WIP
User's Manual

James A. Morgan
and

Radio Astronomy Laboratory, University of California at Berkeley
Laboratory for Astronomical Imaging, University of Illinois
Laboratory for Millimeter Astronomy, University of Maryland

Version 2.1
October 1996
Printed: November 14, 1996

WIP Description

WIP is a graphics software package available (at least) on Sun Workstations, VAX/VMS systems, Linux machines, Cray/UNICOS, and CONVEX systems. WIP is intended to be used to generate high quality graphics (using the PGPLOT graphics library) with a minimum of effort.

Restrictions on Use

Disclaimer

Trademark Acknowledgments

WIP Contact

Please address all communication concerning WIP and this manual to:

WIP
James A. Morgan
Astronomy Program
University of Maryland
College Park, MD 20742
INTERNET: morgan@astro.umd.edu
WWW: http://bima.astro.umd.edu/wip/

Contents

List of Tables

List of Figures

Part I

General Concepts

Chapter 1

How to Use This Manual

This manual serves as a reference guide and cookbook for the WIP interactive graphics package. The manual is split into three parts. The first part introduces you to the WIP package and its general concepts. The next part explains more advanced plotting techniques. The last part of this manual contains appendices with a variety of details as well as some plotting examples.

Part I is a general introduction. It contains Chapter 2 which describes the interactive user interface and how to start up WIP. Chapter 3 describes some general plotting concepts. Chapter 4 describes some of the commands used to generate simple plots and Chapter 5 explains the commands associated with images.

Part II describes more advanced plotting techniques. Chapter 6 illustrates how to perform various fits on data and then display the results. Chapter 7 describes how to define and edit macros. Chapter 8 explains user variables and how to use them. Finally, Chapter 9 shows how to have more control over which commands are executed.

Lastly, Part III contains the appendices. Appendix A describes how to set up a file which can be used to customize your WIP environment. Appendix B discusses the options that may be specified on the command line when starting WIP. Appendix C is a glossary which describes and defines often-used terms for easy reference. Appendix D lists, in alphabetical order, the commands WIP currently understands. Next, Appendix E contains several examples of command files used to create the accompanying plots. And, finally, Appendix F presents a list of frequently asked questions along with explanations and solutions.

Notation:

In the examples throughout this manual the prompt from WIP is shown. In most examples this is done symbolically by the string WIP> followed by the text typed by the user. For example:

```
WIP>  device myfile.ps/vps
```

illustrates the use of the WIP prompt followed by the WIP command to define a new device. When trying this example out, the user should not type the characters of the prompt; just the text that follows it. When the example is illustrating a command issued at the operating system level, it will always be done with the percent sign. For example, the command:

```
% wip -d /tek
```

shows how to start WIP with a device other than the default.

In addition, almost every example illustrates one important feature of WIP: the use of comments. While not required, comments are quite useful in WIP plot files, especially several weeks or months later when you return to adopt a plot file for use in another plot. Comments begin with the comment character (#) and continue to the end of the current line. Comments may appear anywhere except on command lines that require a text string as an argument. In this case, comments should not be included as they will be incorporated into the text string.

There's always `help`:

As mentioned previously, Appendix D lists all the commands currently understood by WIP. This listing is the same as the listing generated on-line by the command `help xxx` where `xxx` is any WIP command name (see Section 3.2). Many times in this manual, a list of commands will appear relevant to a particular style of plotting. Rather than go into great detail in these sections about command arguments and options, the reader should investigate the command syntax presented in Appendix D. Additionally, command arguments or options are more likely to change with improvements to WIP while their general concepts will not. For this reason, the discussions about individual commands will, in general, be postponed until Appendix D where individual commands will be kept up to date. If ever a discrepancy exists between this manual and the on-line `help` command, the on-line version is to be taken as correct.

Other useful sources of help in this manual include examples of actual WIP commands accompanied by the generated plot and a list of frequently asked questions about WIP. The examples can be found in Appendix E. The frequently asked questions, as well as answers and examples, can be found in Appendix F.

READLINE:

The WIP user interface has been written in such a way that, if it exists, the Readline library of routines can be linked into WIP when it is compiled. Doing this gives WIP the command line recall and editing characteristic of `Emacs`. By default, this capability is not present. However, if it is present, a message will appear each time WIP is started identifying which command listing file is being read. These commands will not be executed but will be available to be recalled and edited. This name is also the file which WIP will, on exiting, write the history of commands executed. Even if this option is added when WIP is compiled, it can be dynamically ignored via command line options (see Appendix B).

PGPLOT:

The lower level work of WIP is done primarily by a collection of graphical subroutines (developed by Tim Pearson) called PGPLOT. The PGPLOT subroutines may be obtained free–of–charge because the software was written under Unites States government support, although it is copyrighted by California Institute of Technology. PGPLOT is provided "as is" and carries no promise that it will work in your configuration, but all source code is included. Information/questions about PGPLOT should be addressed care of:

```
Tim Pearson
Astronomy Dept 105-24, Caltech, Pasadena, California 91125, USA
Internet:           tjp@astro.caltech.edu or Pearson_T@caltech.edu
NSI-Decnet (SPAN): Deimos::TJP or 15237::TJP
Telephone:          +1 818 395-4980
WWW:                "http://astro.caltech.edu/~tjp/pgplot/"
```

World Wide Web:

The WIP User Manual[1] can also be viewed via the World Wide Web (WWW). There is also the WIP home page[2] which contains general information about WIP as well as information on how to obtain the latest WIP distribution.

[1] http://bima.astro.umd.edu/wip/manual/wip.html
[2] http://bima.astro.umd.edu/wip/

Chapter 2

The User Interface

There are, at present, two ways to create plots with WIP: interactively or by reading plot command files from the command line. Command line plotting is an efficient way to spool plots to a printer or to load plot command files and macros when starting WIP. Command line plotting will be discussed more fully in Appendix B. Interactive plotting consists of typing the commands that generate the plot, one at a time, and seeing the results. This chapter describes the user interface used by WIP.

When WIP is started, certain commands and parameters are defined by the program as well as by the user (see Appendix A for details on creating your own initialization file). The user is then presented with the standard WIP prompt (WIP>) and commands may be then entered one per line. Command lines in WIP consist of one or more "words" that are separated by some combination of either spaces or commas. These "words" may be alphabetic, numeric, or a combination of both and represent the command names and their respective arguments. Command names in WIP are case insensitive (*i.e.* commands may be entered in either lower case, upper case, or a mixture of both). Arguments to a command that will be displayed (*e.g.* a string of characters that will appear in a label) does, however, remain case sensitive. Command names in WIP may also be abbreviated to uniqueness. If a command typed by the user is not unique, WIP will issue a warning and ignore the ambiguous command. The exception to this rule is exact matches; a complete command name will always match.

As an example of how to type commands in interactive mode, suppose a user wanted to draw a single line from one position to another. The following listing illustrates the commands the user would type and an example of WIP's response to an ambiguously stated command:

```
WIP> move 0 0
WIP> draw 1 1
WIP> en
 Ambiguous command name, choices are:
 end     environment
WIP> end
```

Some of WIP's commands have a number of arguments which must be specified. If a command requires a certain number of arguments to function properly and an insufficient number of arguments are given with the command, then WIP will issue a warning to the user. For example, if the user typed the `move` command without the two required arguments, WIP would warn the user:

```
WIP> move
 Insufficient number of arguments provided.  2 required.
```

Other commands, however, have optional parameters which will operate with default values if they are not supplied by the user. In other words, a command may have a number of arguments that may be optionally left for the program to determine. Optional parameters allow the user to alter the default action of these commands. For example, the command `limits` has four arguments that specify the extreme values for the world coordinates. If the user issues the command with no arguments, WIP will determine an acceptable range based on the data currently loaded into the **X** and **Y** arrays.

We have already introduced the main WIP prompt (WIP>). There are two other prompts in WIP: DEFINE> and INSERT> . These prompts appear in response to the commands `define` and `insert`, respectively, and are discussed further in Section 7.4.

Comments may be entered along with WIP commands. The comment character is the pound sign (#) and everything from this character to the end of the input line is ignored by WIP. One exception to this are commands involving labels (see Section 3.6). Since there is no easy way for WIP to determine whether the comment character is part of the string or not, it generally is assumed to be part of the text and is passed to the command as part of the text to label. Hence, comment characters should not be used with these commands. In most of the examples shown in this manual, comments are used to describe the command or specify arguments in the definition of macros. If you are trying these examples interactively, you need not type the comment character nor any of the text that follows it.

Last, but certainly not least, WIP provides the user with three different types of on-line help. See Section 3.2 for a description of each of these types of help.

Chapter 3

General Graphics Concepts

This chapter describes the general graphical concepts of WIP. Many of WIP's commands are used to set the device type, the device's environment, location and limits of the border, and annotate the plot with labels. Most of these commands are called only one time; others several times. These concepts and some of the commands needed to put them to use are described in the sections that follow.

3.1 Graphical Devices

Plotting devices in WIP provide the destination for the plot. In this sense, any graphics device, whether a computer terminal, a high resolution laser printer, or even a file may be considered a WIP device. An initial device is selected when WIP first begins but new devices may be selected at any time during the interactive session.

There are several ways that the initial device may be chosen when starting WIP. The following list indicates the different ways an initial WIP device is selected:

1. An initial device command has been set up to be `device /xwindow`. This initial device was fixed when WIP was compiled.
2. If a default device is provided in your `.wipinit` file (see Appendix A on the use of this file), it is loaded in place of the hard coded device.
3. If the "`-d device`" flag appears on the command line (see Appendix B for a further description of command line options), it will always override any other initial device selected.

A new device may be selected at any time in WIP. To specify a new device interactively, use the command `device` with the new device name as its argument. For example, the command

```
WIP> device /ps
```

selects the landscape version of the PostScript driver as the new device.

Devices have a standard PGPLOT format of "file-name/device-type" or "device-name/device-type". If the 'file-name' or 'device-name' contains a slash (/), it should be enclosed in double quotation marks. For example, the command

```
WIP> device "/dev/tty"/tek
```

sets the current device to a Tektronix terminal and sends the output to the device named "/dev/tty". Usually there exists a default 'file-name' or 'device-name' (see Table 3.1). The part after the slash (required) identifies the plotter device type ('device-type'). This could be, for example, "`/tek`" for plotting on a Tektronix terminal, or "`/ps`" to generate a PostScript output file. If the `device` command argument is "?" (no preceding slash), then WIP generates a list of all currently recognized device types. The command `ldev` will also generate the list of the currently recognized device types. Table 3.1 lists a few of the typically recognized devices and a short description about each.

NOTE: The `/retro` and `/tek4010` devices may have some problems with the cursor when used interactively (especially over a network) – this is currently being investigated.

NOTE: The `/xwindow` and `/xserve` devices behave differently than other devices. These plotting devices start a server independently of WIP which is used to communicate with each plotting window. The server will stay present on the display long after a WIP session is complete. The difference between the `/xwindow` and `/xserve` drivers is that the `/xserve` remains on the screen after a WIP session. There may be multiple `/xserve` windows on a single display, each having a unique id number associated with it. The id number is used to identify the proper window to WIP. As with many X-window applications, the server associated with these two devices has several application defaults which may be set up in your X-resource file. These resources, acceptable values, and default values are shown in listing that follows. To target options at specific windows, replace the `Win` component of the specification with `win#`, where `#` is the id number of the window.

pgxwin.Win.acceptQuit: Either True or False; the default is False. If true, it allows the window manager to provide a way to destroy the window; even while being used.

pgxwin.Win.iconize: Either True or False; the default is False. By default, the /xserve window remains mapped when WIP finishes. If this option is true, the window will become iconic until opened by another WIP session.

pgxwin.Win.maxColors: Any number; the default is 100. This sets the maximum number of colors to try to allocate on startup. Useful for stopping "colormap flashing".

pgxwin.Win.minColors: Any number; the default is 16. Rejects colormaps with fewer than this number of color entries.

Table 3.1: A Sample of Available Plotting Devices

Device	Description
?	Interactive help to list all available devices.
/null	A null device. Useful for error testing of scripts.
/tek	A Tektronix terminal.
/retro	A VT100 with RETRO graphics card.
/xterm	The Tektronix window associated with an xterm window.
[[id]@][display]/xwindow	A X-window display. Multiple windows may be started on one display by using a unique *id* number. The default *id* is 0, which selects an inactive window if available, or creates a new one. A display may also be specified to override the default (which is to use the `DISPLAY` environment variable).
[[id]@][display]/xserve	A persistent version of the `/xwindow` server. In addition, this device may be dynamically resized during various WIP sessions.
file/ps	File in PostScript printer format (landscape). The default name for the *file* is pgplot.ps.
file/vps	File in PostScript printer format (portrait). The default name for the *file* is pgplot.ps.
size/cps	The same as `/ps` except in color.
size/vcps	The same as `/vps` except in color.

pgxwin.Win.visual: Any one of: default, monochrome, PseudoColor, DirectColor, StaticColor, TrueColor, GrayScale, or StaticGray. The default value is default (any visual will do). This resource indicates the visual that should be tried first.

pgxwin.Win.crosshair: Either True or False; the default is False. If true, this sets the default active cursor to cross hairs.

pgxwin.server.visible: Either True or False; the default is True. If false, the server window will be hidden.

pgxwin.server.iconGeometry: An icon geometry format: `+X+Y` that specifies where the server icon should appear.

pgxwin.Win.iconGeometry: An icon geometry format: `+X+Y` that specifies where the window icon should appear.

pgxwin.Win.geometry: A window geometry format: `WIDTHxHEIGHT+X+Y`. The default is something like 867x670.

Files created for hardcopy devices must be spooled to the printer. Two commands and one string variable exist to simplify this for the user. The commands that spool a plot to a hardcopy device are `hardcopy` and `phard`. The string variable *print* is used to control *how* the plot file will be spooled to the printer.

The command `hardcopy` is used when the current device is a hardcopy (*i.e.* non-interactive) device and you are finished generating output to that device. This command closes the output file and releases it to be printed by WIP. Alternatively, the command `phard` may be used when the current device is an interactive device and the user desires to playback the list of current commands to a hardcopy device and then resume plotting on the initial interactive device. Both commands use the string variable *print* to describe how to spool the resulting plot file.

Since the method needed to spool the file to the selected printer is site and device specific, a flexible method was needed to allow users to spool their plots. Generally, the command to send a plot to a printer is quite simple and may be specified by a single statement. The string variable *print* contains the command needed for the system to plot the file 'file-name' and overrides the initial default value set when WIP was compiled. For example, the default value for *print* on a Unix system is "`lpr %s`" where the "`%s`" represents a variable that is used as a placeholder for the value of the plot file name. The value for *print* may be customized for every instance of WIP if it is defined in your `.wipinit` file (see Appendix A on how to set up this command in this file). Alternatively, the command

```
WIP> set print myprog %s | lpr
```

illustrates how a particular instance for the value of *print* may be defined by using the command `set`. Chapter 8 provides more details about the use the `set` command. In this example, the string variable *print* has been defined to run the program `myprog` with the plot file as its only argument and then pipe the result to the standard printer. Note the use of the `%s` variable to insure that the file is passed to the program `myprog` rather than appended to the end of the command. If the `%s` was not used, the resulting command would have been the same as if the user typed:

```
WIP> set print myprog | lpr %s
```

Sometimes, it is useful to do everything except spool the file to a printer. This might happen when generating color PostScript files and it is desired to screen preview the file before shipping them off to the printer. To override the printing command, set the *print* string to "ignore" as in the example:

```
WIP> set print ignore
```

This will cause the `spool` and `hardcopy` commands to perform all of the post-processing on the hardcopy file but to ignore the print statement. Setting the *print* variable to an empty string will have the same effect.

3.2 Help, Echo, and Show

Help is available on-line in WIP. The location and name of the on-line help file is set when WIP is compiled. Generally, this should never need to change. However, there are two ways the location or name of this file can be reassigned without having to recompile WIP. The first is to set the environment variable "WIPHELP". The other way is for the user to do this via their `.wipinit` file (see Appendix A for details).

In addition to this manual, there are three modes of help that are available within the interactive mode of WIP. The command `help` has an optional argument that, depending on its use and whether or not it is present, provides three different styles of help. They are:

help ? – Lists the names of all currently available commands and the names of the currently defined macros.

help – Lists each command along with a one line description of what the command is suppose to do and the necessary and/or optional arguments.

help *cmd* – Provides a complete description of the command *cmd*. Appendix D provides the equivalent of this option for every available command.

In addition to the on-line help, WIP also provides a way to see the current value of many of the attributes, coordinate system positions, user variables, and image characteristics. This may be done with either the command `echo` or the command `show`. The command `echo` enables the user to display, at the command line, the current value of almost any item available to WIP. The `echo` command will also evaluate expressions and print literal text. Multiple items may also be presented using only one `echo` command. Throughout the rest of this manual, several examples of the command `echo` will be presented. The command `show` displays not only many of the items listed above, but also the user variable "name" (shown in upper case) that is associated with their values (see the topics described in the Chapters of Part II for a more detailed description on how to make use of these names).

3.3 Coordinate Systems

There are two distinct coordinate systems in WIP. They are the *Device* and the *World* coordinates. The device, or *viewport*, coordinates represent positions on the device and are always in units that range between (0 – 1). The *world* coordinates represent what the user usually associates as the limits or range of the plot. For example, a plot of height versus weight might have the abscissa (x-coordinate) represent weight and the ordinate (y-coordinate) height. The actual values of weight and height represent the *world* coordinates. Where these values map onto the device represent the *device* coordinates.

This rectangular region of the *world* coordinate space (termed the *window*) is mapped onto a specified rectangle (termed the *viewport*) on the *view surface* (the screen of an interactive display or a sheet of paper on a hardcopy plotter). The following list of commands affect either the *world* coordinate system or the position and size of the *viewport*.

- World coordinates

  environment – sets the world coordinates based on the input arguments or the extreme minimum and maximum of the current X and Y arrays.

  header – reads header information from the current image file and uses the header values to set the world coordinates. This command expands the world limits by, effectively, the width of half a pixel in each direction.

  limits – sets the World coordinates based on the input arguments or the limits of the current X and Y arrays.

  winadj – sets the world coordinates (as with `limits`) and then adjusts the viewport values to have the same aspect ratio.

- Viewport coordinates

  panel – selects a sub-region within the current viewport as a new, *temporary* viewport. The gap between the panels created with this command may be adjusted using the command `submargin`.

  paper – changes the size of the view surface to a specified width (in inches) and height (based on the input aspect ratio). This, in turn, sets the size of the viewport.

  scale – sets the viewport size scale based on the range of the world coordinates and the scaling factor given as input arguments.

  viewport – sets the viewport size with arguments given in viewport coordinates (ranging from 0 to 1 in all directions for all devices).

  viewsize – sets the viewport size with arguments given in inches.

  vstand – sets the viewport size to the full area of the view surface less a margin of 4 character heights on each side.

  winadj – sets the world coordinates (as with `limits`) and then adjusts the viewport values to have the same aspect ratio.

3.4 Attributes

There are several graphic characteristics of WIP that may be set by either the program or by the user. These have the special behavior that they will remain in effect until either the end of the plot or until they are changed again by the user. These characteristics include the color index, text font, line style and width, fill type, and character size expansion to name a few. Table 3.2 contains a list of the attributes, their default values, and the commands used in WIP to modify them. The values of the attributes listed in Table 3.2 remain intact until changed by another call to the controlling command with one exception:

Table 3.2: Graphical Attributes

Attribute	Reset	Default value	Command
Angle	No	0 (No rotation – degrees)	ANGLE *D*
Prompting	Yes	0 (No new page prompting)	ASK *n*
Color Index	Yes	1 (Foreground color index)	COLOR *n*
Character Size	Yes	1 (In character height units)	EXPAND *n*
Fill Type	Yes	1 (Solid fill)	FILL *n*
Font Style	Yes	1 (Normal simple font)	FONT *n*
Line Style	Yes	1 (Full line)	LSTYLE *n*
Line Width	Yes	1 (One stroke)	LWIDTH *n*
Current Symbol	No	0 (Open square)	SYMBOL *n*

attributes identified with a 'Yes' in the Reset column are reset to their default value every time a new device is selected or the user issues the `reset` command. The examples in Appendix E illustrate how to use several of these graphical attributes and Figure E.7 displays the first 32 graphical markers used by the command `symbol`.

Table 3.3 presents the predefined colors representations along with the corresponding Red-Green-Blue (RGB) and Hue-Lightness-Saturation (HLS) intensities. All but one of the color intensities have values that are in the inclusive range of 0.0 – 1.0, with 1.0 being the maximum intensity. The exception is Hue which is a cyclic quantity expressed as an angle in degrees and has the inclusive range of 0 – 360. If the three RGB intensities are the same, the color is a shade of gray.

3.5 Boxes

Fundamental to the display of graphics in WIP is an item called a box. Boxes are constructs that border a plot and may identify the scale along a particular axis. There are several possible arrangements of the arguments to the command `box` that will lead to very different borders. Many of the attributes listed in section 3.4 also affect the resulting appearance of the box.

Prior to calling the `box` command, the *world* coordinates, or limits, should be set. Refer to section 3.3 for a collection of commands which allow the user to specify the *world* coordinates of the plot.

Two other characteristics of the border may be specified prior to a call to `box`: the major tick mark interval and the number of minor tick marks per major tick. Use the command `ticksize` to input the major interval and number of minor ticks for each axis. If the major tick mark interval is specified as 0 (the default), then `box` will calculate its own interval. Likewise, if the number of minor tick marks per major tick mark is set to 0 (the default), then `box` will attempt to calculate a "nice" interval. Setting the number of minor tick marks to 1 will inhibit any minor tick marks (except for logarithm) regardless

Table 3.3: Default Color Representations

Index	Color	(H, L, S)	(R, G, B)
0	Black (background)	0, 0.00, 0.00	0.00, 0.00, 0.00
1	White (foreground)	0, 1.00, 0.00	1.00, 1.00, 1.00
2	Red	120, 0.50, 1.00	1.00, 0.00, 0.00
3	Green	240, 0.50, 1.00	0.00, 1.00, 0.00
4	Blue	0, 0.50, 1.00	0.00, 0.00, 1.00
5	Cyan (Blue + Green)	300, 0.50, 1.00	0.00, 1.00, 1.00
6	Magenta (Red + Blue)	60, 0.50, 1.00	1.00, 0.00, 1.00
7	Yellow (Red + Green)	180, 0.50, 1.00	1.00, 1.00, 0.00
8	Orange (Red + Yellow)	150, 0.50, 1.00	1.00, 0.50, 0.00
9	Green + Yellow	210, 0.50, 1.00	0.50, 1.00, 0.00
10	Green + Cyan	270, 0.50, 1.00	0.00, 1.00, 0.50
11	Blue + Cyan	330, 0.50, 1.00	0.00, 0.50, 1.00
12	Blue + Magenta	30, 0.50, 1.00	0.50, 0.00, 1.00
13	Red + Magenta	90, 0.50, 1.00	1.00, 0.00, 0.50
14	Dark Gray	0, 0.33, 0.00	0.33, 0.33, 0.33
15	Light Gray	0, 0.66, 0.00	0.66, 0.66, 0.66
16-255	Undefined		

of the arguments to the command `box`. The units of the arguments to `ticksize` are the same used to specify the limits of the *world* coordinates.

There are two arguments to the command `box` which determine how WIP will display the border. Each argument consists of one or more letters (case independent; no intervening spaces) which tell WIP how to construct the box. The current letters (and their meanings) used to construct an argument to `box` are shown in Table 3.4. Unrecognized letters are quietly ignored as are any multiple occurrences of letters. To call `box` with arguments for only one axis, use the null argument (0); this will suppress any drawing along the unused axis. For example, to produce a standard box but only along the y-axis, use

```
WIP> box 0 bcnstv                                  # Draw only a y-axis box.
```

By calling `box` with one group of arguments, then changing one or more attributes, and then calling `box` again with different arguments, one can produce a box with considerable variation. The following list of commands provides such an example.

```
WIP> limits 0 10 0 100                     # Set the World coordinates.
WIP> lstyle 1                          # Set the line style to a solid line.
WIP> color 2                                # Set the color index to red.
WIP> box bcst bcst                       # Draw the sides and tick marks.
```

Table 3.4: Box Option Characters

Option	What It Produces
A	Draw Axis line (X axis is a horizontal line at Y=0, Y axis is a vertical line at X=0).
B	Draw Bottom (X) or Left (Y) edge of the frame.
C	Draw Top (X) or Right (Y) edge of the frame.
D	Include the superscript symbols $^\circ$, $'$, and $''$. If this option is used, the option 'Y' is implicitly implied.
F	Write only the last part of the label for the first time (left or bottom most) tick on the axis (see option 'Z'). (*e.g.* if the full first label is 17 42 34.4, then only write 34.4.)
G	Draw a Grid of vertical (X) or horizontal (Y) lines at each of the major tick mark positions.
H	Include the superscript symbols 'd', 'h', 'm', and 's'.
I	Invert the tick marks; *i.e.* draw them outside the viewport instead of on the inside.
L	Label the axis Logarithmically. If option 'Z' is used, then this option is ignored. The major tick interval is *always* 1.0; the number of minor ticks is *always* 8. The numeric label is 10^x, where x is the world coordinate at the tick mark.
M	Write Numeric labels in the unconventional location Above (X) or to the Right (Y) of the viewport.
N	Write Numeric labels in the conventional location Below (X) or to the Left (Y) of the viewport.
O	Omit leading zeros in numbers < 10 in time labels (*e.g.* $3^h3^m1.^s2$ will be written rather than $03^h03^m01.^s2$). The day field is not affected by this option and this option is ignored unless used with the 'Z' option. This option is ignored when used with the 'V' option as it makes it impossible to align the labels nicely. This option is useful to help save space on the X-axis.
P	Extend ("Project") the major tick marks outside the box (ignored if the option 'I' is also specified).
S	Draw Minor Tick marks (Subticks).
T	Draw Major Tick marks at the major coordinate interval.
V	Orient numeric labels Vertically. This is only applicable to the Y axis. The default is to write the Y-labels parallel to the axis.
Y	Do not include the day field in time labeling. This means that labels are "HH MM SS.S" rather than "DD HH MM SS.S". The hours will accumulate beyond 24, if necessary, in this case.
Z	Use time labeling ((DD) HH MM SS style). The World coordinates of the box (see `limits`) and the tick mark increments (see `ticksize`) should be given in units of seconds.
1	Force decimal labeling, instead of automatic choice.
2	Force exponential labeling, instead of automatic.
0	By itself, a zero produces a null-operation argument; useful to produce a box along only one axis.

```
WIP> color 4          # Set the color index to blue.
WIP> box n nv         # Numerically label the tick marks.
WIP> color 14         # Set the color index to dark gray.
WIP> lstyle 2         # Set the line style to a dashed line.
WIP> box g g          # Draw grid lines at each tick mark.
```

As another example, the command

```
WIP> box bcnsthz bcnstvdyz
```

illustrates how to use the `box` command with astronomical coordinates (*i.e.* right ascension for the x axis and declination for the y axis). The first argument (`bcnsthz`) tells `box` to draw the bottom and top edge of the frame, draw minor and major tick marks, and numerically label the tick marks using time labeling and hour, minute, and second superscripts. The second argument (`bcnstvdyz`) is the same as the first except that it suppresses any day formatting, orients the label vertically, and requests time labeling with degree, minute, and second superscripts.

