wininfo.canvases[active_window];
    canvas  =  can_info->canvas;
    canvas_xv_window  =  canvas_paint_window(canvas);

    for  (i=0;  i<10;  i++)
      if  (xv_get(menu,  MENU_NTH_ITEM,  i+1)  ==  item)
        if  (can_info->alive_plots[i])
          set_active_plot(i);
}
```

```
#include  <math.h>
#include  <X11/Xlib.h>
#include  <xview/xview.h>
#include  <xview/panel.h>
#include  <xview/textsw.h>
#include  "global.h"

extern  int  active_window;
extern  int  old_active_window;
extern  int  total_windows;
extern  int  action;
extern  char  msg[MSG_LENGTH];

/*  main_frame  objects  */
extern  Frame  main_frame;
extern  Panel_item  cmd_panel_item;
extern  Panel  main_frame_info_panel;
extern  Panel_item  active_win_info,  active_plot_info;
extern  Textsw  msgwindow;
extern  Textsw  fileinfo_window;
extern  int  textsw_total_used;
extern  int  textsw_memory_limit;
extern  int  textsw_saves;


top()
{
  /*  default  prompt  */

  /*  clear  the  file  info  window  */
  textsw_reset(fileinfo_window,  0,  0);

  /*  print  the  new  file  info  */
  print_info();
  textsw_normalize_view(fileinfo_window,  1);

  sprintf(msg,  "\n");
  print_msg(msg);
  set_left_footer("Type  a  command");
  set_cmd_prompt("Command:  ");
}


name_col(colnum)
     int  colnum;
{
  /*  prompt  for  the  name  and  unit  (for  naming)  */
  sprintf(msg,  "Type  the  name  and  unit  for  column  %d",  colnum);
  set_left_footer(msg);
  set_cmd_prompt("Name,  Unit:  ");
}


print_fileinfo(txt)
     char  txt[MSG_LENGTH];
{
  /*  prints  the  current  data  states  in  info  window  */

  int  len  =  strlen(txt);
  if  (len  ==  0)  return;

  textsw_insert(fileinfo_window,  txt,  len);
}
```



```
print_msg(txt)
     char  txt[MSG_LENGTH];
{
  /*  prints  a  message  in  the  message  subwindow  */

  int  len;
  Menu  m;
  int  notice_msg;
  char  filename[20];

  len  =  strlen(txt);
  if  (len  ==  0)  return;

  /*  if  buffer  is  half  full,  show  a  warning  */
  if  ((textsw_total_used  <=  0.5*textsw_memory_limit)
     &&  textsw_total_used+len  >=  0.5*textsw_memory_limit)
    warn_textsw_almost_full();

  textsw_total_used  +=  len;

  /*  if  buffer  is  almost  full,  force  a  file  save  or  clear  buffer  */
  if  (textsw_total_used  >=  0.8*textsw_memory_limit)
    {
      sprintf(filename,  ".xlook_msgs_%d",  textsw_saves);
      if  (textsw_full_show_warning[filename]  ==  0)
        {
          textsw_store_file(msgwindow,  filename,  0,  0);
          textsw_saves++;
        }
      textsw_reset(msgwindow,  0,  0);
      textsw_total_used  =  0;
    }

  /*  print  txt  in  msg  window  */
  textsw_insert(msgwindow,  txt,  len);
}


set_cmd_prompt(txt)
     char  txt[128];
{
  xv_set(cmd_panel_item,  PANEL_LABEL_STRING,  txt,  NULL);
}


set_left_footer(txt)
     char  txt[128];
{
  xv_set(main_frame,  FRAME_LEFT_FOOTER,  txt,  NULL);
}


display_active_window(aw)
     int  aw;
{
  char  string[20];

  if  (aw  >  0)  sprintf(string,  "Active  Window:  %d",  aw);
  else  sprintf(string,  "Active  Window:  NONE");

  xv_set(active_win_info,  PANEL_LABEL_STRING,  string,  NULL);
```



```
}


display_active_plot(ap)
     int  ap;
{
  char  string[20];

  if  (ap  >  0)  sprintf(string,  "Active  Plot:  %d",  ap);
  else  sprintf(string,  "Active  Plot:  NONE");

  xv_set(active_plot_info,  PANEL_LABEL_STRING,  string,  NULL);
}


print_info()
{
  int  i  ;
  char  tmp[MSG_LENGTH];

  sprintf(msg,  "ALLOCATION:  max_col  =  %d  ,  max_row  =  %d\t",max_col,max_row);
  sprintf(tmp,  "number  of  records  =  %d\n  ",head.nrec);
  strcat(msg,  tmp);
  print_fileinfo(msg);

  msg[0]  =  '\0';
  for(  i  =  1;  i  <  max_col  ;  ++i  )
    {
      if  (  strncmp(&(head.ch[i].name[0]),"no_val",6)  !=  0)
        {
          sprintf(tmp,"  col  %ld",i)  ;
          strcat(msg,  tmp);
        }
    }
  strcat(msg,  "\n");
  print_fileinfo(msg);

  msg[0]  =  '\0';
  for(  i  =  1;  i  <  max_col  ;  ++i  )
    {
      if  (  strncmp(&(head.ch[i].name[0]),"no_val",6)  !=  0)
        {
          sprintf(tmp,"%12s",head.ch[i].name)  ;
          strcat(msg,  tmp);
        }
    }
  strcat(msg,  "\n");
  print_fileinfo(msg);

  msg[0]  =  '\0';
  for(  i  =  1;  i  <  max_col  ;  ++i  )
    {
      if  (  strncmp(&(head.ch[i].name[0]),"no_val",6)  !=  0)
        {
          sprintf(tmp,"%12s",head.ch[i].units)  ;
          strcat(msg,  tmp);
        }
    }
  strcat(msg,  "\n");
  print_fileinfo(msg);

  msg[0]  =  '\0';
  for(  i  =  1;  i  <  max_col  ;  ++i  )
```



```
    {
      if  (  strncmp(&(head.ch[i].name[0]),"no_val",6)  !=  0)
        {
          sprintf(tmp,"%7d  recs",head.ch[i].nelem)  ;
          strcat(msg,  tmp);
        }
    }
  strcat(msg,  "\n");
  print_fileinfo(msg);
}




/*****************************  error  messages  ************************/

nea()
{
  sprintf(msg,  "Not  enough  arguments.  Command  aborted.\n");
  print_msg(msg);
}

ne()
{
  sprintf(msg,  "Name  Error!  Aborted\n");
  print_msg(msg);
}

coe()
{
  sprintf(msg,  "Error!  Column  not  allocated.  You  can  only  use  up  to  12  colum
  print_msg(msg);
}

cre()
{
  sprintf(msg,  "Error!  Undefined  row  interval.\n");
  print_msg(msg);
}
```

```
#include <X11/Xlib.h>
#include <xview/xview.h>
#include <xview/canvas.h>
#include <xview/panel.h>
#include <math.h>

#include "global.h"

extern char msg[MSG_LENGTH];
extern int active_window;
extern GC gctick, gctitle;
extern char plot_cmd[128];


/******************************** line plot ********************************/

void line_plot_proc(menu, item)
    Menu menu;
    Menu_item item;
{
  int can_num = xv_get(menu, MENU_CLIENT_DATA);
  canvasinfo *can_info;
  int ap;

  set_active_window(can_num);
  can_info = wininfo.canvases[active_window];
  ap = can_info->active_plot;
  display_active_plot(ap+1);

  if (ap == -1)
    {
      sprintf(msg, "There is no plot in this window.\n");
      print_msg(msg);
      return;
    }

  set_line_plot();
}

set_line_plot()
{
  canvasinfo *can_info;

  can_info = wininfo.canvases[active_window];
  can_info->plots[can_info->active_plot]->mouse = 1;
  can_info->plots[can_info->active_plot]->p1 = 0;
  can_info->plots[can_info->active_plot]->p2 = 0;
  can_info->plots[can_info->active_plot]->x1 = 0;
  can_info->plots[can_info->active_plot]->x2 = 0;
  can_info->plots[can_info->active_plot]->y1 = 0;
  can_info->plots[can_info->active_plot]->y2 = 0;
  can_info->plots[can_info->active_plot]->zp1 = 0;
  can_info->plots[can_info->active_plot]->zp2 = 0;

}

do_line_plot()
{
  canvasinfo *can_info;
  int ap;
  plotarray *data;
  float slope, intercept, x1, x2, y1, y2;
  char xstring[20], ystring[20], rowstring[20];
```



```
  Canvas canvas;

  can_info = wininfo.canvases[active_window];
  ap = can_info->active_plot;
  data = can_info->plots[ap];

  canvas = can_info->canvas;

  XDrawLine(xv_get(can_info->canvas, XV_DISPLAY), can_info->win, gctick,
            data->x1, data->y1,
            data->x2, data->y2);

  x1 = (data->x1 - can_info->start_x)/data->scale_x + data->xmin;
  x2 = (data->x2 - can_info->start_x)/data->scale_x + data->xmin;
  y1 = (can_info->start_y - data->y1)/data->scale_y + data->ymin;
  y2 = (can_info->start_y - data->y2)/data->scale_y + data->ymin;

  sprintf(msg, "line plot: (%f, %f)-(%f, %f)\n", x1, y1, x2, y2);
  print_msg(msg);

  slope = (y2 - y1)/(x2 - x1);
  intercept = y1 - slope*x1;

  sprintf(rowstring, "");
  xv_set(xv_get(canvas, XV_KEY_DATA, CAN_ROW), PANEL_LABEL_STRING, rowstring, NULL);
  sprintf(xstring, "Slope: %.5g", slope);
  xv_set(xv_get(canvas, XV_KEY_DATA, CAN_X), PANEL_LABEL_STRING, xstring, NULL);
  sprintf(ystring, "Y intercept: %.5g", intercept);
  xv_set(xv_get(canvas, XV_KEY_DATA, CAN_Y), PANEL_LABEL_STRING, ystring, NULL);
}


/************************** mouse_mu ********************************/
void mouse_mu_proc(menu, item)
    Menu menu;
    Menu_item item;
{
  int can_num = xv_get(menu, MENU_CLIENT_DATA);
  canvasinfo *can_info;
  int ap;

  set_active_window(can_num);
  can_info = wininfo.canvases[active_window];
  ap = can_info->active_plot;
  display_active_plot(ap+1);

  if (ap == -1)
    {
      sprintf(msg, "There is no plot in this window.\n");
      print_msg(msg);
      return;
    }

  set_mouse_mu_proc();
}

set_mouse_mu_proc()
{
  canvasinfo *can_info;

  can_info = wininfo.canvases[active_window];
  can_info->plots[can_info->active_plot]->mouse = 2;
  can_info->plots[can_info->active_plot]->p1 = 0;
```



```
  can_info->plots[can_info->active_plot]->p2 = 0;
  can_info->plots[can_info->active_plot]->x1 = 0;
  can_info->plots[can_info->active_plot]->x2 = 0;
  can_info->plots[can_info->active_plot]->y1 = 0;
  can_info->plots[can_info->active_plot]->y2 = 0;
  can_info->plots[can_info->active_plot]->zp1 = 0;
  can_info->plots[can_info->active_plot]->zp2 = 0;

}

do_mouse_mu()
{
  canvasinfo *can_info;
  int ap;
  plotarray *data;

  can_info = wininfo.canvases[active_window];
  ap = can_info->active_plot;
  data = can_info->plots[ap];

  XDrawLine(xv_get(can_info->canvas, XV_DISPLAY), can_info->win, gctick,
            data->x1, can_info->start_y,
            data->x1, can_info->end_y);
}


/******************************** dist ********************************/
void dist_proc(menu, item)
    Menu menu;
    Menu_item item;
{
  int can_num = xv_get(menu, MENU_CLIENT_DATA);
  canvasinfo *can_info;
  int ap;

  set_active_window(can_num);
  can_info = wininfo.canvases[active_window];
  ap = can_info->active_plot;
  display_active_plot(ap+1);

  if (ap == -1)
    {
      sprintf(msg, "There is no plot in this window.\n");
      print_msg(msg);
      return;
    }

  set_dist_proc();
}

set_dist_proc()
{
  canvasinfo *can_info;

  can_info = wininfo.canvases[active_window];
  can_info->plots[can_info->active_plot]->mouse = 3;
  can_info->plots[can_info->active_plot]->p1 = 0;
  can_info->plots[can_info->active_plot]->p2 = 0;
  can_info->plots[can_info->active_plot]->x1 = 0;
  can_info->plots[can_info->active_plot]->x2 = 0;
  can_info->plots[can_info->active_plot]->y1 = 0;
  can_info->plots[can_info->active_plot]->y2 = 0;
  can_info->plots[can_info->active_plot]->zp1 = 0;
```



```
  can_info->plots[can_info->active_plot]->zp2 = 0;

}

do_dist()
{
  canvasinfo *can_info;
  int ap;
  plotarray *data;
  float dx, dy, da;
  char xstring[20], ystring[20], vstring[20];
  Canvas canvas;
  float x1, x2, y1, y2;

  can_info = wininfo.canvases[active_window];
  ap = can_info->active_plot;
  data = can_info->plots[ap];

  x1 = (data->x1 - can_info->start_x)/data->scale_x + data->xmin;
  x2 = (data->x2 - can_info->start_x)/data->scale_x + data->xmin;
  y1 = (can_info->start_y - data->y1)/data->scale_y + data->ymin;
  y2 = (can_info->start_y - data->y2)/data->scale_y + data->ymin;

  dx = fabs(x1 - x2);
  dy = fabs(y1 - y2);
  da = sqrt((double)((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2)));

  canvas = can_info->canvas;

  sprintf(xstring, "dX: %.5g", dx);
  xv_set(xv_get(canvas, XV_KEY_DATA, CAN_X), PANEL_LABEL_STRING, xstring, NULL);
  sprintf(ystring, "dY: %.5g", dy);
  xv_set(xv_get(canvas, XV_KEY_DATA, CAN_Y), PANEL_LABEL_STRING, ystring, NULL);
  sprintf(vstring, "dV: %.5g", da);
  xv_set(xv_get(canvas, XV_KEY_DATA, CAN_ROW), PANEL_LABEL_STRING, vstring, NULL);
}


/******************************** zoom ********************************/

void zoom_plot_proc(item, event)
    Panel_item item;
    Event *event;
{
  canvasinfo *can_info;
  int ap;

  set_active_window(xv_get(item, PANEL_CLIENT_DATA));

  can_info = wininfo.canvases[active_window];
  ap = can_info->active_plot;
  display_active_plot(ap+1);

  if (ap == -1)
    {
      sprintf(msg, "There is no plot in this window.\n");
      print_msg(msg);
      return;
    }

  set_zoom();
}
```

```
set_zoom()
{
  canvasinfo  *can_info;

  can_info  =  wininfo.canvases[active_window];
  can_info->plots[can_info->active_plot]->mouse  =  4;
  can_info->plots[can_info->active_plot]->p1  =  0;
  can_info->plots[can_info->active_plot]->p2  =  0;
  can_info->plots[can_info->active_plot]->x1  =  0;
  can_info->plots[can_info->active_plot]->x2  =  0;
  can_info->plots[can_info->active_plot]->y1  =  0;
  can_info->plots[can_info->active_plot]->y2  =  0;
  can_info->plots[can_info->active_plot]->zp1  =  0;
  can_info->plots[can_info->active_plot]->zp2  =  0;

}


zoom_get_pt(xloc,  yloc,  p)
    int  xloc,  yloc,  p;
{
  float  xval,  yval;
  int  nrows,  row_num;
  canvasinfo  *can_info;
  plotarray  *data;

  can_info  =  wininfo.canvases[active_window];
  data  =  can_info->plots[can_info->active_plot];

  xval  =  (xloc  -  can_info->start_x)/data->scale_x  +  data->xmin;
  yval  =  (can_info->start_y  -  yloc)/data->scale_y  +  data->ymin;

  nrows  =  data->nrows_x;
  row_num  =  get_row_number(nrows,  xval,  yval);
  if  (row_num  <=  nrows  &&  row_num  !=  -1)
    {
      xval  =  data->xarray[row_num];
      yval  =  data->yarray[row_num];
      draw_xhair(xval,  yval);

      /*  row_num  in  zp1  and  zp2  is  relative  to  this  plot's  begin  index  */
      if  (p  ==  1)
        data->zp1  =  row_num;
      else
        data->zp2  =  row_num;
    }

  /*  else,  do  nothing  */
}


zoom()
{
  float  x1,  x2;
  int  row_num;
  canvasinfo  *can_info;
  plotarray  *data;
  int  begin,  end;


  can_info  =  wininfo.canvases[active_window];
  data  =  can_info->plots[can_info->active_plot];

  x1  =  data->xarray[data->zp1];
```



```
  x2  =  data->xarray[data->zp2];

  if  (x2  <  x1)
    {
      begin  =  data->zp2  +  data->begin;
      end  =  data->zp1  +  data->begin;
    }
  else
    {
      begin  =  data->zp1  +  data->begin;
      end  =  data->zp2  +  data->begin;
    }

  /*  printf("zooming...  %d  %d  %d  %d\n",  data->col_x,  data->col_y,  begin,  end)
    clr_all();  */

  strcpy(plot_cmd,  "plotauto");
  sprintf(msg,  "plotauto  %d  %d  %d  %d",  data->col_x,  data->col_y,  begin,  end);
  do_plot(msg);
}



print_xy(xloc,  yloc)
    int  xloc,  yloc;
{
  canvasinfo  *can_info;
  plotarray  *data;
  char  xstring[20],  ystring[20],  rowstring[20];
  Canvas  canvas;
  float  xval,  yval;

  can_info  =  wininfo.canvases[active_window];
  data  =  can_info->plots[can_info->active_plot];
  canvas  =  can_info->canvas;

  draw_crosshair(xloc,  yloc);

  xval  =  (xloc  -  can_info->start_x)/data->scale_x  +  data->xmin;
  yval  =  (can_info->start_y  -  yloc)/data->scale_y  +  data->ymin;

  sprintf(rowstring,  "");
  xv_set(xv_get(canvas,  XV_KEY_DATA,  CAN_ROW),
         PANEL_LABEL_STRING,  rowstring,  NULL);

  /*  print  the  x  and  y  coord  on  the  panels  */
  sprintf(xstring,  "X:  %.5g",  xval);
  xv_set(xv_get(canvas,  XV_KEY_DATA,  CAN_X),  PANEL_LABEL_STRING,  xstring,  NULL);
  sprintf(ystring,  "Y:  %.5g",  yval);
  xv_set(xv_get(canvas,  XV_KEY_DATA,  CAN_Y),  PANEL_LABEL_STRING,  ystring,  NULL);
  /*  print  the  x  and  y  coord  on  the  msg  window  */
  strcat(xstring,  "  ");
  strcat(xstring,  ystring);
  XDrawString((Display  *)xv_get(canvas,  XV_DISPLAY)  ,  can_info->win,  gctick,  xloc+10
,  xstring,  strlen(xstring));
  strcpy(msg,  xstring);
  strcat(msg,  "\n");
  print_msg(msg);

}


print_xyrow(xloc,  yloc,  draw_string)
    int  xloc,  yloc,  draw_string;
```



```
{
  canvasinfo  *can_info;
  plotarray  *data;
  char  xstring[20],  ystring[20],  rowstring[20];
  char  xyrowstring[60];
  Canvas  canvas;
  float  xval,  yval;
  int  nrows,  row_num;

  can_info  =  wininfo.canvases[active_window];
  data  =  can_info->plots[can_info->active_plot];
  canvas  =  can_info->canvas;

  /*  get  the  x  and  y  (data)  values  */
  xval  =  (xloc  -  can_info->start_x)/data->scale_x  +  data->xmin;
  yval  =  (can_info->start_y  -  yloc)/data->scale_y  +  data->ymin;

  nrows  =  data->nrows_x;
  row_num  =  get_row_number(nrows,  xval,  yval);
  if  (row_num  <=  nrows  &&  row_num  !=  -1)
    {
      /*  get  x  and  y  values  from  data  (not  screen)  */
      xval  =  data->xarray[row_num];
      yval  =  data->yarray[row_num];
      row_num  =  row_num  +  data->begin;
      sprintf(rowstring,  "Row  Number:  %d",  row_num);
      /*  draw  crosshair  on  point  */
      draw_xhair(xval,  yval);
    }
  else
    {
      sprintf(rowstring,  "Row  Number:  None");
      sprintf(msg,  "The  point  picked  was  not  on  the  curve.\n");
      print_msg(msg);
    }
  xv_set(xv_get(canvas,  XV_KEY_DATA,  CAN_ROW),  PANEL_LABEL_STRING,  rowstring,  NULL);
  /*  print  the  x  and  y  coord  on  the  panels  */
  sprintf(xstring,  "X:  %.5g",  xval);
  xv_set(xv_get(canvas,  XV_KEY_DATA,  CAN_X),  PANEL_LABEL_STRING,  xstring,  NULL);
  sprintf(ystring,  "Y:  %.5g",  yval);
  xv_set(xv_get(canvas,  XV_KEY_DATA,  CAN_Y),  PANEL_LABEL_STRING,  ystring,  NULL);

  if  (draw_string  ==  1)
    {
      sprintf(xyrowstring,  "(%.5g,  %.5g)  row  %d",  xval,  yval,  row_num);
      xloc  =  (xval  -  data->xmin)*data->scale_x  +  can_info->start_x;
      yloc  =  can_info->start_y  -  (yval  -  data->ymin)*data->scale_y;
      XDrawString((Display  *)xv_get(canvas,  XV_DISPLAY)  ,  can_info->win,  gctick,  xloc+1
0,  yloc+10,  xyrowstring,  strlen(xyrowstring));
    }

}

/*
print  the  x  and  y  coord  on  the  info  panel  and  msg  window

draw_xy(xloc,  yloc)
    int  xloc,  yloc;
{
  canvasinfo  *can_info;
  plotarray  *data;
  char  xstring[20],  ystring[20],  rowstring[20];
  Canvas  canvas;
  float  xval,  yval;
```



```
  can_info  =  wininfo.canvases[active_window];
  data  =  can_info->plots[can_info->active_plot];
  canvas  =  can_info->canvas;

  xval  =  (xloc  -  can_info->start_x)/data->scale_x  +  data->xmin;
  yval  =  (can_info->start_y  -  yloc)/data->scale_y  +  data->ymin;

  sprintf(rowstring,  "");
  xv_set(xv_get(canvas,  XV_KEY_DATA,  CAN_ROW),
         PANEL_LABEL_STRING,  rowstring,  NULL);


  sprintf(xstring,  "X:  %.5g",  xval);
  xv_set(xv_get(canvas,  XV_KEY_DATA,  CAN_X),  PANEL_LABEL_STRING,  xstring,  NULL);
  sprintf(ystring,  "Y:  %.5g",  yval);
  xv_set(xv_get(canvas,  XV_KEY_DATA,  CAN_Y),  PANEL_LABEL_STRING,  ystring,  NULL);

  strcat(xstring,  "  ");
  strcat(xstring,  ystring);
  strcpy(msg,  xstring);
  strcat(msg,  "\n");
  print_msg(msg);

}

*/



int  get_row_number(nrow,  x1,  y1)
    int  nrow;
    float  x1,  y1;
{
  int  ii,  j,  stop;
  int  *list;
  float  xmax,xmin,ymax,ymin;
  float  dist1,  dist2;
  double  min_dist,  dist;
  canvasinfo  *can_info;
  plotarray  *data;
  int  rownum;
  float  *x,  *y;

  can_info  =  wininfo.canvases[active_window];
  data  =  can_info->plots[can_info->active_plot];

  list  =  (int  *)calloc((unsigned)nrow,sizeof(int));

  xmax  =  data->xmax;
  xmin  =  data->xmin;
  ymax  =  data->ymax;
  ymin  =  data->ymin;
  x  =  data->xarray;
  y  =  data->yarray;

  j  =  0;
  rownum  =  -1;          /*  default  value  for  "nothing  fits"  */

  for(ii  =  0;  ii  <  nrow;  ++ii)
    {
      /*  this  doesn't  do  too  well  if  x[ii]  =  x[ii+1]  */
      if(  (x1  >=  x[ii])  &&  (x1  <=  x[ii+1])  )
```

```
        list[j++] = ii;
      /* in case function is multiply defined, search for */
      /* all cases where "picked x" is between two points */
    }

  /* then calc. dist. to each point and take min. */
  min_dist = 1e100;

  for(ii = 0; ii < j; ++ii)
    {
      /* check both point in list and next point (the two that bracket x1) */
      /* to see which fits best */
      if( (dist = sqrt( (x1-x[list[ii]])*(x1-x[list[ii]]) + (y1-y[list[ii]])*(y1-y[list
]]) ) ) < min_dist)
        {
          min_dist = dist;
          rownum = list[ii];
        }

      if( (dist = sqrt( (x1-x[list[ii]+1])*(x1-x[list[ii]+1]) + (y1-y[list[ii]+1])*(y1-
ist[ii]+1]) ) ) < min_dist)
        {
          min_dist = dist;
          rownum = list[ii]+1;
        }
    }

  return rownum;
  }


draw_xhair(xval, yval)
     float xval, yval;
{
  canvasinfo *can_info;
  int xloc, yloc;
  int start_x, start_y;
  float scale_x, scale_y;
  float xmin, ymin;
  int active_plot;

  can_info = wininfo.canvases[active_window];
  active_plot = can_info->active_plot;
  start_x = can_info->start_x;
  start_y = can_info->start_y;
  scale_x = can_info->plots[active_plot]->scale_x;
  scale_y = can_info->plots[active_plot]->scale_y;
  xmin = can_info->plots[active_plot]->xmin;
  ymin= can_info->plots[active_plot]->ymin;

  xloc = (xval-xmin)*scale_x;
  yloc = (yval-ymin)*scale_y;

  XDrawLine(xv_get(can_info->canvas, XV_DISPLAY), can_info->win, gctitle,
            start_x + xloc - 5,
            start_y - yloc,
            start_x + xloc + 5,
            start_y - yloc);
  XDrawLine(xv_get(can_info->canvas, XV_DISPLAY), can_info->win, gctitle,
            start_x + xloc,
            start_y - yloc - 5,
            start_x + xloc,
            start_y - yloc + 5);
```



```
}

draw_crossbair(xloc, yloc)
     int xloc, yloc;
{
  canvasinfo *can_info;

  can_info = wininfo.canvases[active_window];

  XDrawLine(xv_get(can_info->canvas, XV_DISPLAY), can_info->win, gctitle,
            xloc-5, yloc,
            xloc+5, yloc);

  XDrawLine(xv_get(can_info->canvas, XV_DISPLAY), can_info->win, gctitle,
            xloc, yloc-5,
            xloc, yloc+5);

}
```



```
#include "global.h"
#include <string.h>
#include <xview/notice.h>
#include <xview/frame.h>
#include <X11/Xlib.h>
#include <xview/xview.h>

extern Frame main_frame;
extern int action;


int write_show_warning()
{
  extern Frame main_frame;
  Xv_notice notice;
  int notice_stat;

  notice = xv_create(main_frame, NOTICE,
                     NOTICE_MESSAGE_STRINGS,
                     "The file already exist.",
                     "Do you want to overwrite the file?",
                     NULL,
                     NOTICE_BUTTON_YES, "Yes",
                     NOTICE_BUTTON_NO, "No",
                     NOTICE_STATUS, &notice_stat,
                     XV_SHOW, TRUE,
                     NULL);

  switch(notice_stat)
    {
    case NOTICE_YES:
      xv_destroy_safe(notice);
      return(1);
      break;

    case NOTICE_NO:
      xv_destroy_safe(notice);
      return(0);
      break;
    }
}


warn_textsw_almost_full()
{
  extern Frame main_frame;
  Xv_notice notice;
  int notice_stat;

  notice = xv_create(main_frame, NOTICE,
                     NOTICE_MESSAGE_STRINGS,
                     "The message window is almost out of memory",
                     "You can either discard the whole content,",
                     "or you can save it in a file.",
                     "Click on the message window with the right button",
                     "to get the menu.",
                     NULL,
                     NOTICE_BUTTON_YES, "OK",
                     NOTICE_STATUS, &notice_stat,
                     XV_SHOW, TRUE,
                     NULL);
```



```
  switch(notice_stat)
    {
    case NOTICE_YES:
      xv_destroy_safe(notice);
      break;
    }
}


int textsw_full_show_warning(filename)
     char *filename;
{
  extern Frame main_frame;
  Xv_notice notice;
  int notice_stat;
  char name[128];

  sprintf(name, "Select Save to save the massages in %s", filename);


  notice = xv_create(main_frame, NOTICE,
                     NOTICE_MESSAGE_STRINGS,
                     "The message window is almost full.",
                     name,
                     "or select Discard to flush the messages. ",
                     NULL,
                     NOTICE_BUTTON, "Save", 100,
                     NOTICE_BUTTON, "Discard", 101,
                     NOTICE_STATUS, &notice_stat,
                     XV_SHOW, TRUE,
                     NULL);

  switch(notice_stat)
    {
    case 100:
      xv_destroy_safe(notice);
      return 0;
      break;

    case 101:
      xv_destroy_safe(notice);
      return 1;
      break;
    }
}


all_show_warning_proc(cmd)
     char cmd[128];
{
  extern Frame main_frame;
  Xv_notice notice;
  int notice_stat;


  notice = xv_create(main_frame, NOTICE,
                     NOTICE_MESSAGE_STRINGS,
                     "CAUTION - no buffer for current data array.",
                     "Reallocation will not cause loss of overlapping space",
                     "but if reallocation involves reduction, those parts",
                     "no longer allocated will be LOST and GONE.",
                     "no longer allocated will be LOST and GONE.",
```

```
                    NULL,
                    NOTICE_BUTTON_YES,  "Yes",
                    NOTICE_BUTTON_NO,  "No",
                    NOTICE_STATUS,  &notice_stat,
                    XV_SHOW,  TRUE,
                    NULL);

  switch  (notice_stat)
    {
    case  NOTICE_YES:
      /*  continue  processing  */
      set_left_footer("Input  number  of  rows  and  columns  for  reallocation");
      set_cmd_prompt("NROW  NCOL:  ");
      action  =  ALL_NEED_ROW_COL;
      break;
    case  NOTICE_NO:
      action  =  MAIN;
      set_left_footer("Aborted!");
      break;
    }

  xv_destroy_safe(notice);
}
```



```
#include  "global.h"
#include  <malloc.h>
#include  <stdio.h>


void  nrerror(error_text)
char  error_text[];
{
        void  exit();
        extern  char  msg[MSG_LENGTH];

        sprintf(msg,  "Numerical  Recipes  run-time  error...\n  %s",  error_text);
        print_msg(msg);
        sprintf(msg,"...now  exiting  to  system...\n");
        print_msg(msg);
        exit(1);
}



float  *vector(nl,nh)
int  nl,nh;
{
        float  *v;

        v=(float  *)malloc((unsigned)  (nh-nl+1)*sizeof(float));
        if  (!v)  nrerror("allocation  failure  in  vector()");
        return  v-nl;
}

int  *ivector(nl,nh)
int  nl,nh;
{
        int  *v;

        v=(int  *)malloc((unsigned)  (nh-nl+1)*sizeof(int));
        if  (!v)  nrerror("allocation  failure  in  ivector()");
        return  v-nl;
}

double  *dvector(nl,nh)
int  nl,nh;
{
        double  *v;

        v=(double  *)malloc((unsigned)  (nh-nl+1)*sizeof(double));
        if  (!v)  nrerror("allocation  failure  in  dvector()");
        return  v-nl;
}



float  **matrix(nrl,nrh,ncl,nch)
int  nrl,nrh,ncl,nch;
{
        int  i;
        float  **m;

        m=(float  **)  malloc((unsigned)  (nrh-nrl+1)*sizeof(float*));
        if  (!m)  nrerror("allocation  failure  1  in  matrix()");
        m  -=  nrl;

        for(i=nrl;i<=nrh;i++)  {
```



```
                m[i]=(float  *)  malloc((unsigned)  (nch-ncl+1)*sizeof(float));
                if  (!m[i])  nrerror("allocation  failure  2  in  matrix()");
                m[i]  -=  ncl;
        }
        return  m;
}

double  **dmatrix(nrl,nrh,ncl,nch)
int  nrl,nrh,ncl,nch;
{
        int  i;
        double  **m;

        m=(double  **)  malloc((unsigned)  (nrh-nrl+1)*sizeof(double*));
        if  (!m)  nrerror("allocation  failure  1  in  dmatrix()");
        m  -=  nrl;

        for(i=nrl;i<=nrh;i++)  {
                m[i]=(double  *)  malloc((unsigned)  (nch-ncl+1)*sizeof(double));
                if  (!m[i])  nrerror("allocation  failure  2  in  dmatrix()");
                m[i]  -=  ncl;
        }
        return  m;
}

int  **imatrix(nrl,nrh,ncl,nch)
int  nrl,nrh,ncl,nch;
{
        int  i,**m;

        m=(int  **)malloc((unsigned)  (nrh-nrl+1)*sizeof(int*));
        if  (!m)  nrerror("allocation  failure  1  in  imatrix()");
        m  -=  nrl;

        for(i=nrl;i<=nrh;i++)  {
                m[i]=(int  *)malloc((unsigned)  (nch-ncl+1)*sizeof(int));
                if  (!m[i])  nrerror("allocation  failure  2  in  imatrix()");
                m[i]  -=  ncl;
        }
        return  m;
}



float  **submatrix(a,oldrl,oldrh,oldcl,oldch,newrl,newcl)
float  **a;
int  oldrl,oldrh,oldcl,oldch,newrl,newcl;
{
        int  i,j;
        float  **m;

        m=(float  **)  malloc((unsigned)  (oldrh-oldrl+1)*sizeof(float*));
        if  (!m)  nrerror("allocation  failure  in  submatrix()");
        m  -=  newrl;

        for(i=oldrl,j=newrl;i<=oldrh;i++,j++)  m[j]=a[i]+oldcl-newcl;

        return  m;
}



void  free_vector(v,nl,nh)
float  *v;
```



```
int  nl,nh;
{
        free((char*)  (v+nl));
}

void  free_ivector(v,nl,nh)
int  *v,nl,nh;
{
        free((char*)  (v+nl));
}

void  free_dvector(v,nl,nh)
double  *v;
int  nl,nh;
{
        free((char*)  (v+nl));
}



void  free_matrix(m,nrl,nrh,ncl,nch)
float  **m;
int  nrl,nrh,ncl,nch;
{
        int  i;

        for(i=nrh;i>=nrl;i--)  free((char*)  (m[i]+ncl));
        free((char*)  (m+nrl));
}

void  free_dmatrix(m,nrl,nrh,ncl,nch)
double  **m;
int  nrl,nrh,ncl,nch;
{
        int  i;

        for(i=nrh;i>=nrl;i--)  free((char*)  (m[i]+ncl));
        free((char*)  (m+nrl));
}

void  free_imatrix(m,nrl,nrh,ncl,nch)
int  **m;
int  nrl,nrh,ncl,nch;
{
        int  i;

        for(i=nrh;i>=nrl;i--)  free((char*)  (m[i]+ncl));
        free((char*)  (m+nrl));
}



void  free_submatrix(b,nrl,nrh,ncl,nch)
float  **b;
int  nrl,nrh,ncl,nch;
{
        free((char*)  (b+nrl));
}



float  **convert_matrix(a,nrl,nrh,ncl,nch)
float  *a;
int  nrl,nrh,ncl,nch;
```

```
{
        int i,j,nrow,ncol;
        float **m;

        nrow=nrh-nrl+1;
        ncol=nch-ncl+1;
        m = (float **) malloc((unsigned) (nrow)*sizeof(float*));
        if (!m) nrerror("allocation failure in convert_matrix()");
        m -= nrl;
        for(i=0,j=nrl;i<=nrow-1;i++,j++) m[j]=a+ncol*i-ncl;
        return m;
}


void free_convert_matrix(b,nrl,nrh,ncl,nch)
float **b;
int nrl,nrh,ncl,nch;
{
        free((char*) (b+nrl));
}
```



```
#include "global.h"
#include <stdio.h>
#include <math.h>
#define PI 3.141592654

void perrf(Kamp,Kph,lgth,Vol,Per,pGperm,pGdif,psstor)
double Kamp,Kph,lgth,Vol,Per,*pGperm,*pGdif,*psstor;
{

        static int i;
        static double cond, area, omega, perm, diffuse;
        static double lper,beta,por;
        static double phi, amp, kay, rhog, visc;
        static double alpha, gamma, lambda;
        static double Ag, Pg, Dalp, Dgam, this, last, Num, Dom;
        static double sdg, efpor;
        static double Gperm, Gdif, sstor;

        Gperm = *pGperm;
        Gdif = *pGdif;
        sstor = *psstor;

        beta = 4.58e-10;
        rhog = 9800.0;
        visc = 0.001;
        area = 9.58e-04;

        cond = rhog*Gperm/visc;
        Gdif = cond/sstor;

        omega = 2.0*PI/Per;
        kay = sqrt(omega/(2.0*Gdif)) ;
        lambda = Gperm*area/visc/beta/Vol;
        alpha = lambda/pow(2.0*omega*Gdif,0.5);
        gamma = omega*lgth/pow(2.0*omega*Gdif,0.5);

        Ag = 4.0*alpha*alpha/((2.0*alpha*alpha+1.0)*cosh(2.0*gamma)+(2.0*alpha*alpha-1.0)*
cos(2.0*gamma)+2.0*alpha*(sinh(2.0*gamma)-sin(2.0*gamma)));
        Ag = sqrt(Ag);
        Num = -(sin(gamma)+tanh(gamma)*(2*alpha*sin(gamma)+cos(gamma)));
        Dom = (tanh(gamma)+2*alpha)*cos(gamma)-sin(gamma);
        Pg = atan(Num/Dom);
        if(Pg < 0) Pg=-Pg;
        if(Pg > 2*PI) Pg=Pg-2*PI;
        if(Pg > PI) Pg=Pg-PI;
        if (Dom > 0)
          {if (Num > 0)
            {Pg = Pg-2*PI;}
             else {Pg = -Pg;}}
        else
          {if (Num > 0)
            {Pg = -Pg-PI;}
             else {Pg = Pg-PI;}};
        alpha = 0.1;
        Dgam = 0.1;
        gamma = 0.1;
        while(Dgam>1.0e-09)
        {
        Dalp = 1.0;
        Ag = 4.0*alpha*alpha/((2.0*alpha*alpha+1.0)*cosh(2.0*gamma)+(2.0*alpha*alpha-1.0)*
cos(2.0*gamma)+2.0*alpha*(sinh(2.0*gamma)-sin(2.0*gamma)));
        Ag = sqrt(Ag);

        while(fabs(Ag-Kamp) > 1.0e-10)
```



```
        {
                if(Ag>Kamp)
                {
                        sdg = -1.0;
                        while(Ag>Kamp)
                        {
                                alpha += (Dalp*sdg);
                                if(alpha <= Dalp/10.0) { alpha=Dalp; Dalp/=10.0; if(Dal
.0e-08) break;}
                                Ag = 4.0*alpha*alpha/((2.0*alpha*alpha+1.0)*cosh(2.0*gamma
)+(2.0*alpha*alpha-1.0)*cos(2.0*gamma)+2.0*alpha*(sinh(2.0*gamma)-sin(2.0*gamma)));
                                Ag = sqrt(Ag);
                        }
                        Dalp /= 10.0;
                }
                if(Ag<Kamp)
                {
                        sdg = 1.0;
                        while(Ag<Kamp)
                        {
                                alpha += (Dalp*sdg);
                                Ag = 4.0*alpha*alpha/((2.0*alpha*alpha+1.0)*cosh(2.0*gamma
)+(2.0*alpha*alpha-1.0)*cos(2.0*gamma)+2.0*alpha*(sinh(2.0*gamma)-sin(2.0*gamma)));
                                Ag = sqrt(Ag);
                        }
                        Dalp /= 10.0;
                }
        }
        Num = -(sin(gamma)+tanh(gamma)*(2*alpha*sin(gamma)+cos(gamma)));
        Dom = (tanh(gamma)+2*alpha)*cos(gamma)-sin(gamma);
        Pg = atan(Num/Dom);
        if(Pg < 0) Pg=-Pg;
        if(Pg > 2*PI) Pg=Pg-2*PI;
        if(Pg > PI) Pg=Pg-PI;
        if (Dom > 0)
          {if (Num > 0)
            {Pg = Pg-2*PI;}
             else {Pg = -Pg;}}
        else
          {if (Num > 0)
            {Pg = -Pg-PI;}
             else {Pg = Pg-PI;}}
        perm = visc*beta*Vol*alpha*omega*lgth/area/gamma;
        diffuse = omega*lgth*lgth/2.0/gamma/gamma;
        if(Kph > Pg)
        {
                gamma -= Dgam;
                Dgam /= 10.0;
        }
        else { gamma += Dgam; }

        }
        Num = -(sin(gamma)+tanh(gamma)*(2*alpha*sin(gamma)+cos(gamma)));
        Dom = (tanh(gamma)+2*alpha)*cos(gamma)-sin(gamma);
        Pg = atan(Num/Dom);
        if(Pg < 0) Pg=-Pg;
        if(Pg > 2*PI) Pg=Pg-2*PI;
        if(Pg > PI) Pg=Pg-PI;
        if (Dom > 0)
          {if (Num > 0)
            {Pg = Pg-2*PI;}
             else {Pg = -Pg;}
        }
        else
```



```
          {if (Num > 0)
            {Pg = -Pg-PI;}
             else {Pg = Pg-PI;}
        }
        perm = visc*beta*Vol*alpha*omega*lgth/area/gamma;
        diffuse = omega*lgth*lgth/2.0/gamma/gamma;
        cond = rhog*perm/visc;
        sstor = cond/diffuse;
        efpor = sstor/rhog/beta;

*pGperm = perm;
*psstor = sstor;
fprintf(stderr,"%le\t%le\n",perm,sstor);
return;
}
```

```
/*
        Constructs the descrete weighted least squares approximation by polynomials
        degree less than nterms to given data.