Two string variables associated with the x and y axis arguments of `box` are *xbox* and *ybox*, respectively. If the `box` command is called without any arguments, these strings are substituted in their place. Appendix A shows how to have these strings loaded as the defaults every time WIP is run.

3.6 Text

WIP provides many commands which will display a string of characters. Strings may be altered dramatically by judicious use of many of the graphical attributes shown in Table 3.2. The commands that are used to write labels or titles on the plot are shown in the list below along with a brief description. Use `help cmd` (where `cmd` is one of the following commands) or see Appendix D for a more detailed description of each of the following commands.

id – displays identification information at the bottom of the current plot page.

label – writes a string at the current pen position.

mtext – writes a string relative to a side of the viewport.

putlabel – writes a string at the current pen position but justified by a requested amount. If this amount is negative, the cursor is used to specify a new current pen position (see Section 3.8).

xlabel – writes a centered string below the bottom of the current box.

ylabel – writes a centered string left of the left edge of the current box.[1]

[1] Using `ylabel` along with an annotated box generated using the "Z" option may not produce an optimally placed label. Try using `mtext L 3.5 0.5 0.5` *ylabel-text* instead of `ylabel` (as shown in Example E.10).

All of the text commands above (except `id`) require a label as an argument. This label may be modified by any of the escape sequences shown in Table 3.5. About half of the escape sequences listed in Table 3.5

Table 3.5: Label Escape Sequences

Escape Sequence	What It Produces
\u	Either starts a superscript or ends a subscript.
\d	Either starts a subscript or ends a superscript.
	(NOTE: \u and \d must always appear in pairs).
\b	Backspace (*i.e.* do not advance the text pointer after plotting the previous character).
\\	The backslash (escape) character (\).
\x	The multiplication sign.
\A	Ångström symbol (Å).
\m*n*	The graph marker *n* (*n* may be 1 or 2 digits).
\g*x*	Greek letter corresponding to the roman letter *x*.
\fn	Switch to Normal font (1).
\fr	Switch to Roman font (2).
\fi	Switch to Italic font (3).
\fs	Switch to Script font (4).
\(*n*)	Hershey number *n* (1 to 4 decimal digits); the closing parenthesis may be omitted if the next character is neither a digit nor ')'. This makes a number of special characters (*e.g.* mathematical, musical, astronomical, and cartographical) available. See section E.8 to see how to display the Hershey symbols available.
\[*x*]	User variable *x* (see Section 8). The current value of the specified user variable is substituted into the string. The closing bracket may be omitted only if the user variable is the last item in the string.

remain in effect until the end of the string or until another escape sequence terminates its scope. The following list of commands illustrates the use of a few of the text commands along with several escape sequences.

```
WIP> xlabel \ga (1950)             # Labels x axis as α (1950).
WIP> label HC\d3\uN                # Makes the label HC3N.
WIP> mtext t 2 .5 .5 Title         # Label at top center of the panel.
```

By far, the most flexible text command is `mtext`. The first four arguments to `mtext` specify where the label will appear on the viewport; the remaining arguments represent the label to display. The first argument is a character describing along which viewport side the label will appear. The character must be either 'b', 'l', 't', or 'r' to specify, respectively, the bottom, left, top, or right viewport side. In addition, an optional character 'v' may be appended (as in 'lv' or 'rv') to the left or right character to specify that the label be presented perpendicular to the viewport side. The second argument to `mtext` is a displacement (in character height units) away from the viewport edge. Positive values position the label "away" from the edge; negative values move the label "inside" the viewport. The next two arguments (arguments three and four to `mtext`) specify the justification of the label. The first of these two arguments specifies a position along the viewport. This position is a value relative to the total extent of the viewport edge (*i.e.* running from 0 to 1 left to right and bottom to top). The second of these two arguments specifies how to justify the label relative to this position with a value of 0 indicating left justified; 0.5 centered; and 1 right justified.

3.7 Data Files

Data files are simple ASCII text files that can be read to generate point and line graphics (*i.e.* 1-dimensional plots). The command `data` opens a file and prepares it for reading. The data is assumed to be column ordered with any two columns separated by a comma or blank space (or both). Currently, there is no restriction on the contents of the data in each column except that lines beginning with the comment character (#) are ignored. Non-existent and empty data files are warned about and then ignored.

Data is usually read in by column format into WIP with the commands `xcolumn` and `ycolumn`. The required argument to these commands is the column number from which to read the data (*i.e.* each line read is parsed into words and the command evaluates only the selected word). Incorrect data lines are usually printed out with a warning message.

Which lines are actually read from the data file may be restricted by the user. The command `lines` limits the range that the input data file is read from line *L1* to *L2*, where *L1* and *L2* are the two required arguments given with the command. If the data file has fewer than *L2* lines, reading is stopped at the end of the file.

In addition to the **X** and **Y** arrays discussed above, there are presently two other predefined arrays: the **ERR** and **PSTYLE** arrays. The **ERR** array is used to store error bar distances from the corresponding **X** and **Y** positions (see the command `errorbar` in Section 4.2) and the **PSTYLE** array is used to hold a symbol number for each **X** and **Y** position (see the command `points` in Section 4.2). The **ERR** and **PSTYLE** arrays are used by other commands for other purposes too; these usage variations will be identified when the specific commands are discussed later.

Another use of the data files is to assign strings to string variables. String variables are a type of User Variable and will be introduced in Chapter 8. The command `string` can be used to assign a word or a string to a string variable. The only line read in the data file is defined by the value of *L1* (see above) and which (space separated) words are assigned to the string variable is determined by the arguments to the command. Use `help string` or see Appendix D for a detailed description of the arguments to this command.

Dynamic Data Files

There is a straightforward way to input data directly on the command line within WIP. If the argument to the `data` command is `stdin`, then the current mode is switched to a data input mode. WIP will stay in that mode until the string `enddata` appears on a line by itself. While in data input mode, every line is treated as input to this dynamic data set. When the input mode is terminated, the data may be read using any of the techniques described above. The data set created will remain active until either another data set is opened (with the `data` command) or the user exits from WIP.

3.8 Cursor Interaction

The command `cursor` permits a simple means of interacting with the plot. Several keys have been assigned to the `cursor` command which allow complicated plotting commands to be easily constructed. Each command typed while interactively using the cursor can be saved in the command buffer (see Chapter 7 for a general description of macros and the command buffer) and may be edited (using the techniques described in section 7.4) or re-executed (with the command `playback`). The keystroke commands that are currently recognized are shown in the list below and are *not* case sensitive.

A – Display the present cursor position and, if an image is loaded, the image intensity at that position,

D – Draw to the current cursor position.

M – Move to the present cursor position.

P – Draw a point of the current symbol type at the present cursor position.

S – Toggle the ability to save the keystrokes that follow as commands.

X – Exit the interactive cursor routine.

? – Present the list of current keystroke commands (this is the same as the command `help cursor`).

There are a few commands in WIP which use the cursor to specify the location where the command is to act. Most significant is the command `putlabel`, which takes two arguments: a justification value and a string to write. The justification value ranges from 0 – 1 where 0 means left justification relative to the current cursor position, 1 means right justification, and 0.5 means center the text on the cursor position. However, WIP allows negative justifications. Negative justifications signal WIP to activate the cursor prior to displaying the text so that the user may specify a new current cursor position. The minus sign is just a flag to WIP, the justification that will eventually reach the text displaying routine will still range between 0 – 1 (*i.e.* abs(justification)).[2] As an example, suppose a user were to type the command

```
WIP>  putlabel -0.5 A string to center
```

[2] Note that to left justify the text *and* activate the cursor, the justification argument should be something like -0.001.

The cursor would then become active and allow the user to move to any position on the screen. Now, suppose the user moved the cursor to the position (5, 8) and hit any key. In this case, the keystroke commands are "turned off" and the first key hit by the user terminates the cursor routine. The string entered would appear centered at the position (5, 8) just as if the user had typed the following two lines:

```
WIP> move 5 8
WIP> putlabel 0.5 A string to center.
```

In fact, if this command were typed interactively (or when reading an input file off the command line), these latter two commands are exactly what would be saved in the user's command buffer.

There are a few instances, however, when the cursor command will not function interactively (as shown in the previous example). In these cases, the cursor will only be used to return the cursor position selected; the keystroke commands will be ignored. These instances are when either of the commands `cursor` or `putlabel` are (i) executed from inside a macro; (ii) entered into the command buffer with the command `insert`; or (iii) executed using the command `input`. For these special cases, the cursor will only return the current cursor position and ignore any subsequent keystrokes.

Chapter 4

Basic Plotting

This chapter introduces the commands used to generate one and two dimensional plots. This first section deals mainly with commands that do not require data; the second section with plotting commands that require data. Data may be loaded using the commands of section 3.7. Chapter 3 introduced commands that define the current graphics device, border drawing commands like `box` (Section 3.5), commands needed to control some of the graphical attributes (Section 3.4), and commands that can adjust the values of the coordinate systems (Section 3.3). Many of the concepts described in Chapter 3 will be useful here.

4.1 Simple Plots

This section describes some of the WIP commands that may be called that do not generally act on input data. Examples of many of these commands are shown in Appendix E. An explanation of the optional and required arguments as well as a more detailed description of each of the following commands is available both in Appendix D and via the on-line help using the `help` command.

arc – Draws an arc relative to the current cursor position. The command requires the major and minor axes as inputs and may be drawn over a specified angular extent, start at a particular angle, and be rotated by a specified amount. The attribute “fill” determines whether the arc will be filled or outlined. All angles are in units of degrees, start from the right side of the viewport, and increase counter-clockwise. NOTE: This command is very dependent on the aspect ratio of the plot region; especially if the arc is rotated.

arrow – Draws an arrow from the current cursor position to the specified location. The arrow shape (acute angle of the arrow point and the fraction of the triangular arrow-head that is cut away from the back) may be controlled by use of optional arguments to the command. The attribute “fill” determines whether the arrow will be filled or outlined.

autolevs – Automatically computes (fills) the contour level array elements for use with contour plots (see Chapter 5).

beam – Draws a beam at the current cursor position.

dot – Draws the current symbol type at a requested position or the current cursor position.

draw – Draws from the current cursor position to the specified position.

erase – Erases the graphics screen. If the "ask" attribute is set, then the user will be prompted before the screen will be allowed to be erased.

globe – Draws a projection globe to use with positions generated with the `aitoff` command.

hardcopy – Spool the current plot to the printer (applicable only when the current device is a hardcopy type).

hls – Sets the color of the specified color index to the input Hue-Lightness-Saturation level.

input – Read and execute the WIP commands from the named file.

ldev – List the devices that are currently available to WIP. This is equivalent to the command `device ?`.

levels – Specify the levels for a contour plot (see Chapter 5).

lines – Specify which range of lines of a data file to read (see Section 3.7).

lookup – Specify a lookup table for gray scale plots (see Chapter 5).

macro – Read and define, but do not execute, the WIP macro definitions from the named file.

move – Move the current cursor position to the specified position.

minmax – List the minimum and maximum of the current image.

palette – Select a lookup table for color halftone plots (see Chapter 5).

phard – Replay the current plot commands to a specified device and then return to the current device. This is useful for spooling a plot to a hardcopy device.

playback – Repeat (execute) the commands in the command buffer or any named macro.

plotfit – Plots the most recent results from the `fit` command.

read – Read, but do not execute, the WIP commands in the named file.

rect – Draw a rectangle bordered by the specified vertices. Many of the attributes listed in Section 3.4 affect the resulting rectangle.

reset – Resets the current graphics state (*i.e.* attributes) to the way it is when the device is first selected.

rgb – Sets the color of the specified color index to the input Red-Green-Blue values.

set – Discussed more fully in Section 8.3.

show – Discussed in Section 3.2.

slev – Permits scaling of the contour level array (see Chapter 5).

string – Discussed more fully in Section 8.3.

submargin – Specifies the gap distance between individual frames. For use with the `panel` command.

wedge – Draws a halftone wedge to illustrate the range of halftone values (see Chapter 5).

write – Writes the command buffer or named macro(s) to the named file.

The following is a simple example of the use of some of these commands, including how to save a list of plot commands out to a file for later use (see Section 7.6 for a more detailed explanation of saving and recalling commands).

```
WIP> limits 0 10 0 100         # Set the World limits (§ 3.3).
WIP> color 3                   # Set the color to green (§ 3.4).
WIP> rect 3 5 80 90            # Draw a filled green rectangle.
WIP> move 5 80                 # Move the pen position.
WIP> color 2                   # Set the color to red.
WIP> draw 2 40                 # Draw a red line.
WIP> symbol 2                  # Set current symbol to "+".
WIP> move 4 40                 # Move the pen position.
WIP> dot                       # Draw a "+" at the pen position.
WIP> erase                     # Erase the graphics display.
WIP> playback                  # Redraw everything up to the erase command.
WIP> write myfile.wip          # Writes commands to the file "myfile.wip".
WIP> delete 1 100              # Clear the command buffer (see § 7.4).
WIP> erase                     # Erase the graphics display.
WIP> input myfile.wip          # Execute the commands in "myfile.wip".
WIP> phard /ps                 # Spool this plot to a PostScript printer.
WIP> end                       # Exit WIP.
```

4.2 Single Variable Plots

Plots generally referred to as *scatter* or *xy* plots are usually the result of single variable expressions. Displaying these plots can be done in point mode (*e.g.* `points`) or in line mode (*e.g.* `connect`). WIP has a variety of commands available to vary how these plots are generated and displayed. Useful in distinguishing various data sets (and how the subsequent plot will look) are the various attributes described in Section 3.4 and the various techniques used to border the plot (see Section 3.5). The following is a list of the current single variable plot commands and a brief description of what they generate. For a more detailed description and a list of any required or optional arguments, use `help cmd` (where `cmd` is one of the following commands) or look at the listing in Appendix D.

aitoff – converts longitude and latitude positions into the equivalent x and y positions for use in an Aitoff (more correctly, Hammer) projection.

bar – draws a bar graph at each of the positions read into the current **X** and **Y** arrays. The bar may extend in either the $\pm x$ or $\pm y$ direction. The color of each bar is determined by the corresponding values in the current **ERR** array. The current color index is used if the **ERR** array is empty.

bin – draws a stepped line plot (histogram). A user supplied parameter may specify how to handle irregularly spaced arrays and arrays with direction reversals.

connect – draws line segments to each of the positions read into the current **X** and **Y** arrays.

errorbar – draws error bars on each of the positions read into the current **X** and **Y** arrays with the magnitude of the error bar set by the values in the **ERR** array. The error bar may extend in either the $\pm x$ or $\pm y$ direction.

histogram – generates a histogram from the data in the current **X** array.

lcur – interactive routine to generate a polyline. Vertices are selected using the cursor and are stored in the current **X** and **Y** arrays.

logarithm – changes the named array to it's base-10 logarithm (with an optional scaling).

ncurse – interactive routine to generate a set of data points. Points are selected using the cursor and are stored in the current **X** and **Y** arrays (in increasing x order).

olin – interactive routine to generate a set of data points. Points are selected using the cursor and are stored in the current **X** and **Y** arrays (in the order they were added).

points – draws points of the current point style at the positions of the current **X** and **Y** arrays. Point styles for each point may be stored in the current **PSTYLE** array. Colors for each point are retrieved from the current **ERR** array.

poly – fills a polygonal area defined by the current **X** and **Y** arrays.

vector – draws arrows using the current point positions from the current **X** and **Y** arrays and extending a radius (specified by the values stored in the current **PSTYLE** array) at an angle (degrees positive counter-clockwise from the $+x$-direction and specified in the current **ERR** array). Each arrow's shape (acute angle of the arrow point and the fraction of the triangular arrow-head that is cut away from the back) may be controlled by use of optional arguments to the command. The attribute "fill" determines whether the arrow will be filled or outlined.

Most of the above commands operate on the current **X** and **Y** arrays (see section 3.7). There are also two pre-defined arrays called **PSTYLE** and **ERR** which are used along with the **X** and **Y** arrays to affect the look of the plot. For example, the command `errorbar` uses the **ERR** array to determine the length of the error bar drawn at each point. As another example, the command `points` uses the integer part of the **PSTYLE** array value to set the symbol number for each point. The symbol number corresponds to the type of symbol drawn. If the symbol number is less than -2, it draws a filled polygon of N sides where N is the absolute value of the symbol number. If the symbol number is -1 or -2, it draws a dot of the smallest possible size and the current line width. If the symbol number is between 0 and 31, it corresponds to the symbols shown in Figure E.7; between 33 and 127, the symbol corresponds to the ASCII character in the currently selected font; and larger than 127, it corresponds to the Hershey symbol of the same number (see section E.8). If a fractional part of a **PSTYLE** item exists, then it is treated as a fractional expansion of the symbol size (fractional parts less than 0.01 are treated as no fractional expansion). In addition, the command `points` uses the **ERR** array to determine which color index to use to draw each point.

The commands `lcur`, `ncurse`, and `olin` use the cursor to add and delete points from the current **X** and **Y** arrays. The keystroke commands used by these commands are different from the interactive commands available with the `cursor` command. With these commands, the (case independent) keystroke commands are described in the list that follows:

A – Add a point at the current cursor location.

D – Remove either the last point entered (`lcur` and `olin`) or the point closest to the current cursor location (`ncurse`).

X – Exit the interactive part of the command.

NOTE: Many devices with a 3-button mouse have the keys 'A', 'D', and 'X' bound (assigned) to the left, middle, and right buttons, respectively.

Chapter 5

Images

This chapter will describe the image capabilities of WIP. The first section focuses on the image files and file formats. The next section details the commands WIP uses to select or exclude parts of the image. The following section identifies some techniques commonly used to get the image in the right location and at the right aspect ratio. The last section concentrates on the WIP commands that generate plots using image data and the WIP commands that can be used to enhance these types of plots.

5.1 Image Files

Currently, there are a number of image file formats that WIP can recognize. The number will change as more image file format drivers become available to WIP and are included in the distribution. The currently acceptable formats are:

- Miriad image data sets.
- FITS image files.
- Basic image files.

A brief description of each type will appear later in this section.

In the current configuration of WIP, only one active image may be present at any one time. This is primarily because of the current way that WIP reserves memory for the image and also because of the limitation on memory space that many images can impose.

The `image` command is used to "load" an image into WIP. The command has three arguments but only the first one is required. The required first argument is the image file name. How the image file name is specified may be dependent on the format of the image (this will be explained in the section that follows). The second argument is optional and selects the plane number to read. This defaults to the first plane if this argument is not present or a non-existent plane is requested. The third argument is also optional and specifies a mask value. If the image has masked (undefined) data, WIP will set the masked data (*e.g.* magic numbers) to a value of -99 unless the third argument is present. If the mask value is present, then all masked data will be set to the requested value.

When the `image` command is called, WIP attempts to open the image file based on the different types of drivers available. If WIP can identify the image type, it will attempt to read the header information (if present) and then load the image into memory.

When an image has more than 3 dimensions, the plane number argument ($\mathcal{P}$) may take on any value in the range:

$$1 \leq \mathcal{P} \leq \prod_{i=3}^{\mathrm{NAXIS}} \mathrm{NAXIS}_i,$$

where NAXIS is the total number of dimensions in the data set and NAXIS_i is the dimension size of the ith axis.[1] Hence, WIP will be able to correctly assign the plane number if a plane from a higher dimension is desired.

The Miriad Image Format

Miriad images are a subset of the Miriad data set format. Currently, the Miriad image is one "item" of the full data set and the header information is also stored as individual "items." Refer to the BIMA Project manuals for a more detailed description of Miriad and its data structure.

The FITS Image Format

The FITS file format is a standard data transfer format used to pass data between different computers. In general, FITS images will contain a header section followed by the image data (in binary form). WIP expects that the header will be in standard 80 character format and *must* contain the standard keywords: SIMPLE, BITPIX, NAXIS, NAXISi (for $i = 1, \ldots,$ NAXIS), and END. WIP will *only* recognize FITS files with BITPIX set to 16 (16 bit integers), 32 (32 bit integers), or -32 (32 bit floating point). In addition to the standard keywords, WIP requests the following keywords (but does not require them to be present): BSCALE, BZERO, BLANK,[2] DATAMAX, and DATAMIN. Also, the keywords CRVALi, CRPIXi, CDELTi, and CTYPEi, (where i is either 1 or 2) are searched to help set up the transformation matrix.[3] If the optional keywords are not found, then reasonable default values are assigned.

[1] If NAXIS only has a value of 2, the plane number may only be 1.

[2] According to the FITS guidelines, the BLANK keyword should be used only with data with positive BITPIX values.

[3] Astronomical images displayed in right ascension and declination should provide CRVALi and CDELTi values in units of degrees and degrees per pixel, respectively.

The Basic Image Format

Basic images are simple binary files that contain a multi-dimensional array along with (possibly) optional header information at the beginning of the file. Most image file formats will be structured something like this, so the Basic file format should suffice if a particular image driver has not yet been written.[4] The necessary information needed to load the Basic file is contained in the file name argument to the **`image`** command. The name syntax is:

```
filename‘columnsXrows[‘offset]
```

where **`filename`** is the name of the file on the disk, **`columns`** is the number of columns (in pixels), and **`rows`** is the number of rows (in pixels). The single character **`X`** defines the pixel data type and separates the **`columns`** value from the **`rows`** value. The character **`X`** may be only one of the following characters:

`b`: unsigned bytes
`s`: signed 2 byte integers
`l`: signed 4 byte integers
`r`: 4 byte floating point
`d`: 8 byte double precision floating point

Finally, **`offset`** represents the offset from the beginning of the file (in units of bytes). If this part of the name is not present, it defaults to 0 bytes. The first back quote in the name syntax is required. The second back quote should not be present unless an offset value is given. If a Basic file has any header information at the beginning of the file, this needs to be skipped; this is provided for with the **`offset`** parameter.

Some machines byte-swap their data. For all image types except Basic, the data is byte-swapped as needed. However, since Basic images are usually created and stored locally, they are probably already stored in the machine's native format. Hence, by default, Basic images are not byte-swapped. If however, on those machines, there are images that need to be byte-swapped when loaded into WIP, byte-swapping can be forced by using an upper case character to identify the pixel data type. For example, use ‘**`L`**’ to force byte-swapped a signed 4 byte integer array. NOTE: Machines that are not byte-swapped will always ignore this byte-swapping option.

5.2 Image Characteristics

This section describes the methods WIP uses to associate a pixel in an image with a world coordinate position, how to select a subsection of the image, how to specify contour levels, and how to access the image header file.

[4] Because of the way Basic Image files are loaded, header information may not be extracted automatically, or, for that matter, even accessible.

The Coordinate Transfer Function

All of the plotting commands discussed in Section 5.4 require a transfer function. Put simply, the transfer function tells WIP where a given pixel should be located relative to the world coordinates. The command to specify the transfer function is `transfer`. This command has six arguments that define the mapping from image pixel to world coordinate. If (`I, J`) corresponds to a position in the image array (a particular pixel), then (x, y), the corresponding position in world coordinates, is related by:

$$\begin{aligned} x &= a + b\mathrm{I} + c\mathrm{J}, \\ y &= d + e\mathrm{I} + f\mathrm{J}. \end{aligned}$$

The constants a, b, c, d, e, and f are the six arguments to the `transfer` command. As an example, if the limits of a plot are the same as the number of pixels in an image (*i.e.* `limits` 1 Nx 1 Ny), then the default transfer function constants of (0, 1, 0, 0, 0, 1) will correctly map the image onto the display. This configuration is particularly useful for images with large limits and very small step sizes (*e.g.* very high resolution astronomical images).

Another simple example deals with images that are flipped in one direction. For an image with the y-axis flipped and the world coordinates set as in the example above, a transfer function with constants (0, 1, 0, $(Ny + 1)$, 0, -1) will dynamically orient the image correctly.

If, however, you have loaded an astronomical image with the world coordinate limits set by the image header variables `CRVAL`, `CRPIX`, and `CDELT`, then a more useful set of values for the transfer function would be:

$$\begin{aligned} a &= \mathrm{CRVALX} - (\mathrm{CRPIXX} \times \mathrm{CDELTX}), \\ b &= \mathrm{CDELTX}, \\ c &= 0, \\ d &= \mathrm{CRVALY} - (\mathrm{CRPIXY} \times \mathrm{CDELTY}), \\ e &= 0, \\ f &= \mathrm{CDELTY}. \end{aligned}$$

This transfer function can be, in fact, set up for the user via the command `header` when these header variables are present in the image file. If these header variables are not present in the image header, they may be set manually by the user (since they are standard user variables). The user must set these user variables *after* the image is read into WIP but *before* the `header` command.

Using the command `transfer` without any arguments will automatically reset the transfer function to the default value of (0, 1, 0, 0, 0, 1).

Rotated Transfer Functions

Some images do not have the `cdelt` keywords but rather specify the pixel step sizes in a matrix. An example of that is a FITS image file that contains the keywords `CD1_1`, `CD1_2`, `CD2_1`, and `CD2_2`. To set up the transfer function (and the limits) for this type of FITS image like the **header rd** command would, a macro could be written as follows:

```
WIP> define doheader
WIP> new cd11 cd21 cd12 cd22
WIP> new xconvert yconvert
WIP> set xconvert 3600.0 / 15.0
WIP> set yconvert 3600.0
WIP> set cd11 CD1_1 * xconvert / cosd(crvaly)
WIP> set cd21 CD2_1 * yconvert
WIP> set cd12 CD1_2 * xconvert / cosd(crvaly)
WIP> set cd22 CD2_2 * yconvert
WIP> set \1 (xconvert * crvalx) - (crpixx * cd11) - (crpixy * cd12)
WIP> set \2 (yconvert * crvaly) - (crpixx * cd21) - (crpixy * cd22)
WIP> transfer \1 cd11 cd12 \2 cd21 cd22
WIP> set \11 \1 + (cd11 * (subx1 - 0.5)) + (cd12 * (suby1 - 0.5) )
WIP> set \12 \1 + (cd11 * (subx2 + 0.5)) + (cd12 * (suby2 + 0.5) )
WIP> set \13 \2 + (cd21 * (subx1 - 0.5)) + (cd22 * (suby1 - 0.5) )
WIP> set \14 \2 + (cd21 * (subx2 + 0.5)) + (cd22 * (suby2 + 0.5) )
WIP> limits \11 \12 \13 \14
WIP> subimage 1 nx 1 ny
WIP> free xconvert yconvert
WIP> free cd11 cd21 cd12 cd22
WIP> end
```

Although this is only one example, the others probably can be solved in a similar fashion.

Selecting subsections of the Image

Two commands exist that allow easy selection of a sub-portion of an image. The command **subimage** sets the range of the indices such that only a subimage is displayed (regardless of the range of the transfer function or the limits). The other command, **quarter** (which should be called right after the **image** command) is used to select a particular quadrant of the current image. By default, the **quarter** command selects the inner quarter of the image, but may be requested to select other quadrants.

If the user would like as much of the image as possible to be displayed, but only the part that fits within a particular region, then the user should not specify the sub portion of the image (*i.e.* do not use the two commands **subimage** or **quarter**) but rather fix the *world* coordinates to the desired range. This method of fixing the *world* coordinates to clip the image is very powerful; especially when overlaying several image plots (see Section 5.3 below and the example in Section E.12).

Setting Contour Levels

Currently, there are two commands used to specify the contour levels and one command that can be used to scale the contour levels. The commands to specify contour levels are **levels** and **autolevs**. The arguments to **levels** are the individual contour level values. The arguments to **autolevs** are the number of levels desired, the type of contour levels (**linear** or **logarithmic**), and, optionally, the minimum and maximum over which to range the computed levels. If the minimum and maximum are not specified, they default to the minimum and maximum of the current image.

The command **slev** permits the user to scale the contour level array by a fixed amount. There are two arguments to **slev**; the first required and the second optional. The first argument is a string which specifies the type of scaling to perform and may be either **absolute** or **percent**. The type **absolute** is the default and specifies that the scaling value be multiplied directly; type **percent** means to use the scaling value as a percentage of the peak intensity of the plot. The second (optional) argument defines the scaling value and defaults to 1 (no scaling). As an example, the commands

```
WIP>  levels 1 2 3 4 5 6 7 8 9 10
WIP>  slev P 10
```

will set the contour levels in intervals of 10% of the image peak intensity starting at 10% of the image peak intensity. Likewise, the commands

```
WIP>  autolevs 10 LIN 1 10
WIP>  slev A 1.0E-4
```

will set the contour levels to 1.0E-4, 2.0E-4, . . . , 9.0E-4, and 1.0E-3.

The Header Command

The command **header** allows the user to read in certain header variables and set up particular defaults for the current image. The defaults established by a call to **header** are the transfer function and the *world* coordinates. The transfer function is set relative to the user variables **CRVALX**, **CRVALY**, **CRPIXX**, **CRPIXY**, **CDELTX**, and **CDELTY** (see the examples above). The limits (or *world* coordinates) are set by the header variables and the range of the indices specified by either **subimage** or **quarter**.

If the image header does not contain the proper header values or the image type does not allow them to be retrieved, the user may specify them using the **set** command. This, however, should be done *after* the image is loaded and *before* the **header** command is called.

Wedge Displays

Sometimes, it will be helpful to be able to annotate a halftone plot with a wedge. These are useful in that they help to delineate the gray or color range displayed. This is done with the **wedge** command.

The arguments to `wedge` command are similar to the `mtext` command. There are required arguments which specify which side of the current viewport to display the wedge (this also fixes the orientation); the displacement from the viewport edge (in character height units); and the thickness of the wedge (also in character height units). If the displacement is negative, the wedge will appear inside the viewport; if it is positive, the wedge will be outside. Optional arguments include the minimum and maximum values to be used to display as the extremes of the wedge and a label which will be displayed on the outside of the wedge. If a label is desired, the minimum and maximum must be specified. If the minimum and maximum are not given, the `wedge` command will display the wedge in such a way that the entire range of values displayed by the most recent `halftone` command will be visible.

The Image Transfer Function

Sometimes the raw image is not desired to be displayed; the image needs to be mapped differently. For example, often it is useful to display the logarithm of the image. The command `itf` facilitates this. There are currently three types of image transfer functions available:

- linear
- logarithmic
- square-root

Remember that the logarithm is a base-10 logarithm. Initially, the image transfer function is set to linear and is reset to linear whenever a new device is selected.

The Image Palette

For halftone plots (see below), the default is to use a gray scale ramp. On devices that permit the use of colors beyond the color index 15, different color lookup tables may be used. The `palette` command loads a new palette. On some interactive devices, the change is immediately displayed. However, the user must be careful to be sure to load the desired palette *prior* to displaying a halftone on a hardcopy device. There are several pre-defined palettes available. The `palette` command has one required argument which is the number of the palette desired. The list below identifies the current list of pre-defined palettes.

1. Gray scale.
2. A rainbow.
3. A heat scale.
4. The IRAF scale.
5. The AIPS scale.

6. The PGPLOT scale.
7. A Saoimage A scale.
8. A Saoimage BB scale.
9. A Saoimage HE scale.
10. A Saoimage I8 scale.
11. A DS scale.
12. A cyclic scale (Red to Green to Blue to Red...).

If the required argument is negative, the color palette is flipped.

If a pre-defined palette is not acceptable, one may be defined by the user using the `lookup` command. The `lookup` command maps the values in the **X** array to the red colors, the **Y** array to the green colors, and the **ERR** array to the blue colors. The data in the **PSTYLE** array identifies the ramp intensity level for the corresponding RGB values. The value in each of the four arrays must be between 0 and 1. Colors on the ramp are linearly interpolated from neighboring levels. An optional argument to `lookup` may be used to flip the resulting color table. For example, a trivial gray scale ramp is defined below:

```
WIP>  data stdin                    # Read data from the command line.
Data Input>  0                      # Enter the data.
Data Input>  1
Data Input>  enddata                # Finish entering data.
WIP>  xcolumn 1                     # Load the red colors.
WIP>  ycolumn 1                     # Load the green colors.
WIP>  ecolumn 1                     # Load the blue colors.
WIP>  pcolumn 1                     # Load the intensity level values.
WIP>  lookup                        # Load the lookup table.
```

5.3 Window Adjustments for Images

Sometimes the hardest job in dealing with images is getting it to fit properly with the rest of the plot. In the previous section, many commands were identified that help to specify how the image pixel maps onto the world coordinate system. This section goes one step further by dealing with how a particular image should map onto the viewport of a particular window. In other words, this section will discuss how to correctly set the aspect ratio of the figure.

At first glance, it might be thought that the best place for WIP to control the aspect ratio of an image is within the `header` command. After all, it loads the coordinate transfer function and sets the world coordinate limits based on the information from the image header. This would be fine if only one image was ever dealt with on a plot. But this will fail whenever one image is to be overlaid on top of another image and the two images have different dimensions. Hence, it was decided to keep this task under the user's control. Unfortunately, this means more work for the user.

The rest of this section will introduce the commands necessary to set the aspect ratio for one image and try to illustrate how to properly set the aspect ratio for multiple (overlaid) images. Additional examples of these techniques will be found in Appendix E.

The Window Adjustment Commands

In general, the image plot should be displayed in such a way that the size of a pixel in the x-direction is the same as the size of a pixel in the y-direction. The `winadj` command permits the user to reconfigure the viewport such that the scale in the x- and y-directions are equal. This command must be used *after* the image is loaded and *before* the `header` command is called (*i.e.* before the world coordinates of the plot window are set). To illustrate, suppose we have an image named "myimage" which is dimensioned 80 by 33. The following commands constrain the viewport so that the halftone plot fits properly (*i.e.* has an aspect ratio of 80:33):

```
WIP> image myimage          # Load the image and header information.
WIP> winadj 0 nx 0 ny       # Set viewport to an 80:33 aspect ratio.
WIP> header                 # Use the header info to set transfer and limits.
WIP> halftone               # Draw the halftone.
```

Note the use of the user variables `nx` and `ny` to specify the dimensions of the plot. Also, note that because pixels have a 'size' associated with them, the width of the window (*i.e.* the number of pixels) must have room for `nx` + 1 pixels. In order for this to be true, the range of values in the x-direction must be from 0 to `nx`. (likewise for the y-direction).

One of the most common mistakes in generating image plots is that user will (a) forget to adjust the window aspect ratio; or (b) call the `winadj` command *after* the `header`, `transfer`, or `limits` commands. Because the `winadj` command changes both the viewport *and* world coordinates, it is important that the limits (world coordinates) of the image be established *after* this command is called.

Panel Movements

To illustrate how to correctly use the `winadj` command with the `panel` command, consider an example where two images should be displayed side by side. Because the two images have the same dimensional range, and for this example, it is desired that there should not be any gap between the images (*i.e.* the boxes should butt up against each other). The following macro definition and list of commands might be used:

```
WIP>  define mytask                          # $1=Panel number; $2=Image plane.
DEFINE>  panel -2 1 $1                       # Set the requested panel.
DEFINE>  image myimage.fits $2               # Load the image.
DEFINE>  winadj 0 nx 0 ny                    # Set the aspect ratio.
DEFINE>  header rd                           # Set the transfer function and limits.
DEFINE>  halftone                            # Draw the halftone.
DEFINE>  box bcstz bcstz                     # Draw a box with tick marks.
DEFINE>  if ($1 < 2) box nhz nvdzy           # Label the left box.
DEFINE>  if ($1 > 1) box nhfz 0              # Label bottom of right box.
DEFINE>  end                                 # Finish the macro definition.
WIP>  viewport 0.1 0.9 0.1 0.9               # Set an initial viewport.
WIP>  mytask 1 3                             # Load the first image.
WIP>  mytask 2 4                             # Load the second image.
WIP>  panel 1 1 1                            # Reset the panel for common RA DEC labels.
WIP>  xlabel \ga (1950)
WIP>  mtext L 3.5 0.5 0.5 \gd (1950)
```

These commands will actually generate a plot with a gap between the images! The gap exists even though the **panel** command was called with a negative argument. The gap would remain regardless of the value of the user variable *xsubmar*.

The reason for the gap is because of the call to **winadj**. Remember that the **winadj** command sets the world coordinate system limits and then attempts to adjust the viewport coordinate system to have the same aspect ratio. The **panel** command has established a viewport (which, for this example, is split into two vertical halves). Next, the **winadj** command tries to adjust the aspect ratio of this viewport to match the image. In general, the resulting viewport from **panel** will not match the aspect ratio desired by the **winadj** command and the viewport will adjust to obtain the proper ratio. As a result, either the top or the right edge (or both) will need to move inward.

The correct solution to the problem above is to find the aspect ratio desired for one image and then to scale it by the number of images in each direction. Adjusting the initial viewport to this range will not only correct the problem seen in the example above but also remove the need for the call to the **winadj** command inside the macro. In this example, there are two panels in the x-direction and only one panel in the y-direction. Hence, the commands that will produce the properly scaled plot with the boxes butted together is shown in the listing that follows (with the **winadj** command removed from the definition of the **mytask** macro above):

```
WIP>  viewport 0.1 0.9 0.1 0.9                   # Set an initial viewport.
WIP>  image myimage.fits 1                       # Do this to get Nx and Ny.
WIP>  set \0 2 * nx                              # Compute the x-direction width.
WIP>  winadj 0 \0 0 ny                           # Adjust the viewport.
WIP>  mytask 1 3                                 # Display the first image.
WIP>  mytask 2 4                                 # Display the second image.
WIP>  panel 1 1 1                                # Reset the panel for common RA DEC labels.
WIP>  xlabel \ga (1950)
WIP>  mtext L 3.5 0.5 0.5 \gd (1950)
```

The variable \0 is a user variable and is discussed in Chapter 8.

Overlaid Images

A common plotting task is to overlay one image graph on top of another. Many times, the two (or more) images will have the same dimensions and this will present no problem. However, there are many examples where a smaller image is overlaid on top of a larger image (or vice-versa) or one image has a completely different scaling (resolution) than another. Making sure that the aspect ratio of these two images is maintained does not have to be difficult.

The simple case is when one image is used to define the limits of the world coordinates. Since the aspect ratio is defined by one image and its limits specify the world coordinate range of the plot, there is no need to compute the aspect ratio for the other image. The world coordinate limits need to be saved, however, and then loaded after the second image's header variables are read and transfer function defined. The following is an example of this method (the user variables \1–\4 are discussed in Chapter 8):

```
WIP>  viewport 0.1 0.9 0.1 0.9                   # Set an initial viewport.
WIP>  image first.fits 1                         # Do this to get Nx and Ny.
WIP>  winadj 0 nx 0 ny                           # Adjust the viewport.
WIP>  header rd                                  # Set the transfer function.
WIP>  set \1 x1                                  # Save the current x...
WIP>  set \2 x2                                  # ...world coordinates.
WIP>  set \3 y1                                  # Save the current y...
WIP>  set \4 y2                                  # ...world coordinates.
WIP>  halftone                                   # Display the first image.
WIP>  box bcnsthz bcnstvdyz                      # Draw and label a box.
WIP>  image second.fits 2                        # Load the second image.
WIP>  header rd                                  # Load it's transfer function.
WIP>  limits \1 \2 \3 \4                         # Reset the world limits.
WIP>  levels 2 3.5 4 5.5                         # Set the contour level values.
WIP>  contour                                    # Contour the second image.
```

The more complicated case is when none of the images define the limits of the world coordinates of the plot. In this case, the limits need to be established by the user. If these limits are simple (*i.e.* not astronomical coordinates), then the `winadj` command can be used to establish the world coordinates and the aspect ratio. When astronomical coordinates are used, the aspect ratio must be calculated based on the plot range. In other words, determine the aspect ratio range using similar coordinate systems (*e.g.* in arcsec offsets) and then set the limits in the absolute coordinate system. The technique shown in the example above of saving the world coordinates and then reloading them can be well utilized for this case.

5.4 Image Plots

This section presents the various ways of generating a plot with an image. Each of the commands below requires that certain characteristics are defined prior to being called. Most of the image characteristics are described in Section 5.2.

Contour Plots

Before any contour plots may be generated, the user must specify the contour levels using one or more of the techniques discuss in Section 5.2.

There is only one command in WIP to generate a contour plot: `contour`. This command, however, permits access to many different methods to use for drawing the contour plots. There are up to three optional arguments to the `contour` command.

The first (optional) argument specifies the method used to draw the contour plot. By default (or if the method `s` is selected), WIP draws a very simple contour plot. Selecting contour method `t` means the plot will be drawn with continuous contour lines (*i.e.* it starts a contour line and follows it until it ends). Contour method `b` means that the contour should be drawn (just like contour method `s`) but also allows a blanking value to be specified. Contour method `l` draws the same contour plot as method `t` except that it includes contour level labeling. The second and third optional arguments are ignored by the contour methods `s` and `t`.

The contour method `t` will draw positive contours values with solid lines (`lstyle 1`) and negative contour values with dashed lines (`lstyle 2`). This change of line styles may be overridden by specifying the contour method as `-t`. Using the method `-t` will mean that the `contour` command will draw both positive and negative contour values with the same (*i.e.* the current) line style. Note that the other two methods of drawing contour plots always uses the current line style regardless of the value of the contour level.

If contour method `b` is selected, an additional optional argument may be given that specifies the blanking value (image array values equal to this blanking value are ignored in the plot). The blanking value, if omitted, defaults to 0.

If contour method `l` is selected, the second (optional) argument may be provided to specify the spacing along the contour between labels (in grid cells) and the third (optional) argument is used to specify the minimum number of cells that must be crossed before drawing the first contour label. If not present, the second argument defaults to 16 and the third to 8.

Gray-Scale Plots

Gray scale and color image plots are generated in WIP using the `halftone` command. By default, a gray scale halftone is generated. To create color, the display or hardcopy device must be capable of handling color maps and one must be loaded either with the `palette` or `lookup` command (see Section 5.2). The `halftone` command has two optional arguments that may be used to delimit the minimum and maximum intensity range of the gray scale. By default, the minimum and maximum of the image are used.

Currently, WIP uses the most recently loaded lookup table when displaying a halftone. Some devices permit this to be changed after the halftone is displayed but since this is not the case for hardcopy devices, the user should be careful to insert the desired palette prior to calling `halftone` when generating a hardcopy plot.

Halftones will only be drawn if the device is capable of accessing color indices larger than 15. Use the command `'echo cmin cmax'` to display the current values of *CMIN* and *CMAX*).

Cross Section Plots

There is a crude cross section plot available in WIP. It is the `hi2d` command. Each cross section slice consists of a hidden line histogram. A bias is applied to each successive slice in order to raise it "above" (the y-direction) the previous slice. This bias is the only required argument and is entered in the same units as the image data.

Two other optional arguments may be specified to control the display of the cross section plot. The first optional argument controls the slant of the plot by acting as an offset to successive cross sections in the x-direction. If this argument is positive (> 0), it creates a plot that appears to slant to the right; if negative (< 0), slant left. The default value for this argument is no slant effect (argument is 0). The last (optional) argument is a flag to specify whether the indices in the x-direction are centered or represent the left edges of the bins. The default value for this argument is to center the bins (argument is 0).

Drawing A Beam

The `beam` command was created because many astronomical image files contain information about the beam size and orientation used to observe the source. This information can be used to display the beam providing a graphical presentation of the resolution of the image. The `beam` command draws a beam at the current cursor position (with an optional offset). This beam is outlined in the current color using the current line properties. The required arguments are the beam major axis, the minor axis, and the position angle. As in all astronomical images, the position angle is defined as 0 degrees towards the north and increases counter-clockwise.

The optional arguments include offsets, a scaling factor, a fill color index, and a background rectangle color index. The offsets permit the center of the beam to be shifted relative to the current cursor position. The scaling factor provides greater control over the aspect ratio. By default, the scaling factor (or if it is provided and is negative) is set to $15 \times \cos(\text{declination})$ (which is proper for RA-Dec type scaling). The fill color index is used to draw the beam with a filled color. The background color index (if greater than or equal to zero) is used as the color to draw the (filled) bounding box of the beam.

The optional offsets are provided in units of the width and height of the beam. This permits the user to easily shift the beam location. As an example of how to use the optional offsets, consider drawing a beam in the lower right corner of the window. Rather than computing the position and then guessing how much to move it so it is slightly offset from the box (and then having to redo it all again if the coordinates need to be changed!), the offsets permit an easy alternative:

```
WIP> image myimage                      # Load the image and header information.
WIP> header rd                          # Set transfer fn and limits.
WIP> halftone                           # Draw the halftone.
WIP> move x2 y1                         # Move to the lower right corner.
WIP> beam bmaj bmin bpa -0.8 0.8        # Draw the beam.
```

In the example above, the parameters `bmaj bmin bpa` are dynamically read from the image header. It is possible some conversion of these terms may be necessary. In this example it was assumed that the major and minor sizes were in the same units as the Y-axis and the position angle was in degrees.

Part II

Advanced Concepts

Chapter 6

Data Fitting

This chapter describes methods of how to fit data. There are currently five different ways to fit data loaded in the **X** and **Y** arrays. Two of these methods fit linear data, one fits gaussians, and two fit polynomials. In addition, only a subset of the data need be fit. Once a fit has been found, the fit may also be plotted. The commands that perform these tasks are described below (and also illustrated in Figure E.9).

6.1 Fitting

The `fit` command is what is used to do all the fitting of data. The first argument to `fit` is required and is the type of fit to perform. The first argument must be one of the following values:

- lsqfit
- medfit
- polynomial
- legendre
- gaussian

Minimum match is in effect when identifying the fit type.

There are two different ways to fit linear data: the Least Squares Method and the Method of Least Absolute Deviations. Data that is exponential in form can also be fit with either of these linear methods by first taking the logarithm of the data. There are also two different ways to fit polynomial data. One is a simple polynomial and the other is a fit of Legendre polynomials. In addition to fitting linear and polynomial data, one can also fit gaussians. More than one gaussian curve can be fit simultaneously. Information particular to each type of fitting are discussed below.

In each method, the data from the **X** and **Y** arrays will be fit. And, except for the "medfit" method, a measure of the error in the Y-direction will be taken from the data in the **ERR** array. If no data has been previously loaded in the **ERR** array, then values of unity will be used.

Additionally, each time a fit is performed, information about the fit will be displayed on the command line. The information will contain the fit parameters as well as a measure of the goodness of the fit. Additional (optional) parameters provided to the `fit` command permit the user to assign these fit terms and their errors to user variables.

Least Squares Method

The method of Least Squares trys to fit the input data to a straight line model of the form $y(x) = a + bx$. This is done by minimizing the chi-squared function:

$$\chi^2(a,b) = \sum_{i=1}^{N} \left(\frac{y_i - a - bx_i}{\sigma_i} \right)^2$$

where σ_i is the error estimate for each data point (the **ERR** data).

When a fit is found, this method returns the parameters a and b, the chi-squared value for the fit, and the probable uncertainties in the estimates of a and b. If the data in the **ERR** array was used to specify the errors on the observation points (**Y** array), a "goodness of fit" is also returned. If, however, no errors are specified (there are no points in the **ERR** array), then the correlation coefficient (r) is returned in place of the "goodness of fit". If the "goodness of fit" term (represented by the letter 'q') is larger than about 0.1, then the fit is probably reasonable. Values for the correlation coefficient (r) near ± 1 are considered good.

Method of Least Absolute Deviations

This is similar to the Least Squares Method described above except that the uncertainties in the data (the **ERR** array data) are ignored and, instead of minimizing the chi-squared function, the merit function to minimize is

$$\sum_{i=1}^{N} |y_i - a - bx_i|$$

The use of the median in this type of fitting provides a more robust method of fitting than the Least Squares method and is especially useful when the data contains outlying points.

When a fit is found, this method returns the parameters a and b and a measure of the mean absolute deviation (in y) of the data points from the fitted line.

Simple Polynomial Fitting

A simple polynomial has the form:

$$y(x) = \sum_{i=0}^{N-1} (a_i x^i)$$

where a_j is the jth coefficient of a polynomial of degree N. The method used to find a fit is just a generalization of the Least Squares method extended to include more than two coefficients.

When a fit is found, this method returns the N coefficients and then the N probable uncertainties in the estimates of the coefficients. Finally, the chi-squared value of the fit is returned.

Legendre Polynomial Fitting

The method is identical to the Simple Polynomial Fit except that it uses Legendre Polynomials. Legendre Polynomials have the following recurrence relation:

$$(n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x)$$

where $P_0(x) = 1$ and $P_1(x) - x$.

When a fit is found, this method returns the N Legendre coefficients and then the N probable uncertainties in the estimates of the coefficients. Finally, the chi-squared value of the fit is returned.

Gaussian Fitting

This method tries to fit one or more Gaussians to the data. In addition to the style (which must be "`gaussian`"), the *number* of Gaussians to fit must be supplied. Additional parameters may also be supplied to help the fitting routine. For each Gaussian fit, there are three parameters used to describe it: the amplitude, the X-peak position, and the full width at half the peak amplitude (FWHM). Or as an equation:

$$G_i(x) = A_i \times \exp\left(\frac{(x - B_i)^2}{C_i^2}\right)$$

where A_i is the amplitude of the ith Gaussian; B_i, the peak position; and C_i, the FWHM. The resulting curve will be the sum of the N Gaussians:

$$y(x) = \sum_{i=1}^{N} G_i(x)$$

Note that if any of the supplied Gaussian terms are negative, they will be held fixed during the fitting.

When a fit is found, this method returns three coefficients for each Gaussian fit and the chi-square value of the fit.

6.2 Limiting and Displaying Fits

Limiting A Fit

There will be some data sets that should only have a subset fit. This could be done by limiting the amount of data loaded into WIP. But that is cumbersome at the least and error prone. The `range` command allows a portion of the data to be selected and subsequent fitting will only pertain to this subset. The range command can limit data in the X-axis direction only or both the X and Y-axis directions. If the range arguments in either direction are equal, then the range is set as unlimited. Examples of how to use this command are shown in Figure E.9.