        Inputs;
                x[i], f[i] i=0 .. npoint-1 gives the abscissae and ordinates of t
n data
                        to be fit
                w npoint vector containing the positive weights to be used
                npoint number of data points.
                nterms the order (degree + 1) of the polynomial approximant ( <=

        Work areas;
                pjml, pj arrayes of length npoint to containe the values at the
  two most
                recent orthogonal polynomials.

        Outputs;
                error npoint vector containing the error at the x's of the polynor
roximant to
                the given data.

        Output via externals;
                b,c arrayes contining the coefficients for the three-term recurrence
  generates
                the orthogonal polynomials.
                d coefficients of the polynomial approximant to the given data with
t to the
                sequence of orthogonal polynomials.
                The value of the approximant at a point y may be obtained by a
o ortval(y).

        Method:
                The sequence p0, p1,...,pnterms-1 of orthogonal polynomials with res
o the discrete
                inner product
                        (p,q) = sum(p(x[i])*q(x[i])*w[i], i=0,....npoint-1
                is generated in terms of their three-term recurrence
                        pjp1(x)=(x-b[j+1])*pj(x)-c[j+1]*pjm1(x)
                and the coefficient d[j] of the weighted least squares approximatior
e given data
                is obtained concurrently as
                        d[j+1] = (f,pj)/(pj,pj) j=0,.....nterms-1
                actually, in order to reduce cancellation, (f,pj) is calculated as
pj) with
                error=f initially, and for each j, error reduced by d[j+1]*pj as
[j+1]
                becomes available.
*/
static double b[20],c[20],d[20]; /* external variables used by ortval */
static int numterm;

void ortpol(x,f,w,npoint,pjml,pj,error,nterms)
        double x[],f[],pj[],pjml[],w[],error[];
        int npoint,nterms;
{
        int i,j;
        double p,s[20];

        numterm=nterms;

        for(j=0;j<nterms;j++)
                b[j]=d[j]=s[j]=0.0;
        c[0]=0.0;
```



```
        for(i=0;i<npoint;i++)
        {
                d[0]+=(f[i]*w[i]);
                b[0]+=(x[i]*w[i]);
                s[0]+=w[i];
        }
        d[0]/=s[0];
        for(i=0;i<npoint;i++)
                error[i]=f[i]-d[0];
        if(nterms == 1) return;
        b[0]/=s[0];
        for(i=0;i<npoint;i++)
        {
                pjml[i]=1.0;
                pj[i]=x[i]-b[0];
        }

        for(j=1; j<nterms;j++)
        {
                for(i=0;i<npoint;i++)
                {
                        p=pj[i]*w[i];
                        d[j]+=(error[i]*p);
                        p*=pj[i];
                        b[j]+=(x[i]*p);
                        s[j]+=p;
                }
                d[j]/=s[j];
                for(i=0;i<npoint;i++)
                        error[i]-=(d[j]*pj[i]);
                if(j == nterms-1) return;
                b[j]/=s[j];
                c[j]=s[j]/s[j-1];
                for(i=0;i<npoint;i++)
                {
                        p=pj[i];
                        pj[i]=(x[i]-b[j])*pj[i]-c[j]*pjml[i];
                        pjml[i]=p;
                }
        }
}

/*
        Returns the value at x of the polynomial of degree less than nterms
                d[1]*po(x)+d[2]*p1(x)+.....+d(nterms)*pnterms-1(x)
        with the sequence p0,p1,... of orthogonal polynomials generated by the
        recurrence
                pjp1(x)=(x-b[j+1])*pj(x)-c(j+1]*pjml(x), all j
*/

double ortval(x)
        double x;
{
        extern double b[20],c[20],d[20];
        extern int numterm;
        double prev,yval,prev2;
        int k;

        prev=0.0;
        yval=d[numterm-1];
        if(numterm == 1) return(yval);
        for(k=numterm-2;k>=0;k--)
        {
                prev2=prev;
```



```
                prev=yval;
                yval=d[k]+(x-b[k])*prev-c[k+1]*prev2;
        }
        return(yval);
}
```



```
/*      Non-linear inverse routine using the levenberg-marquardt method and
                svd

This version is set up to invert for the parameters of a 2 state variable
rate and state model

last modified
        26/7/94 changed the way the L-M lamda parameter is modified. It now
        gets reduced only after 3 steps in the right direction.
        12/8/94 changed format of op_file to include param info at bottom
        14/8/94 changed to allow data table to be written automatically
*/
#include "global.h"
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <sys/types.h>
#include <sys/timeb.h>
#include <strings.h>
#define CLOSE(a,b) (fabs((a-(int)a) - b) < 0.01)

extern char msg[MSG_LENGTH];


FILE    *op_file, *data_table;

int exec_gl(converg_tol, lamda, wc, rsp)
    double converg_tol, lamda, wc;
    struct rs_parameters *rsp;   /*rate state stuff, defined in global.h*/
{
  time_t tp;
  char *calloc(), buf[80], mess1[200], *final_mess;
  char *cp, tmp_msg[MSG_LENGTH];
  char weight_vec_only=FALSE;
  int i, j, k;
  int  iter, iter_max, ndata, ma, temp;
  int  cont = TRUE;
  int  neglig;
  int  get_mu_at_x();
  int  msvdfit();
  double       *x, *y, *a, *wv, temp_d;
  double       **u,**v,*w,*a_unscaled, *std_a, *std_a_unscaled;
  double       chisq, prev_chisq, lamda_max;
  double       sqrt(), pow();
  double       calc_chisq();
  double       **dm();
  void free_dm();
  void set_rs_parameters(), cfree();

  ma=6;                         /*set defaults*/
  rsp->one_sv_flag=FALSE;
  final_mess = (char *)calloc((unsigned)1000,sizeof(char));

  if(rsp->dc2 < 0)
    {
      ma=4;
      rsp->one_sv_flag=TRUE;
    }
  ndata=rsp->last_row - rsp->first_row+1;

  rsp->vs_row++;                /* to account for fortran c, here vs_row is rel
rst row, which = 1 */
  rsp->weight_row++;
  rsp->end_weight_row++;
```

```
for(i=1;i<=rsp->n_vsteps;i++) /*don't do 0*/
  rsp->vs_row_list[i]++;

/*allocate space*/
a = (double *)calloc((unsigned)ma+1,sizeof(double));
w = (double *)calloc((unsigned)ma+1,sizeof(double));
wv = (double *)calloc((unsigned)ndata+1,sizeof(double)); /*weighting vector*/
std_a = (double *)calloc((unsigned)ma+1,sizeof(double));
std_a_unscaled = (double *)calloc((unsigned)ma+1,sizeof(double));
a_unscaled = (double *)calloc((unsigned)ma+1,sizeof(double));
u = dm(ndata,ma);
v = dm(ma,ma);

if(rsp->one_sv_flag)                    /*to handle 1sv case*/
  {
    rsp->b2 = a[5] = 0.0;
    rsp->dc2 = a[6] = 1e30;
  }

if(lamda >= 0)                  /*if less than 0, don't write log files*/
  {
    /* write info to log */
    strcpy(buf,"qi_log_");
    strcat(buf,head.title);
    if(rsp->op_file_flag & 0x10)      /*sixteens bit means use separate file for e
ep*/
      {
        sprintf(messl,"%d",rsp->vs_row-1);
        strcat(buf,"_");
        strcat(buf,messl);
      }
    /*twos bit on means create rather than append*/
    if(rsp->op_file_flag & 0x2)
      op_file = fopen(buf, "w"); /* write to default file */
    else
      op_file = fopen(buf, "a"); /* append to default file */

    if(rsp->op_file_flag & 0x08) /*eights bit on means write data table*/
      {
        data_table = fopen("data_table", "a"); /* append to default file */
        if( (int)ftell(data_table) == 0 )     /*write header if first line*/
          fprintf(data_table,"%8s,%8s,%9s,%9s,%9s,%8s,%8s,%9s,%9s,%8s,%8s,%9s,%s,%1s
,%s,%6s,%10s\n","mu_o","std mu_o","a","std a","b1","std b1","Dc1","std Dc1","b2","std b2
"Dc2","std Dc2","a-b","lin_term","wc","law","step","run date");
      }
    fprintf(op_file,"\n**************************************************\n");
    time(&tp);
    fprintf(op_file,"%d state variable fit for Exp: %s \t\t Date: %s", (rsp->one_
) ? 1: 2, head.title, ctime(&tp));
    fprintf(op_file,"result to column %d of look file\n",rsp->mu_fit_col);
    fprintf(op_file,"recs fit: %d to %d, vs_row=%d, fric_lin_term=%g, v_init. = %g
= %g, k=%g,\n\tconvergence_tol=%g, L-M_lamda=%g\n", rsp->first_row, rsp->last_row, rsp-
s_row-1, rsp->lin_term, rsp->vo, rsp->vf, rsp->stiff, converg_tol, lamda);

    if(rsp->n_vsteps >1)
      {
        fprintf(op_file,"Fitting multiple velocity steps,");
        for(i=1;i<rsp->n_vsteps;i++)          /*1st new vel in i=1, 3rd v overall",
          fprintf(op_file,"\tvs_row=%d v=%g%c",rsp->vs_row_list[i]-1,rsp->vel_list[i],(i
==rsp->n_vsteps-1) ? '\n': ',');
      }

    if(wc > 0)
      sprintf(mess1,"Weighting with wv(i) = (ndata/i)^%g, first weighted row (i=1)
```



```
\n",wc, rsp->weight_row-1);

      else if(wc < 0)
        sprintf(mess1,"Weighting data from row %d to row %d by a factor of %g\n",
ht_row-1, rsp->end_weight_row-1,-wc);

      else
        sprintf(mess1,"no weighting\n");

      fprintf(op_file,mess1);
      sprintf(msg, "%s\n", mess1);
      print_msg(msg);

      strncpy(final_mess,mess1,strlen(mess1));          /*save */
      switch(rsp->law)
        {
        case 'o':
          sprintf(buf,"Ruina law -using original (old) code");
          break;
        case 'r':
          sprintf(buf,"Ruina-Dieterich, slip law");
          break;
        case 'd':
          sprintf(buf,"Dieterich-Ruina, slowness law");
          break;
        case 'p':
          sprintf(buf,"Perrin-Rice law");
          break;
        case 'j':
          /*check this later, just a guess now 15/7/94   */
          sprintf(buf,"Rice law");
          break;
        }
      strcat(final_mess,buf);          /*save */
      strcpy(mess1,"Inverting for the ");
      strcat(mess1,buf);
      strcat(mess1,"\n");
      fprintf(op_file,mess1);
      sprintf(msg, "%s\n", mess1);
      print_msg(msg);

      strcpy(msg, "");
      if(rsp->one_sv_flag)
        sprintf(msg, "Doing one sv fit,\t");
      if(rsp->lin_term)
        strcat(msg, "\tadding linear term to friction law: %g*dx\n",rsp->lin_term);
      else
        strcat(msg, "\n");
      print_msg(msg);

      fprintf(op_file,"Initial guesses for parameters:\n\tmu_init= %g,\ta= %g,\tb1= %g
c1= %g,\tb2= %g,\tdc2= %g\n", rsp->mu_o, rsp->a, rsp->b1, rsp->dc1, rsp->b2, rsp->dc
    }

  else
    op_file=stderr; /*redirect op to stderr if no log file used*/

  /* a[1] = mu_o, a[2]=a, a[3]=b1, a[4]=log(dc1), a[5]=b2, a[6]=log(dc2) */
  a[1] = rsp->mu_o;
  a[2] = rsp->a;
  a[3] = rsp->b1;
  a[4] = rsp->dc1;
  a[5] = rsp->b2;
  a[6] = rsp->dc2;
```



```
/*solve for log of Dc, following RWLM*/
a[4] = log(a[4]);
if(!rsp->one_sv_flag)
  a[6] = log(a[6]);

/* gen func. for setting a-b, mu_f etc */
set_rs_parameters(rsp,a,NULL);

if(lamda < 0)
  {
    if( get_mu_at_x(rsp->disp_data,rsp->model_mu,ndata,rsp) == -1)
      {
        sprintf(msg, "error from get_mu_at_x \n");
        print_msg(msg);
      }

    for(i=1;i<=ndata;i++)
      wv[i]= 1.0;
    chisq = prev_chisq = calc_chisq(a, rsp->disp_data, rsp->mu_data, wv, ma, ndat
;
    sprintf(msg, "chisq=%g\n",chisq);
    print_msg(msg);
    return(i);
  }

/*weighting*/
if( CLOSE(wc,0.1234) )       /*so I can see weighting vector*/
  {
    wc =-0.1234;
    weight_vec_only=TRUE;
  }

if(wc < 0)                                  /*weight between two points*/
  {
    for(i=1; i<(rsp->weight_row - rsp->first_row); ++i)
      wv[i] = 1.0;
    for(i=rsp->weight_row - rsp->first_row; i<=rsp->end_weight_row - rsp->first_row;
}
      wv[i] = -wc;
    for(i=rsp->end_weight_row - rsp->first_row +1; i<=ndata; ++i)
      wv[i] = 1.0;
    wc *= -1.0;
  }

else                            /*weight with decreasing function*/
  {   /*for mul. v steps, default is to weight with std wv between steps*/
    /*first part*/
    for(i=1;i< rsp->vs_row - rsp->first_row; ++i)
      wv[i] = 1.0;

    for(k=1;k<=rsp->n_vsteps;k++)
      {                             /*temp will = ndata if n_vsteps = 1*/
        temp = rsp->vs_row_list[k] - rsp->vs_row_list[k-1] +1;
        for(j=1, i=rsp->vs_row_list[k-1] - rsp->first_row;
            i <= rsp->vs_row_list[k] - rsp->first_row; ++i)
          wv[i] = pow((double)((double)temp/(double)j++),wc);
      }
  }

if( weight_vec_only ) /*so I can see weighting vector*/
  {
    for(i=1;i<=ndata;i++)
      rsp->model_mu[i] = wv[i];
```



```
    sprintf(msg, "\n\nweighting vector in mod_mu col, wc=%g\n\n",wc);
    print_msg(msg);
    return(2);
  }

/*initialize*/
chisq = prev_chisq = calc_chisq(a, rsp->disp_data, rsp->mu_data, wv, ma, ndata,
if(chisq == -1)
  {
    fclose(op_file);
    return(-1);
  }

fprintf(op_file,"chisq for initial model is: %f\n",chisq);

lamda *= -1.0;                              /*for initialization*/
if(msvdfit(rsp->disp_data, rsp->mu_data, wv, ndata, a, ma, u, v, w, &chisq, rsp
==-1)
  {
    sprintf(msg, "problem in msvdfit: initialization%c\n",BELL);
    print_msg(msg);
    fclose(op_file);
    return(-1);
  }


lamda_max = 5000; iter_max=100;
iter=neglig=0;
while( cont )                       /*main loop for iterations*/
  {
    if(msvdfit(rsp->disp_data, rsp->mu_data, wv, ndata, a, ma, u, v, w, &chisq,
mda)==-1)
      {
        sprintf(msg, "problem in msvdfit: iteration = %d chisq=%g%c\n",iter,chisq,
        print_msg(msg);
        fclose(op_file);
        return(-1);
      }

    /* check for convergence */
    /*don't count if it went up*/
    if((chisq < prev_chisq) && (1.0-chisq/prev_chisq < converg_tol))
      neglig++;

    if(neglig > 1)                  /* stop on the 2nd time */
      cont = FALSE;
    iter++;

    /*uniform variance is chisq/(ndata-ma) */
    /*ones bit of op_flag is verbose option*/
    if( (iter % 2)==0 || (rsp->op_file_flag & 0x1))
      {
        fprintf(op_file,"new params for a\n");
        for(i=1;i<=ma; ++i)
          fprintf(op_file," %f%c",((i==4 || i==6) ? exp(a[i]) : a[i]), (i % ma)
: '\n');
        fprintf(op_file,"Iteration: %d \tchisq=%f\tdel_chisq=%g\tl-chisq/prev_chisq=%g\
L-M lamda=%g\n", iter,chisq,(chisq-prev_chisq),1-chisq/prev_chisq,lamda);
      }

    sprintf(msg, "new params for a\n");
    print_msg(msg);
    sprintf(msg, "");
    for(i=1;i<=ma; ++i)
```

```
      {
        sprintf(tmp_msg, " %f%c",((i==4 || i==6) ? exp(a[i]) : a[i]), (i % ma)
'\n');
        strcat(msg, tmp_msg);
      }
    print_msg(msg);

    sprintf(msg, "Iteration: %d \tchisq=%f\tdel_chisq=%g\tl-chisq/prev_chisq=%g\tL-M la
da=%g\n", iter,chisq,(chisq-prev_chisq),1-chisq/prev_chisq,lamda);

    prev_chisq = chisq;              /* keep previous error */

    if( (lamda > lamda_max || iter > iter_max) && (rsp->op_file_flag & 0x4) ) /
t on means interactive mode */
      {
        sprintf(msg, "\n\tlamda too large or too many iterations. Try again with
imum values for both.\n");
        /* scanf("%lf %d",&lamda_max, &iter_max); */
        /* changed: just exit and ask the user to retry. */
        print_msg(msg);
        return(-1);
      }
  }                                /*end of main iteration loop*/

/* do undamped, non-weighted L-M and free work space on final call */
lamda = 0.0;
if(wc)
  for(i=1;i<=ndata;++i)       /*remove weighting if nec. */
    wv[i]=1.0;
/*covar is returned in v*/
if(msvdfit(rsp->disp_data, rsp->mu_data, wv, ndata, a, ma, u, v, w, &chisq, rsp,
==-1)
  {
    sprintf(msg, "problem in msvdfit: final call to set covar%c\n",BELL);
    print_msg(msg);
    fclose(op_file);
    return(-1);
  }

/* mu for best fit params */
if( get_mu_at_x(rsp->disp_data,rsp->model_mu,ndata,rsp) == -1)
  {
    fclose(op_file);
    return(-1);       /*signal problem to caller*/
  }

/*variance = chisq/(ndata-ma)*/
chisq = calc_chisq(a, rsp->disp_data, rsp->mu_data, wv, ma, ndata, rsp);
if(chisq == -1)
  {
    sprintf(msg, "problem from chisq calc: final call after getting covar%c\n",BEL
    print_msg(msg);
    fclose(op_file);
    return(-1);
  }

fprintf(op_file,"std covar:\n");
for(i=1;i<=ma;++i)
  for(j=1; j<=i; ++j)
    fprintf(op_file,"%11.6g%c",v[i][j],((i==j) ? '\n' : '\t'));

for(i=1;i<=ma;++i)         /*calculate scaled covariance : covar*variance */
  {
    for(j=1; j<=i; ++j)
```