Displaying A Fit

The `plotfit` command provides an easy way to display the results of a fit. With no arguments, the fit is displayed over the entire X-axis range. The first two optional arguments provide a way to limit the X-axis extent of the displayed fit. The third argument is an optional step size. This permits a way to display the fit with finer or coarser resolution. By default, the command will use the data in the X array to guess a reasonable step size. Figure E.9 provides an example of how this command is used.

Chapter 7

Macros

7.1 General Concepts

Macros provide a way to execute a list of WIP commands repetitively. In much the same way, a programmer defines a subroutine which will do a particular task for different programs or different aspects of the same program. Hence, macros provide an easy way to tie together several plotting instructions under one name. Whenever this defined name is called, the commands associated with it are executed. Macros in WIP may be defined, edited, executed, and deleted. In fact, there always exists at least one defined macro: the command buffer.

Most of the commands that the user types are saved in a macro called the command buffer. There are a few commands, however, which do not typically need to be saved (*e.g.* `echo`, `show`, and `help`). These commands are generally used diagnostically and not graphically. For the most part, WIP "knows" which commands to save and which to not save in the command buffer. If a command is not saved, it may still be stored in the command buffer by explicitly entering it using the `insert` command (see Section 7.4).

As an example of which commands are typically saved and which commands are not, suppose that during a typical WIP interactive session a user typed the following commands:

```
WIP>  device /xwindow
WIP>  limits 0 10 0 5
WIP>  box bcnst bcnstv
WIP>  show c
WIP>  xlabel time (sec)
WIP>  ylabel Number
WIP>  erase
WIP>  box bcnst bcnstv
WIP>  move 0 2
```

```
WIP>  draw 0 3
WIP>  draw 2 3
```

At this point, if the user issued the WIP command `list`, only the following commands would appear:

```
WIP>  list
   1. limits 0 10 0 5
   2. box bcnst bcnstv
   3. xlabel time (sec)
   4. ylabel Number
   5. box bcnst bcnstv
   6. move 0 2
   7. draw 0 3
   8. draw 2 3
WIP>
```

Note that the `device`, `show`, and `erase` commands are not present in the listing. This "listing" actually holds the contents of the command buffer or, more precisely, the macro named *buffer*.

The sections that follow will explain how to define macros; pass arguments into the macros; edit, execute, and remove macros. In Chapter 9, more advanced use of macros will be illustrated. Appendix A presents how to have macros automatically defined whenever WIP is started. Finally, Appendix E illustrates several examples of macro definitions and their uses.

7.2 Creating Macros

Macros are created using the `define` command. The `define` command has only one required argument: the macro name. Macros may have any name as long as it does not conflict with any WIP command or any other previously defined macro. Macro names are case insensitive (*i.e.* `MyLabel` is considered the same macro name as `mylabel`). All valid commands entered after the `define` command are inserted into the named macro. When WIP receives a proper `define` command, the prompt changes to DEFINE> to remind the user they are in definition mode, and this prompt remains until the next `end` command terminates the definition.

Each command entered as part of the definition of a macro is checked that it is either a valid command or another defined macro and is then stored for later use. For this reason, macros may not reference other macros that have not yet been defined.[1] Of course, previously defined macros may be used within another macro definition.

[1] This is to prevent macros from calling themselves.

If a macro already exists with the same name as the macro being defined a warning is issued to the user and the previous definition is removed. The warning is issued to indicate to the user that a conflict of names exists and that the user should be careful that the proper macro is going to be used in the plot file.

7.3 Macro Arguments

There are 10 special arguments that may be used within the definition of a macro: \$1, \$2, ..., \$9, and \$0. When a macro is executed, any parameters following the macro name are assigned, in order, to the ten variables. Whenever a command in the macro references one of these variables, the command line parameter is substituted in its place. The input arguments to a macro are specified in the same fashion as when providing arguments to a command: tokens separated by white space or commas. And, as usual, arguments may be either numbers or text strings. If a requested argument is not found, the special argument is skipped. As an example of how to define a macro with arguments, consider the following example (remember the # character denotes the beginning of a comment that lasts until the end of the line):

```
WIP> define mylabel                              # Use as mylabel x y label
DEFINE> move    $1   $2                          # Move to position (x, y).
DEFINE> label $3 $4 $5 $6
DEFINE> end                                      # End of macro definition.
WIP> limits 0 10 0 10                            # Set the world coordinates.
WIP> mylabel 1   4 A    string  to   present.    # Call the macro.
WIP> mylabel 1 8 6th    argument omitted.        # Pass only 5 args.
WIP> mylabel 4 5 A string of 5 elements.         # Pass too many args.
```

In short, `mylabel` moves to the specified position and writes a label consisting of up to four words separated by single spaces. Notice that the second time the `mylabel` macro is called, it is only called with five arguments. In this case, the sixth argument is being ignored and has no effect on the text labeled. When the macro is executed, the command parser will substitute the command line arguments. When the parser does not find a requested argument (in this example, the sixth), it silently ignores it. The last time the macro is called, too many arguments are passed into it. Unneeded arguments to macros are ignored. So, in this example, the string that will actually appear will only consist of "A string of 5"; the word "elements" will be discarded.

Note also that, in the above example, the number of spaces separating the arguments in the first two calls to the macro does not matter. This is also true of the `move` command in the macro definition. However, in the `label` command, the spacing and location of the arguments is important! The macro command

```
DEFINE> label $3 $4 $5 $6
```

will produce quite different results from the statement

DEFINE> `label $3$4$5$6`

This distinction may be quite useful when defining a macro. In particular, a macro command like

DEFINE> `set \0 0$1`

will set the user variable \0 to 0 if the macro is called without any arguments. This is because the macro argument `$1` will be skipped if no arguments are present and the resulting command will be the same as if the command typed was:

DEFINE> `set \0 0`

An advanced feature of macro arguments is that of returned values. If used properly, a macro argument may be set in a macro and the new value "returned" at the end. In this way, internal counters for the `loop` command (see section 9.2) can be updated. Several examples of this technique are shown in the section containing sample plot files and figures (Appendix E). It is important to note that the macro argument that is used (and returned) is always a user variable (see Chapter 8) and never a literal value or string.

Notice that comments are used whenever possible in the macro definition, but are not present at the end of any text labeling command. A useful convention to follow is to identify the arguments to a macro in the definition of the macro and identify which user variables, if any, are changed or used by the macro. When working interactively, comments are ignored and are not saved in the command buffer. Working in INSERT mode, as well as when plot files are created external to WIP in an editor, will, however, retain what comments are typed. Hence if you define a macro externally or modify the plot file in an editor, you can place comments and label the inputs to the macro. This will greatly help you at a later time when you return to this macro and try to figure out what it does and what the inputs should be when calling it. Neither comments or identification of arguments are required, but like good programming, they are extremely valuable.

7.4 Editing Macros

All macros, including the command buffer may be edited. The three commands used to edit a macro and the command buffer are described in the sections that follow. It should be understood that whenever the word macro is used in the discussions that follow, it includes all user defined macros *and* the command buffer.

List

Whenever a user edits a macro, the edit command usually requires a line number as an argument. The line number tells the edit command where in the macro to begin to edit. These line numbers are associated with the numbers presented whenever the macro is listed. The `list` command provides a way to display the contents of a macro.

Without any arguments, `list` will display the entire contents of the command buffer. Each command will be proceeded by a number representing the order it is executed. As the number of commands in a macro increases, this may make it difficult to "see" all the commands on your display.

If only one argument is given to the `list` command, it is assumed to be the starting command line number. All commands, including the one specified, are displayed until the end of the buffer. So, the command

```
WIP> list 5
```

will list the fifth command in the command buffer, then the sixth, and so on, until the end of the command buffer.

If a second argument is present, it is the line number at which the listing will stop. This is useful when the user only wants to display the first few lines of a macro. If this second number is greater than the number of commands in the buffer, then the `list` command will present all commands from the starting line number to the end of the buffer. In this way, the command

```
WIP> list 5 10000
```

will present every command in the command buffer starting with the fifth. (This is identical to the command shown before.)

Finally, a third argument, if present, will inform the `list` command which macro to list. If the user wishes to display the entire macro, then the command

```
WIP> list 1 10000 mymacro
```

will surely be sufficient to cover the entire listing of the macro `mymacro`. Small sections of the macro may be specified by judicious selection of the input line number arguments.

Insert

Commands are never typed in the way we want them on the first time through a plot. Often, a color change or line style adjustment extends further in the plot file than we first expected. Another command is needed just a few lines prior to the current command. Or perhaps we have, too hastly, finished a definition of a macro and forgot to include an important command. These situations can be quite easily fixed with the `insert` command. The `insert` command may have up to two optional arguments (discussed below).

When the `insert` command is called, it places the user into insert mode. This is demarked by the prompt changing to INSERT>. All commands typed at this point are checked that they are valid commands (if not, they are ignored) and are inserted in the proper order (without being executed). Insert mode is terminated by entering the `end` command.

Without any arguments, `insert` starts inserting commands at the end of the command buffer. This is the same as interactive mode except that the commands typed are not executed. If one argument is present, it is considered the insert line number and all commands will be inserted *before* that line in the macro. Hence, if the command buffer has three commands currently in it and the user types

```
WIP>  insert 10000
```

the effect will be the same as if the user had typed

```
WIP>  insert 4
```

or just

```
WIP>  insert
```

If two arguments are present, then the first argument specifies which line number to insert commands before (as above) and the second argument is the name of the macro in which commands will be inserted. As an example, if the user has a macro defined called `mymacro`, then the command

```
WIP>  insert 1 mymacro
```

will allow the user to insert commands at the beginning of the named macro.

Delete

Just as users may wish to insert commands in either the command buffer or their own macros, they will want to remove certain commands. The `delete` command permits the user to specify lines of commands or even entire macros to be deleted. A macro is automatically deleted whenever the number of commands it contains reaches zero.[2] Deleting an entire macro is sometimes useful if the user wishes to define another macro with the same name.

The `delete` command may have up to three optional arguments. With no arguments present, it removes the last line in the command buffer. The first two arguments are considered as command line numbers. With only one argument, the specified command line in the command buffer is removed. If the line number is greater than the number of commands present in the macro, the last command is removed. If

[2] The command buffer is never removed; even if there are no commands in it.

two arguments are present, then the inclusive range of commands are removed from the command buffer. Finally, if all three optional arguments are present, then the lines removed are from the named macro. Hence, the command

WIP> `delete 4 4 mymacro`

will remove only the forth line from the macro `mymacro`. The command

WIP> `delete 1000 1000 mymacro`

will remove only the last command from the macro `mymacro` (if there are fewer than 1000 commands in the macro).

7.5 Executing Macros

Once a macro has been defined, it is executed just like any of the other WIP commands. To execute a macro, just call it by name. The macro name may also be followed by any required or optional arguments as is necessary.

In addition, a macro may be executed without being saved in the command buffer. This is accomplished with the `playback` command. Without any arguments, `playback` executes the commands in the command buffer. This is an extremely useful command when the constructed plot begins to become quite complicated. The optional arguments to `playback` is the name of the macro to be executed and any arguments needed by the macro. Since the `playback` command is not saved in the command buffer, this permits a user the opportunity to test their macros. However, the real usefulness of the `playback` command is in repeating the plot file constructed.

7.6 Saving and Recalling Macros

Once a user has defined a useful macro, they will undoubtedly wish to save them to a file for use again. This is accomplished with the `write` command. The `write` command permits named macros and the command buffer to be written to an ASCII text file. The only required argument to the `write` command is the name of the output file. With no other arguments, the command buffer is written to the named file. If any other arguments are present after the file name, they are considered to be the macros to be saved and these named macros are written to the specified file. Multiple macros may be written to the same file with only one `write` command. If the macro name listed is "all", then all macros currently defined *and* the command buffer will be written to the named file.

To retrieve macro definitions, WIP provides three commands. Each command treats the input file differently and are explained in brief detail in the following list.

macro *filename* – Used to define macros. Only macro definitions are read in; any other commands are ignored. The macro is not executed but is now available for further execution.

read *filename* – Used to read in plot commands into the command buffer. Commands are placed into the command buffer without executing them (this is similar to the `insert` command). If the file contains macro definitions, they will be not placed into the command buffer or defined; they must be entered using the `macro` command.

input *filename* – Commands are executed just as if they were typed in interactive mode but are not saved in the command buffer. Macro definitions are acceptable but will overwrite (redefine) the macro. A message is printed when a macro is redefined. Only exact matches to the macro name are overwritten. Because of this, care should be taken to insure that the proper macro is being executed.

Chapter 8

User Variables

8.1 General Concepts

Available to the WIP user are a host of variables. Many of these variables are associated with the general graphical attributes discussed in Chapter 3; some with the current image (Section 5); and some are available to perform simple to complex calculations. Whenever the user accesses a graphics tool in WIP, they are probably setting the value of an internal variable. In general, these variables need not (and should not) be accessed directly. These variables may, however, be used to probe the current values and may also be used as arguments to their respective commands. Some of the variables, however, can only be accessed directly and are designed for that use. The sections that follow will describe the variables that are currently available to the user (Section 8.2) and the tools used to access them (Section 8.3). In addition, there are ways the user can create and destroy their own user variables (Section 8.4).

User variables enable the WIP user to perform quite complicated tasks easily. In addition, it allows the user to perform commands without knowledge of the actual current *value* of the user variable. For example, as will be shown in Section 8.2, the user variables that specify the limits of the current world (or User coordinate) window are `x1`, `x2`, `y1`, and `y2`. Suppose an astronomer reads in a list of x and y values that correspond to right ascension and declination positions, respectively. If the limits are determined automatically with the **`limits`** command (*i.e.* the command called with no arguments), then the x axis values will increase to the right. But right ascension increases to the left. The following WIP listing shows how, with the use of user variables, this can be corrected... without knowing the actual values of `x1`, `x2`, `y1`, and `y2`:

```
WIP> data myfile             # Access the data file.
WIP> xcolumn 1               # Read the RA positions.
WIP> ycolumn 2               # Read the Dec. positions.
WIP> limits                  # Autoscale the limits.
WIP> limits x2 x1 y1 y2      # Flip only the x axis limits.
```

Note that by calling `limits` with the four user variables as arguments, the user does not need to know the *actual* values or contents of the user variables, just the names of the variables. Furthermore, by switching the order of the x-axis arguments, the orientation of the x-axis direction has been flipped.

This works because WIP checks for arguments and expects literal numeric values whenever it is passed a command. If WIP cannot "understand" the argument as a number it searches the list of user variables for a match. If a match is made, the current value of the user variable is substituted internally for the corresponding command argument.

8.2 Pre-Defined Variables

There are two types of user variables available: numeric variables and string variables. These two types cannot be mixed at any time! Attempting to mix these types of user variables will either create an error message or produce unpredictable and undesirable results.

Tables 8.1 – 8.3 identify the current predefined numeric user variable names, a description, and a (typical) command which may be used to set it (the index at the end of the manual will direct the reader to a more detailed description of each of the commands mentioned as well as possibly a few more that are applicable). The tables are broken up into areas where the user variable is most applicable.

In addition, Table 8.4 identifies the current predefined numeric user variable array (vector) names, the maximum defined size of the array, and the typical commands used to load the array. Some of these entries have the maximum size set by the user variable **`maxarray`**. This is initialized to 20,000 by WIP but may be overridden by setting it to a larger or smaller number. This must, however, be done right when WIP starts. The best way to insure this is done correctly is to specify a different value in the user's `.wipinit` file (see Appendix A for further details). For example, to set this value to 10,000, the following command would be put in the user's `.wipinit` file:

```
WIP> set maxarray 10000      # Limits array size.
```

In general, though, the default value should be sufficient for almost all needs.

The string user variables may be fewer in number but are just as powerful as the numeric user variables. The current pre-defined string user variables are listed in Table 8.5 along with the primary command used to set each, a description of each variable, and the corresponding default string value.

Table 8.1: User Variables – Coordinate Systems

Variable	Command	Description
x1	`limits`	World Coordinate left x-axis limit value.
x2	`limits`	World Coordinate right x-axis limit value.
y1	`limits`	World Coordinate lower y-axis limit value.
y2	`limits`	World Coordinate upper y-axis limit value.
vx1	`viewport`	Viewport Coordinate left x axis limit.
vx2	`viewport`	Viewport Coordinate right x axis limit.
vy1	`viewport`	Viewport Coordinate lower y axis limit.
vy2	`viewport`	Viewport Coordinate upper y axis limit.
cx	`cursor`	Current World Coordinate x value (pen position).
cy	`cursor`	Current World Coordinate y value (pen position).
xtick	`ticksize`	Major tick mark interval along x axis (World units).
ytick	`ticksize`	Major tick mark interval along y axis (World units).
nxsub	`ticksize`	Number of minor tick marks along the x axis.
nysub	`ticksize`	Number of minor tick marks along the y axis.
xsubmar	`submargin`	Panel gap size in x-direction (char. height units).
ysubmar	`submargin`	Panel gap size in y-direction (char. height units).

Table 8.2: User Variables – Images

Variable	Command	Description
subx1	`subimage`	Left x pixel number of current subimage.
subx2	`subimage`	Right x pixel number of current subimage.
suby1	`subimage`	Lower y pixel number of current subimage.
suby2	`subimage`	Upper y pixel number of current subimage.
crvalx	`header`	Reference pixel value in the x-direction (World units).
crpixx	`header`	The reference pixel in the x-direction.
cdeltx	`header`	The x-direction step size between pixels (World units).
crvaly	`header`	Reference pixel value in the y-direction (World units).
crpixy	`header`	The reference pixel in the y-direction.
cdelty	`header`	The y-direction step size between pixels (World units).
slevel	`slev`	The scaling factor applied to the contour level array.
immin	`image`	Minimum intensity value of the current image.
immax	`image`	Maximum intensity value of the current image.
nx	`image`	Number of pixels in the current image's x direction.
ny	`image`	Number of pixels in the current image's y direction.

Table 8.3: User Variables – Graphical Attributes

Variable	Command	Description
angle	`angle`	Current rotation angle value (in degrees).
expand	`expand`	Current character expansion value (character height units).
color	`color`	Current color index value.
fill	`fill`	Current fill type value.
font	`font`	Current font type.
lstyle	`lstyle`	Current line style.
lwidth	`lwidth`	Current line width.
cmin	`device`	Current device's minimum color index.
cmax	`device`	Current device's maximum color index.
bgci	`bgci`	Value of argument to the most recent call to `bgci`.
itf	`itf`	Value of argument to the most recent call to `itf`.
cursmode	`set`	Value determines type of cursor drawn.
hardcopy	`device`	1 if current device is hardcopy; 0 otherwise.
maxarray	`set`	Initializes the maximum size of the **X**, **Y**, **ERR**, and **PSTYLE** arrays.
nsig	`set`	Number of significant digits used to print a variable.
pi	(none)	The value of π.
\0 – \20	`set`	Storage registers 0 – 20.

Table 8.4: User Variables – Numeric Arrays

Variable	Maximum Size	Command
x	`maxarray`	`xcolumn`
y	`maxarray`	`ycolumn`
err	`maxarray`	`ecolumn`
pstyle	`maxarray`	`pcolumn`
transfer	6	`transfer header`
levels	40	`levels autolevs slev`

Table 8.5: String User Variables

Variable	Command	Description	Default
device	`device`	Name of the current device.	/xwindow
datafile	`data`	Name of the current data file.	empty
imagefile	`image`	Name of the current image file.	empty
helpfile	`set string`	Name of the on-line help file.	$WIPHELP
xbox	`set string`	x-axis argument for `box`.	bcnst
ybox	`set string`	y-axis argument for `box`.	bcnstv
print	`set string`	The print plot file command.	lpr
xheader	`set string`	x-axis argument for `header`.	rd
yheader	`set string`	y-axis argument for `header`.	rd
levtype	`slev`	Type of scaling for contour levels.	absolute

8.3 Accessing the User Variables

This section describes the WIP commands that access and change the value of the user variables. Remember that there are two types of user variables: numeric and string. The primary command used to access all User Variables is `set`. In addition, another command used to set the string variables is called `string`. The command `echo` can be used to display the value of many of the user variables (both numeric and string) on the command line. Also of use is the `show` command. The `show` command presents many of the user variables along with their current values. The user variables are listed by `show` in upper case along with their current value.

The SET Command

The command `set` requires at least two arguments. The first (required) argument is an acceptable user variable name. This may be any predefined user variable (see Tables 8.1 – Table 8.5) or any user variable defined with the `new` command (see below). The remaining arguments are treated differently depending on the type of user variable being set. The next two sections discuss how `set` treats (i) numeric and variable array (vector) user variables and (ii) string user variables.

Numeric User Variables

When evaluating numeric user variables (both simple numeric and vector), the first argument will be set to the result of the expression consisting of the tokens appearing in the rest of the command. The expression is used to determine what value will be assigned to the user variable. The syntax of the `set` command expression in this case is just a standard mathematical expression with the following exceptions:

1. The inner most expression enclosed by parenthesis is always evaluated first.
2. All expressions are evaluated left to right.
3. All functions must have a set of parenthesis delimiting their argument.
4. All operators that act on two variables (binary operators) must appear between their arguments.

In many ways, the `set` command is similar to a basic calculator.

The most straightforward use is to set the user variable equal to something. For example (and note the absence of the = sign),

WIP> `set x1 0.1` # Sets the user variable `x1` to the value of 0.1.

Table 8.6 identifies the operations that act on two variables. These operators allow user variables to be added, subtracted, and so on. They also allow two user variables (or expressions) to be compared with the validity of the comparison as the result. As mentioned already, there is no precedence of the various operators except that of being evaluated from left to right. This may be explicitly specified by enclosing various operations within a matching pair of parentheses. Note that the logical operations will only return a value of 1 if the expression evaluated is true or a value of 0 if it is false. Only these two values are returned regardless of the value of the two arguments used to test the conditional.

Table 8.7 lists functions that operate on single variables. Functions are identified by WIP when it sees a recognized function name followed by parenthesis which enclose the variable. The argument within the parenthesis is evaluated and then passed to the function. Hence, the expression within the parenthesis may be any valid expression (even another function) as long as it evaluates to a single variable. Note that functions appearing in the top part of Table 8.7 require a *numeric value* as the argument to the function and functions listed in the bottom part require a *vector name*.

One note about two of the functions listed in Table 8.7. The functions rand(x) and gasdev(x) use a negative integer argument to help seed the random number generator functions. By default, a seed is pre-supplied by WIP. Hence, the user only needs to supply an argument of zero each time one of these functions is called. If the user wishes to start with a different seed value, then the first time one of these functions is called, the user should supply an odd, negative integer value. On subsequent calls, the user should still supply a value of zero. This is because the seed is retained internally and it is used if any positive argument is passed to the function.

Note that an argument in the expression of the `set` syntax can be easily made negative by placing the minus sign before the user variable name. For example, the commands

```
WIP>  set \0 -\0
WIP>  set \1 tand(-angle)
WIP>  set \2 2 ** -x1
```

Table 8.6: Double Variable Operations

Binary	Result
+	Addition of the two arguments.
−	Subtraction of second from first argument.
*	Multiplication of the two arguments.
/	Division of first argument by second argument.
\	Integer result of division of first by second argument.
%	Remainder of first argument divided by second argument.
max	Larger value of the first and second argument.
min	Smaller value of the first and second argument.
**	First argument raised to the power of the second argument.
Logical	**Result is 1 only if the relationship is true; 0 otherwise**
>	True if the first argument is greater than the second.
<	True if the first argument is less than the second.
>=	True if the first argument is greater than or equal to the second.
<=	True if the first argument is less than or equal to the second.
==	True if the first argument is equal to the second.
!=	True if the first argument is not equal to the second.
$\vert\vert$	True if either the first or second argument is not equal to zero.
&&	True if both the first and second arguments are not equal to zero.
^	True if first or second argument (but not both) is not equal to zero.

Table 8.7: Single Variable Functions

Function	Numeric Argument	Result
$\sin(\theta)$	(θ) – angle in radians	$\sin(\theta)$
$\text{sind}(\theta)$	(θ) – angle in degrees	$\sin(\theta)$
$\text{asin}(x)$	$-1 \leq (x = \sin(\theta)) \leq 1$	Angle $(-\pi/2 \leq \theta \leq \pi/2)$ in radians.
$\text{asind}(x)$	$-1 \leq (x = \sin(\theta)) \leq 1$	Angle $(-90 \leq \theta \leq 90)$ in degrees.
$\cos(\theta)$	(θ) – angle in radians	$\cos(\theta)$
$\text{cosd}(\theta)$	(θ) – angle in degrees	$\cos(\theta)$
$\text{acos}(x)$	$-1 \leq (x = \cos(\theta)) \leq 1$	Angle $(0 \leq \theta \leq \pi)$ in radians.
$\text{acosd}(x)$	$-1 \leq (x = \cos(\theta)) \leq 1$	Angle $(0 \leq \theta \leq 180)$ in degrees.
$\tan(\theta)$	(θ) – angle in radians	$\tan(\theta)$
$\text{tand}(\theta)$	(θ) – angle in degrees	$\tan(\theta)$
$\text{atan}(x)$	$x = \tan(\theta)$	Angle $(-\pi/2 \leq \theta \leq \pi/2)$ in radians.
$\text{atand}(x)$	$x = \tan(\theta)$	Angle $(-90 \leq \theta \leq 90)$ in degrees.
$\text{sqrt}(x)$	Any argument $x > 0$	Non-negative square root.
$\ln(x)$	Any argument $x > 0$	Base e logarithm.
$\log(x)$	Any argument $x > 0$	Base 10 logarithm.
$\text{log10}(x)$	Any argument $x > 0$	Base 10 logarithm.
$\exp(x)$	Any argument $x < 85$	Base e exponent function.
$\text{abs}(x)$	Any argument x	Absolute value of the argument.
$\text{int}(x)$	Any argument x	Integer value of the argument.
$\text{nint}(x)$	Any argument x	Nearest integer of the argument.
$\text{rand}(x)$	An integer seed x	Uniform random deviate [0, 1).
$\text{gasdev}(x)$	An integer seed x	Normal (Gaussian) random deviate [0, 1).