```
      v[i][j] = v[i][j]*(chisq/(ndata-ma));

    /*std_a is std dev of log Dc's */
    std_a_unscaled[i] = std_a[i] = sqrt(v[i][i]);

    a_unscaled[i] = a[i];     /*unscaled model vector*/
  }

fprintf(op_file,"covar*variance (for logDc's) is:\n");
for(i=1;i<=ma;++i)
  for(j=1; j<=i; ++j)
    fprintf(op_file,"%11.6g%c",v[i][j],((i==j) ? '\n' : '\t'));

for(i=1;i<=ma;++i)                    /*calculate correlation coef*/
  for(j=1; j<=i; ++j)
    u[i][j] = v[i][j]/(sqrt(v[i][i])*sqrt(v[j][j]));

fprintf(op_file,"correlation coefficient matrix:\n");
for(i=1;i<=ma;++i)                    /*print correlation coef*/
  for(j=1; j<=i; ++j)
    fprintf(op_file,"%11.6g%c",u[i][j],((i==j) ? '\n' : '\t'));

a_unscaled[4] = exp(a[4]);                              /*unscaled model params */
std_a_unscaled[4] = exp(a[4]+std_a[4])-exp(a[4]);     /*unscaled std. dev. of a...l
ms */
if(!rsp->one_sv_flag)
  {
    a_unscaled[6] = exp(a[6]);
    std_a_unscaled[6] = exp(a[6]+std_a[6])-exp(a[6]);          /*treat +,- separat  */
  }

fprintf(op_file,"\nunweighted Chi square error is %g\t\tvariance=chisq/dof = %g\n\n
sq,chisq/(ndata-ma));

/*
    fprintf(op_file,"parameters (best fit) which minimize chi_sq = |A a - b    1"
    fprintf(op_file,"mu_o a b1 dc1 b2 dc2 a-b\n");
    for(i=1;i<=ma; ++i)
    fprintf(op_file,"%11.6g%c",a_unscaled[i], ( (i % ma) ? ' ' : '\n'));
    fprintf(op_file,"%11.6g\n",rsp->amb);
    if(!rsp->one_sv_flag))
    fprintf(op_file,"%8.6f %8.6f %8.6f %8.3f\n",a_unscaled[1],a_unscaled[2],a_unscaled
[3],a_unscaled[4],rsp->amb);
    else
    fprintf(op_file,"%8.6f %8.6f %8.6f %8.3f %8.6f %8.3f\n",a_unscaled[1],a_unsc
[2],a_unscaled[3],a_unscaled[4],a_unscaled[5],a_unscaled[6],rsp->amb);

    fprintf(op_file,"std_dev's (real Dc's)\n");
    for(i=1;i<=ma; ++i)
    fprintf(op_file,"%11.6g%c",std_a_unscaled[i], ( (i % ma) ? '\t' : '\n'));
    if(rsp->one_sv_flag)
    fprintf(op_file,"%10c\t%10c\t%10c\t%11.6g\n",' ',' ',' ',exp(a[4]-std_a[4])-exp
]);
    else
    fprintf(op_file,"%10c\t%10c\t%10c\t%11.6g\t%10c\t%11.6g\n",' ',' ',' ',exp(a[4]-std_
[4])-exp(a[4]),' ',exp(a[6]-std_a[6])-exp(a[6]));

    */

  /*best fit values*/
  sprintf(msg, "parameters (best fit) which minimize chi_sq = |A a - b|^2\n");
  fprintf(op_file,"parameters (best fit) which minimize chi_sq = |A a - b|^2\n");
```



```
  if(rsp->one_sv_flag)
    {
      sprintf(msg, "mu_o a b1 dc1 a-b\n");
      print_msg(msg);

      fprintf(op_file,"mu_o a b1 dc1 a-b\n");
      sprintf(msg, "%8.6f %9.6f %9.6f %8.3f %9.6f\n",a_unscaled[1],a_unscaled[2],a_unsc
d[3],a_unscaled[4],rsp->amb);
      print_msg(msg);

      fprintf(op_file,"%8.6f %9.6f %9.6f %8.3f %9.6f\n",a_unscaled[1],a_unscaled[2],a_uns
aled[3],a_unscaled[4],rsp->amb);
    }
  else
    {
      sprintf(msg, "mu_o a b1 dc1 b2 dc2 a-b\n");
      print_msg(msg);

      fprintf(op_file,"mu_o a b1 dc1 b2 dc2 a-b\n");
      sprintf(msg, "%8.6f %9.6f %9.6f %8.3f %9.6f %8.3f %9.6f\n",a_unscaled[1],a_unsc
2],a_unscaled[3],a_unscaled[4],a_unscaled[5],a_unscaled[6],rsp->amb);
      print_msg(msg);

      fprintf(op_file,"%8.6f %9.6f %9.6f %8.3f %9.6f %8.3f %9.6f\n",a_unscaled[1],a_uns
ed[2],a_unscaled[3],a_unscaled[4],a_unscaled[5],a_unscaled[6],rsp->amb);
    }

  /*std deviations*/
  sprintf(msg, "std_dev's (real Dc's)\n");
  print_msg(msg);
  fprintf(op_file,"std_dev's (real Dc's)\n");
  if(rsp->one_sv_flag)
    {
      sprintf(msg, "%8.6f %9.6f %9.6f %8.3f\n",std_a_unscaled[1],std_a_unscaled[2],std_a_
nscaled[3],std_a_unscaled[4]);
      print_msg(msg);
      sprintf(msg, "%37.3f\n",exp(a[4]-std_a[4])-exp(a[4]));
      print_msg(msg);
      fprintf(op_file,"%8.6f %9.6f %9.6f %8.3f\n",std_a_unscaled[1],std_a_unscaled[2],std_
a_unscaled[3],std_a_unscaled[4]);
      fprintf(op_file,"%37.3f\n",exp(a[4]-std_a[4])-exp(a[4]));
    }
  else
    {
      sprintf(msg, "%8.6f %9.6f %9.6f %8.3f %9.6f %8.3f\n",std_a_unscaled[1],std_a_unsc
d[2],std_a_unscaled[3],std_a_unscaled[4],std_a_unscaled[5],std_a_unscaled[6]);
      print_msg(msg);
      sprintf(msg, "%37.3f %18.3f\n",exp(a[4]-std_a[4])-exp(a[4]),exp(a[6]-std_a[6])-exp(a
[6]));
      print_msg(msg);
      fprintf(op_file,"%8.6f %9.6f %9.6f %8.3f %9.6f %8.3f\n",std_a_unscaled[1],std_a_u
aled[2],std_a_unscaled[3],std_a_unscaled[4],std_a_unscaled[5],std_a_unscaled[6]);
      fprintf(op_file,"%37.3f %18.3f\n",exp(a[4]-std_a[4])-exp(a[4]),exp(a[6]-std_a[6])-ex
p(a[6]));
    }

  sprintf(buf,"; fric_lin_term=%g \n", rsp->lin_term);
  strcat(final_mess,buf);
  fprintf(op_file,"%d state variable fit for %s; recs fit: %d to %d, vs_row=%d\n",
ne_sv_flag) ? 1: 2, head.title,rsp->first_row, rsp->last_row, rsp->vs_row-1);
  fprintf(op_file,final_mess);

  if(rsp->op_file_flag & 0x08) /*write to data table*/
    {
```



```
    if( !rsp->one_sv_flag && (a_unscaled[6] < a_unscaled[4]) )        /*dc1 < 2
      {
        temp_d=a_unscaled[4];                          /*dc's*/
        a_unscaled[4] = a_unscaled[6];
        a_unscaled[6] = temp_d;
        temp_d=a_unscaled[3];                          /*b's*/
        a_unscaled[3] = a_unscaled[5];
        a_unscaled[5] = temp_d;
        temp_d=std_a_unscaled[4];                      /*dc's std*/
        std_a_unscaled[4] = std_a_unscaled[6];
        std_a_unscaled[6] = temp_d;
        temp_d=std_a_unscaled[3];                      /*b's std*/
        std_a_unscaled[3] = std_a_unscaled[5];
        std_a_unscaled[5] = temp_d;
      }
    strcpy(buf,head.title);
    sprintf(mess1,"%d",rsp->vs_row-1);
    strcat(buf,"_");
    strcat(buf,mess1);
    cp = ctime(&tp);
    strcpy(mess1,cp);
    for(i=0;i<strlen(mess1);i++)      /*remove spaces from date*/
      if(mess1[i] == ' ')
        mess1[i] = '_';


    if(rsp->op_file_flag & 0x08) /*eights bit on means write data table*/
      {
        if(rsp->one_sv_flag)
          fprintf(data_table,"%8.6f,%8.6f,%9.6f,%9.6f,%9.6f,%9.6f,%8.4g,%8.4g,%9s,%9s,%8
s,%8s,%9.6f,%11g,%3g,%c,%s,%s",a_unscaled[1],std_a_unscaled[1],a_unscaled[2],std_a_unsc
d[2],a_unscaled[3],std_a_unscaled[3],a_unscaled[4],std_a_unscaled[4]," "," "," "," "  sp
amb,rsp->lin_term,wc,rsp->law,buf,mess1);
        else
          fprintf(data_table,"%8.6f,%8.6f,%9.6f,%9.6f,%9.6f,%9.6f,%8.4g,%8.4g,%9.6f,%9.6
f,%8.4g,%8.4g,%9.6f,%11g,%3g,%c,%s,%s",a_unscaled[1],std_a_unscaled[1],a_unscaled[2],std_a
_unscaled[2],a_unscaled[3],std_a_unscaled[3],a_unscaled[4],std_a_unscaled[4],a_unscaled[5]
,std_a_unscaled[5],a_unscaled[6],std_a_unscaled[6],rsp->amb,rsp->lin_term,wc,rsp->law,l
mess1);
        fclose(data_table);
      }
    }

  cfree(a);
  cfree(w);
  cfree(wv);
  cfree(std_a);
  cfree(std_a_unscaled);
  cfree(a_unscaled);
  free_dm(u,ndata,ma);
  free_dm(v,ma,ma);
  fclose(op_file);
  return(0);
}

#undef CLOSE


/*************************************************************************/
/* calculation of the chisq err for a given vector of model parameters a[i]'

        a[1] = mu_o, a[2]=a, a[3]=b1, a[4]=dc1, a[5]=b2, a[6]=dc2

        wv is a weighting vector, set in main()
```

```
 */
double calc_chisq(a, x, y, wv, ma, ndata, rsp)
     double      a[], x[], y[], wv[];
     int         ma, ndata;
     struct rs_parameters *rsp;
{

  char *calloc();
  int i;
  double dy, chisq, *mod_mu;
  int get_mu_at_x();
  void cfree();

  mod_mu = (double *) calloc((unsigned)ndata+1,sizeof(double));

  chisq=0.0;

  /*this function supplies mu at the displacements in x for the parameters in a'

  if(get_mu_at_x(x,mod_mu,ndata,rsp)==-1)
    return(-1); /*signal problem to caller*/

  for(i=1;i<=ndata;i++)
    {
      dy = y[i]-mod_mu[i];
      chisq += dy*dy*wv[i];
    }

  cfree(mod_mu);

  return(chisq);
}


/*************************************************************************/

double **dm(row,col)
     int row,col;
{

   int i;
  double **m;

  m = (double**)calloc((unsigned)row+1,sizeof(double*));
  for(i=0;i<=row;++i)
    {
      m[i] = (double*)calloc((unsigned)col+1,sizeof(double));
      if(!m[i])
        {
          sprintf(msg, "dm(): allocation error, memory full? \n");
          print_msg(msg);
          return NULL;
        }
    }
  return m;
}


/*************************************************************************/

void free_dm(m,row,col)
     double **m;
     int row,col;
{
```



```
  int i;

  for(i=0;i<=row;++i)
    cfree(m[i]);
  cfree(m);
}


/*************************************************************************/
/*Below are functions for the svd and lm solution to
  the non-linear inverse problem. Several aspects of the approach
  are based on the method of Reinen and Weeks 1993, JGR 98, 15937-15950.
  */

/* a[1] = mu_o, a[2]=a, a[3]=b1, a[4]=log(dc1), a[5]=b2, a[6]=log(dc2) */

/*
   da is the change in model parameters used to calculated the derivative
   a_corr is the correction to the model parameters a based on the linearized s
   to the inverse problem.
   */
#define DA_TOL 1.0e-10    /*probably no point in making it smaller than this*/
#define DELTA 0.01

int msvdfit(x,y,wv,ndata,a,ma,u,v,w,chisq,rsp,lamda)
     double x[],y[],wv[],a[],**u,**v,w[],*chisq,*lamda;
     int ndata,ma;
     struct rs_parameters *rsp;
{
  char *calloc();
  char covar_err=0,tmp_str[4096],err_str[4096];
  int i, j, k;
  int do_lm_reduce;
  int get_mu_at_x();
  static double wmax,thresh, *b, dyda, **alpha, *beta, **u_norm, **alpha_norm, **v_
alpha_inv;
  static double *jac_norm, *a_corr, *a2, *atry, *da, ochisq, *mod_mu, *mod_mu2;
  double calc_chisq(), **dm(), sqrt(), exp();
  int msvdcmp();
  void msvbksb();
  void set_rs_parameters();
  void free_dm(), zero_sv();

  if(*lamda<0)                    /*for initialization*/
    {
      do_lm_reduce=0;
      *lamda *= -1;               /* change sign */
      alpha = dm(ma,ma);
      alpha_norm = dm(ma,ma);
      alpha_inv = dm(ma,ma);
      u_norm = dm(ndata,ma);
      v_tmp = dm(ma,ma);
      beta = (double *) calloc((unsigned)ma+1,sizeof(double));
      b = (double *) calloc((unsigned)ndata+1,sizeof(double));
      mod_mu = (double *) calloc((unsigned)ndata+1,sizeof(double));
      mod_mu2 = (double *) calloc((unsigned)ndata+1,sizeof(double));
      jac_norm = (double *) calloc((unsigned)ma+1,sizeof(double));
      a2 = (double *) calloc((unsigned)ma+1,sizeof(double));
      da = (double *) calloc((unsigned)ma+1,sizeof(double));     /* for derivative st
ze */
      a_corr = (double *) calloc((unsigned)ma+1,sizeof(double)); /*corrections to cu
a*/
      atry = (double *) calloc((unsigned)ma+1,sizeof(double));
```



```
      for(k=1;k<=ma;++k)
        a_corr[k]= (k==4 || k==6) ? 10*exp(a[k]) : 10*a[k];     /*make 10x a so th
arts with 10% of a*/
      return(0);                                                /* use rea
l dc's, which can't be zero */
    }

  for(k=1;k<=ma;k++) /* derivative step size is 0.01 of last correction to model
    da[k] = (fabs(a_corr[k]*DELTA) < DA_TOL) ? DA_TOL : a_corr[k]*DELTA;

  ochisq=*chisq;                        /*chisq for current a is sent here*/

  /*calculate the Jacobian matrix of partial */
  /*derivatives: dy[i]/da[j]; i:1,ndata,j:1,ma*/
  if(get_mu_at_x(x,mod_mu,ndata,rsp) == -1)
    return(-1);

  for(j=1;j<=ma;j++)
    {
      for(k=1;k<=ma;k++)                /*set up for derivative*/
        a2[k] = (k==j) ? a[k]+da[k] : a[k];
      set_rs_parameters(rsp,a2,da);     /* set rs params */
      /*get mu for new a vector*/
      if(get_mu_at_x(x,mod_mu2,ndata,rsp) == -1)
        return(-1);
      for(i=1;i<=ndata;i++)                     /* calc. jacobian matrix */
        u[i][j] = (mod_mu2[i]-mod_mu[i])/da[j]; /*do weighting with alpha*/
    }

  /*this is vector of residuals for a*/
  for (i=1;i<=ndata;i++)
    b[i]=(y[i]-mod_mu[i]);              /*unweighted at this point*/

  for(j=1;j<=ma;++j)                    /*normalize jacobian -follow KWDM*/
    {
      jac_norm[j]=0.0;
      for(i=1;i<=ndata;i++)
        jac_norm[j] += u[i][j]*u[i][j];
      jac_norm[j] = sqrt(jac_norm[j]);
      for(i=1;i<=ndata;i++)
        u_norm[i][j] = u[i][j]/jac_norm[j]; /* u_norm is normalized jacobian */
    }

  for(i=1;i<=ma;++i)                    /*alpha_norm = (u_normT wvT u_norm) */
    {
      for(j=1;j<=ma;++j)
        {
          alpha[i][j]=alpha_norm[i][j]=0;
          for(k=1;k<=ndata;k++)
            {
              alpha_norm[i][j] += u_norm[k][i]*wv[k]*u_norm[k][j];
              alpha[i][j] += u[k][i]*wv[k]*u[k][j];
            }
        }
      alpha_norm[i][i] += *lamda;               /*Levenberg-Marquardt*/
      alpha[i][i] += *lamda;            /*Levenberg-Marquardt*/
    }

  for(j=1;j<=ma;++j)                    /*calculate beta = (uT wvT b) */
    {
      beta[j]=0;
      for(k=1;k<=ndata;k++)
        beta[j] += u_norm[k][j]*wv[k]*b[k];     /*beta uses normalized u*/
    }
```



```
/*
   sprintf(msg, "alpha_norm u:\n");
   print_msg(msg);

   for(i=1;i<=ma; ++i)
   {
   for(j=1;j<=ma; ++j)
   sprintf(msg, "%f, ",alpha_norm[i][j]);
   sprintf(msg, "\n");
   }
   sprintf(msg, "\n");

   sprintf(msg, "da for deriv\n");
   for(i=1;i<=ma; ++i)
   sprintf(msg, " %f%c",da[i], ( (i % ma) ? '\t' : '\n'));
*/


  /* svd of alpha_norm into U=alpha_norm, w, v */
  if (msvdcmp(alpha_norm,ma,ma,w,v) == -1)
      return(-1);

  zero_sv(w,ma,1e-5,(rsp->op_file_flag & 0x1));     /* zero singular values */

  if(*lamda==0)                 /* signal convergence, finish up, clean up */
    {
      for(i=1;i<=ma;++i)        /*eigenvalues, account for svd zeros */
        w[i] = (w[i]) ? (1.0/w[i]) : 0;         /* use 1/w instead of 1/w^2 to
normalization */

      /*calculated covariance, put it in alpha_norm */
      for(j=1;j<=ma;j++)        /* use eq'n 14.3.20 from Num. Rec.*/
        {
          for(k=1;k<=ma;k++)
            {
              alpha_norm[j][k]=0;
              for(i=1;i<=ma;i++)
                alpha_norm[j][k] += v[j][i]*v[k][i]*w[i];
              /*remove normalization*/
              alpha_norm[j][k] /= (jac_norm[j]*jac_norm[k]);
            }
        }

      for(i=1;i<=ma;++i)
        for(j=1;j<=ma;j++)
          v[i][j] = alpha_norm[i][j];   /*return covar in v */

      /* get inverse of alpha (un norm) to check covar */
      if( msvdcmp(alpha,ma,ma,w,v_tmp) == -1)
        return(-1);

      zero_sv(w,ma,1e-8,(rsp->op_file_flag & 0x1));                     /* zero singula
alues */

      for(i=1;i<=ma;i++)
        w[i] = (w[i]) ? (1/w[i]) : 0;   /* use 1/w */

      for(i=1;i<=ma;i++)
        {
          for(j=1;j<=ma;j++)
            {
              alpha_inv[i][j]=0;
```