Function	Vector Argument	Result
$\text{npts}(x)$	Any vector name x	Current size of the named vector.

will set, respectively, the user variable \0 to the negative of its current value; the user variable \1 to the tangent of the negative of the current angle variable; and \2 to the value of two raised to the negative of the current value of `x1`.

String User Variables

When the user variable is a string variable, the first argument to `set` is the string variable name and the remaining arguments are treated as a string to be assigned to that variable. This is the easiest way to set a string variable. For example,

```
WIP> set print ignore
```

will set the string variable *print* to the string `ignore`. This overrides the default value such that when a plot file is ready to be spooled to the printer, the spool command will not send the file to the printer.

The syntax of the `set` command is such that it requires at least one argument: the string variable name. If no other arguments are present, the string variable is set to an empty string. If other arguments are present, then the remaining arguments are copied, exactly as entered, into the string variable. Hence, string variables may contain several tokens (words) in its translation.[1]

The string variables in Table 8.5 not directly associated with the `set` command (*device*, *datafile*, *imagefile*, etc.) are set by the listed commands and generally should not be changed with the `set` command. The other string variables listed in Table 8.5 are specifically designed to be accessed by the user to allow WIP to be customized according to their needs. The `set` command changes the value of the variable until either the end of the plotting session, until another `set` command, or until it is reset by a command. Additionally, `set` commands may be placed in the user's `.wipinit` file (see Appendix A for more details) to always override WIP's initial values.

The `string` command is another method used to set string user variables. The `string` command allows the user to set the string variable using the contents of an external file. Only one line is read by the command but the user may specify which words of that line are actually loaded into the string variable. This command, along with its arguments, is discussed further in Section 3.7 and Appendix D.

8.4 Creating and Removing User Variables

New user variables may easily be defined with the `new` command. Likewise, the user defined variables may be removed with the `free` command. The ability to define user variables allows the user to create local variables in macros, retrieve and assign image header variables, and associate a value with a variable name more in keeping with its use. The only limitation on user defined user variables is that they may not have the same name as any other user variable, vector, string variable, or function.[2]

[1] The current maximum length for the value of a string variable is 256 characters.

[2] Just like predefined commands, no predefined user variable, vector, string variable, or function may be removed.

As an example of local usage of user variables, consider the following macro:

```
WIP> define radec                                   # $1–$6=rah ram ras decd decm decs
DEFINE> new ra dec                                  # Create new user variables: "ra" and "dec".
DEFINE> set ra ((($1 * 60.0) + $2) * 60.0) + $3
DEFINE> set dec (((abs($4) * 60.0) + $5) * 60.0) + $6
DEFINE> if $4 < 0 set dec -dec
DEFINE> move ra dec                                 # Move to the position (ra, dec).
DEFINE> free ra dec                                 # Remove the user variables ra and dec.
DEFINE> end                                         # Finish the definition.
```

This macro will convert a Right Ascension and Declination position into units of seconds of time and seconds of arc and moves the current position to that location. By creating the user variables `ra` and `dec` within the macro and then removing them prior to leaving it, the user has limited the scope of these variables. In addition, the user has also removed the possibility of inadvertently altering their value by another macro.

The user should be aware, however, that defining user variables is a global operation. Just because a variable is defined within a macro does not mean that it is only seen by that macro or is active only while in that macro. On the contrary, user defined variables exist in same way as predefined user variables and do so until they are removed. For this reason, care is needed when defining local variables. In the example above, another macro could define these same variables and then call the macro `radec`. This would cause the macro `radec` to abort when the `new` command is reached. Therefore, make sure your local names are "really" local (*i.e.* unique).

Chapter 9

Control Flow

This chapter describes methods of how to control the execution of plot commands. Most WIP commands are called once and, generally, only when the user needs them. If a command or group of commands are going to be called several times the user usually creates a macro using the techniques described in Chapter 7. If a macro or command needs to be executed a number of times or only under certain conditions, then tools are needed to enable the user to do this. The following sections describe these advanced tools and provide examples of their use.

NOTE: If you are unfamiliar with the definition and use of macros and user variables, then you should first refer to Chapters 7 and 8 before continuing with the following sections.

9.1 IF

An Example of How to Use the IF Command

To demonstrate how to use the `if` command, let's create a plot where the use of the command would be the only logical way to solve the problem. Suppose we want to draw several plots on one view surface and want to draw a box around each plot but only have the left most and bottom most boxes labeled with numbers along with the tick marks. Since we are going to draw essentially the same plot in each box, the easy solution will be to define a macro that uses the `panel` command to select various regions on the view surface and then only label select panels when drawing the box.

To make this example a bit more realistic, suppose we have an image cube that contains 12 planes and they are to be plotted on a 3×4 grid. Because each of the 12 panels on the display is going to have contours drawn in them, it would probably be desired to have as much space available for each plot window as possible. Hence, it will be best to have no space between neighboring panels. Also, remember we will only (numerically) identify the tick marks on the left most and bottom most panels;

the other panels will be bordered and have unlabeled tick marks. The following fragment shows the macro definition and the commands used to perform this task.

```
WIP>    define contbox                     # $1=counter; uses \1.
DEFINE>  set $1 $1 + 1                      # Increment plane number.
DEFINE>  panel -3 -4 $1                     # Select the active panel.
DEFINE>  image myimage.img $1               # Load this image plane.
DEFINE>  contour                            # Draw the contour plot.
DEFINE>  box bcstz bcstz                    # Draw a simple unlabeled box.
DEFINE>  set \1 ($1 - 1) % 3                # If a left panel, \1 = 0.
DEFINE>  if (\1 < 1) box 0 nvdyz            # Label the y-axis only.
DEFINE>  set \1 ($1 - 1) \ 3                # If a bottom panel, \1 = 0.
DEFINE>  if (\1 < 1) box ndz 0              # Label the x-axis only.
DEFINE>  ...                                # Perhaps more macro commands....
DEFINE>  end                                # End of the macro definition.
WIP>    autolevs 10 lin 1 10                # Set up 10 contours.
WIP>    slev P 10                           # Draw the contours at every 10%.
WIP>    set \0 0                            # Initialize so the first is plane 1.
WIP>    contbox \0                          # Draw lower left box with plane 1.
WIP>    contbox \0                          # Move right and draw plane 2.
:
:                                           # Call macro contbox 9 more times.
WIP>    contbox \0                          # Move right and draw the final plane (12).
WIP>    ...                                 # Maybe more WIP commands....
```

In the macro definition, user variables are used to keep track of the current plane number and the current panel frame. Furthermore, this index is used to determine whether or not the current frame is located along the bottom or the left side of the composite picture. The line

```
DEFINE>  set \1 ($1 - 1) % 3
```

stores in the user variable \1 the value of $((plane - 1) \bmod 3)$. This will only evaluate to zero when the current panel is on the left edge of the composite plot. Likewise,

```
DEFINE>  set \1 ($1 - 1) \ 3
```

performs an integer divide of $(plane - 1)$ by 3. This will only be zero when the current panel is along the bottom of the plot.

The use of the `if` command should now become more apparent. Prior to testing whether the current panel is on either the left or bottom edges, a call to the `box` command is made with only the most basic options specified (see Table 3.4 for a listing of the `box` command options). Then, if the current frame is located on the left edge (*i.e.* the value of the user variable \1 is less than 1), the command `box` is called again but with a NULL argument for the x-axis and a numeric labeling argument for the y-axis. If the frame is not on the left edge (*i.e.* the value of the user variable \1 is greater than 0), then this `box` command is silently ignored. In the same fashion, if the current panel is along the bottom edge, then the next `if` command will call `box` to provide a numeric label for the x-axis and a NULL argument for the y-axis; otherwise the `box` command will be skipped.

Other Uses of the IF Command

The section on `if` in Appendix D as well as the on-line command `help if` will describe the conditional operations that are acceptable as arguments to this command. They are essentially the logical operations presented in Table 8.6.

The `if` command can also permit the user to leave a macro before the last command defined in it is reached. For example, if a macro is drawing points on a plot and the user only wants certain symbol types to be drawn, then an `if` command could allow the user to exit the macro (gracefully) before reaching the undesired plot commands. Another example might be to check for boundary limits before executing further steps in a macro.

It should be noted that the `if` conditional will only apply to the command or macro that follows on the same line. Furthermore, only one command or macro may be present on a single line. To have one conditional result in several commands being executed either chain several `if` commands together or create a macro and have the macro name as the result of the `if` command. For example, if the user had a situation where there were only certain conditions that a labeled box should be drawn along with a dotted line at $y = 0$, then the user could use the following group of commands:

```
WIP>  if (\1 > 0) box bcnst bcnstv
WIP>  if (\1 > 0) lstyle 4
WIP>  if (\1 > 0) move x1 0
WIP>  if (\1 > 0) draw x2 0
WIP>  if (\1 > 0) lstyle 1
```

Writing the conditional commands this way is useful only if the conditional commands are used once or if the number of conditional commands are few. The next listing will produce the same results as the example before except that a macro is used. This listing is not much more compact than the previous example but would be far easier to enhance and maintain than the previous example. In addition, it groups all the commands that are affected by the conditional command in one place.

```
WIP>  define myifbox
DEFINE>  box bcnst bcnstv
DEFINE>  lstyle 4
DEFINE>  move x1 0
DEFINE>  draw x2 0
DEFINE>  lstyle 1
DEFINE>  end
  ⋮
WIP>  if (\1 > 0) myifbox                # If conditional is true, execute macro.
```

The IF Command with Complicated Expressions

All of the above examples have shown only one argument on each side of the conditional operation. Also, only one conditional operation was present. The entire conditional expression, however, can be made arbitrarily complex. For example, in the example that started this section, a test was made to determine if the current panel was on the left edge and if so, draw a labeled box. The two commands used to perform this action were:

```
DEFINE>  set \1 ($1 - 1) % 3             # If a left panel, \1 = 0.
DEFINE>  if (\1 < 1) box 0 nvdyz         # Label the y-axis only.
```

This can be reduced to one command as in

```
DEFINE>  if ((($1 - 1) % 3) < 1) box 0 nvdyz
```

Along the same line, if the macro `myifbox` was defined as above, but now is only to be executed when `\1 > 0` *and* the line style is solid, then the following command will execute the macro properly:

```
WIP>  if ((\1 > 0) && (lstyle == 1)) myifbox
```

However, as these commands become more complex, they may be more difficult to understand the meaning or intent and also are more difficult to track down when errors appear.

9.2 LOOP

The command `loop` provides the user with the ability to execute a macro several times with only one call. This command has two required arguments: (1) the number of times the named macro should be executed, and (2) the macro to execute. Any other parameters present with the command are considered as arguments to be passed to the macro (see Section 7.3).

To illustrate the use of the command `loop`, let us refer back to the example plot file fragment illustrated in Section 9.1. To review, a macro was defined which will display several contour plots on one plot labeling only the left and bottom most panels. After the macro was defined, the image counter (\0) was initialized such that the first plane (and first panel) was correctly set when the macro was called. Next, the macro was called twelve (12) times and then some other WIP plotting commands were possibly issued. Instead of calling the macro twelve times, the command `loop` might have been used to perform the same function. Hence, the commands that appear after the macro definition in that example may be replaced with the WIP commands shown in the following code fragment:

```
WIP>  set \0 0                        # Initialize so the first plane will be 1.
WIP>  loop 12 contbox \0              # Draw all 12 contour plots.
WIP>  ...                             # Possibly more WIP commands....
```

Unlike the `if` command, the `loop` command may only operate on a macro (*i.e.* a user may not execute a single command with a `loop` command).[1]

Up until now, nothing has been said about the required argument to the example macro `contbox` (defined on page 66). Recalling the first part of the definition:

```
WIP>     define contbox               # $1=counter; uses \1.
DEFINE>  set $1 $1 + 1                # Increment plane number.
DEFINE>  ...                          # The rest of the definition.
DEFINE>  end                          # End of the macro definition.
```

note the form of the `set` command. Because the macro was called with a simple user variable,[2] (\0), the `set` command, when expanded by the macro processor, becomes a valid WIP command. That means that the command

```
WIP>  contbox \0
```

will produce, internally, the command

```
WIP>  set \0 \0 + 1                   # Increment plane number.
```

Every time the macro is called, this `set` command will increase the value of the user variable \0 by 1. This technique permits macro users to iterate with the same macro but with different counters. This also, in essence, provides a way to *return* values to the macro caller. Refer to the plot files and figures in Appendix E for further examples of this technique.

[1] A very simple solution to this would be to define a macro with only the one command needed and then use `loop` to execute it repetitively.

[2] Note that calling this macro with a vector, string variable, or function is an error and will cause the macro to abort.

Part III

Appendices

Appendix A

The `.wipinit` File

WIP has set up most of the default attributes so that the user should not have to "worry" about them until they want to alter them to enhance their plot. There may be a time, however, when a user will want to customize characteristics a certain way. For example, suppose a user typically works with a tektronix device. The hard coded device type is the `/xwindow`. To override this default device, the user could start WIP with the command

```
% wip -d /tek
```

every time they use WIP. Obviously, a simpler solution is desired. Placing the line

```
set device /tek
```

in the file `.wipinit` in the user's home directory will cause WIP to always act as if the hard coded initial device was a tektronix terminal.[1]

An another example, suppose a user typically displays images with the x-axis representing Right Ascension and the y-axis Declination. Then, this user might like to have a call to `box` without any arguments to correctly label the x-axis in hours, minutes, and seconds and the y-axis in degrees, minutes, and seconds. By default, WIP defines *xbox* as the string "BCNST" and *ybox* as "BCNSTV" (see Table 8.5 for an explanation of these characters). One alternative might be that the user would specify the string variables *xbox* and *ybox*

```
WIP>  set xbox bcnsthz
WIP>  set ybox bcnstvdyz
```

whenever they entered WIP. A better alternative, however, would be for the user to specify these string

[1]This default device may still be overridden with the command line option *-d device-type* (see Appendix B).

variables in their `.wipinit` file as:

```
set xbox bcnsthz
set ybox bcnstvdyz.
```

Then, whenever WIP is started, these values will become the default values for the string variables *xbox* and *ybox*.

In addition to setting these variables, a user may also want to automatically load files containing macro definitions or set other user variables to certain values whenever WIP is started. A user should use caution when using the `set` command in their `.wipinit` file: calculations made with variables established by the device (*e.g.* X1, VY1, FILL, etc.) are undefined at this point and, hence, so are results based on their values. For this reason, the command `set` should generally be used cautiously in the user's `.wipinit` file except for string variables and the most rudimentary of calculations.

As an example of how to define macros automatically, suppose a user has a file named `mymacro.wip` that contains a macro definition that is used often. Placing the line

```
macro mymacro.wip
```

in the user's `.wipinit` file will automatically load the macro definitions from the named file every time WIP is started. Remember everything in the named file but macro definitions will be ignored by the `macro` command.

Table A.1 lists the commands and string variables that may be specified in your `.wipinit` file. Note that the user variable `maxarray` can only be set in the `.wipinit` file. This is because once these arrays are initialized, they can not have their maximum size changed. In addition, comments may be placed anywhere in the `.wipinit` file as long as they begin with the comment character (#). For a further explanation of how to define macros, see Chapter 7. The command `set` and user variables are discussed more fully in Chapter 8.

A typical `.wipinit` file might contain the following lines:

```
set device /xserve
set xbox BCNSTHZ
set ybox BCNSTVDYZ
set print lp -d ps -w
macro /myroot/subdir/macro1.wip
macro /myroot/macro2.wip
```

where the user has two files containing macros (`/myroot/subdir/macro1.wip` and `/myroot/macro2.wip`) that are always loaded whenever WIP begins. In addition, this example illustrates how to set the default graphics device to the `/xserve` X window driver; change the default arguments for the `box` command; and set the print command to send the plot (under Solaris) to the printer named `ps` with a notification to the user after it is printed.

Table A.1: Acceptable commands in the `.wipinit` File

Command	Required Argument	Default value
set device	"device-name/device-type"	/xwindow
set xbox	Box options (Table 3.4) for the X-axis	BCNST
set ybox	Box options (Table 3.4) for the Y-axis	BCNSTV
set print	Print command (System dependent)	`lpr`
set xheader	X-axis header options (Section 5.2)	rd
set yheader	Y-axis header options (Section 5.2)	rd
set levtype	Contour scaling type (Section 5.2)	absolute
set helpfile	The on-line help file (System dependent)	$WIPHELP
set maxarray	The maximum size of the **X, Y, ERR**, and **PSTYLE** arrays. This must be set in the `.wipinit` file.	20,000
set	Set options (see Chapter 8)	(none)
macro	Macro file name	(none)
new	New user variable name (Section 8.4)	(none)
free	User variable name to remove (Section 8.4)	(none)

Appendix B

Command Line Interface

WIP can always be run by specifying parameters and plot files on the command line. The command line interface is quite appropriate for command procedures and shell scripts.[1] It is also very useful for getting a plot command file loaded into WIP.

The format for calling WIP from the command line is

```
% wip [optionalArgs] [plotfile ...]  [-e wipcmd]
```

where `optionalArgs` includes the following options:

```
[-d device] [--] [-x] [-r] [-b] [-g] [-q] [-h] [-?]
```

Except for the `-e` option (discussed below), the WIP options are, by convention, placed before any of the plot file names. The options may, however, appear anywhere on the command line (after the WIP command); but they are *always* evaluated before reading any of the plot files.

All items enclosed in square brackets ([...]) are optional. The following list explains the command line arguments and what function they perform.

`-d` *device-type* – Sets the initial graphics device to *device-type*. This option overrides the default device *and* any device specified in the user's `.wipinit` file. See Section 3.1 on how to specify the *device-type*.

`--` – Specifies `stdin` as an input plot file. This may have some use when using WIP in a script. Note that this option also forces the `-x` option because input ends when all of data from stdin has arrived.

[1] If you are not familiar with these terms, don't worry, they refer to advanced VMS and UNIX concepts you can learn about. Check with someone more experienced than you or with a system manager to learn more.

-x – Effectively "turns off" the interactive state of WIP. After WIP has processed all of the plot files on the command line, a close plot command is issued and WIP exits. If the current graphics device is a hardcopy device, then the `hardcopy` command is issued when the last plot file has finished.

-r – If WIP has been compiled with the `READLINE` library, then each time a user exits from WIP, a history file is written which contains all of the commands typed. This history file is read in by the `READLINE` routines when a user enters WIP again and the commands are available for recalling, editing, and executing. Use of this option suppresses the writing of the history file when the user exits from WIP. Any previous commands will still be available for recall and the `READLINE` editing capabilities will still exist.

-b – This option completely disables the `READLINE` capabilities. This option may be useful for terminals that do not permit the escape sequences to be passed to the `READLINE` routines.

-g – Opens WIP in debug mode. This is not generally useful to the user but might provide some additional comments in cases where the usual diagnostic messages don't help.

-q – Turns off all informational messages from WIP. This is useful when the selected output file for the device is `stdout`. This option is disabled, however, when WIP enters interactive mode.

-h or -? – Writes a usage statement listing all of the command line options available within WIP and then immediately exits. No plotting is done.

The remaining optional arguments to the WIP command (up to the optional `-e` flag) are the names of either macro or plot command files. These files are only read[2] after the `.wipinit` file has been read, the command line arguments set, and the initial device (if any) has been opened. The remaining arguments are then processed just as if the user had typed the command lines interactively. When the last file is finished being read, WIP moves into interactive mode (unless the "`-x`" option is used). All macros defined in the input files are now defined in WIP and all plot commands are present in the command buffer.

As an example of how to use the command line arguments, consider the following command:

```
% wip -x -d myfile.ps/vps myfile.wip
```

The file `myfile.wip` is a simple ASCII text file of commands (similar to the listings that appear in this manual or those saved with the `write` command). This type of file is a useful way to insert a plot command listing, for playing back commands (say to another device), or for defining commonly used macros. The argument `-d myfile.ps/vps` is the command line switch that WIP uses to redefine the initial device. This example illustrates how to specify a file name (`myfile.ps`) and associate with a hardcopy device (`/vps`). For more details on graphical devices, see Section 3.1. Finally, the `-x` flag tells WIP to exit when it is finished reading all of the command files on the command line (in this case, just the file `myfile.wip`).

[2]Files on the command line are read in the order in which they are typed on the command line (left to right) and are read until either an error occurs or until the interactive state terminates (an `end` command not associated with either a `define` or `insert` command).

Therefore, this command line example illustrates a quick way to spool a plot off to the printer without having to enter the interactive mode of WIP. This might be useful, perhaps, when putting together a paper where there are figures that will be used are generated at different times in the analysis stage. The plot commands can be saved to a file (like `myfile.wip`) as they are created. And then later, when the paper is ready to go out, they can be easily spooled off to the printer. Users familiar with Unix shell scripts will discover that this option can be quite useful (and welcome).

Finally, if the `-e` optional flag is present, it signals that the next and any remaining arguments are a single command to be issued inside WIP. That command can be any internal WIP command or a defined macro. The command is executed after all of the files on the command line have been loaded. If WIP is to be run interactively, this command is executed just before WIP enters interactive mode.

The usefulness of this option is seen as a generalization of the example above. Suppose the file `myfile.wip` defined a plot but required a file name as an argument. Rather than editing `myfile.wip` for each file, WIP can be called with the `-e` option and each file can be passed on the command line. For example, if the file `myfile.wip` defines a macro called `dojob`, then multiple files could be input as follows:

```
% wip -x -d file1.ps/vps myfile.wip -e dojob file1
% wip -x -d file2.ps/vps myfile.wip -e dojob file2
```

Appendix C

Glossary

The following is a loose list of terms used throughout the WIP manual, and perhaps not always explained in detail.

ASCII – A numeric code standard for characters. Literally, ASCII stands for American Standard Code for Information Interchange. This is the most common character to integer translation code.

BASIC – An image type used by WIP to reflect many different image formats. The Basic image file is generally a binary file with an optional header section followed by the packed data. The image name used with the `image` command generally describes the format of the data and the size of the optional header section.

BIMA The Berkeley-Illinois-Maryland Association.

C-shell – In UNIX, this is one of the possible shells that can be used to communicate with UNIX. Other shells are `tcsh` (the T-shell), `sh` (the Bourne Shell), and `bash` (the GNU Shell).

FITS – This stands for the Flexible Image Transport System and is a standard data format used to interchange and archive data between different computers. It can be used for image as well as other types of data.

host – Your local computer. The term *host* is often used as in *host interpreter* and *host commands*, and is really meant to warn you that in this context commands may differ depending on which host/machine (often VMS versus UNIX) you work.

Miriad – The Multichannel Image Reconstruction, Image Analysis and Display data reduction package is developed by the BIMA group and is used here to refer to the format of the images generated with that package.

PGPLOT – A graphics subroutine package developed by Tim Pearson at Caltech used to generate publication quality graphics on various graphic display devices. The PGPLOT subroutines are the graphics routines used by WIP.

PostScript file – An ASCII text file in the PostScript language. PostScript is a page description language, and has become an industry standard for printing high quality text and graphics. There are PGPLOT device drivers which can create a PostScript file. This file can be sent to a printer to get hardcopy output.

READLINE – The Readline Library is a set of routines for providing `Emacs` style line input distributed by the Free Software Foundation. The library is not included within the WIP distribution. If, however, it is present on the system when WIP is built, then a compile time option can incorporate the library into the user interface of WIP providing `Emacs` style command line recall and editing.

shell – A supposedly easier to handle front-end command processor used to communicate with a lower level, and often more difficult to handle program or operating system.

WIPHELP – The name of the on-line help file. This environment variable defines the full directory path and file name of the file used by WIP to present on-line help.

X-WINDOWS – The name of a windowing environment developed by MIT and available (free) via anonymous ftp.

Appendix D

Table of Current Command Names

This Appendix presents, alphabetically, each of the current command names along with a description of the arguments to each command. Each entry in this file is equivalent to requesting help with the command `help cmd` where `cmd` is the command name. Command arguments inside square brackets are optional parameters; the description will describe what value will be substituted if these arguments are not supplied.

If there is a difference between the descriptions that follow and the corresponding on-line version, the on-line help message should be assumed to be correct.

`AITOFF [L B]` – converts L-b coordinate values to equivalent x-y positions.
: `AITOFF` will convert the X values (longitude) from the most recent `XCOLUMN` command and the Y values (latitude) from the most recent `YCOLUMN` command and convert them to the equivalent x-y positions for use in an Aitoff (more precisely, a Hammer) projection. The `AITOFF` command MUST be called every time new data is inserted either with the `XCOLUMN` or `YCOLUMN` commands. If the optional arguments `L` and `B` are present, they are converted to the equivalent x-y positions for use in an Aitoff projection. For the equivalent Aitoff "box", see the `GLOBE` command.

`ANGLE D or ANGLE X1 X2 Y1 Y2` – sets angle of text or points to D degrees.
: `ANGLE` will cause text from `LABEL` and `PUTLABEL` commands to be presented at `D` degrees counterclockwise from the horizontal (+x direction). Alternatively, if there are four arguments present, then they are taken to be the World (user) coordinates of two positions and the `ANGLE` is set to the slope of the line.