```
            for(k=1;k<=ma;k++)
              alpha_inv[i][j]  +=  v_tmp[i][k]*w[k]*alpha[j][k];
          }
      }

    strcpy(err_str,"row,col\t(1-alpha_inv/svd_covar)\n");
    k=0;
    for(i=1;i<=ma;i++)                /* compare covar from alpha_inv and covar fro
/w */
      for(j=1;j<=ma;j++)
        if(  fabs(1-(alpha_inv[i][j]/v[i][j]))  >  1e-3)
          {
            covar_err  =  1;
            k++;
            sprintf(tmp_str,"%d,%d:%16.6g%c",i,j,1-alpha_inv[i][j]/v[i][j],  (k  %  3)  ?''
' : '\n');
            strcat(err_str,tmp_str);
          }

    if(covar_err  &&  (rsp->op_file_flag  &  0x1))       /*verbose option*/
      {
        sprintf(msg,  "%cCovar from inverse of unnormalized alpha and eigenvector/ei
lues  disagree\nCheck  op_file  for  a  list\n",BELL);
        print_msg(msg);

        fprintf(op_file,"Covar from inverse of unnormalized alpha and eigenvector/ei
alues  disagree\nCheck  the  following  entries\n");
        fprintf(op_file,"%s\n\n",err_str);
      }

    free_dm(alpha,ma,ma);
    free_dm(alpha_norm,ma,ma);
    free_dm(alpha_inv,ma,ma);
    free_dm(u_norm,ndata,ma);
    free_dm(v_tmp,ma,ma);
    cfree(beta);
    cfree(b);
    cfree(mod_mu);
    cfree(mod_mu2);
    cfree(jac_norm);
    cfree(a2);
    cfree(da);
    cfree(a_corr);
    cfree(atry);
    return(0);                        /*stop  here*/
  }

 for(i=1;i<=ma;++i)               /*save  old  values  in  case  new  ones  are  ng*/
  a2[i]=a_corr[i];

 msvbksb(alpha_norm,w,v,ma,ma,beta,a_corr);  /*a_corr  will  contain  the  corrections  t
o  a*/

 for(i=1;i<=ma;++i)                   /*set  up  to  see  if  these  are  better*/
  {
    a_corr[i]  /=jac_norm[i];   /*  remove  normalization*/
    atry[i]  =  a[i]+a_corr[i];
  }


 set_rs_parameters(rsp,atry,a_corr);          /*  re-set  params  */
 /*check  if  better*/
 *chisq  =  calc_chisq(atry,  x,  y,  wv,  ma,  ndata,  rsp);
 if(*chisq  ==  -1)
```



```
    return(-1);

  if(*chisq  <  ochisq)
    {
      do_lm_reduce++;
      if(do_lm_reduce  >  2)
        {
          *lamda  *=  0.1;                        /*reduce  L-M  factor*/
          do_lm_reduce=0;
        }
      for(i=1;i<=ma;++i)
        a[i]  =  atry[i];          /*save  new  parameters  */
    }
  else
    {
      do_lm_reduce=0;
      fprintf(op_file,"in  svdfit:  chisq  worse  for  new  guesses,  prev_chisq=%f,  new  c
\n",ochisq,*chisq);
      *lamda  *=  10.0;           /*increase  L-M  factor*/
      *chisq  =  ochisq;          /*old  values  were  better  */
      for(i=1;i<=ma;++i)
        a_corr[i]=a2[i];
    }

  set_rs_parameters(rsp,a,a_corr);      /*  re-set  params  */

  /*
    fprintf(op_file,"new  params  for  a\n");
    for(i=1;i<=ma;  ++i)
    fprintf(op_file,"  %f%c",((i==4  ||  i==6)  ?  exp(a[i])  :  a[i]),  (i  %  ma)  ?    t
;
    sprintf(msg,  "new  params  for  a\n");
    for(i=1;i<=ma;  ++i)
    sprintf(msg,  "  %f%c",((i==4  ||  i==6)  ?  exp(a[i])  :  a[i]),  (i  %  ma)  ?  '\t'

    */
}
#undef  DA_TOL
#undef  DELTA



/*****************************************************************/

/*  general  function  to  set  a-b  and  mu_f.
    This  should  be  called  whenever  a,  b1,  b2  is  changed,  such  as  by  inversi...  r
    Callers  should  send  null  if  they  don't  want  to  update  from  a[]  or  da[]

    a[1]  =  mu_o,  a[2]=a,  a[3]=b1,  a[4]=log(dc1),  a[5]=b2,  a[6]=log(dc2)

    note  that  a[4]=dc1  and  a[6]=dc2  are  assumed  to  be  log(dc1)  and  log(dc2)

*/

#define  BIG_NUM  1e30

void  set_rs_parameters(rsp,a,da)
      struct  rs_parameters  *rsp;
      double  a[],  da[];
{

  double  log(),exp(),fabs();

  if(a  !=  NULL)
```



```
    {                                 /*  update*/
      rsp->muo  =  a[1];
      rsp->a  =  a[2];
      rsp->b1  =  a[3];
      rsp->dc1  =  exp(a[4]);          /*a[4]  is  really  log  of  dc1*/
      rsp->b2  =  (rsp->one_sv_flag)  ?  0.0  :  a[5];
      rsp->dc2  =  (rsp->one_sv_flag)  ?  BIG_NUM  :  exp(a[6]);  /*a[6]=log(dc1)*/
    }

  if(da  !=  NULL)
    {                 /*  these  are  size  of  last  step  (update)  from  svd  solution*
      rsp->a_step  =  da[2];
      rsp->b1_step  =  da[3];
      rsp->dc1_step  =  exp(a[4]+da[4])-exp(a[4]);
      rsp->b2_step  =  (rsp->one_sv_flag)  ?  0.0:  da[5];
      rsp->dc2_step  =  (rsp->one_sv_flag)  ?  0.0:  exp(a[6]+da[6])-exp(a[6]);
    }

  rsp->amb  =  rsp->a  -  (rsp->b1  +  rsp->b2);
  rsp->muf  =  rsp->muo  +  rsp->amb  *  log(rsp->vf/rsp->vo);

}
#undef  BIG_NUM


/*****************************************************************************/

#define  DISP_TOL  1e-3  /*  I  assume  this  is  in  microns  */
#define  ONE      1.0
#define  TWO      2.0
#define  SMALL_NUM  1e-30
#define  EPSILON  1e-5
#define PGROW   -0.200
#define  PSHRINK -0.250
#define  PCOR    1.0/15.0
#define  SAFETY  0.9
#define  ERRCON  6.0e-4

/*  This  function  calculates  friction  for  the  displacements  in  x[]
    (and  puts  it  in  mod_mu[])  for  4  Rate/State  friction  laws.

    Ruina  law  (old  method)
    mu  =  mu_init  +  a  ln(v/v_init)  +  b  psi  +  c  dx  ,  where  c  (lin_term)
    is  a  term  to  describe  linear  hardening/weakening
    d_psi/dt  =  -v/Dc[psi+  ln  (v/v_init)]
    d_mu  =  k[v  -  v_init]                ,  v  is  slider  velocity  after  step,  v_init
point  v

    See  office  notes  from  about  15/7/94  for  other  laws.

    a  is  the  ma  vector  of  rs  parameters:
    a[1]  =  mu_o,  a[2]=a,  a[3]=b1,  a[4]=log(dc1),  a[5]=b2,  a[6]=log(dc2)

    x  is  the  ndata  vector  of  displacements  at  which  mod_mu[i]  is  needed

    rsp  contains  things  like  stiffness  etc.  It's  defined  in  global.h
    rsp->vs_row  is  the  point  at  which  the  vel.  step  occurred  (in  the  look  file)

    Modifications:
    modified  to  handle  multiple  velocity  steps
    15/7/94  modified  to  handle  other  rs  laws
    25/7/94  modified  approach  used  for  multiple  velocity  steps,
    use  initial  vo  like  reference  velocity
    */
```



```
int  get_mu_at_x(x,  mod_mu,  ndata,  rsp)
     double  x[],  mod_mu[];
     int  ndata;
     struct  rs_parameters  *rsp;
{
  double  h,  hh,  next_h,  bdid,  tnow,  ttnd,  v;
  double  mu,  p_mu,  old_mu,  psi1,  p_psi1,  old_psi1,  psi2,  p_psi2,  old_psi2;
  double  disp;
  double  mu_err_scale,  psi_err_scale,  err,  max_err;
  double  mu_err,  psi1_err,  psi2_err;
  double  fabs(),  log(),  exp();
  double  constant[5];                  /*for  rk  calc*/
  int  i,  j,  h_err;
  int  do_rk();

  constant[0]=constant[1]=0.0;
  constant[2]=constant[3]=0.5;
  constant[4]=1.0;

  rsp->vf  =  rsp->vel_list[0];          /*reset  vf,  which  gets  changed  for  multiple
s  */

  mu_err_scale  =  rsp->muo;             /*  use  constant  fractional  errors*/
  psi_err_scale  =  fabs(log(rsp->vo/rsp->vf));

  /*  initial  time  step  */
  /*  This  is  the  first  point  after  (at)  the  velocity  step  */
  next_h  =  (x[(rsp->vs_row-rsp->first_row)+1]-x[(rsp->vs_row-rsp->first_row)])/(10.0*rsp-
>vo);

  rsp->vo_dc1  =  rsp->vf  /  rsp->dc1;          /*std  settings*/
  rsp->vo_dc2  =  rsp->vf  /  rsp->dc2;

  switch(rsp->law)                       /*  set  up  initial  value  for  psi*/
    {
      /*        in  case  of  multiple  steps,  use
                initial  vo  like  true  ref  velocity,  e.g.
                change  only  vf  for  each  additional  v  step  */

    case  'o':
      psi1  =  psi2  =  -log(rsp->vo/rsp->vf);
      break;
    case  'r';
    case  'd':
      psi1  =  rsp->dc1/rsp->vo;  /*this  will  be  the  same  for  theta  version  of      ar
*/
      psi2  =  rsp->dc2/rsp->vo;
      break;
    case  'p':
      rsp->vo_dc1  =  rsp->vf/(TWO*rsp->dc1);          /*modify  for  Perrin-Rice  law*/
      rsp->vo_dc2  =  rsp->vf/(TWO*rsp->dc2);
      psi1  =  (TWO*rsp->dc1)/rsp->vo;
      psi2  =  (TWO*rsp->dc2)/rsp->vo;
      break;
    case  'j':  /*check  this  later,  just  a  guess  now  1
5/7/94  */
      psi1  =  rsp->dc1/(rsp->vf+rsp->vo);
      psi2  =  rsp->dc2/(rsp->vf+rsp->vo);
      break;
    }

  if(rsp->one_sv_flag)
    rsp->vo_dc2  =  SMALL_NUM;  /*b2  is  zero  for  1sv  model,  set  dc2  to  very  large
```

```
rsp->k_vf=  rsp->stiff  *  rsp->vf;
mu  =  rsp->muo;                    /*  initial  mu,  muf  varies  with  a-b*/
v  =  rsp->vo;

for(i=1;i<=(rsp->vs_row-rsp->first_row);i++)  /*fill  with  prejump  mu*/
  mod_mu[i]  =  mu+rsp->lin_term*(x[i]-x[(rsp->vs_row-rsp->first_row)]);

i=(rsp->vs_row-rsp->first_row+1);              /*first  point  after  vs_row*/
disp  =  x[(rsp->vs_row-rsp->first_row)];

j=1;                                  /*j=0  is  the  initial  step*/

while(  i  <=  ndata  )
  {
   if(i==(rsp->vs_row_list[j]-rsp->first_row+1)  )    /*add  one  to  get  v  step  in  ri
lace*/
     {
       rsp->vf=rsp->vel_list[j];  /*new  *final*  velocity*/
       /*  use  muo  for  first  step  as  reference  velocity  */
       rsp->muf  =  rsp->muo  +  rsp->amb  *  log(rsp->vf/rsp->vo);        /*use  current
r  vo*/
       rsp->k_vf  =  rsp->stiff  *  rsp->vf;  /*new  constants*/

       rsp->vo_dc1  =  rsp->vf  /  rsp->dc1;            /*std  settings*/
       rsp->vo_dc2  =  rsp->vf  /  rsp->dc2;
       switch(rsp->law)
         {
         case  'p':
           rsp->vo_dc1  =  rsp->vf/(TWO*rsp->dc1);  /*modify  for  Perrin-Rice  law
*/
           rsp->vo_dc2  =  rsp->vf/(TWO*rsp->dc2);
           break;
         case  'j':  /*check  this  later,  just  a  gue
ss  now  15/7/94                                          */

           break;
         }
        if(rsp->one_sv_flag)
         rsp->vo_dc2  =  SMALL_NUM;  /*b2  is  zero  for  1sv  model,  set  dc2  to  very
mber*/
       j++;
     }

    ttnd  =  fabs(x[i]-x[i-1])/rsp->vf;        /*  time  to  next  data*/
    tnow=0;                                   /*  time  since  last  data*/

    while(tnow  <  ttnd)
      {
       h_err=TRUE;

       if(tnow+next_h  >=  ttnd)                        /*set  time*/
         h  =  ttnd-tnow;
       else
         h  =  next_h;

       while(h_err)
         {
           /*  save  old  values  */
           old_mu  =  p_mu  =  mu;
           old_psi1  =  p_psi1  =  psi1;
           old_psi2  =  p_psi2  =  psi2;

           hh  =  h/TWO;                            /*  half-steps*/
```



```
           if((do_rk(&mu,  &psi1,  &psi2,  &hh,  &v,  rsp,  constant)  ==  -1)  ||
             /*1st  half  step*/
             (do_rk(&mu,  &psi1,  &psi2,  &hh,  &v,  rsp,  constant)  ==  -1)  ||
             /*2nd  half  step,  using  updated  vals*/
             (do_rk(&p_mu,  &p_psi1,  &p_psi2,  &h,  &v,  rsp,  constant)  ==  -1)  /*  f
step,  updated  vals  in  p_  */)
           {
             sprintf(msg,  "\terror  from  do_rk,  row=%d,  exiting  early\t",i);
             print_msg(msg);

             fprintf(op_file,"\terror  from  do_rk,  row=%d,  exiting  early\t",i);
             rsp->vf  =  rsp->vel_list[0];
             return(-1);
           }

          if(!h)                                  /*time  step  too  small?  */
           {
             sprintf(msg,  "\terror  from  do_rk,  time_step  =  0,  row=%d,  exiting
",i);
             print_msg(msg);
             fprintf(op_file,"\terror  from  do_rk,  time_step  =  0,  row=%d,  exiting
y\t",i);
             rsp->vf  =  rsp->vel_list[0];
             return(-1);
           }

          max_err=0;                                /*  evaluate  error*/
          mu_err  =  mu-p_mu;
          err  =  fabs(mu_err/mu_err_scale);
          max_err  =  (err  >  max_err)  ?  err  :  max_err;

          psi1_err  =  psi1-p_psi1;
          err  =  fabs(psi1_err/psi_err_scale);
          max_err  =  (err  >  max_err)  ?  err  :  max_err;

          psi2_err  =  psi2-p_psi2;
          err  =  fabs(psi2_err/psi_err_scale);
          max_err  =  (err  >  max_err)  ?  err  :  max_err;

          max_err  /=  EPSILON;                  /*  scale  relative  to  tolerance*/

          if(max_err  <=1.0)
           /*step  succeeded,  compute  size  of  next  step*/
           {
             bdid  =  h;
             next_h        =  (max_err  >  ERRCON)  ?  SAFETY*h*exp(PGROW*log(max_err))
4.0*h;
             h_err=FALSE;
           }
          else
           {
             h  =  SAFETY*h*exp(PSHRINK*log(max_err));       /*truncation  error  too
ge,  reduce  step  size*/
             mu  =  old_mu;
             psi1  =  old_psi1;
             psi2  =  old_psi2;
           }
         }                          /*  loop  for  adaptive  step  size  control*/
        tnow  +=  bdid;
        mu  +=  mu_err*PCOR;                    /*  fifth  order  bit  */
        psi1  +=  psi1_err*PCOR;
        psi2  +=  psi2_err*PCOR;
        disp  +=  rsp->vf*bdid;
```



```
       }                          /*  loop  for  time  to  next  data  */

     if(  fabs(disp-x[1])  >  DISP_TOL  )
       {
         fprintf(op_file,"\tget_mu_at_x:  error,  displacement  not  correct,  row=%d,  mea
d_disp=%f,  numer_disp  =%f,  exiting  early\t",i+rsp->first_row,x[i],disp);
         return(-1);
       }

     mod_mu[i]  =  mu+rsp->lin_term*(x[i]-x[(rsp->vs_row-rsp->first_row)]);
     ++i;  /*  increment  data  counter  */

   }          /*  end  of  loop  for  data  points  */

  rsp->vf  =  rsp->vel_list[0];                 /*reset  vf*/
  return(0);    /*  success!  */
}
#undef  SMALL_NUM
#undef  ONE
#undef  TWO
#undef  PGROW
#undef  PSHRINK
#undef  PCOR
#undef  SAFETY
#undef  ERRCON
#undef  EPSILON
#undef  DISP_TOL


/*************************************************************************/

/*      This  func.  does  most  of  the  work  of  solving  the  coupled
        equations  for  rate/state  friction  and  elastic  interaction.  It's  called
        from  get_mu_at_x().
        a[1]  =  mu_0,  a[2]=a,  a[3]=b1,  a[4]=log(dc1),  a[5]=b2,  a[6]=log(dc2)
        */

#define  BIGNUM  1e10
#define  ONE     1.0
#define  TWO     2.0
#define  FIVE    5.0
#define  PI6     ONE/6.0

/*  these  are  defined  in  get_mu_at_x.c
        constant[0]=0;
        constant[1]=constant[2]=0.5;
        constant[3]=1;
        */

int  do_rk(mu,  psi1,  psi2,  H,  v,  rsp,  constant)
     double  *mu,  *psi1,  *psi2,  *H,  *v,  *constant;
     struct  rs_parameters  *rsp;
{

  int   calc_bombed=1;
  int    i;
  int   isnan();
  double        alpha,arg;
  double        J[5],  K[5],  M[5];
  double        w_psi1,  w_psi2,  w_mu;
  double        old_psi1,  old_psi2,  old_mu,  old_H,  old_v;
  double        exp(),  log();

  while(calc_bombed)
```



```
  {
   old_psi1  =  *psi1;  /*  save  last  vals  in  case  calc  bombs  ...*/
   old_psi2  =  *psi2;
   old_mu  =  *mu;
   old_H  =  *H;
   old_v  =  *v;
   J[0]=K[0]=M[0]=0;
   /*  Runga  Kutta  calc  */
   switch(rsp->law)
    {
    case  'o':                              /*old,  original  Runia  slip  law*/
     for(i=1;i<5;++i)
      {
        w_psi1  =      *psi1  +  J[i-1]*constant[i];
        w_psi2  =      *psi2  +  K[i-1]*constant[i];
        w_mu  =                  *mu  +  M[i-1]*constant[i];
        alpha  =  exp((w_mu  -  rsp->muf  -  rsp->b1*w_psi1  -  rsp->b2*w_psi2)/rsp->a);
        J[i]  =  (*H)  *  -rsp->vo_dc1  *  alpha  *  (w_psi1  +  log(alpha));
        K[i]  =  (*H)  *  -rsp->vo_dc2  *  alpha  *  (w_psi2  +  log(alpha));
        M[i]  =  (*H)  *  rsp->k_vf  *  (ONE  -  alpha)  ;
      }
     *psi1  +=  (J[1]  +  TWO*J[2]  +  TWO*J[3]  +  J[4])*PI6;
     *psi2  +=  (K[1]  +  TWO*K[2]  +  TWO*K[3]  +  K[4])*PI6;
     *mu  +=  (M[1]  +  TWO*M[2]  +  TWO*M[3]  +  M[4])*PI6;

     *v  =  rsp->vf  *  exp((*mu  -  rsp->muf  -  (rsp->b1*(*psi1))  -  (rsp->b2*(*psi2))
;
     break;

    case  'r':                      /*Ruina,  slip  law,  using  theta  as  sv*/
     for(i=1;i<5;++i)
      {
        w_psi1  =      *psi1  +  J[i-1]*constant[i];
        w_psi2  =      *psi2  +  K[i-1]*constant[i];
        w_mu  =                  *mu  +  M[i-1]*constant[i];
        alpha  =  exp((w_mu  -  rsp->muf  -  rsp->b1*log(rsp->vo_dc1*w_psi1)  -  rsp->b
p->vo_dc2*w_psi2)  )/rsp->a);
        arg  =  alpha  *  rsp->vo_dc1  *  w_psi1;
        J[i]  =  (*H)  *  -arg  *  log(arg);
        arg  =  alpha  *  rsp->vo_dc2  *  w_psi2;
        K[i]  =  (*H)  *  -arg  *  log(arg);
        M[i]  =  (*H)  *  rsp->k_vf  *  (ONE  -  alpha);
      }
     *psi1  +=  (J[1]  +  TWO*J[2]  +  TWO*J[3]  +  J[4])*PI6;
     *psi2  +=  (K[1]  +  TWO*K[2]  +  TWO*K[3]  +  K[4])*PI6;
     *mu  +=  (M[1]  +  TWO*M[2]  +  TWO*M[3]  +  M[4])*PI6;