`ARC MAJX MAJY [ANG [STARTANG]]` – draws an arc with major axes `MAJX`, `MAJY`.
: `ARC` draws an arc at the current cursor position with major axes `MAJX` and `MAJY` (in World coordinate units). The curve is drawn as a polygon with the characteristics specified by the command `FILL`. The number of points used to generate the curve can be controlled by adjusting the value of the character expansion (see the command `EXPAND`). The optional parameter `ANG` defines the angular extent of the arc in degrees (set to 360 degrees if not given). An additional optional parameter, `STARTANG`, may be used to determine an initial offset angle of the arc relative to `MAJX` (this defaults to 0 degrees if this parameter is not present). Use the command `ANGLE` to set the rotation of `MAJX` with respect to the positive x-axis. All angles are in units of degrees and are defined to be zero toward the right of the viewport (regardless of the direction of the

World Coordinate system) and increasing counter-clockwise. See also the `BEAM` command. NOTE: If the World Coordinate system is currently RA and DEC, a rotated arc may not plot correctly. Use `'header px px'` or `'header so so'` (or some other equivalent offset option) to set the coordinate system to something comparable in scale.

`ARROW X Y [ANGLE [VENT]]` – Draws an arrow.
`ARROW` draws an arrow from the current cursor position to the World (user) coordinate (`X,Y`). The acute angle of the arrow point, in degrees, is specified by the optional input `ANGLE`; angles in the range 20.0 to 90.0 give reasonable results. If `ANGLE` is not present on the command line, a value of 45.0 degrees is used. The fraction of the triangular arrow-head that is cut away from the back is specified by the optional parameter `VENT`. A value of 0.0 gives a triangular wedge arrow-head; 1.0 gives an open `>`. Values of 0.3 to 0.7 give reasonable results. The default value for VENT (if it is not present) is 0.3. `EXPAND` can be used to vary the length of the arrowhead. For `EXPAND` set to 1.0, the default size of the arrowhead is 1/40th of the smaller of the width or height of the view surface.

`ASK K` – Changes the current device prompt state.
The command `ASK` will set the prompt state to ON if `K` is set to 1 and the current device selected is interactive; otherwise, `ASK` will set the prompt state to OFF. If `ASK` is set ON, then every command that erases the page will prompt the user before clearing the screen. The default state of this is OFF; also, every time a new device is selected (see `DEVICE`), `ASK` is set to OFF.

`AUTOLEVS NLEV [TYPE [MIN [MAX]]]` – sets up the contour levels automatically.
`AUTOLEVS` will fill an internal contour level array with `NLEV` contours between the values `MIN` and `MAX`. `NLEV` must be input, but the rest of the parameters are optional. If either of or both `MIN` and `MAX` are not given, they are default to the minimum and maximum of the current array loaded using the command `IMAGE`. The optional parameter `TYPE` is a string of characters specifying the type of contour levels to generate. If `TYPE` is `LIN`, then the contour levels are linearly spaced (this is the default if `TYPE` is not present); if `TYPE` is `LOG`, then the levels are spaced evenly in the logarithm.

`BAR K [THRESH [GAP]]` – draws bar graphs on (x,y) pairs in direction `90(K-1)`.
`BAR` is analogous to the command `BIN` except that it will draw a bar graph on all (x,y) coordinates read by `XCOLUMN` and `YCOLUMN` in the direction specified by the argument `K`. Set the argument `K` to 1 to put the bar extending towards the +X direction; 2 for +Y; 3 for -X; and 4 for -Y. The values read by the most recent call to `ECOLUMN` determine which color to draw individual bars. If this array is empty, the current color index is used. The optional argument `THRESH` may be used to specify the "bottom" of the bar. If this optional argument is present, then it is used to anchor the beginning of the bar graph and should be entered in the units of that axis. If this optional argument is not present, the box edge opposite the direction the bar extends acts as the threshold. An additional optional argument `GAP` may be used to specify the "width" of the bar. If this is not present, the width is defined as the difference between the mean of the current point and the next point and the mean of the current point and the previous point. If `GAP` is present, it should be in entered in units of that axis (i.e. for `K == 2`, `GAP` should be in the units of the X-axis) and the bar will be centered on the point with half the width on either side.

`BEAM MAJ MIN PA [OFFX OFFY [SCALE [FILLCOLOR [BGRECT]]]]` – draws a beam.
`BEAM` draws a beam at the current cursor position outlined using the current color and line width properties. It has a major axis `MAJ` and minor axis `MIN` and is rotated by a position angle `PA` (which is in units of degrees defined as zero along the positive Y-axis (regardless of the current World Coordinate system) and increasing counter-clockwise). The optional parameters `OFFX` and `OFFY` (which both default to 0) define the offset of the center of the beam from the current cursor position. These offsets are in units of the width or height of the box that bounds the beam. The beam will first be drawn solid using the optional input `FILLCOLOR` and is then drawn hollow with the current color. If `FILLCOLOR` is not provided, it defaults to color index 15. By

default, the axes will be scaled as if `header rd rd` has been previously called. This means that the X-axis will be scaled by 15*cos(declination). To override this scaling, the optional input `SCALE` can be provided and set to whatever scaling is desired (no scaling can be achieved by setting `SCALE` to 1). If `SCALE` is set to any negative value (or is not present), then RA-Dec type scaling will be done. The major and minor axes of the beam should be in the same units as the two axes. This means if `SCALE` is set to anything but 1, then they should both be in the same units as the Y-axis. If the optional argument `BGRECT` is provided (it defaults to 0) and is not negative, then a solid rectangle will be drawn in that color (color index `BGRECT`) first. The rectangle will only be as large as the width and height of the beam.

`BGCI N` – sets the text background color index to `N`.

`BGCI` sets the text background color index for subsequent text. By default, text does not obscure underlying graphics. If the text background color index is positive, however, text is opaque: the bounding box of the text is filled with the color specified by the value of `N` before drawing the text characters. Use color index 0 to erase underlying graphics before drawing text. If `N` is less than 0, the text will be transparent; if greater than or equal to 0, the text will be drawn on an opaque background with color index `N`. The initial value of the text background color index is -1 (transparent) and is reset to -1 whenever a new device is selected.

`BIN [K [GAP] ]` – draws a histogram of (x,y) pairs.

`BIN` connects the coordinates read by `XCOLUMN` and `YCOLUMN` as a histogram. If the optional argument `K` is present and equal to 0, then the individual X values denote the lower left edge (in X) of the bin. If `K` is set to 1 or is not present, then the individual X values denote the center location of the bin. If the optional parameter `GAP` is present, it represents the distance (in X axis units) that signals a gap in the histogram not to be connected. This is useful for histograms with irregular spacing or directional changes. A good value for `GAP` is 2.

`BOX [x [y] ]` – makes a box labeled according to `LIMITS` and `TICKSIZE`.

`BOX` annotates the viewport with a frame, axes, numeric labels, etc. The two arguments `X` and `Y` are strings of options that determine the type of box and the labeling that will be drawn. The option letters may be in any order and do not depend on case. If either parameter is omitted, the missing parameter is given the default value set by the string variables XBOX and YBOX (see the command `SET`). If none of the letters 'A', 'B', or 'C' are specified, then tick marks will not be drawn. The options currently available are found in Table 3.4.

`BUFFER` – predefined macro name that refers to the entire command buffer.

Typing `BUFFER` will execute the contents of the command buffer (the same as the command "`playback`" or "`playback buffer`"). `BUFFER` may also be used as an argument for commands requiring a macro name (e.g. `LIST` and `WRITE`).

`COLOR N` – select color for lines and characters.

`COLOR` defines the current color index used for filling polygons, drawing lines, and labeling characters for the selected device. `COLOR` is originally set to 1 and is reset to this value whenever a new device is selected. The number of colors available is device dependent (the range can be displayed with the command '`echo cmin cmax`'). If the requested color index is not available on the selected device, a color index 1 will be substituted. The first 16 colors usually have predefined values. Additional values are device dependent. Each color associated with the color index `N` is listed in Table 3.3.

`CONNECT` – connects (x,y) pairs with line segments.

`CONNECT` draws line segments connecting the coordinates read by `XCOLUMN` and `YCOLUMN`. The lines are drawn using the current line style (set by the command `LSTYLE`), in the current color (set by the command `COLOR`), and the current thickness (set by the command `LWIDTH`).

`CONTOUR [N [BLANK [MIN]]]` – makes a contour plot of an array read with `IMAGE`.

`CONTOUR` makes a contour plot of an array read with the `IMAGE` command at the contour levels set using either the `LEVELS` or `AUTOLEVS` commands. The contour levels MUST be set before a call to `CONTOUR`. The contour level array will be modified according to the current value of the user variable SLEVEL and the string variable LEVTYPE (also see the command `SLEVEL`) before calling any of the contour methods below.

Currently, there are four contour routines available. The optional parameter `N` allows the user to select which of four contour drawing methods will be used to generate the plot. If `N` is the character 's', then the command will produce a fast contour plot (this is the default if `N` is not present); if `N` is 't', then a smoother contour is produced; if `N` is 'b', then the same contour plot as with 's' is produced but with blanking; and 'l' produces the same contour plot as with 't', but with contour level labeling.

For `N` set to 'b', an optional argument `BLANK` may be given to specify the blanking value such that array elements equal in value to `BLANK` are ignored (blanked). For `N` set to 'l', the optional argument `BLANK` defines the spacing along the contour between labels (in grid cells) and the optional parameter `MIN` specifies the minimum number of cells that must be crossed before drawing the first contour label. The default value for `BLANK` is 0.0 when `N` = 'b' and 16 for `N` = 'l'; `MIN` defaults to 8. The optional parameters are ignored by the contour options `N` = 's' and 't'.

By default, each contour line is drawn with the current line attributes (color index, line style, and line width). However, for options 't' and 'l', the default is to draw negative contours values with dashed and positive contours values with solid lines. If, however, the parameter `N` is set to '-t' (option 't' prepended with a minus sign), then negative contours values will be drawn with the same line style as positive contours values.

`CURSOR [x y]` – enables cursor and returns x-y location and the key pressed.

`CURSOR` enables the graphics cursor and returns the x-y position of the cursor along with the key pressed. `CURSOR` may also display the image value of the selected position if an array is currently loaded (see the `IMAGE` command) and the cursor position lies within the scope of the image region (see the command `TRANSFER`). The two optional arguments may be used to determine the starting position of the cursor. If both parameters are given, they are used as the World (user) coordinates for the starting position. If only one parameter is present, it is used as the X starting position and the Y position is set as the current cursor position. Finally, if no arguments are present, the current cursor position is used as a starting position.

If `CURSOR` is called inside a macro (see the command `DEFINE`), with the command `PLAYBACK`, or is executed inside a file via the command `INPUT`, then only one keystroke is allowed and no saving of commands will be done. If, on the other hand, `CURSOR` is called interactively, then `CURSOR` may perform several functions, depending on the character typed. Internal calls to `CURSOR` will occur repetitively until the letter 'X' is typed. On some devices, the left most mouse button will be associated with the keystroke 'A'; the middle with 'D'; and the right with 'X'.

Other keystroke functions are:

```
A : Display the cursor position and image intensity.
D : Draw to the current cursor position.
M : Move to the present cursor position.
P : Draw a point of the current symbol type.
S : Toggles ability to save following keys as commands.
X : Exit the cursor routine.
? : Present this list of key commands.
```

`DATA fspec` – opens the file "fspec" for reading data.

`DATA` opens an external file for reading data. The file is assumed to have data in columns separated by spaces, tabs or commas. Any column can be read as x or y coordinates with the `XCOLUMN` and `YCOLUMN` commands.

`DEFINE xxx` – creates the macro xxx and enters define mode.
: `DEFINE` creates an entry for the macro `xxx`, and subsequent commands (until the `END` command) are considered the body of the macro. The prompt changes to `DEFINE>` to indicate the mode. A macro is just a collection of commands (or other macros) that can be executed with one call statement. A macro is invoked by using its name as a command. The contents of the macro may be listed by using the macro name as an argument to the command `LIST`. Macros may be saved using the command `WRITE` and may be played back using the command `PLAYBACK`. Macros may be passed arguments or variables whenever they are executed. The ten tokens used inside the macro to identify the macro arguments are \$1, \$2, ..., \$9, and \$0. These tokens may be used in the body of the macro, and when the macro is invoked its arguments will be substituted for these tokens. These tokens MAY NOT be redefined inside the macro (for example, using the command `SET`). If an argument is not present when the macro is invoked, the token is replaced with a blank space.

`DELETE [N1 [N2 [MACRO]]]` – removes the commands `N1 - N2` from a macro buffer.
: `DELETE` removes commands from the macro listing (numbered according to what is shown with the command `LIST`). If no arguments are present, then the last command in the command buffer (see `BUFFER`) is deleted. If `N2` is not present, then only the command numbered `N1` in the command buffer (`BUFFER`) is removed. If both `N1` and `N2` are present, then all commands from `N1` to `N2` (inclusive) are deleted from the command buffer (`BUFFER`). If the optional macro name is given, then lines `N1` to `N2` (inclusive) are removed from `MACRO`. For example, to remove line 4 from a macro named "`dojob`", use the command `delete 4 4 dojob`. Note that the command buffer macro name is "`BUFFER`".

`DEVICE device` – initializes output to a graphics device.
: `DEVICE` defines the "device specification" for the plot device. The specification is system dependent, but usually has the form "device/type" or "file/type". If the argument is a question mark ("?"), the user will be prompted to supply a device name after a list of the currently available devices is presented. When a new device is selected, the `PANEL` command is reset and the `COLOR`, `EXPAND`, `FILL`, `FONT`, `LSTYLE`, and `LWIDTH` commands are called to reset each to a value of 1. The color palette (see the command `PALETTE`) is also reset to the default palette and the coordinate transfer function (`TRANSFER`) is reset to linear. Finally, the inquire state is turned off (see the command `ASK`).

`DOT [X Y]` – makes a point of the current style at the current location.
: `DOT` draws a point at the current location (set by `MOVE`, `DRAW`, etc.) (or at the position (X,Y) if provided) in the style determined by the latest call to `SYMBOL` or `PCOLUMN`. The symbol's size is governed by the last call to the command `EXPAND`.

`DRAW X Y` – draws a line to `(X,Y)` from the current coordinate position.
: `DRAW` draws a line to the World (user) coordinate (X,Y) from the current location (set by a previous call to `MOVE`, `DRAW`, etc.). `DRAW` then makes (X,Y) the current position.

`ECHO EXPRESSION [...]` – displays the result of `EXPRESSION` on the screen.
: `ECHO` acts just like the command `SET` except that rather than assigning the result of the `EXPRESSION` to anything it displays the result on the screen. See the command `SET` for an explanation of `EXPRESSION`. There may be more than one `EXPRESSION` present, but each one should be delimited by enclosing parenthesis. Literal strings may also be displayed by enclosing the text in double quotes. String variables may be displayed by listing their names. For example, to display the current cursor position, use the command: `echo "cx = " cx "cy = " cy`.

`ECOLUMN N` – reads error bar data from column `N` of the data file.
: `ECOLUMN` reads column N from the data file as the magnitudes of the error bar displacements from the corresponding (x,y) coordinates. Note that the command `LOGARITHM` does not adjust the error values to logarithmic scale. `ECOLUMN` values are also used by the command `LOOKUP`.

`END` – terminates define mode, insert mode, or exits from the program.

In DEFINE mode, `END` finishes the macro definition and returns to EXECUTE mode. In INSERT mode, `END` terminates inserting and returns to EXECUTE mode. In EXECUTE mode, `END` exits from the program altogether. The commands `EXIT` and `QUIT` may be used as a substitute for the command `END`.

`ENVIRONMENT [X1 X2 Y1 Y2 [N [K]]]` – sets the user limits and draws a box.

`ENVIRONMENT` can clear the current page, set up the World (user) limits and draw a standard box. The first four optional arguments establish the World (user) limits of the plot region. If they are not present, then the data read from the most recent call to the commands `XCOLUMN` and `YCOLUMN` is used to autoscale the limits. If the limits are present, then two other optional parameters may be selected. The first of these parameters, `N`, will set the scales of the X and Y axes to be equal, if it is equal to 1; otherwise the two axes will be scaled independently. The final optional parameter `K` will determine what type of box to draw and may have the following values:

```
-2 : draw no box, axes or labels;
-1 : draw box only;
 0 : draw box and label it with coordinates;
 1 : same as K=0, but also draw the coordinate axes (X=0, Y=0);
 2 : same as K=1, but also draw grid lines at the major
   : increments of the coordinates;
10 : draw box and label X-axis logarithmically;
20 : draw box and label Y-axis logarithmically;
30 : draw box and label both axes logarithmically.
```

Both `N` and `K` default to 0 if not provided.

`ERASE` – erases the graphics screen.

`ERASE` erases the graphics screen. If the current device is a hardcopy device, `ERASE` will cause the plot to advance to the next page. If `ASK` is set ON and the current device is not a hardcopy device, then the user will be prompted before advancing to the next page.

`ERRORBAR K` – draws error bars on (x,y) pairs in the direction `90(K-1)`.

`ERRORBAR` is analogous to `POINTS`; it draws error bars at each of the (x,y) points (read by the most recent call to the commands `XCOLUMN` and `YCOLUMN`) at a length specified by the points read by the command `ECOLUMN`. The argument `K` should be 1 to put the bar along the +x direction; 2 for +y; 3 for -x; and 4 for -y. If `K` is 5, then error bars are drawn in both the +x and -x directions. Likewise, if `K` is 6, then error bars are drawn in both the +y and -y directions. Use `EXPAND` to govern the size (extent) of the caps. Setting the expansion to zero means no caps are drawn.

`ETXT` – erases the text from the view surface without affecting graphics.

`ETXT` erases the text from the view surface (if the device is capable of doing this). This should erase text without affecting the currently displayed graphics. Nothing is done on devices without this capability.

`EXPAND E` – expands all characters and points by a factor `E`.

`EXPAND` sets the expansion for all characters and points. It is initially set to 1.0 and is reset to 1.0 whenever a new device is chosen.

`FILL N [ANGLE [SEPN [PHASE]]]` – Sets the fill area style to `N`.

`FILL` sets the fill type. The fill type is identified by `N`, where `N` refers to one of the following:

```
1 : solid
2 : hollow
3 : hatched
4 : cross-hatched
```

The fill type is initially set to 1 (solid) and is reset to 1 whenever a new device is chosen. If the fill type is hatched or cross-hatched, optional arguments can be supplied that affect the way the hatching will be done. Up to three additional arguments may be provided. `ANGLE` specifies the angle the hatch lines make with the horizontal, in degrees, increasing counterclockwise. `SEPN` is the spacing of the hatch lines. The unit spacing is 1 percent of the smaller of the height or width of the view surface. `PHASE` is an offset value and is a real number between 0 and 1. The hatch lines are displaced by this fraction of `SEPN` from a fixed reference. Adjacent regions hatched with the same `PHASE` have contiguous hatch lines. To hatch a region with alternating lines of two colors, fill the area twice: once with `PHASE=0.0` in one color and `PHASE=0.5` in the other color. The default value for `ANGLE` is 45.0 degrees, `SEPN` is 1.0, and `PHASE` is 0.0.

`FIT style [N] [P1 P2 P3 ...]` – Fits a curve to the (x,y) data pairs.

`FIT` will solve either for a linear fit of the form: "`y = a + bx`" by either the (a) Least Squares method ("lsqfit"); or (b) Criterion of Least Absolute Deviations ("medfit"); or will solve for a Gaussian fit of the form "`y = a * exp(-((x - b)/c)**2)`". The data that comprise the "x" and the "y" in these equations are from the most recent call to the `XCOLUMN` and `YCOLUMN` commands. For a linear fit, there is only one required argument, `STYLE`, which specifies which type of linear fit to perform. The only possible values of `STYLE` are "lsqfit" and "medfit". The fit coefficients and the corresponding errors in the fit are displayed at the command line. When using the "lsqfit" technique, estimates of the measurement errors may be included by reading the error values with the `ECOLUMN` command. The optional parameters provide a way to assign the fits and the error estimates to User Variables (see the `SET` command). The "lsqfit" returns (in this order) the parameters "a" and "b" along with a chi-square for the fit and sigma errors on the estimates of "a" and "b". If data in the most recent call to `ECOLUMN` was used to specify errors on the observation points, a "goodness of fit" is also returned. If, however, no errors are specified, the correlation coefficient is returned in place of the "goodness of fit". The "medfit" returns (in this order) the fit parameters "a" and "b" along with "absdev" (the mean absolute deviation in the "y" direction). As an example, the command `fit lsqfit \0 \1` will only assign the fit parameters "a" and "b" to the User Variables \0 and \1 and the remaining items will be ignored (but still displayed at the command line).

For Gaussian fits, there are two required arguments: `STYLE` and `N`, where `STYLE` must be "gaussian" and `N` specifies the number of Gaussians to fit. The number of optional parameters that may follow is governed by the value of `N`. Optional arguments should follow in groups of 3's in the following order: amplitude, mean, and full width at half maximum. Assignment of the output values follows as for linear fits. If the `3*N+1` parameter is present, it is set to the chi-square of the fit. Setting any parameter to it's negative holds that value fixed during the fit. For example:

```
set \1 1.0   \# Guess at amplitude.
set \2 5.0   \# First guess at X-position of peak.
set \3 -2.3  \# Input FHWM and hold this fixed during fit.
fit gaussian 1 \1 \2 \3 \4  \# Fit 1 Gaussian and store fit
      \# and chi-square in \1, \2, \3, and \4, respectively.
```

`FONT N` – Sets the font type to type `N`.

`FONT` sets the font type to one of the available fonts. The argument `N` selects the font used for subsequent text plotting. The font type is initially set to 1 ("normal") and is reset to 1 whenever a new device is chosen. Currently, four fonts are available and can be set by setting the parameter `N` to one of the following values:

```
1 : (default) a simple single-stoke font (``normal'' font)
2 : roman font
3 : italic font
4 : script font
```

The font may also be temporarily changed within a text string by using the escape sequences: \fn, \fr, \fi, or \fs.

`FREE item1 [item2 [...]]` – releases items created with the `NEW` command.
`FREE` will remove the user specified variable(s) from WIP. In addition, it releases (i.e. frees) the memory associated with each `ITEM`. At least one `ITEM` is required and each `ITEM` listed specifies the name of the string variable, user variable, or vector (without any index) to release.

`GLOBE [nlong nlat]` – draws a "globe" with nlong/nlat long/lat lines.
`GLOBE` draws a "globe" for use in Aitoff (Hammer) projections. The "globe" that is drawn is an outline of the entire sky centered on the position (L, b) = (0, 0). There are two optional parameters. The first optional parameter `NLONG` represents the number of longitude lines that are drawn and the optional parameter `NLAT` represents the number of latitude lines that are drawn. If these arguments are omitted, the values of `NLONG = 5` and `NLAT = 3` will be used for defaults. Also see the command `AITOFF`.

`HALFTONE [MIN MAX [N [BLANK]]]` – produces a halftone plot of an image.
`HALFTONE` draws a plot of the array read with the `IMAGE` command over the subimage region selected with the `SUBIMAGE` command according to the coordinate transformation matrix specified by the `TRANSFER` command and the current image transfer function (see the `ITF` command). If the device permits, colors can be preassigned to the halftone values with any of the `PALETTE`, `LOOKUP`, `RGB`, or `HLS` commands (whenever a new device is selected, the palette is reset). The first two optional parameters allow the user to specify the intensity range for the plot. If the optional parameters `MIN` and `MAX` are not present, then `HALFTONE` will use the image minimum and maximum. The next optional parameter, `N`, if present, indicates that histogram equalization should be performed using `N` bins. If this parameter is present and is set to zero, then the full number of color table entries will be used in the binning. If this is not present, no equalization will be done. The last optional parameter `BLANK` is used to specify a blank value in the image to ignore when doing histogram equalization. It defaults to -99. Histogram equalization is only applied to data within the current subimage region and within the minimum and maximum intensity.

NOTE: Histogram equalization changes the current LUT. The LUT must be reset before calling the `HALFTONE` command again.

`HARDCOPY` – causes a stored printer plot to be plotted.
If the current device selected is a hardcopy unit, then the `HARDCOPY` command will cause the output file to be closed and the file spooled to the printer. How the file is printed is determined by the string variable "print" (see the command `SET`). By setting "print" to the string "ignore", automatic printing can be disabled. By default, the string variable "print" is set to "lpr" (use the command '`echo print`' to display the current value of "print").

`HEADER [XTYPE [YTYPE]]` – loads header information of the current image.
`HEADER` sets the transfer function and World (User) limits of the box region for the current image. This function makes use of the current `SUBIMAGE` values to fix the World coordinate range. If the optional arguments `XTYPE` and `YTYPE` are not present, they are set to the current values of the string variables "XHEADER" and "YHEADER", respectively. If only one argument is present, both `XTYPE` and `YTYPE` are set to this value. Currently, there are 7 possible values for `XTYPE` and `YTYPE` used to control the resulting header arrangement:

```
rd : Right ascension/declination (absolute coordinates).
so : Arcsecond offset positions.
mo : Arcminute offset positions.
po : Pixel offset positions.
px : Absolute pixel positions.
```

```
gl : General linear coordinates.
go : General linear offset coordinates.
```

All offset arguments are relative to the value of the user variables CRVALX and CRVALY. The option "rd" includes a 15*cos(dec) factor applied to the right ascension term. Incorrect options are treated as if "px" was selected. The options "gl" and "go" apply no additional scaling factor to the header variables. If the image headers CRVAL, CRPIX, and CDELT are not present for that axis or are all set to 0, the coordinate transfer function will be set to its default value (see the command `TRANSFER`) and the world limits to the image range in pixels. NOTE: Currently, the conversion is fixed to be from radians to arcseconds (or seconds of time) for MIRIAD images; degrees to arcseconds for FITS images; and no conversion for BASIC images.

`HELP xxx` – prints an explanation of the command `xxx`.
Without any argument, `HELP` prints a one line description of every command. With the name of a command as an argument, `HELP` prints more information on that particular command. The command '`help ?`' will provide a list of all available command and macro names.