     *v  =  rsp->vf  *  exp((*mu  -  rsp->muf  -  (rsp->b1*log(rsp->vo_dc1  *  *psi1))  -
og(rsp->vo_dc2  *  *psi2))/rsp->a);
     break;

    case  'd':                      /*Dieterich,  slowness  law*/
     for(i=1;i<5;++i)
      {
        w_psi1  =      *psi1  +  J[i-1]*constant[i];
        w_psi2  =      *psi2  +  K[i-1]*constant[i];
        w_mu  =                  *mu  +  M[i-1]*constant[i];
        alpha  =  exp((w_mu  -  rsp->muf  -  rsp->b1*log(rsp->vo_dc1*w_psi1)  -  rsp->b
p->vo_dc2*w_psi2)  )/rsp->a);
        arg  =  alpha  *  rsp->vo_dc1  *  w_psi1;
        J[i]  =  (*H)  *  (ONE  -  arg);
        arg  =  alpha  *  rsp->vo_dc2  *  w_psi2;
        K[i]  =  (*H)  *  (ONE  -  arg);
        M[i]  =  (*H)  *  rsp->k_vf  *  (ONE  -  alpha);
```

```
        }
      *psi1 += (J[1] + TWO*J[2] + TWO*J[3] + J[4])*P16;
      *psi2 += (K[1] + TWO*K[2] + TWO*K[3] + K[4])*P16;
      *mu += (M[1] + TWO*M[2] + TWO*M[3] + M[4])*P16;

      *v = rsp->vf * exp(*mu - rsp->muf - (rsp->bl*log(rsp->vo_dc1 * *psi1)) -
og(rsp->vo_dc2 * *psi2))/rsp->a);
      break;

    case 'j':                /*Rice Law*/
      sprintf(msg, "no rice law yet \n");
      print_msg(msg);
      return(-1); /* inform caller of problem*/
      break;

    case 'p':          /*Perrin-Rice Quadratic law*/
      for(i=1;i<5;++i)
        {
          w_psi1 =     *psi1 + J[i-1]*constant[i];
          w_psi2 =     *psi2 + K[i-1]*constant[i];
          w_mu =               *mu + M[i-1]*constant[i];
          alpha = exp((w_mu - rsp->muf - rsp->bl*log(rsp->vo_dc1*w_psi1) - rsp->b2*l
p->vo_dc2*w_psi2) )/rsp->a);
          arg = alpha * rsp->vo_dc1 * w_psi1;
          J[i] = (*H) * (ONE - arg*arg);
          arg = alpha * rsp->vo_dc2 * w_psi2;
          K[i] = (*H) * (ONE - arg*arg);
          M[i] = (*H) * rsp->k_vf * (ONE - alpha);
        }
      *psi1 += (J[1] + TWO*J[2] + TWO*J[3] + J[4])*P16;
      *psi2 += (K[1] + TWO*K[2] + TWO*K[3] + K[4])*P16;
      *mu += (M[1] + TWO*M[2] + TWO*M[3] + M[4])*P16;

      *v = rsp->vf * exp(*mu - rsp->muf - (rsp->bl*log(rsp->vo_dc1 * *psi1)) -
og(rsp->vo_dc2 * *psi2))/rsp->a);
      break;

    default :
      sprintf(msg, "no rs law chosen. --how could this have happened?\n");
      print_msg(msg);
      return(-1); /* inform caller of problem*/
    }

  if( isnan(*v) || isnan(*mu) || (fabs(*v) > BIGNUM) )
    {
      fprintf(op_file,"do_rk: calculation bombed, retry #%d,\tcurrent parameters are:\
%g, bl=%g, dc1=%g, b2=%g, dc2=%g,\nmu_o=%g, mu_f=%g, time step=%g, v=%g old_H=%g, o
\n",calc_bombed,rsp->a,rsp->bl,rsp->dc1,rsp->b2,rsp->dc2,rsp->muo,rsp->muf,*H,*v,old_H, ol
d_v);
      if(++calc_bombed <= 3 ) /* give up if problem persists */
        {
          sprintf(msg, "do_rk: calculation bombed out\tcurrent parameters are:\na=%g,
%g, dc1=%g, b2=%g, dc2=%g,\nmu_o=%g, mu_f=%g, time step=%g, v=%g old_H=%g, old_v=%g\
->a,rsp->bl,rsp->dc1,rsp->b2,rsp->dc2,rsp->muo,rsp->muf,*H,*v,old_H, old_v);
          print_msg(msg);
          fprintf(op_file,"do_rk: calculation bombed out\tcurrent parameters are:\na=%
bl=%g, dc1=%g, b2=%g, dc2=%g,\nmu_o=%g, mu_f=%g, time step=%g, v=%g old_H=%g, old_v=
rsp->a,rsp->bl,rsp->dc1,rsp->b2,rsp->dc2,rsp->muo,rsp->muf,*H,*v,old_H, old_v);
          return(-1); /* inform caller of problem*/
        }
      else
        {
          *psi1 = old_psi1; /* install old vals */
          *psi2 = old_psi2;
```



```
          *mu = old_mu;
          *H = old_H/FIVE; /* reduce H -stablize calc? */
          *v = old_v;
        }
    }
  else
    calc_bombed = FALSE;          /* all OK, stop and return */

 }      /* end of calc_bombed loop */
 return(0);
}
#undef ONE
#undef FIVE
#undef TWO
#undef P16
#undef BIGNUM

/*************************************************************************/

#include <math.h>

static double at,bt,ct;
#define PYTHAG(a,b) ((at=fabs(a)) > (bt=fabs(b)) ? \
                     (ct=bt/at,at*sqrt(1.0+ct*ct)) : (bt ? (ct=at/bt,bt*sqrt(1.0+ct ...,)
0.0))

static double maxarg1,maxarg2;
#define MAXIM(a,b) (maxarg1=(a),maxarg2=(b),(maxarg1) > (maxarg2) ?\
                    (maxarg1) : (maxarg2))
#define SIGN(a,b) ((b) >= 0.0 ? fabs(a) : -fabs(a))

int msvdcmp(a,m,n,w,v)
     double **a,*w,**v;
     int m,n;
{
  char *calloc();
  int flag,i,its,j,jj,k,l,nm;
  double c,f,h,s,x,y,z;
  double anorm=0.0,g=0.0,scale=0.0;
  double *rv1;

  if (m < n)
    {
      sprintf(msg, "nrerror: SVDCMP: You must augment A with extra zero rows\
      print_msg(msg);
      return(-1);
    }

  rv1=(double *)calloc((unsigned)n+1,sizeof(double));
  for (i=1;i<=n;i++)
    {
      l=i+1;
      rv1[i]=scale*g;
      g=s=scale=0.0;
      if (i <= m)
        {
          for (k=i;k<=m;k++) scale += fabs(a[k][i]);
          if (scale)
            {
              for (k=i;k<=m;k++)
                {
                  a[k][i] /= scale;
                  s += a[k][i]*a[k][i];
                }
```



```
              f=a[i][i];
              g = -SIGN(sqrt(s),f);
              h=f*g-s;
              a[i][i]=f-g;
              if (i != n)
                {
                  for (j=l;j<=n;j++)
                    {
                      for (s=0.0,k=i;k<=m;k++) s += a[k][i]*a[k][j];
                      f=s/h;
                      for (k=i;k<=m;k++) a[k][j] += f*a[k][i];
                    }
                }
              for (k=i;k<=m;k++) a[k][i] *= scale;
            }
        }

      w[i]=scale*g;
      g=s=scale=0.0;
      if (i <= m && i != n)
        {
          for (k=l;k<=n;k++) scale += fabs(a[i][k]);
          if (scale)
            {
              for (k=l;k<=n;k++)
                {
                  a[i][k] /= scale;
                  s += a[i][k]*a[i][k];
                }
              f=a[i][l];
              g = -SIGN(sqrt(s),f);
              h=f*g-s;
              a[i][l]=f-g;
              for (k=l;k<=n;k++) rv1[k]=a[i][k]/h;
              if (i != m)
                {
                  for (j=l;j<=m;j++)
                    {
                      for (s=0.0,k=l;k<=n;k++) s += a[j][k]*a[i][k];
                      for (k=l;k<=n;k++) a[j][k] += s*rv1[k];
                    }
                }
              for (k=l;k<=n;k++) a[i][k] *= scale;
            }
        }
      anorm=MAXIM(anorm,(fabs(w[i])+fabs(rv1[i])));
    }
  for (i=n;i>=1;i--)
    {
      if (i < n)
        {
          if (g)
            {
              for (j=l;j<=n;j++)
                v[j][i]=(a[i][j]/a[i][l])/g;
              for (j=l;j<=n;j++)
                {
                  for (s=0.0,k=l;k<=n;k++) s += a[i][k]*v[k][j];
                  for (k=l;k<=n;k++) v[k][j] += s*v[k][i];
                }
            }
          for (j=l;j<=n;j++) v[i][j]=v[j][i]=0.0;
        }
      v[i][i]=1.0;
```



```
      g=rv1[i];
      l=i;
    }
  for (i=n;i>=1;i--)
    {
      l=i+1;
      g=w[i];
      if (i < n)
        for (j=l;j<=n;j++) a[i][j]=0.0;
      if (g)
        {
          g=1.0/g;
          if (i != n)
            {
              for (j=l;j<=n;j++)
                {
                  for (s=0.0,k=l;k<=m;k++) s += a[k][i]*a[k][j];
                  f=(s/a[i][i])*g;
                  for (k=i;k<=m;k++) a[k][j] += f*a[k][i];
                }
            }
          for (j=i;j<=m;j++) a[j][i] *= g;
        }
      else
        {
          for (j=i;j<=m;j++) a[j][i]=0.0;
        }
      ++a[i][i];
    }
  for (k=n;k>=1;k--)
    {
      for (its=1;its<=30;its++)
        {
          flag=1;
          for (l=k;l>=1;l--)
            {
              nm=l-1;
              if (fabs(rv1[l])+anorm == anorm)
                {
                  flag=0;
                  break;
                }
              if (fabs(w[nm])+anorm == anorm) break;
            }
          if (flag)
            {
              c=0.0;
              s=1.0;
              for (i=l;i<=k;i++)
                {
                  f=s*rv1[i];
                  if (fabs(f)+anorm != anorm)
                    {
                      g=w[i];
                      h=PYTHAG(f,g);
                      w[i]=h;
                      h=1.0/h;
                      c=g*h;
                      s=(-f*h);
                      for (j=1;j<=m;j++)
                        {
                          y=a[j][nm];
                          z=a[j][i];
                          a[j][nm]=y*c+z*s;
```

```
                    a[j][i]=z*c-y*s;
                  }
                }
              }
            }
          z=w[k];
          if (l == k)
            {
              if (z < 0.0)
                {
                  w[k] = -z;
                  for (j=1;j<=n;j++)  v[j][k]=(-v[j][k]);
                }
              break;
            }
          if (its == 30)
            {
              sprintf(msg, "nrerror: No convergence in 30 SVDCMP iterations\n");
              print_msg(msg);
              return(-1);
            }
          x=w[l];
          nm=k-1;
          y=w[nm];
          g=rv1[nm];
          h=rv1[k];
          f=((y-z)*(y+z)+(g-h)*(g+h))/(2.0*h*y);
          g=PYTHAG(f,1.0);
          f=((x-z)*(x+z)+h*((y/(f+SIGN(g,f)))-h))/x;
          c=s=1.0;
          for (j=l;j<=nm;j++)
            {
              i=j+ 1;
              g=rv1[i];
              y=w[i];
              h=s*g;
              g=c*g;
              z=PYTHAG(f,h);
              rv1[j]=z;
              c=f/z;
              s=h/z;
              f=x*c+g*s;
              g=g*c-x*s;
              h=y*s;
              y=y*c;
              for (jj=1;jj<=n;jj++)
                {
                  x=v[jj][j];
                  z=v[jj][i];
                  v[jj][j]=x*c+z*s;
                  v[jj][i]=z*c-x*s;
                }
              z=PYTHAG(f,h);
              w[j]=z;
              if (z)
                {
                  z=1.0/z;
                  c=f*z;
                  s=h*z;
                }
              f=(c*g)+(s*y);
              x=(c*y)-(s*g);
              for (jj=1;jj<=m;jj++)
                {
```



```
                  y=a[jj][j];
                  z=a[jj][i];
                  a[jj][j]=y*c+z*s;
                  a[jj][i]=z*c-y*s;
                }
            }
          rv1[l]=0.0;
          rv1[k]=f;
          w[k]=x;
        }
    }
  cfree(rv1);
  return(0);
}

#undef SIGN
#undef MAXIM
#undef PYTHAG


/*****************************************************************/

void msvbksb(u,w,v,m,n,b,x)
     double **u,w[],**v,b[],x[];
     int m,n;
{
  char *calloc();
  int jj,j,i;
  double s,*tmp;

  tmp=(double *)calloc((unsigned)n+1,sizeof(double));
  for (j=1;j<=n;j++)
    {
      s=0.0;
      if (w[j])
        {
          for (i=1;i<=m;i++)  s += u[i][j]*b[i];
          s /= w[j];
        }
      tmp[j]=s;
    }
  for (j=1;j<=n;j++)
    {
      s=0.0;
      for (jj=1;jj<=n;jj++)  s += v[j][jj]*tmp[jj];
      x[j]=s;
    }
  cfree(tmp);
}


/*****************************************************************/

void zero_sv(w,ma,tol,op_flag)
     double w[],tol;
     int ma,op_flag;
{
  int j;
  double wmax, thresh;

  wmax=0.0; /* zero singular values */
  for (j=1;j<=ma;j++)
    if (w[j] > wmax)
      wmax=w[j];
```



```
  thresh=tol*wmax;
  for (j=1;j<=ma;j++)
    {
      if (w[j] < thresh)
        {
          if(op_flag)
            fprintf(op_file,"singular value zeroed, w[%d]=%g, wmax=%g, tol.=%g\n",j,w[j]
max,tol);
          w[j]=0.0;
        }
    }
}
```



```
/* simplex is a method for optimising coefficients in an equation.
 *
 * if the form of a function to fit some data is known, then values
 * for the coefficients in the equation can be found, according to
 * some error criterion, with total error better than some specified value
 *
 * this version needs to be compiled with functions
 *
 * void get_initial_values( temp, errormax, first_step, &n_param, &iter_max)
 *      double temp[] enters the initial guesses for the parameters
 *      double errormax[] gives the criteria for stopping the iteration
 *      float first_step[] enters the initial offsets to the guesses
 *      int n_param gives the # parameters being optimised
 *      int iter_max enters the max # iterations, if the stopping criteria are
 *
 * void table_write( temp)
 *      double temp[] passes the solution, so that a table of results may be p
 *
 * double error( funcname, temp, x, y, ndata)
 *      char funcname[] contains name of function to fit to data
 *      double temp[] contains a guess at up to MAX_PARAM coefficients
 *      float x[ndata][10], y[ndata] is the data read from a file specified as
line argument
 * --it returns a value for the total error between the predicted values and
 *
 * Simon Cox, June 1985.
 *          revised to accept multiple independent variables, May 1986.
 * Greg Boitmott
 *          adapted for use in lookv3 with some corrections, Sept 1988.
 *
   Chris Harone
            adapted for use with rate and state friction calculation, June 1

   Lokman Alwi
            adapted for use with xlook, Jan 1995.
 */

#include "global.h"
#include <time.h>
#include <sys/types.h>
#include <sys/timeb.h>
#include <strings.h>


/*-----------------------------------------------------------------*/

extern do_scm_2();


extern int action;
extern int simp_func_action;
extern char msg[MSG_LENGTH];

extern int l, first, last, temp_int, simp_xch[MAX_COL], simp_ych;
extern char t_string[300];

FILE  *op_file;
time_t tp;
int       done;
static int     n, i, j;
int       ndata, iter_max, scrn_write, sw_num;
int high[MAX_PARAM+1], low[MAX_PARAM+1];
float first_step[MAX_PARAM];
double centre[MAX_PARAM];
```

```
double  errormax[MAX_PARAM+1],  error(),  err[MAX_PARAM+1];
double  simp_rate_state_mod();
char    *strcat();
char    buf[80];
int  max_iter;


do_simp_func()
     /*(infile,free,xch,ych,funcname,first,last,max_iter)
       FILE  *infile;
       int  free,  xch[],  *ych;
       char  funcname[];
       int  first,  last,  max_iter;  */
{

  max_iter  =  temp_int;

  n_param  =  1;
  ndata  =  last  -  first  +  1;

  /*
   *  get  the  starting  guesses  for  the  parameters  ..  in  the  array  simp[0][]
   *  the  stopping  criteria  ..  in  the  array  errormax[]
   *  the  initial  offsets  ..  in  first_step[]
   *  the  #  of  parameters  being  optimised
   *  and  the  max  #  of  iterations  allowed
   *
   *  and  then  send  a  nicely  formatted  copy  to  outf
   */

  if(  strncmp(t_string,  "rs_fit",  6)  ==  0  )
    {
      if(n_param  ==  4)          /*  two  state  variable  fit  */
        {
          simp[0][0]  =  temp[0]  =  rs_param.a;     /*  these  are  set  in  look  */
          simp[0][1]  =  temp[1]  =  rs_param.bl;
          simp[0][2]  =  temp[2]  =  rs_param.dc1;
          simp[0][3]  =  temp[3]  =  rs_param.dc2;

          errormax[0]  =  rs_param.a_er;
          errormax[1]  =  rs_param.bl_er;
          errormax[2]  =  rs_param.dc1_er;
          errormax[3]  =  rs_param.dc2_er;
          errormax[4]  =  rs_param.total_er;

          first_step[0]  =  rs_param.a_step;
          first_step[1]  =  rs_param.bl_step;
          first_step[2]  =  rs_param.dc1_step;
          first_step[3]  =  rs_param.dc2_step;
        }

      else  if(n_param  ==  2)            /*  only  doing  1  state  variable  fit...  */
        {
          simp[0][0]  =  temp[0]  =  rs_param.a;     /*set  in  look  */
          simp[0][1]  =  temp[1]  =  rs_param.dc1;

          errormax[0]  =  rs_param.a_er;
          errormax[1]  =  rs_param.dc1_er;
          errormax[2]  =  rs_param.total_er;

          first_step[0]  =  rs_param.a_step;
          first_step[1]  =  rs_param.dc1_step;
        }
      else
```



```
        {
          sprintf(msg,"simplex  got  confused  about  whether  it  was  a  1  or  2  state
it\n");
          print_msg(msg);
        }

      iter_max  =  max_iter;
      scrn_write  =  (int)(max_iter/10  -1);       /*  keep  user  informed  of  progr
      sw_num  =  1;
      simp_func_final();

    }
  else                /*  -one  of  the  other  simplex  look  functions...  */
    {