`HI2D bias [slant [center]]` – draws a histogram of the data read by `IMAGE`.
`HI2D` plots a series of cross-sections of the current image (see `IMAGE`). Each cross-section is plotted as a hidden line histogram. The one required argument, `BIAS`, specifies a bias value to be applied to each successive cross-section (in order to raise it above the previous one). `BIAS` is in the same units as the image data. If the optional parameter `SLANT` is present, it is an integer value offset to be applied to each successive cross-section (in the X-direction). If `SLANT` is greater than 0, then the plot slants to the right; less than 0, it slants to the left. The default value for `SLANT` is 0 (no slant). If the optional argument `CENTER` is present and equal to 1 (the default), then the individual X values denote the center location of the bin; otherwise, the X values denote the lower edge (in X) of the bin.

`HISTOGRAM [MIN MAX [N]]` – draws a histogram of the data read by `XCOLUMN`.
`HISTOGRAM` connects the coordinates read by `XCOLUMN` as a histogram. If the optional arguments `MIN` and `MAX` are present, they are used to specify the minimum and maximum of the histogram. Furthermore, an optional parameter `N` can specify the number of bins to divide up the histogram. At present, `N` may not exceed 200. By default, `MIN` and `MAX` are set to the minimum and maximum of the data read by `XCOLUMN` and `N` is set to 5. The user should set the limits (see the command `LIMITS`) of the box before calling the `HISTOGRAM` command.

`HLS K hue light sat` – sets the color representation using the HLS system.
`HLS` sets the color representation of the color index `K` to the provided values of Hue-Lightness-Saturation. `HUE` is represented by an angle in degrees, with red at 120, green at 240, and blue at 0 (or 360). `LIGHT` ranges from 0 to 1, with black at lightness of 0 and white at lightness 1. `SAT` ranges from 0 (gray) to 1 (pure color). `HUE` is irrelevant when `SAT` is 0. If `K` is greater than the maximum color index of the device, then this call will be ignored (use the command '`echo cmin cmax`' to display the current color range).

`ID` – puts an identification label at the bottom of a plot.
`ID` puts the name of the user and the current date and time at the bottom of the plot region. If the environment variable PGPLOT_IDENT is defined and the current device is a hardcopy device (the command '`echo hardcopy`' will return 1 if this is true), then this call is automatically done when the device is finally closed (see the command `HARDCOPY`).

`IF EXPRESSION XXX [...]` – executes `xxx` if `EXPRESSION` is true.
`IF` allows conditional execution of macros and commands. The macro or command `XXX` is executed only if the conditional test of the expression is TRUE (non-zero). The `EXPRESSION` may be a simple `(X OP Y)` expression where the parameters `X` and `Y` may be any numerical value or user variable (see the `SET` command) and `OP` is the condition to test and is defined TRUE for the following:

```
    op : Test
------------------------------------------------------
    == : X equal to Y;
    != : X not equal to Y;
    <  : X less than Y;
    >  : X greater than Y;
    <= : X less than or equal to Y;
    >= : X greater than or equal to Y;
    && : X and Y are non-zero;
    || : X or Y are non-zero;
    ^  : X or Y, but not both, are non-zero.
```

If the test consists of more than just a simple `(X op Y)` test, then the entire expression MUST be enclosed by parenthesis. As an example,

```
if ((x1 < x2) && (y1 < y2)) limits x2 x1 y1 y2
```

will only execute the `LIMITS` command if X1 is less than X2 AND Y1 is less than Y2. Note also that this example illustrates that the usual parameters/arguments may follow the macro or command name. See the command `SET` for a more detailed description of `EXPRESSION`.

`IMAGE fspec [plane [badpixel [order]]]` – reads in an image from file "fspec".

`IMAGE` loads an array into WIP for use with plotting commands like `CONTOUR` and `HALFTONE`. Currently, WIP tests the file to determine its file type (i.e. FITS, MIRIAD, BASIC, etc.) and then loads it into available memory. The optional parameter `PLANE` selects which plane of a data cube to load. The default value for `PLANE` is 1. `IMAGE` trys to read available header information which will be useful for setting the coordinate transformation matrix (see the command `TRANSFER`) after the image is read into memory (with the command `HEADER`). The current subimage values (see the command `SUBIMAGE`) are reset to the full array size of the new image and the user variables CRVALX/Y, CRPIXX/Y, and CDELTX/Y may be set to the newly read header values (if they can be read). The subimage values, 6 user variables, and the transformation matrix are used to determine where the plot will appear. In order to correctly set the transformation matrix with the command `HEADER`, the values of CRVAL, CRPIX, CDELT, and CTYPE must be present for the first two axes. The next optional parameter `BADPIXEL` may be specified (after `PLANE`) as the value for blanked or masked pixels. By default, these will be set to -99. This value may be helpful for use with the `CONTOUR` command. The final optional parameter `ORDER` permits the image to be smoothed on reading. The parameter `ORDER` may take on a value of 0, 1, or 2 and represents the order of the fit. A value of 0 (or if it is not provided) means to apply no smoothing (the default).

NOTE: Current image types include: basic, Miriad, and FITS. BASIC image name syntax is: filename'columnsXrows['offset] where rows and columns are in pixels and the optional offset is in bytes from the beginning of the file. The character 'X' determines the size of each element and is 'b' for unsigned bytes; 's', signed 2 byte integers; 'l', signed 4 byte integers; 'r', 4 byte floating point; 'd', 8 byte double precision floating point.

`INITIALIZE V N EXPRESSION` – Sets `V` to the result of `EXPRESSION`.

`INITIALIZE` evaluates the `EXPRESSION` (see the command `SET`) and sets every element of the array (vector) user variable `V` to the result. If `N` is less than zero, the current size of the vector variable `V` is used. If `N` is zero, no expression is needed and the vector is set to size zero (empty). If `N` is greater than zero and an expression is present, the first `N` elements of the vector are set to the result of the expression. If the expression is missing, the size of the vector will be reset.

`INPUT fspec` – reads plot commands from file "fspec" and executes them.

`INPUT` reads plot commands from a file and treats them exactly as if they were typed: commands are executed or can be inserted or macros may be defined. Unlike interactive mode, individual commands are NOT saved in the command buffer (see `BUFFER`). NOTE: Macros are not redefined so care should be taken to insure that the proper macro is being executed.

`INSERT [N [macro]]` – commands are inserted before command `N` in a macro.
`INSERT` enters insert mode (indicated by the `INSERT>` prompt) and subsequent commands are inserted, without execution, into the named macro until an `END` command is encountered. Commands are numbered as shown by the command `LIST`. Without any arguments, `INSERT` inserts commands at the end of the command buffer. If `N` is present but no macro name is given, then commands are inserted before line `N` in the command macro (`BUFFER`). With the `MACRO` argument, `INSERT` starts inserting just before line number `N` in macro `MACRO`. NOTE: The command buffer macro name is `BUFFER`.

`ITF N` – sets the current image transfer function to `N`.
`ITF` sets an internal parameter such that all subsequent image drawing functions (`IMAGE` and `WEDGE`) act on the transformed image. Acceptable values for `N` and their meanings follow:

```
0 : linear
1 : logarithmic
2 : square-root
```

The initial value of the image transfer function is 0 and is reset to 0 whenever a new device is selected.

`LABEL STR` – writes the string `STR` at the current cursor position.
`LABEL` writes the string `STR` (which starts at the current cursor position and continues to the last non-space character in the label string). The string's size and the angle it is displayed are controlled by the most recent call to the commands `EXPAND` and `ANGLE`.

`LCUR [NPTS]` – draws a line using the cursor.
`LCUR` allows a polyline to be drawn using the cursor. Vertices may be added or removed by locating the cursor at a desired point and selecting the key 'A' (Add) or 'D' (Delete) or may terminate the command with the key 'X' (eXit). The vertices are stored in the `XCOLUMN` and `YCOLUMN` arrays and may be redrawn by subsequent calls to this command. If the optional parameter `NPTS` is given, it is used as the number of points in the previous call to this command. To initialize the command, set `NPTS` to 0. By default, `NPTS` is set to the current value of the number of points in the most recent call to the `XCOLUMN` and `YCOLUMN` commands.

`LDEV` – lists the devices currently available.
`LDEV` lists (to standard output) the names of the installed devices known to the current version of WIP.

`LEVELS [L1 L2 ...]` – sets the contour levels for a contour plot.
`LEVELS` establishes the contour values for use in a contour plot. Currently, the maximum number of levels available is 40. This array of levels may be augmented if a call to the command `SLEVEL` is done prior to a call to the `CONTOUR` command.

`LIMITS [X1 X2 Y1 Y2]` – sets the World limits of the plot.
`LIMITS` sets the World (user) coordinate system of the plot window. All coordinates from `XCOLUMN`, `YCOLUMN`, `MOVE`, `DRAW`, etc., are relative to these limits. If no arguments are present, then the data from the most recent `XCOLUMN` and `YCOLUMN` commands are used to set the limits.

`LINES L1 L2` – limits the `X, Y, E,` and `PCOLUMN` file read to lines `L1-L2`.
`LINES` sets the range of lines read from the data file by the commands `XCOLUMN`, `YCOLUMN`, `ECOLUMN`, and `PCOLUMN`. This is useful to avoid non-data lines. The argument `L1` is also used to specify the line that the command `STRING` will use to extract a string variable.

`LIST [L1 [L2 [xxx]]]` – lists the commands of macro `xxx`.

`LIST` lists the commands of the named macro. If there are no arguments, the macro `BUFFER` (the command buffer) is listed. If one argument is given, then the commands from `L1` to the end of the macro `BUFFER` is shown. With two arguments, lines `L1` to `L2` are shown. If the macro name `xxx` is also entered as the third argument, then that macro is listed over the inclusive line range of `L1` to `L2`. The numbers assigned to commands by `LIST` should be the ranges used by calls to the commands `DELETE` and `INSERT`. NOTE: The command buffer macro name is `BUFFER`.

`LOGARITHM name [scale]` – takes the scaled logarithm of vectors and images.

`LOGARITHM` takes the base 10 logarithm of either vectors or images. The argument `NAME` specifies either a vector name (e.g. "x") or an image name (currently, only the name "image" is acceptable). If the value of the vector or image is greater than zero, the logarithm is taken and this result is multiplied by the value of `SCALE`. If `SCALE` is not present, a value of 1.0 is used. `SCALE` may be any valid numeric expression (see the command `SET`). `LOGARITHM` uses -50 for the logarithm of 0 or negative numbers and does not scale these values.

`LOOKUP [N]` – loads a RGB color lookup table.

`LOOKUP` uses the values read by the most recent call to the commands `XCOLUMN`, `YCOLUMN`, `ECOLUMN`, and `PCOLUMN` to load values into the color lookup table. This is a more compact (and correct) method for loading an entire table than the single color index commands `RGB` and `HLS`. All values in the `XCOLUMN`, `YCOLUMN`, `ECOLUMN`, and `PCOLUMN` data must be between 0 and 1. The `XCOLUMN` data correspond to the red values, `YCOLUMN` to green, and `ECOLUMN` to blue. The `PCOLUMN` data identifies the ramp intensity level for the corresponding RGB values. Colors on the ramp are linearly interpolated from neighboring levels. If the optional parameter `N` is present and is negative, then the resulting lookup table will be flipped. By default (if the optional argument `N` is missing or is positive), the RGB table is loaded directly (not flipped).

`LOOP count xxx [p1 p2 ...]` – executes the macro `xxx COUNT` times.

`LOOP` allows for multiple execution of a macro. The macro `xxx` is executed `COUNT` times. The optional parameters `P1, P2, ...` are any arguments needed to be passed to the macro (see the `DEFINE` command). The arguments `P1, P2, ...` passed to the macro are NOT altered during each execution of the loop.

`LSTYLE N` – sets the current line style to `N`.

All subsequent lines except for points and characters are drawn with line style `N`, where `N` refers to lines of the following form:

```
1 : solid
2 : dashed
3 : dot - dash - dot - dash
4 : dotted
5 : dash - dot - dot - dot
```

If `N` is some number other than one of the line style values above, it is set to 1. Also, the line style is set to 1 initially and is reset to 1 whenever a new device is selected.

`LWIDTH N` – sets the current line width attribute to `N`.

`LWIDTH` sets an internal parameter such that all subsequent lines, text, and graph markers are drawn with lines of width `N`. This number may range from the default of 1 to a maximum of 201. Any other value is set to 1. The actual thickness drawn is device dependent, but is approximately equal to 0.005 inches times the value of `N`. The initial value of the line width is 1 and is reset to 1 whenever a new device is selected.

`MACRO filename` – Used to define macros using an external file.

`MACRO` will read the commands from the file `filename` and define macros exactly as if they were typed interactively. This allows macros saved from previous WIP sessions (using the `WRITE` command) to be read into WIP and defined. Only macro definitions are allowed.

`MINMAX [N]` – list the maximum and minimum values of the current image.

`MINMAX` list the maximum and minimum values of the image that has been read by the `IMAGE` command. This command may be useful for setting contour levels, or adjusting a halftone plot. The minimum and maximum of the current array are stored in the user variables IMMIN and IMMAX. (The minimum and maximum may also be displayed using the command 'echo immin immax'.) If the optional parameter `N` is present and set to any value greater than 0, then the minimum and maximum are explicitly recalculated. This may be useful if the minimum and maximum has been found and then the logarithm of the image is computed.

`MOVE X Y` – sets the current World (user) position to `(x,y)`.

`MOVE` sets the current World (user) location to (x,y) without drawing any lines or text. This simply provides a way to specify the current position of the cursor.

`MTEXT side disp coord just str` – writes the string `STR` relative to `SIDE`.

`MTEXT` writes text at the position specified relative to a viewport side. The parameter `SIDE` specifies which side of the viewport to place the string. `SIDE` must be either 'B', 'L', 'T', or 'R' to specify the Bottom, Left, Top, or Right side of the viewport. Additionally, if `SIDE` is 'LV' or 'RV', then the string will be written perpendicular to the frame rather than parallel to it. The parameter `DISP` specifies the displacement from the frame edge (in character height units). If `DISP` is negative, then the string is written inside the viewport. The parameter `COORD` is the location of the string as a fractional length of the specified edge of the viewport. `JUST` specifies how to justify the string relative to `COORD` with `JUST = 0` meaning left justified; 0.5, centered; and 1.0, right justified. The text `STR` is the (multiword) string to display. As an example, "good" values used to place a string along the X axis, the Y axis, and the top of the plot box are:

```
mtext B 3.2 0.5 0.5 Xaxis label string
mtext L 2.2 0.5 0.5 Yaxis label string
mtext T 2.0 0.5 0.5 Top label string
```

The first two examples above correspond to the syntax used by the commands `XLABEL` and `YLABEL`.

`NCURSE [n]` – marks a set of points using the cursor.

`NCURSE` allows a set of points to be drawn using the cursor. Points may be added or removed by locating the cursor at a desired point and selecting the key 'A' (Add) or 'D' (Delete) or may terminate the command with the key 'X' (eXit). The points are stored in the `XCOLUMN` and `YCOLUMN` arrays and may be redrawn by subsequent calls to this command. If the optional parameter `N` is given, it is used as the number of points in the previous call to this command. To initialize the command, set `N` to 0. By default, `N` is set to the current number of points in the most recent call to the `XCOLUMN` and `YCOLUMN` commands. On return, the cursor points are returned in increasing order of X (unlike the command `OLIN`).

`NEW item [...]` – creates a new string variable, user variable, or vector.

`NEW` creates a new variable. If more than one `ITEM` is present on the command line, then each will represent the name of a new variable to be created. Each `ITEM` name must be unique and may not correspond to a pre-defined variable or any other defined variable. Once these variables are created, they may be used just like the pre-defined variables: the command `SET` will permit the value to be set/changed and the command `ECHO` may be used to display the current value. Use the command `FREE` to release the memory associated with each `ITEM`. If the `ITEM` is enclosed in double quotes (e.g. `"ITEM"`), a string variable is created. If the `ITEM` has an optional size argument included, a vector is created. Note that when defining a vector variable, the maximum number of elements should be enclosed in brackets and appear affixed to the end of the `ITEM` name (e.g. `array[10]`). If `ITEM` is not a vector or string variable, then `ITEM` represents a simple user variable.

`OLIN [n]` – marks a set of points using the cursor.

`OLIN` allows a set of points to be drawn using the cursor. Points may be added or removed by locating the cursor at a desired point and selecting the key 'A' (Add) or 'D' (Delete) or may terminate the command with the key 'X' (eXit). The points are stored in the `XCOLUMN` and `YCOLUMN` arrays and may be redrawn by subsequent calls to this command. If the optional parameter `N` is given, it is used as the number of points in the previous call to this command. To initialize the command, set `N` to 0. By default, `N` is set to the current number of points in the most recent call to the `XCOLUMN` and `YCOLUMN` commands. On return, the cursor points are returned in the order they were added (unlike the command `NCURSE`).

`PALETTE K [LEV]` – Sets the color palette to entry `K`.
`PALETTE` sets the current image color array to a known look up table (LUT). The argument `K` specifies which palette is loaded and legal values are displayed below. If `K` is not associated with a legal palette value, a warning is issued and the palette remains unchanged. The value of `K` can be negative in which case, the absolute value of `K` is used and the sense of the LUT is reversed. NOTE: Not all devices have color LUTs so this command may have no effect on them. Whenever a device is changed, the palette resets to the default.

```
    K   Palette set to\dots
-------------------------------
    0 : Background to Foreground (default).
    1 : Gray scale.
    2 : A rainbow.
    3 : Heat scale.
    4 : IRAF scale.
    5 : AIPS scale.
    6 : PGPLOT scale.
    7 : Saoimage A scale.
    8 : Saoimage BB scale.
    9 : Saoimage HE scale.
   10 : Saoimage I8 scale.
   11 : DS scale.
   12 : Cyclic scale.
```

The optional parameter `LEV` may be provided to specify the number of levels into which to squeeze the palette. If `LEV` is not present (or is negative), then the number of levels is not changed. If `LEV` is present and set to 0, the number is reset to the maximum range for the current device. If `LEV` is present and positive, the number of color levels is set to this value.

`PANEL nx ny k` – Sets the plot location to a subpanel.
`PANEL` makes the current plot location panel `K`, where there are `NX` panels across and `NY` panels vertically. The direction the panels run depends on the sign of `K`. If `K` is positive, then the panels are counted across from the lower left and upwards (`K = 1` specifies the first panel). If `K` is negative, then the panels are counted across from the upper left and downwards. `PANEL` retains a copy of the `VIEWPORT` coordinate system that is present when `PANEL` is first called. To reset to those values, use the command `panel 1 1 1`; this will reset the plot location to the previous `VIEWPORT` area (this is also done automatically every time the current device is changed). Be sure to also call `panel 1 1 1` prior to changing the plot location with any other `VIEWPORT` command! The separation between adjacent panels is controlled by the value of `EXPAND` as well as by the user variables XSUBMAR and YSUBMAR (see the `SUBMARGIN` command for more details on these variables). If either `NX` or `NY` is negative, then the gap between the subpanels in that direction is removed. The absolute value of `NX` and `NY` is always used to determine the number of subpanels.

`PAPER width aspect` – change the size of the view surface.
`PAPER` changes the size of the view surface to a specified width (inches) and aspect ratio (height/width).

This command should only be used immediately after a call to the DEVICE command.

PCOLUMN n – reads point type data from column N of the current data file.
PCOLUMN reads column N from the data file as the values of a symbol array. The data read by PCOLUMN is used by the commands POINTS and BIN (as well as others) to represent additional data information.

PHARD devname [xxx args] – spool a plot to an alternative device.
PHARD allows the commands in macro xxx (or the commands in the command buffer) to be spooled to an alternative device. The one required argument DEVNAME is the name of the alternative device (see the command DEVICE). If no other arguments are present, then the commands in the command buffer are played back (see the command PLAYBACK); otherwise, the named macro (xxx) is called (with any further arguments on the command line used as input arguments to the named macro). If the named device is a hardcopy device, the HARDCOPY command is automatically executed at the end of the playback. Finally, the current device is set to the device that was present when this command was called.

PLAYBACK [xxx args] – replay macro xxx or commands in the command buffer.
PLAYBACK executes the macro xxx without storing the commands in the command buffer. With an argument of BUFFER (or no argument), PLAYBACK replays all commands in the command buffer. The optional arguments (args) are passed to the macro.

PLOTFIT [X1 X2 [STEP]] – draws a plot of the most recent fit.
PLOTFIT uses the parameters from the most recent call to FIT to display a smooth curve. The optional parameters X1 and X2 can be used to limit the X-range to be plotted. They default to the full range of the window. The other optional parameter is the STEP size of the plot. It provides a means to make the plot smoother or more coarse. The STEP defaults to the mean of the difference between data in the most recent call to the XCOLUMN command.

POINTS [K] – draws points of the current style at each (x,y).
POINTS draws points of the current style (see the command SYMBOL) and size (see the command EXPAND) at the coordinates read by the commands XCOLUMN and YCOLUMN. The command PCOLUMN may be used to specify a SYMBOL corresponding to each (x,y) point plotted. The format of the data read by the PCOLUMN command is the same as the argument for the SYMBOL command, except that an optional fractional part may be added to each symbol. If the entry has a fractional part, it is treated as an expansion factor (fractional parts less than 0.01 gives the default expansion). For example, an entry of 93.5 in a point column is the same as symbol 93, expand 0.5. If the optional argument K is present, the command ECOLUMN may be used to specify a color corresponding to each (x,y) point plotted. Each item of data read by ECOLUMN corresponds to a color index.

POLY – draws a polygon.
POLY will draw a closed polygon based on the data from the most recent call to the XCOLUMN and YCOLUMN commands. POLY will draw the polygon according to the graphical characteristics such as FILL and COLOR. Also see the command RECT.

PUTLABEL just str – writes justified text STR at the current location.
PUTLABEL writes a string at the current cursor location with rotation and size specified by the ANGLE and EXPAND commands (exactly like LABEL). The label is justified with respect to the current location according to the argument JUST: 0 for left justified; 0.5 for centered justified; and 1 for right justified. If JUST is negative, the cursor is activated and the position of the cursor when any key is struck is used as the current location and the absolute value of JUST is used to determine the string justification. NOTE: Because -0 will be interpreted as +0 by the string parser, use something like -0.001 for left justification when using the cursor to locate the position.

QUARTER [N] – allows quick selection of a subsection of the current image.
QUARTER is a short cut to selecting a quadrant of the current image. Rather than determining pixel ranges

and then calling SUBIMAGE, QUARTER allows the user to quickly set the corresponding pixel range. If the optional parameter N is present and set to 1, QUARTER selects the lower left quadrant; set to 2, the lower right; 3, the upper left; and 4, the upper right. QUARTER, with any other value for N or if no argument is present, selects the inner quarter of the current image.

RANGE [X1 X2 Y1 Y2] – limits the range over which to fit.
RANGE permits a way to limit the data used by the FIT command. By default, the FIT command uses all the data read by the most recent call to the XCOLUMN and YCOLUMN commands. Specifying the limits in one direction restricts which data will be used by the FIT command. If the limits in either direction are equal, then the fit is unbounded in that direction.

READ fspec – reads plot commands from file "fspec".
READ reads plot commands from a file and places them in the command buffer. Commands are not executed and macros are ignored. To execute the commands, see the command PLAYBACK; to define macros, see the commands DEFINE and MACRO.

RECT xmin xmax ymin ymax – draw a rectangle, using fill-area attributes.
RECT is like the command POLY but draws a rectangle bordered by the World (user) coordinate vertices (XMIN, YMIN) and (XMAX, YMAX). RECT uses the current fill, line style, and line color characteristics.

RESET – full reset of the graphics state of the current plotting device.
RESET executes a full reset of graphics state of the current plotting device. The coordinate transformation matrix is reset to the default value (see the command TRANSFER); the current angle is set to 0 (see the command ANGLE); the tick mark interval and the number of minor tick marks is reset to 0 (see the command TICKSIZE); the image transfer function is reset to linear (see the command ITF); the text background color index is set to transparent (see the command BGCI); and the color, expand, fill, font, line style, and line width are all set to 1 (see the commands COLOR, EXPAND, FILL, FONT, LSTYLE, and LWIDTH). Finally, the Palette (see the command PALETTE) is reset to the default palette and the number of halftone levels is reset to the maximum range possible for the current device.

RGB K red [green blue] – sets the color representation using the RGB system.
RGB sets the color representation of color index K to the Red-Green-Blue values. The range of values for the color value inputs are [0 - 1], inclusive. If only the first two arguments are present, the RED value is also used to set the GREEN and BLUE values (useful when creating just a gray scale). If K is larger than the maximum color index of the device (use the command 'echo cmin cmax' to display the color index range), then this call is ignored.

SCALE X [Y [K]] – sets the viewport size scale.
SCALE allows the viewport (device) size scale to be set based on the range of the World (user) coordinate range. The viewport is changed to the World range divided by the input scale factor (i.e. X and Y are in World units h per desired unit). The optional parameter K specifies the desired conversion the two scale parameters will be performing and scales as shown below:

```
    K   X and Y are dimensioned as\dots
---------------------------------------------------------------
    0 : World (user) units per normalized device coordinate.
    1 : World units per inch.
    2 : World units per mm.
    3 : World units per pixel.
```

If K is not present on the command line, then it is set to 2. The value of Y may be omitted, in which case it is set to the same value as X. However, if a value other than K = 2 is desired, both X and Y MUST be present. If K is not one of the above accepted values, no conversion is performed. Also, if either X or Y are less than or equal to 0, no conversion is performed. As an example, if the limits of the current plot are 0 - 10 in the x-direction and 0 - 12 in the y-direction, `scale 2 3 1` will produce a viewport region 5x4 inches in size. Hence, it is possible to produce a viewport larger than the view size.