      /*  get  args  for  get_initial_values
        get_initial_values(errormax,  first_step,  &iter_max);
        init_values_write(  errormax,stdout);

        then  calculate  the  initial  error
        for  (  j=0;  j<n_param;  ++j)  simp[0][j]  =  temp[j];

        then  call  simp_func_final()
        */

      set_left_footer("Type  the  maximum  number  of  iterations  needed");
      set_cmd_prompt("Max  Iter:  ");

      action  =  SIMP_FUNC_GET_MAX_ITER;
      return;
    }

}


do_get_initial_values(arg)
     char  arg[128];
{
  static  int  i;

  nocom(arg);

  sscanf(arg,  "%d",  &iter_max);

  for(i=0;  i<n_param;  ++i)
    {
      sscanf(arg,  "  %lf",&temp[i]);
    }

  for(i=0;  i<n_param;  ++i)
    {
      errormax[i]  =  0.0;
    }

  for(i=0;  i<n_param;  ++i)
    {
      sscanf(arg,  "  %f",&first_step[i]);
    }

  init_values_write(  errormax,stdout);

  for  (  i=0;  i<n_param;  ++i)  simp[0][i]  =  temp[i];
}
```



```
simp_func_final()
{

  simp[0][n_param]  =  error(  t_string,  simp_xch,  simp_ych,  ndata,  first);

  get_starting_simplex(t_string,first_step,simp_xch,simp_ych,ndata,first);
  sprintf(msg,  "starting  simplex:\n");
  print_msg(msg);
  printSimplex(stdout,  t_string);

  if(  strncmp(t_string,  "rs_fit",  6)  ==  0  )
    {
      strcpy(buf,"simplex_log_");
      strcat(buf,head.title);
      op_file  =  fopen(buf,  "a");        /*  append  to  default  file  */
        fprintf(op_file,"\n**********************************************\n\n");
      time(&tp);
      fprintf(op_file,"Fit  for  Exp:  %s  \t\t  Date:  %s\n",  head.title,  ctime(&tp));
      fprintf(op_file,"recs  fit:  %d  to  %d,  v_init.  =  %5.3f,  v_f  =  %5.3f,  mu_init.
 mu_f  =  %5.4f\nRows  %d  to  %d  weighted  by  %5.3f\tPeak  row  (%d)  weighted  by  %5.3
param.vs_row,  rs_param.last_row,  rs_param.vo,  rs_param.vf,  rs_param.muo,  rs_param.muf,
param.vs_row+1,  rs_param.weight_pts,  rs_param.weight,  rs_param.peak_row,  rs_param.weight
.000,  ((rs_param.weight_control  <  0)  ?  ("\n***Weighting  adjusted:  Pre-peak  overestim
  favored  over  underestimate.")  :  ("  "))  );
      if(n_param  ==  4)
        fprintf(op_file,"Simplex  parameters:  max_interations  %d,\n  Tolerances:  a=  %5.3
tbl=  %5.3g,\tdc1=  %5.3g,\tdc2=  %5.3g,\ttotal  error=  %5.3g\n  Init.  step  size:\t  a=  %
tbl=  %5.3g,\tdc1=  %5.3g,\tdc2=  %5.3g\n",  max_iter,  rs_param.a_er,  rs_param.bl_er,  rs_p
.dc1_er,  rs_param.dc2_er,  rs_param.total_er,  rs_param.a_step,  rs_param.bl_step,  rs_param
c1_step,  rs_param.dc2_step);
      else
        fprintf(op_file,"Simplex  parameters:  max_interations  %d,  Tolerances:  a=  %5.3g,\
c=  %5.3g\ttotal  error=  %5.3g\n  Initial  step  size:\t  a=  %5.3g,\tdc=  %5.3g\n",  max_it
_param.a_er,  rs_param.dc1_er,  rs_param.total_er,  rs_param.a_step,  rs_param.dc1_step);

      fprintf(op_file,  "starting  simplex:\n");
      printSimplex(op_file,  t_string);
    }
/*
 *  find  the  high  and  low  values  for  each  parameter
 */

  for  (  i=0;  i<=n_param;  ++i)  {          /*  initialise  */
    low[i]=0;
    high[i]=0;
  }
  order(  high,  low);

  /**********************************
   *  START  ITERATION
   */

  for  (  n=0,  done=0;  n<iter_max  &&  done==0;  ++n)
    {

      find_centroid(centre,high[n_param]);

      reflect_worst(centre,high[n_param]);

      /*
       *  TEST  IT:        BETTER  THAN  BEST?
       */
```



```
      if(  (temp[n_param]  =  error(t_string,simp_xch,simp_ych,ndata,first))  <=  simp[  w
aram]  ][n_param]  )
        {
          saveAs_new_vertex(  high[n_param]);
          expand_reflection(centre,high[n_param]);

          /*
           *  BETTER  STILL  ?
           */

          if(  (temp[n_param]  =  error(t_string,simp_xch,simp_ych,ndata,first))  <=  simp[
[n_param]  ][n_param]  )
            saveAs_new_vertex(  high[n_param]);
        }

      /*
       *  OR  ORLY  BETTER  THAN  WORST..?
       */

      else  if(  temp[n_param]  <=  simp[  high[n_param]  ][n_param]  )
        saveAs_new_vertex(  high[n_param]);

      /*
       *  IF  WORSE  THAN  WORST,..
       */

      else
        {
          contract_worst(centre,high[n_param]);

          /*
           *  NOW  BETTER  THAN  WORST?
           */

          if(  (temp[n_param]  =  error(t_string,simp_xch,simp_ych,ndata,first))  <=  simp[
h[n_param]  ][n_param]  )
            saveAs_new_vertex(  high[n_param]);
          else
            {
              contract_all(t_string,centre,low[n_param],simp_xch,simp_ych,ndata,first);
            }
        }

      /*
       *  CHECK  FOR  CONVERGENCE
       */

      order(  high,  low);
      for  (  i=0,  done=1;  i<=n_param;  ++i)  {
        err[i]  =  fabs(  (simp[  high[i]  ][i]  -  simp[  low[i]  ][i])/simp[  high[i]  ][i]
        /*  flag  to  stop  if  parameter  is  essentially  zero  */
        if  (  err[i]  >  errormax[i]  &&  simp[high[i]][i]  >  1e-7  )          done      %
      }
      if(  strncmp(t_string,  "rs_fit",  6)  ==  0  )
        {
          if(n==0)
            {
              sprintf(msg,"Iteration:  0  ");
              print_msg(msg);
            }
          else  if(n  >  scrn_write)
            {
              sw_num++;
```

```
            scrn_write  =  (max_iter/10)  *  sw_num  -1;
            sprintf(msg,"%d  ",  n);
            print_msg(msg);
          }
        if(  temp[n_param]  <  0  )
          {
            sprintf(msg,"\nSimplex  main  loop:  error  in  fit.  error()  returned  %g  -i
 loop  early....\n",  temp[n_param]);
            print_msg(msg);
            fprintf(op_file,"\nSimplex  main  loop:  error  in  fit.  error()  returned  %
ting  loop  early....\n",  temp[n_param]);
            done  =  1;
          }
      }
    }

  /****  END  ITERATION
   ************************************************
   *
   *  average  final  results
   *  and  send  the  results  to  outf
   */

  for  (  i=0;  i<=n_param;  ++i)  {
    temp[i]  =  0;
    for  (  j=0;  j<=n_param;  ++j)
      temp[i]  +=  simp[j][i];
    temp[i]  /=  (  n_param+1  );
  }

  solution_write(n,err,stdout,t_string);

  if(  strncmp(t_string,  "rs_fit",  6)  ==  0  )
    {
      solution_write(n,err,op_file,t_string);
      if(n_param  ==  2)
        {
          rs_param.a  =  temp[0];
          rs_param.dc1  =  temp[1];
          rs_param.b1  =  rs_param.a  -  rs_param.amb;
          fprintf(op_file,"\nBest  simplex  fit  (vs_row=%d):  a=  %g\tb=  %g\tdc=  %g\ta-b
n",  rs_param.vs_row,  rs_param.a,  rs_param.b1,  rs_param.dc1,rs_param.amb);
        }
      else  if(n_param  ==  4)
        {
          rs_param.a  =  temp[0];
          rs_param.b1  =  temp[1];
          rs_param.dc1  =  temp[2];
          rs_param.dc2  =  temp[3];
          rs_param.b2  =  -(rs_param.amb-rs_param.a+rs_param.b1);
          fprintf(op_file,"\nBest  simplex  fit  (vs_row=%d):  a=  %g\tb1=  %g\tb2=  %g\tdc1=
tdc2=  %g\ta-b  =  %g\n",  rs_param.vs_row,  rs_param.a,  rs_param.b1,  rs_param.b2,  rs_para
,  rs_param.dc2,rs_param.amb);
        }
      else
        {
          sprintf(msg,"simplex  got  confused  about  1  or  2  state  variable  model.  Numi
 probably  GARBAGE!\n");
          print_msg(msg);
        }

      simp_rate_state_mod();            /*  to  set  final  (averaged)  values  in  look  ar
      fclose(op_file);
    }
```



```
  switch(simp_func_action)
    {
    case  SCM:
      do_scm_2();
      break;

    case  SIMPLEX:
      sprintf(msg,  "SIMPLEX:  DONE\n");
      print_msg(msg);
      action  =  MAIN;
      top();
      break;

    }

}

/*-------------------------------------------------------------------*/

init_values_write(  errormax,outf)
     FILE       *outf;
     double     errormax[];
{
  int  i;

  if  (outf  ==  stdout)
    {
      sprintf(msg,  "starting  values:\n");
      print_msg(msg);

      for  (  i=0;  i<n_param;  ++i)
        {
          sprintf(msg,  "  %c  %.3e\n",  (65+i),  temp[i]);
          print_msg(msg);
        }
    }

  else
    {
      fprintf(outf,  "starting  values:\n");
      for  (  i=0;  i<n_param;  ++i)
        {
          fprintf(outf,  "  %c  %.3e\n",  (65+i),  temp[i]);
          /*
            fprintf(outf,  "  allowed  error:  %.2e\n",  errormax[i]);
          */
        }
      /*
        fprintf(outf,  "  total  allowed  error:  %.2e\n",  errormax[n_param]);
      */
    }
}


/*-------------------------------------------------------------------*/

get_starting_simplex(funcname,  first_step,p_xch,p_ych,ndata,first)
     char  funcname[];
     int        ndata,  p_xch[],  *p_ych;
     float      first_step[];
     int  first;
{
  int  i,j;
```



```
  float p[MAX_PARAM],q[MAX_PARAM];
  double        error();
  /*
   *  find  initial  vertices  offset
   */

  for  (  j=0;  j<n_param;  ++j)  {
    p[j]  =  first_step[j]*(  sqrt((float)n_param+1)  +  n_param  -  1)/(  n_param*sqrt(2.)
    q[j]  =  first_step[j]*(  sqrt((float)n_param+1)  -  1)/(  n_param*sqrt(2.)  );
  }

  /*
   *  find  starting  simplex
   */

  for  (  i=1;  i<=n_param;  ++i)  {
    for  (  j=0;  j<n_param;  ++j)  simp[i][j]  =  simp[0][j]  +  q[j];
    simp[i][i-1]  =  simp[0][i-1]  +  p[i-1];
    for  (  j=0;  j<n_param;  ++j)  temp[j]  =  simp[i][j];
    simp[i][n_param]  =  error(funcname,p_xch,p_ych,ndata,first);
  }
}
/*-------------------------------------------------------------------*/

solution_write(n,  err,  outf,  f_name)

    FILE       *outf;
    int        n;
    double     err[];
    char       f_name[];
{
  static  int    i,j;

  if  (outf  ==  stdout)
    {
      sprintf(msg,  ".......stopped  after  %d  iterations...\n",  n);
      print_msg(msg);

      sprintf(msg,  "The  final  simplex  is:\n");
      print_msg(msg);

      printSimplex(outf,  f_name);

      sprintf(msg,  "  the  mean  values  are:\n");
      print_msg(msg);
      for  (  i=0;  i<n_param;  ++i)
        {
          sprintf(msg,  "%14.5e",  temp[i]);
          print_msg(msg);
        }

      sprintf(msg,  "\n");
      print_msg(msg);

      sprintf(msg,  "  and  the  estimated  fractional  error  is:\n");
      print_msg(msg);

      for  (  i=0;  i<=n_param;  ++i)
        {
          sprintf(msg,  "%14.5e",  err[i]);
          print_msg(msg);
        }

      sprintf(msg,  "\n");
```



```
      print_msg(msg);
    }

  else
    {
      fprintf(outf,  ".......stopped  after  %d  iterations...\n",  n);
      fprintf(outf,  "The  final  simplex  is:\n");

      printSimplex(outf,  f_name);

      fprintf(outf,  "  the  mean  values  are:\n");
      for  (  i=0;  i<n_param;  ++i){
        fprintf(outf,  "%14.5e",  temp[i]);
      }
      fprintf(outf,  "\n");
      fprintf(outf,  "  and  the  estimated  fractional  error  is:\n");
      for  (  i=0;  i<=n_param;  ++i)
        fprintf(outf,  "%14.5e",  err[i]);
      fprintf(outf,  "\n");
    }
}

/*-------------------------------------------------------------------*/

printSimplex(outf,  function_name)
     FILE       *outf;
     char       function_name[];
{
  int  i,j;

  if(  strncmp(function_name,  "rs_fit",  6)  ==  0  )
    {
      if(  n_param  ==  4)
        fprintf(outf,"  a  b1  dc1  dc2  mis
fit\n");
      else
        fprintf(outf,"  a  dc  misfit\n");

    }
  else
    {
      for  (  i=0;  i<n_param;  ++i)
        fprintf(outf,  "  %c  ",(65+i));
      fprintf  (outf,  "  misfit\n");
    }
  for  (  i=0;  i<=n_param;  ++i)  {
    for  (  j=0;  j<=n_param;  ++j)
      fprintf(outf,  "%14.5e",  simp[i][j]);
    fprintf(outf,  "\n");
  }
  fprintf(outf,  "\n");
}
/*-------------------------------------------------------------------*/

order  (  high,  low)
     int        high[],  low[];
{
  int  i,  j;

  for  (  i=0;  i<=n_param;  ++i)
    {
      for  (  j=0;  j<=n_param;  ++j)
        {
          if  (  simp[i][j]  <  simp[  low[j]][j]  )  low[j]  =  i;
```

```
        if ( simp[i][j] > simp[ high[j]][j] ) high[j] = i;
      }
    }
}

/*-------------------------------------------------------------------*/

find_centroid(centre,h)
    int       h;
    double    centre[];
{
  int  i,j;

  for ( i=0; i<n_param; ++i)
    centre[i] = 0;              /* initialise */
  for ( j=0; j<=n_param; ++j)
    {
      if ( j != h )
        {               /* exclude worst */
          for ( i=0; i<n_param; ++i)
            centre[i] += simp[j][i];
        }
    }
  for ( i=0; i<n_param; ++i) centre[i] /= n_param;
}
/*-------------------------------------------------------------------*/

reflect_worst(centre,h)
    int       h;
    double    centre[];
{
  int  i;

  for ( i=0; i<n_param; ++i)
    {
      temp[i] = ( ALPHA+1 )*centre[i] - ALPHA*simp[h][i];
    }
}
/*-------------------------------------------------------------------*/

expand_reflection(centre,h)
    int       h;
    double    centre[];
{
  int  i;

  for ( i=0; i<n_param; ++i)
    {
      temp[i] = GAMMA*simp[h][i] + ( 1-GAMMA )*centre[i];
    }
}
/*-------------------------------------------------------------------*/

contract_worst(centre,h)
    int       h;
    double    centre[];
{
  int  i;

  for ( i=0; i<n_param; ++i)
    {
      temp[i] = BETA*simp[h][i] + ( 1-BETA )*centre[i];
    }
}
```



```
/*-------------------------------------------------------------------*/

contract_all(funcname,centre,l,p_xch,p_ych,ndata,first)
    char      funcname[];
    int       l,ndata;
    int       p_xch[],  *p_ych;
    double    centre[];
    int  first;
{
  int  i,j;

  for ( i=0; i<n_param; ++i)
    {
      for ( j=0; j<n_param; ++j)
        {
          simp[i][j] = (simp[i][j] + simp[l][j])*BETA;
          temp[j] = simp[i][j];
        }
      simp[i][n_param] = error( funcname, p_xch, p_ych, ndata, first);
    }
}
/*-------------------------------------------------------------------*/

saveAs_new_vertex (h)
    int       h;
{
  int i;

  for ( i=0; i<n_param; ++i)  simp[h][i] = temp[i];
}

/*-------------------------------------------------------------------*/
/*
get_initial_values(errormax, first_step, p_iter_max)
    double  errormax[];
    float  first_step[];
    int  *p_iter_max;
{
  static int i;
  sprintf(msg,"maximum number of iterations -> ");

  fscanf(infile,"%d",p_iter_max);
  sprintf(msg,"\ninitial guesses for free parameters\n");
  for(i=0; i<n_param; ++i)
    {
      if(infile==stdin) sprintf(msg,"%c -> ",(char)(65+i));
      fscanf(infile,"%lf",&temp[i]);
    }
 *
 *.....................................................................
 *    BECAUSE of lack of use, error criterion for end of iteration is dis   d
 *          To enable, input of maximum total error must be added
 *          and a few print statements which have been commented out
 *          must be re-instated.
 *.....................................................................
 *
 *    if(infile==stdin) sprintf(msg,"desired precision of free parameters\n");
 *    for(i=0; i<n_param; ++i)
 *    {
 *          if(infile==stdin) sprintf(msg,"%c -> ",(char)(65+i));
 *          fscanf(infile,"%lf",&errormax[i]);
 *    }
 *
  for(i=0; i<n_param; ++i)
```



```
    {
      errormax[i] = 0.0;
    }
  if(infile==stdin) sprintf(msg,"initial step-size for each parameter\n");
  for(i=0; i<n_param; ++i)
    {
      if(infile==stdin) sprintf(msg,"%c -> ",(char)(65+i));
      fscanf(infile,"%f",&first_step[i]);
    }
  if(infile==stdin) sprintf(msg,"\n");
}
*/
/*-------------------------------------------------------------------*/

double  error(name,p_xch,p_ych,nd,first)
    char  name[];
    int  p_xch[],  *p_ych,  nd,  first;
{
  static  int  i;
  int  ob_flag;  /* FALSE means simplex parameters are *not* out of bounds */
  static  double  u,  s;
  double  err;
  double  simp_rate_state_mod(), power1(), power2(), normal(), chisqr(), scchisqr(), g
(), ExpLin(), gensin(), rclow(), rcph(), Poly4();


  if(strncmp(name,"rs_fit",6)==0)
    {
      ob_flag = FALSE;
      if(n_param == 2)
        {
          /* constrain a, dc */
          if(temp[0] < 0 || temp[1] < 0.01)
            ob_flag = TRUE;
          rs_param.a = temp[0];
          rs_param.dc1 = temp[1];
          rs_param.b1 = rs_param.a - rs_param.amb;
        }
      else if(n_param == 4)
        {
          /* constrain a, b1, dc1, or dc2 */
          if(temp[0] < 0 || temp[1] < 0 || temp[2] < 0.01 || temp[3] < 0.1)
            ob_flag = TRUE;
          rs_param.a = temp[0];
          rs_param.b1 = temp[1];
          rs_param.dc1 = temp[2];
          rs_param.dc2 = temp[3];
          rs_param.b2 = -(rs_param.amb-rs_param.a+rs_param.b1);
        }
      else
        sprintf(msg,"simplex got confused about 1 or 2 state variable model. Numbe
robably GARBAGE!\n");

      if(!ob_flag)
        /* parameters defined in the global structure rs_param */
        err = simp_rate_state_mod();
      else
        err = 1000;

    }
  else if(strncmp(name,"Power1",6)==0)
    {
      err = 0.0;
      for(i=first; i<first+nd; ++i)
```



```
        {
          u = power1((double)darray[p_xch[0]][i],temp[0],temp[1]);
          s = (double)darray[*p_ych][i];
          err += fabs(u-s);
        }
    }
  else if(strncmp(name,"Power2",6)==0)
    {
      err = 0.0;
      for(i=first; i<first+nd; ++i)
        {
          u = power2((double)darray[p_xch[0]][i],temp[0],temp[1],temp[2]);
          s = (double)darray[*p_ych][i];
          err += fabs(u-s);
        }
    }
  else if(strncmp(name,"normal",6)==0)
    {
      err = 0.0;
      for(i=first; i<first+nd; ++i)
        {
          u = normal((double)darray[p_xch[0]][i],temp[0],temp[1]);
          s = (double)darray[*p_ych][i];
          err += fabs(u-s);
        }
    }
  else if(strncmp(name,"chisqr",6)==0)
    {
      err = 0.0;
      for(i=first; i<first+nd; ++i)
        {
          u = chisqr((double)darray[p_xch[0]][i],temp[0]);
          s = (double)darray[*p_ych][i];
          err += fabs(u-s);
        }
    }
  else if(strncmp(name,"scchisqr",8)==0)
    {
      err = 0.0;
      for(i=first; i<first+nd; ++i)
        {
          u = scchisqr((double)darray[p_xch[0]][i],temp[0],temp[1],temp[2]);
          s = (double)darray[*p_ych][i];
          err += fabs(u-s);
        }
    }
  else if(strncmp(name,"genexp",6)==0)
    {
      err = 0.0;
      for(i=first; i<first+nd; ++i)
        {
          u = genexp((double)darray[p_xch[0]][i],temp[0],temp[1],temp[2],temp[3]);
          s = (double)darray[*p_ych][i];
          err += fabs(u-s);
        }
    }
  else if(strncmp(name,"Poly4",5)==0)
    {
      err = 0.0;
      for(i=first; i<first+nd; ++i)
        {
          u = Poly4((double)darray[p_xch[0]][i],temp[0],temp[1],temp[2],temp[3],temp[
          s = (double)darray[*p_ych][i];
          err += fabs(u-s);
```

```
        }
    }
  else  if(strncmp(name,"gensin",6)==0)
    {
      err  =  0.0;
      for(i=first;  i<first+nd;  ++i)
        {
          u  =  gensin((double)darray[p_xch[0]][i],temp[0],temp[1],temp[2],temp[3]);
          s  =  (double)darray[*p_ych][i];
          err  +=  fabs(u*u-s*s);
        }
    }
  else  if(strncmp(name,"ExpLin",6)==0)
    {
      err  =  0.0;
      for(i=first;  i<first+nd;  ++i)
        {
          u  =  ExpLin((double)darray[p_xch[0]][i],temp[0],temp[1],temp[2],temp[3]);
          s  =  (double)darray[*p_ych][i];
          err  +=  fabs(u*u-s*s);
        }
    }
  else  if(strncmp(name,"rclow",5)==0)
    {
      err  =  0.0;
      for(i=first;  i<first+nd;  ++i)
        {
          u  =  rclow((double)darray[p_xch[0]][i],temp[0],temp[1]);
          s  =  (double)darray[*p_ych][i];
          err  +=  fabs(u*u-s*s);
        }
    }
  else  if(strncmp(name,"rcph",4)==0)
    {
      err  =  0.0;
      for(i=first;  i<first+nd;  ++i)
        {
          u  =  rcph((double)darray[p_xch[0]][i],temp[0],temp[1]);
          s  =  (double)darray[*p_ych][i];
          err  +=  fabs(u*u-s*s);
        }
    }
  else  sprintf(msg,"can't  find  function\n");
  return(err);
}
```



```
void  sort(n,ra)
int  n;
float  ra[];
{
        int  l,j,ir,i;
        float  rra;

        l=(n  >>  1)+1;
        ir=n;
        for  (;;)  {
                if  (l  >  1)
                        rra=ra[--l];
                else  {
                        rra=ra[ir];
                        ra[ir]=ra[1];
                        if  (--ir  ==  1)  {
                                ra[1]=rra;
                                return;
                        }
                }
                i=l;
                j=l  <<  1;
                while  (j  <=  ir)  {
                        if  (j  <  ir  &&  ra[j]  <  ra[j+1])  ++j;
                        if  (rra  <  ra[j])  {
                                ra[i]=ra[j];
                                j  +=  (i=j);
                        }
                        else  j=ir+1;
                }
                ra[i]=rra;
        }
}
```



```
#include  "global.h"
#include  "nr.h"
/*  ------------------------------------------------------------------------*/
void  line(colx,coly,first,last,result)
int  *colx  ,  *coly  ,  *first  ,  *last;
float  result[];
{
double  sumx=0.0  ,  sumy=0.0  ,  sumxy=0.0  ,  sumx2=0.0  ;
int  i  ;

    for(  i  =  (*first);  i  <=  (*last);  ++i  )
    {
        sumx  +=  darray[*colx][i]  ;
        sumy  +=  darray[*coly][i]  ;
        sumxy  +=  darray[*colx][i]  *  darray[*coly][i]  ;
        sumx2  +=  darray[*colx][i]  *  darray[*colx][i]  ;
    }

    i  =  *last  -  *first  +  1;
    result[0]  =  (sumx  *  sumy  -  (double)i  *  sumxy)  /
                          (sumx  *  sumx  -  (double)i  *  sumx2)  ;
    result[1]  =  (sumy  -  result[0]  *  sumx)  /  (double)i  ;
}
/*------------------------------------------------------------------------*/
void  stats(col,first,last)
     int  *col  ,  *first  ,  *last  ;
{
  int  i;
  double  rec;
  double  max,  min;
  double  mean,  sum_sq,  sum_dev;
  double  sqrt();

  max  =  (double)darray[*col][*first]  ;
  min  =  (double)darray[*col][*first]  ;
  mean  =  sum_sq  =  0.0;
  sum_dev  =  0.0;
  for(  i  =  (*first);  i  <=  (*last);  ++i)
    {
      max  =  (darray[*col][i]<(float)max)?max:(double)darray[*col][i]  ;
      min  =  (darray[*col][i]<(float)min)?(double)darray[*col][i]:min  ;
      mean  +=  (double)darray[*col][i]  ;
      sum_sq  +=  (double)darray[*col][i]  *  (double)darray[*col][i]  ;
    }
  rec  =  (double)(*last)  -  (double)(*first)  +  1.0;
  i  =  *last  -  *first  +  1;
  col_stat.rec  =  i  ;
  col_stat.mean  =  (float)(mean  /  rec)  ;
  col_stat.max  =  (float)max  ;
  col_stat.min  =  (float)min  ;
  for(  i  =  (*first);  i  <=  (*last);  ++i)
    {
      sum_dev  +=  pow((double)darray[*col][i]-col_stat.mean),2.0);
    }
  col_stat.stddev  =  (float)sqrt(  (sum_dev  /  (rec-1.0)  )  );
}

/*  ------------------------------------------------------------------------*/
void  median_smooth(col,new_col,start,end,window)
int  col,  new_col,  start,  end,  window;
{
static  float  *data,  median;
int     i,j,k;
int     half_window;
```



```
    void    median1();

            window=2*(int)(window/2)  +  1;  /*  force  wind_size  to  be  odd  */
            half_window  =  (int)(window/2);

            data  =  (float  *)calloc(window+1,sizeof(float));

            for(j=start,  i=0;  i<  half_window;  ++i)
                    darray[new_col][j]  =  darray[col][j++];

            for(j=start+half_window,  i=start  ;  i<=end-window+1;  ++i)
            {
                    for(k=0;  k  <  window;  ++k)
                            data[k+1]  =  darray[col][i+k];

                    median1(data,  window,  &median);
                    darray[new_col][j++]  =  median;
            }

            for(i=end-(int)(window/2);  i<=end;  ++i)
                    darray[new_col][i]  =  darray[col][i];

            cfree(data);
    }
    /*  ------------------------------------------------------------------------*/
    void  smooth(col,new_col,start,end,window)
    int  col,  new_col,  start,  end,  window;
    {

            double  sum;
            int  i,j,k,half_wind;

            for(i=start;  i<=  end;  ++i)
                    darray[0][i]  =  darray[col][i];


            k  =  half_wind  =  0;
            sum  =  0;

            window=2*(int)(window/2)  +  1;  /*  force  wind_size  to  be  odd  */
            half_wind  =  (int)window/2;


            for(i=0;  i  <  half_wind;  ++i)
                    sum  +=  darray[0][start];


            for(i=0;  i  <  window;  ++i){
              if(i  <  half_wind)
                    sum  +=  darray[0][start  +  i];

              else  {
                     sum  +=  darray[0][start  +  i];
                    darray[new_col][start  +  k]  =  sum  /  (double)window;
                    sum  -=  darray[0][start];
                    ++k;
              }
            }

            j  =  window  -1;

            for(i=start  +  window;  i  <=  end;  ++i){
                    sum  +=  darray[0][i];
                    darray[new_col][i-half_wind]  =  sum  /  (double)window;
```

```
            sum -= darray[0][i-j];
        }

        for(i=1; i <= half_wind; ++i){
            sum += darray[0][end];
            darray[new_col][end - half_wind +i] = sum / (double)window;
            sum -= darray[0][end-window];
        }
}
/* ----------------------------------------------------------------------*/
void o_slope(col1,col2,new_col,start,end,window)
int col1, col2, new_col, start, end, window;
{

        double xsum, xyprod, ysum, xsqr;
        float *y;
        int i,j,k,half_wind;

        y = (float *) calloc(head.nrec,sizeof(float));

        for(i=start; i<= end; ++i)
        {
            darray[0][i] = darray[col1][i];
            y[i] = darray[col2][i];
        }

        k = half_wind = 0;
        xsum = ysum = xyprod = xsqr = 0;

        window=2*(int)(window/2) + 1; /* force wind_size to be odd */
        half_wind = (int)window/2;


        for(i=0; i < half_wind; ++i)
        {
            xsum += darray[0][start];
            xsqr += darray[0][start] * darray[0][start];
            ysum += y[start];
            xyprod += darray[0][start] * y[start];
        }

        for(i=0; i < window; ++i)
        {
          if(i < half_wind)
          {
            xsum += darray[0][start + i];
            xsqr += darray[0][start +i] * darray[0][start +i];
            ysum += y[start + i];
            xyprod += darray[0][start +i] * y[start + i];
          }
          else
          {
            xsum += darray[0][start + i];
            xsqr += darray[0][start +i] * darray[0][start +i];
            ysum += y[start + i];
            xyprod += darray[0][start +i] * y[start + i];
            darray[new_col][start + k] = ((double)window*xyprod-xsum*ysum)/((double)w
ndow*xsqr-xsum*xsum);
            xsum -= darray[0][start];
            xsqr -= darray[0][start] * darray[0][start];
            ysum -= y[start];
            xyprod -= darray[0][start] * y[start];
            ++k;
```



```
          }
        }

        j = window -1;

        for(i=start + window; i <= end; ++i)
        {
            xsum += darray[0][i];
            xsqr += darray[0][i] * darray[0][i];
            ysum += y[i];
            xyprod += darray[0][i] * y[i];
            darray[new_col][i-half_wind] = ((double)window*xyprod-xsum*ysum)/((double)
window*xsqr-xsum*xsum);
            xsum -= darray[0][i-j];
            xsqr -= darray[0][i-j] * darray[0][i-j];
            ysum -= y[i-j];
            xyprod -= darray[0][i-j] * y[i-j];
        }


        for(i=1; i <= half_wind; ++i){
            xsum += darray[0][end];
            xsqr += darray[0][end] * darray[0][end];
            ysum += y[end];
            xyprod += darray[0][end] * y[end];
            darray[new_col][end - half_wind +i] = ((double)window*xyprod-xsum*ysum)
double)window*xsqr-xsum*xsum);
            xsum -= darray[0][end-window];
            xsqr -= darray[0][end-window] * darray[0][end-window];
            ysum -= y[end-window];
            xyprod -= darray[0][end-window] * y[end-window];
        }

        cfree(y);
}
/* ----------------------------------------------------------------------*/
/* ----------------------------------------------------------------------*/
void slope(xcol,ycol,newcol,interv)
int *xcol , *ycol , *newcol , *interv;
{
    float strend[2];
    int ppoints , npoints , i;
    int recs;

      if(head.ch[*xcol].nelem > head.ch[*ycol].nelem)
      {
            recs = head.ch[*xcol].nelem;
            name(newcol,*xcol);
      }

      else
      {
            recs = head.ch[*ycol].nelem ;
            name(newcol,*ycol) ;
      }


    for ( i = 0; i < recs ; ++i )
    {
      if ( (i - (*interv)) < 0 )
      {
        npoints = 0 ;
      }
      else
```



```
      {
        npoints = (*interv) ;
      }

      if ( (i + (*interv)) > head.nrec )
      {
        ppoints = 0 ;
      }
      else
      {
        ppoints = (*interv) ;
      }
      npoints *= -1 ;
      npoints += i ;
      ppoints += i ;
      line(xcol,ycol,&npoints,&ppoints,strend) ;
      darray[*newcol][i] = strend[0] ;
    }
}
/* ----------------------------------------------------------------------*/
```



```
#include <stdio.h>
#include <string.h>
#include <X11/Xlib.h>
#include <xview/panel.h>

extern Panel_item cmd_panel_item;


nocom(str)
    char *str;
{
  int n=0;

  while (str[n]==' ' || str[n]==',' || str[n]=='\t' || str[n]=='\n')
    {
      n++;
    }

  while (str[n] != '\0')
    {
      if (str[n] == ',') str[n] = ' ';
      n++;
    }
}


int getcol(str, num)
    char *str;
    int num;
{
  int i=0, n=0, a=0;
  int stop = 0;
  int colnum;

  num -= 1;

  while (str[n]==' ' || str[n]==',' || str[n]=='\t' || str[n]=='\n')
    {
      n++;
    }

  while (str[n] != '\0')
    {
      if (i == num)
        break;

      if (str[n] == ' ' || str[n] == ',' || str[n] == '\t' || str[n] ==
        {
          i++;
          /*     printf("found a separator.. %s\n", str+n); */
          stop = n;

          while (str[n]==' ' || str[n]==',' || str[n]=='\t' || str[n]=='\n')
            {
              n++;
            }
          /*     printf("found a new arg.. %s\n", str+n); */
          continue;
        }
      n++;
    }

  /* printf("found arg: %s\n", str+n);
     sscanf(&(str[n]), "%d", &colnum);
```

```
    printf("col to name: %d arg#: %d\n", colnum, num+1); */
  return (colnum);
}


char * strip(str, num)
    char *str;
    int num;
{
  char tmp[128];
  int i=0, n=0;

  strcpy(tmp, str);

  while (tmp[n] != '\0')
    {
      if (i == num) break;
      if (tmp[n] == ' ' || tmp[n] == ',' || tmp[n] == '\t' || tmp[n] == '\n')
        {
          i++;
          /*     printf("found a separator\n"); */

          while (tmp[n]==' ' || tmp[n]==',' || tmp[n]=='\t' || tmp[n]=='\n')
            {
              n++;
            }
          /*     printf("found a new arg\n"); */
          continue;
        }
      n++;
    }

  /* printf("[%s],[%s] : [%s]\n", str, &str[0], &str[n]); */
  if (i != num)
    return (NULL);

  return(&str[n]);
}


/* splits a string into two at a location specified by num. after num words, t
nto the input string and sends the rest of the string to command line. this p
fies the input string! */

stripper(str, num)
    char *str;
    int num;
{
  int i=0, n=0;
  int stop = 0;

  while (str[n]==' ' || str[n]==',' || str[n]=='\t' || str[n]=='\n')
    {
      n++;
    }

  while (str[n] != '\0')
    {
      if (i == num)
        {
          xv_set(cmd_panel_item, PANEL_VALUE, &str[n], NULL);
          break;
        }
```



```
    if (str[n] == ' ' || str[n] == ',' || str[n] == '\t' || str[n] == '\n')
      {
        i++;
        /*     printf("found a separator\n"); */
        stop = n;

        while (str[n]==' ' || str[n]==',' || str[n]=='\t' || str[n]=='\n')
          {
            n++;
          }
        /*     printf("found a new arg\n"); */
        continue;
      }
    n++;
  }
 str[stop] = '\0';

 /* printf("[%s],[%s] : [%s]\n", str, &str[0], &str[n]); */
}
```



```
/* tpulse - Solves the analytical solution of Carslaw and Jaeger for the */
/*          pulse decay problem. The solution is given in section 3:13:v */
/*          The solution for pulse decay with no sample storage is also */
/*          given for comparison.                                       */
/*                                                                      */
/* version adapted for use with lookv3                                  */
/*                                                                      */
/*     bugs - does not seem to work with 200 roots                      */
/*                                                                      */

#include "global.h"
#define PI 3.141592654

double tpulse(perm,sstor,length,vol,time)
double perm,sstor,length,vol,time;
{
        double root();
        double ezhead, truehead, diffuse, cond, area;
        double rhog, visc, rt[200];
        double inc,accur,rootsq,num,denom,node,beta,por,limit,req,alpha;
        int i,nroots,j,k,l;
        double ehch;

        nroots = 50;
        accur = 1.0e-09;
        area = 9.58e-04;
        beta = 4.58e-10;
        visc = 0.001;
        rhog = 9800.0;
        por = sstor/rhog/beta;
        ehch = por*area/vol;
        inc = PI/100.0/length;

        for (i=0; i<nroots; ++i)
        {
                node = (double)i*PI/length;
                limit = ((double)i+(0.5))*PI/length;
                rt[i] = root(&node,ehch,length,&inc,limit,accur);
        }
        cond = rhog*perm/visc;
        truehead = 0.0;
        ezhead = 0.0;
        diffuse = cond/sstor;
        for (i=0; i<nroots; ++i)
        {
                rootsq = pow(rt[i],2.0);
                num = 2.0*(rootsq+ehch*ehch)*exp((-diffuse*rootsq*time))*sin(rt[i]*length)
;
                denom = rt[i]*(length*(rootsq+ehch*ehch)+ehch);
                truehead -= num/denom;
        }
/*      ezhead = exp((-cond*area*time/beta/vol/length/rhog)); */
return(-truehead);
}
/*-------------------------------------------------------*/
double atpulse(perm,sstor,length,vol,time,beta,visc)
double perm,sstor,length,vol,time,beta,visc;
{
        double root();
        double ezhead, truehead, diffuse, cond, area;
        double rhog, rt[200];
        double inc,accur,rootsq,num,denom,node,por,limit,req,alpha;
        int i,nroots,j,k,l;
        double ehch;
```



```
        nroots = 50;
        accur = 1.0e-09;
        area = 9.58e-04;
        rhog = 9800.0;
        por = sstor/rhog/beta;
        ehch = por*area/vol;
        inc = PI/100.0/length;

        for (i=0; i<nroots; ++i)
        {
                node = (double)i*PI/length;
                limit = ((double)i+(0.5))*PI/length;
                rt[i] = root(&node,ehch,length,&inc,limit,accur);
        }
        cond = rhog*perm/visc;
        truehead = 0.0;
        ezhead = 0.0;
        diffuse = cond/sstor;
        for (i=0; i<nroots; ++i)
        {
                rootsq = pow(rt[i],2.0);
                num = 2.0*(rootsq+ehch*ehch)*exp((-diffuse*rootsq*time))*sin(rt[i]*leng
;
                denom = rt[i]*(length*(rootsq+ehch*ehch)+ehch);
                truehead -= num/denom;
        }
/*      ezhead = exp((-cond*area*time/beta/vol/length/rhog)); */
return(-truehead);
}
/* function =root */
/*  */
/*  */
/*  */
#define TRUE 1
#define FALSE 0
#define EQUATION (var*tan(var*ll)-hh)

double root(a,hh,ll,b,limit,accuracy)
double *a,hh,ll,*b,limit,accuracy;

{

     double var,loop;
     double result[2];
     int root_found = FALSE;
     int first_loop = TRUE;

     var= *a;
     loop= *b;


while ( var < limit && root_found == FALSE )
{
     result[0] = EQUATION;

     if ( first_loop == TRUE )
     {
          result[1] = result [0];
          first_loop = FALSE;
     }
     else
     {
       var += loop;
```

```
        if(var>limit)  var  =  limit;
/*      fprintf(stderr,"%le  ",var);  */
     }

     if  (  (result[0]  <  0.0  &&  (result[0]*-1.0)  <  accuracy)  ||  (result[0]  >  0.0  &
) <  accuracy  ) )
     {
/*  fprintf(stderr,"root  at  %le\n",var-loop);*/
         root_found  =  TRUE;
     }
     else  if  (  result[0]/result[1]  <  0.0  )
     {
              if  (  (result[0]  <  0.0  &&  (result[0]*-1.0)  <  accuracy)  ||  (result[0]
&&  result[0]  <  accuracy  ) )
              {
/*  fprintf(stderr,"root  between  %le  and  %le\n",var-2.0*loop,var-loop);*
/
                  root_found  =  TRUE;
              }
              else
              {
                  var  -=  2.0*loop;
                  loop  =  loop/10.0;
                  first_loop  =  TRUE;
/*  fprintf(stderr,"flag  indicating  change  of  increment  %e\n",var);*/
              }
     }
     result[1]  =  result[0];

}
*a  =  var-loop;
return  (var-loop);
}
```