`SET V EXPRESSION` – sets the user variable `V` to result of `EXPRESSION`.
`SET` sets the symbolic variable `V` to the result from evaluating the statement `EXPRESSION`. `V` is a string variable if it is enclosed in double quotes. If `V` is a string variable, then `EXPRESSION` represents the value to assign to the variable (Example: 'set "print" lpr -Pps'). For user and vector variables, `EXPRESSION` may be a simple syntax statement that will be evaluated. Precedence in the expression is ALWAYS left to right with the inner most set of parenthesis evaluated first. Hence `(5 * 3 + 2) != (2 + 3 * 5)` but `((5 * 3) + 2)` does equal `(2 + (3 * 5))`. Currently, a maximum of 20 nested levels of parenthesis is permitted. Also note that a set of parenthesis is understood to mean "( )", "[ ]", or "{ }". At the most basic level, `EXPRESSION` may just be an item that is either a number, a pre-defined user variable, or a vector element. More complex `EXPRESSION`'s may be generated using the operators and functions listed below. If an item begins with '-', it is treated as a unary minus sign. The current defined operations that act on two variables are:

```
 +           add |  /         divide | max                maximum
 -      subtract |  %  modulo divide | min                minimum
 *      multiply |  \ integer divide |  **          exponentiation
==      equal to |  >   greater than |  >= greater than or equal
!= not equal to  |  <      less than |  <=    less than or equal
||    logical OR | &&    logical AND |   ^             logical XOR
```

Note that `min` and `max` are used as `(a min b)` and `(a max b)`. Also, note that logical operations return 1 if true; 0 if false. Current pre-defined functions (i.e. f(x)) are:

```
sqrt(x)          square root |    abs(x)       absolute value
 int(x)   integer truncation |   nint(x)          nearest int
  ln(x)    natural logarithm |    log(x)    base-10 logarithm
 exp(x) base-e antilogarithm |  log10(x)    base-10 logarithm
 sin(x)      sine in radians |   sind(x)      sine in degrees
asin(x)   arcsine in radians |  asind(x)   arcsine in degrees
```

The four types of sin functions apply just as well for the four functions of cos and tan. Note that either `log(x)` or `log10(x)` may be used for the log base 10. Many of the symbolic and user variables are displayed using the `SHOW` command, where symbolic names are displayed in CAPS. All other variables can be displayed using the `ECHO` command. All of the user variable and vectors can then be used as arguments to any other command. NOTE: The macro arguments ($1, $2, ...) may be redefined using this command but only if the macro argument is a numeric user variable.

`SHOW [what]` – shows current limits and attributes.
`SHOW` lists the values of some the symbolic and user variables, including the current location and plot region limits in World (user) and Viewport (device) coordinates, the value of the expansion and angle variables, the line style and width, etc. If no arguments are present, then everything below is displayed. If the optional argument is present, then it represents a string used to define what will be shown. The options that are currently available are:

```
B : Box attributes.
C : Coordinate system information.
D : Device, Printer, and Data file information.
```

```
G : Graphical attributes.
I : Image characteristics.
R : Register variables (\0 - \20).
```

More than one option may be selected (e.g. 'show bci') and unknown options are ignored. The command ECHO may also be used to display user variable values.

SLEVEL type [value] – sets the type and value used to scale contour levels.
SLEVEL is used to change the TYPE and VALUE used to scale the contour level array. Initially, TYPE (the user string variable "LEVTYPE") is set to A and VALUE (the user variable SLEVEL) is set to 1.0 so that no scaling is performed. TYPE may only be either A or P (all other values are treated as A) and defines the type of contour scale factor. If TYPE is P, then scaling is done by percentage; A for absolute scaling. VALUE represents how much the array will be scaled. For example, slevel P 1 will multiply each contour level by 1% of the image peak value. Likewise, slevel A 1E-4 would multiply each contour level by 0.0001. If the parameter VALUE is not given, SLEVEL will not change. The contour level array affected is defined with the commands LEVELS or AUTOLEVS and is altered internally every time the command CONTOUR is called.

STRING name [W1 [W2]] – sets a string variable "name" from a file.
STRING allows certain string variables to be redefined using data in an external file. STRING reads one line (see the commands DATA and LINES) and assigns the string starting with word W1 and ending with W2 to the string variable NAME. If W1 is omitted or set to 0, the entire line is assigned. If W1 is greater than 0, it represents the starting word of the assignment. If W2 is not present, it is set to the value of W1. If W2 is set to 0, it corresponds to the end of the line; if it less than zero, the absolute value represents the number of words to read; if it is greater than zero, it specifies the last word to read.

Currently, the pre-defined string variables are: XBOX and YBOX (which are the default X and Y arguments for a call to the command BOX and are initially set to "BCNST" and "BCNSTV", respectively); PRINT (which is the default print command to spool a plot when the command HARDCOPY or PHARD is used); and HELPFILE (which is the full name of the help file). Other pre-defined string names include XHEADER and YHEADER (see the command HEADER) and LEVTYPE (see the command SLEVEL). New string variables may be defined using the NEW command, released with the command FREE, and may also be assigned to using the SET command.

SUBIMAGE xmin xmax ymin ymax – sets the index range of a subimage.
SUBIMAGE allows the user to specify the indices of a subimage read by the IMAGE command. When IMAGE reads in a new image, the minimum values are set to 1 and the maximum values are set to the number of pixels in the X and Y directions (user variables NX and NY). Use this command to select a subportion of the image by entering the starting and ending pixels of interest. Using the command HEADER after the command SUBIMAGE will reset the limits of the plot region (World coordinates) to the entire subimage region selected.

SUBMARGIN xmarg [ymarg] – sets the gap between individual panels.
SUBMARGIN may be used to fix the gap size between individual frames when using the command PANEL. The gap between individual panels is controlled by the user variables XSUBMAR and YSUBMAR (which are initially set to 2 in character height units). This command is a convenience function to facilitate setting these variables. The arguments XMARG and YMARG are used to set each of the two user variables. If this command is called with only one argument, it is used to set both XSUBMAR and YSUBMAR.

SYMBOL N – sets the current point symbol to N.
SYMBOL causes points to be drawn as the symbol N, where N refers to the graph marker or a Hershey symbol number. The symbol number corresponds to the type of symbol drawn. If the symbol number is -1, it draws a dot of the smallest possible size. If the symbol number is between 0 and 31, it corresponds to

a set of pre-defined symbols; between 33 and 127, it corresponds to the ASCII character in the currently selected font; and larger than 127, corresponds to the Hershey symbol of the same number.

`TICKSIZE XTICK NXSUB YTICK NYSUB` – sets tick intervals for the `BOX` command.
`TICKSIZE` determines tick intervals for the command `BOX`. The argument `NXSUB` refers to the number of intervals between major tick marks on the X-axis and `XTICK` refers to the interval between large ticks in World (user) coordinates. If `XTICK` or `NXSUB` are 0, the box routine will supply it's own intervals according to the World coordinate limits (at least 3 major tick marks). Likewise for `NYSUB` and `YTICK`, except for the Y-axis. Setting `NXSUB` or `NYSUB` to 1 will inhibit any minor tick marks regardless of the arguments to the `BOX` command.

`TRANSFER T1 T2 T3 T4 T5 T6` – specifies the image coordinate transformation.
`TRANSFER` specifies the coordinate transformation between indices I and J of the image array and World (user) coordinates. All two dimensional plot commands (for example, `CONTOUR` or `HALFTONE`) require a coordinate transformation function to place the image on the plot surface. The World coordinates (X, Y) of the array point (I, J) are related by:

```
X = T1 + (T2 * I) + (T3 * J)
Y = T4 + (T5 * I) + (T6 * J)
```

Usually `T3` and `T5` are zero unless the coordinate transformation involves a rotation or sheer. The default transformation is (0, 1, 0, 0, 0, 1). Calling `TRANSFER` with no arguments resets the array to these default values. Usually, the command `HEADER` is used to set the transformation matrix.

`VECTOR [ANGLE [VENT]]` – Draws a vector field as a sequence of arrows.
`VECTOR` draws a vector field as a number of arrows. The tails of the arrows lie at points (x,y) read by `XCOLUMN` and `YCOLUMN`, the lengths of the arrows "r" are read by `PCOLUMN` and the direction of the arrow (counter-clockwise from +X in degrees) is read with `ECOLUMN`. See the command `ARROW` for a description of the optional parameters `ANGLE` and `VENT`. NOTE: The use of `PCOLUMN` in this context requires `SYMBOL` to be re-executed before plotting points.

`VIEWPORT VX1 VX2 VY1 VY2` – sets the physical location of the plot.
The plot region is a rectangle within the coordinates allowed by the plotting device (for example, the Viewport is the region where the command `BOX` will draw its grid). Vectors and points are truncated at the edge of this plot region. `VIEWPORT` specifies (in normalized device coordinates) where the plot region is located. Viewport arguments for this command range from 0 to 1, inclusive, in each direction for all devices. Note that the command `VSTAND` is called to reset the Viewport coordinates every time a new device is opened.

`VSIZE VX1 VX2 VY1 VY2` – sets the physical location of the plot in inches.
`VSIZE` specifies (in inches) where the plot region is located. See also the `VIEWPORT` command.

`VSTAND` – sets the standard (default) viewport.
`VSTAND` defines the standard viewport scale. Currently, this is a box with margins on each side of about 4 character heights. This command is executed every time a new device is chosen. See also the `VIEWPORT` and `VSIZE` commands.

`WEDGE SIDE DISP THICK [MIN MAX BOXARG]` – draws a halftone wedge.
`WEDGE` illustrates the range of values drawn by the halftone command. The `SIDE` argument is used to specify the size and orientation of the wedge. The parameter `SIDE` must be either 'B', 'L', 'T', or 'R' to specify the Bottom, Left, Top, or Right side of the viewport. The `DISP` argument specifies the displacement from the frame edge (in character height units). If `DISP` is negative, then the wedge is drawn inside the viewport. The parameter `THICK` is the thickness of the wedge (in character height units). The optional arguments

MIN and MAX allow the user to specify the intensity range of the wedge. By default, MIN and MAX are the values used by the most recent call to the HALFTONE command. Providing the values of MIN and MAX will cause WEDGE to display a different range of intensities when labeling the wedge box. The optional argument BOXARG permits the user to control how the box around the wedge is drawn (or omit it altogether). By default, a simple box is drawn setting BOXARG to BCSTN/M (depending on the orientation); to eliminate the box, set BOXARG to 0 (for more information on permitted box arguments, see the BOX command). The arguments from the most recent call to the TICKSIZE command are used to draw a box around the wedge and numerically label it. To use the optional BOXARG argument, the MIN and MAX arguments must be provided.

WINADJ [xmin xmax ymin ymax] – sets limits and viewport to same aspect ratio.
WINADJ changes the World (user) limits and adjusts the viewport so that the World coordinate scales are equal in x and y. If the four optional parameters are present, they are used as the new World coordinate values; otherwise, the minimum and maximum values of the XCOLUMN and YCOLUMN arrays are used. Typical usage is (after an image is loaded) to rescale the viewport so that the image has the proper aspect ratio. For example, using the command 'winadj 0 nx 0 ny' after the IMAGE command BUT before the HEADER command will produce square pixels.

WRITE fspec xxx – writes macro xxx to file "fspec".
WRITE the commands making up a macro to a file. If the macro name XXX given is all, all macros currently defined and the entire command buffer (BUFFER) are written to the file. If the macro name XXX is macros, then only the currently defined macros are written to the file (i.e. the command buffer BUFFER is not written). Otherwise, XXX represents a list of macro names to be written in the output file. NOTE: This command will OVERWRITE any currently existing file.

XCOLUMN N – reads X data from column N of the current file.
XCOLUMN reads column N from the data file as the values of the X coordinates of the data points.

XLABEL str – writes the label STR centered under the X axis.
XLABEL writes a string centered under the X axis drawn by the command BOX. Also see the command MTEXT.

YCOLUMN N – reads Y data from column N of the current file.
YCOLUMN reads column N from the data file as the values of the Y coordinates of the data points.

YLABEL str – writes the label STR centered left of the Y axis.
YLABEL writes a string centered left of the Y axis drawn by the command BOX. Also see the command MTEXT; especially if the displacement of the text is insufficient to avoid the axis numbers.

Appendix E

Sample Plots

This appendix contains several examples of plot files generated using many of the WIP commands described in this manual. Because of the obvious limitations of print, commands like `color` are omitted from this appendix. Wherever possible, each command is followed by a comment statement. While these comments, marked by the comment character (#), are not required, they do provide information as to why certain commands are included or are necessary to generate a certain aspect of a plot. The user is strongly encouraged to include comments whenever possible; especially in macro definitions. All plots in this appendix may be reproduced by typing the commands listed in the accompanying plot files.

Figure E.1 is a very basic plot demonstrating many of the commands that control the simple graphical attributes described in Chapter 3.

Figure E.2 demonstrates how to read data in from an external file.

Figure E.3 shows several examples of how to annotate a plot with text and also illustrates the various fonts available in WIP.

Figure E.4 displays the lower and upper case Greek letters and the Roman characters needed to generate them.

Figure E.5 presents a simple example of how to partition a single plot surface into sub-regions or panels; illustrates the various ways that the `box` command may be called; and demonstrates how to define and pass arguments to a macro.

Figure E.6 presents a more complicated example of how to use macros and how to generate array entries. This example also illustrates the use of the `set` and `loop` commands.

Figure E.7 plots the standard graph markers used by the `symbol` command.

Figure E.8 continues the previous example by illustrating how to display the Hershey symbols used by WIP. The macro, as listed, will display all currently available Hershey symbols, but only the first page is shown in this example. This example also makes use of macros, panels, and the `loop` command.

Figure E.9 is an example of how to generate a fit and display it. The different methods of fits are illustrated.

Figure E.10 is a simple example of how to generate an image halftone and wedge and also lists a typical header for a FITS image file.

Figure E.11 expands on the previous example by demonstrating how to extract a subimage and display it in a different region on the view screen. In addition, it illustrates the power of user variables and macros in moving to various regions of the display.

Figure E.12 expands on the previous two image examples by showing how to overlay two images that have different resolutions.

E.1 Basic graphics commands

```
viewport 0.2 0.8 0.3 0.9  # Set the location of the window.
expand 1.2                # Set the character size.
font 2                    # Set the font.
limits 0 10 0 100         # Set the world coordinate limits.
fill 1                    # Fill solid.
rect 7 9 80 90            # Draw a filled rectangle.
fill 2                    # Fill hollow.
rect 3 5 80 90            # Draw a hollow rectangle.
lstyle 1                  # Solid line.
lwidth 2                  # Thicker line.
move 5 80
draw 2 40
lstyle 2                  # Dashed line.
lwidth 1                  # Normal thickness.
move 9 80
draw 6 40
lstyle 1                  # Solid line.
symbol 2                  # Set symbol to a '+' sign.
move 2 20
dot                       # Draw the symbol.
symbol 3                  # Set symbol to a '*' sign.
move 3 20
dot                       # Draw the symbol.
symbol 4                  # Set symbol to an open circle.
move 4 20
dot                       # Draw the symbol.
symbol 5                  # Set symbol to a 'x' sign.
move 5 20
dot                       # Draw the symbol.
symbol 6                  # Set symbol to an open square.
move 6 20
dot                       # Draw the symbol.
symbol 7                  # Set symbol to an open diamond.
move 7 20
dot                       # Draw the symbol.
symbol 14                 # Set symbol to a hollow '+' sign.
move 8 20
dot                       # Draw the symbol.
box bcnst bcnstv          # Draw a box with numbers and tick marks.
```

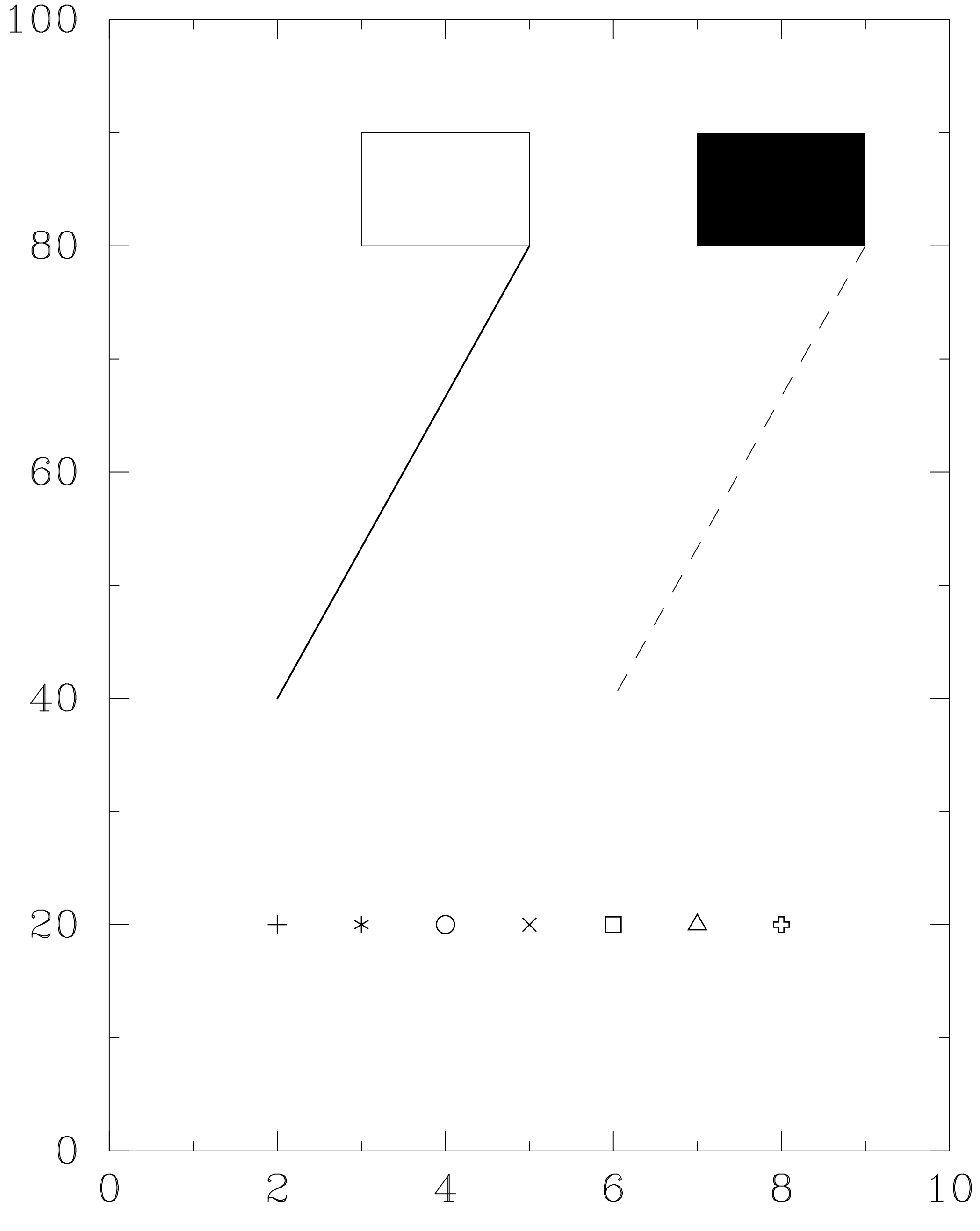

Figure E.1: A simple plot.

E.2 Reading data from an external file

```
viewport 0.2 0.8 0.3 0.8   # Set the viewport.
data data.dat              # Open the data file.
xcolumn 1                  # Read the x data from column 1.
ycolumn 2                  # Read the y data from column 2.
pcolumn 3                  # Read the symbol data from column 3.
limits 0 10 0 20           # Set world coordinates.
box bcnst bcnst            # Draw a box.
xlabel (x)
ylabel (y)
mtext T 2 0.5 0.5 A Simple Graph
points                     # Draw symbol for each point.
connect                    # Connect the points.
```

Contents of the external file data.dat

```
 0.00   0.00  9.0
 0.20   0.04  2.5
 0.40   0.16  2.5
 0.60   0.36  2.5
 0.80   0.64  2.5
 1.00   1.00  9.0
 1.20   1.44  2.5
 1.40   1.96  2.5
 1.60   2.56  2.5
 1.80   3.24  2.5
 2.00   4.00  9.0
 2.20   4.84  2.5
 2.40   5.76  2.5
 2.60   6.76  2.5
 2.80   7.84  2.5
 3.00   9.00  9.0
 3.20  10.24  2.5
 3.40  11.56  2.5
 3.60  12.96  2.5
 3.80  14.44  2.5
 4.00  16.00  9.0
 4.20  17.64  2.5
 4.40  19.36  2.5
 4.60  21.16  2.5
 4.80  23.04  2.5
 5.00  25.00  9.0
 5.20  27.04  2.5
 5.40  29.16  2.5
 5.60  31.36  2.5
 5.80  33.64  2.5
```

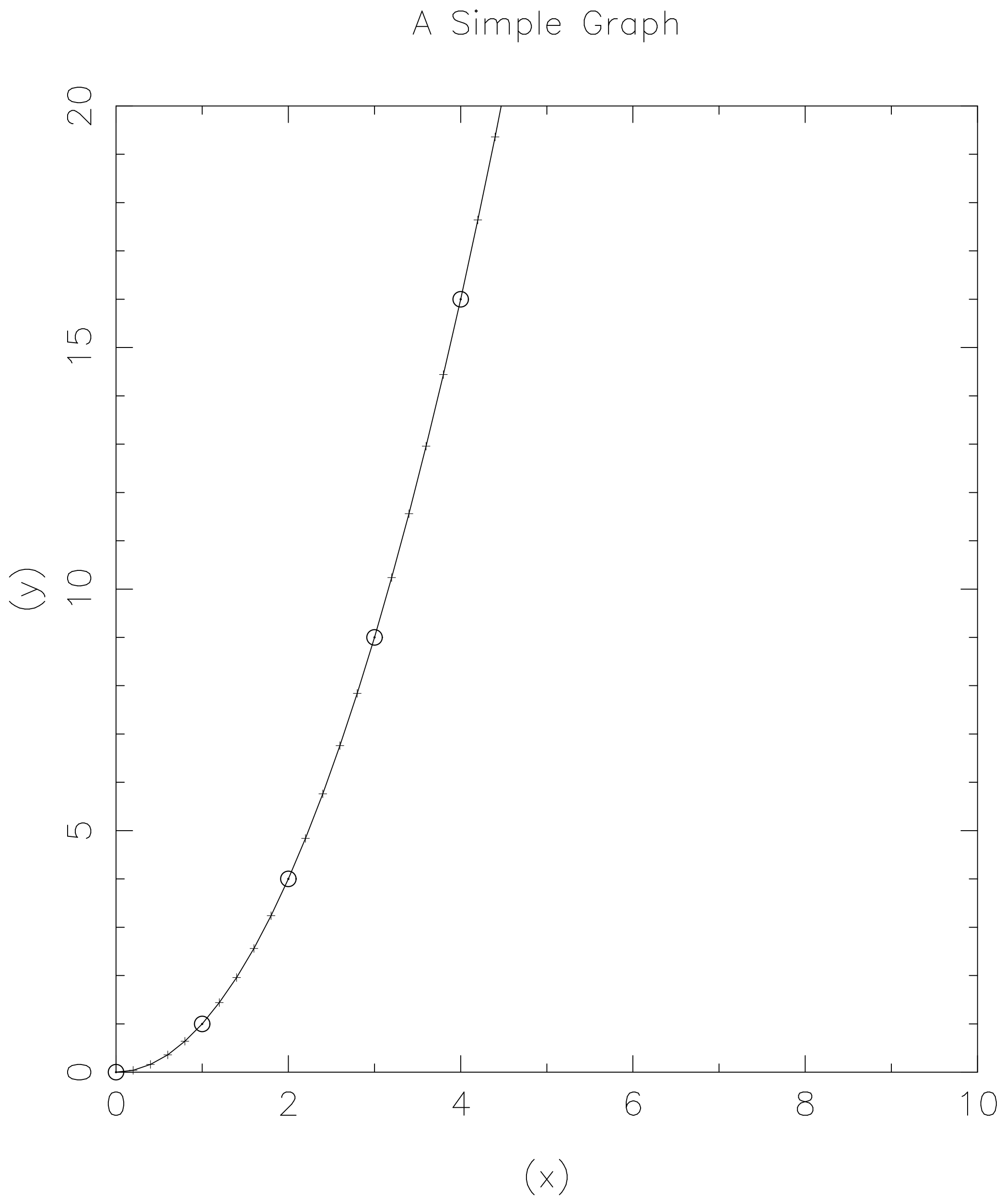

Figure E.2: A plot using an external data file.

E.3 Fonts and different ways to present text

```
viewport 0.2 0.8 0.2 0.9
limits 0 1 16 0
box bc bc
move 0.05 1
label Normal:  \fnABCDQ efgh 1234 \ga\gb\gg\gd \gL\gH\gD\gW
move 0.05 2
label Roman:  \frABCDQ efgh 1234 \ga\gb\gg\gd \gL\gH\gD\gW
move 0.05 3
label Italic:  \fiABCDQ efgh 1234 \ga\gb\gg\gd \gL\gH\gD\gW
move 0.05 4
label Script:  \fsABCDQ efgh 1234 \ga\gb\gg\gd \gL\gH\gD\gW
move 0.05 5
label \fif\fr(\fix\fr) = \fix\fr\u2\dcos(2\gp\fix\fr)e\u\fix\fr\u2
move 0.05 6
label \fiH\d0\u \fr= 75 \(2233) 25 km s\u-1\d Mpc\u-1\d
move 0.05 7
label \fsL/L\d\(2281)\u\fr = 5.6 (\gl1216\A)
expand 1.5
lwidth 3
move 0.05 10
label Bigger (1.5)
expand 0.5
lwidth 1
move 0.5 10
label Smaller (0.5)
expand 1
symbol 3
move 0.5 12
dot
putlabel 0 Left justified (0.0)
move 0.5 13
dot
putlabel 0.5 Centered (0.5)
move 0.5 14
dot
putlabel 1.0 Right justified (1.0)
move 0.5 15
dot
angle 45
putlabel 0 Angle = 45\(2218)
angle 0
```

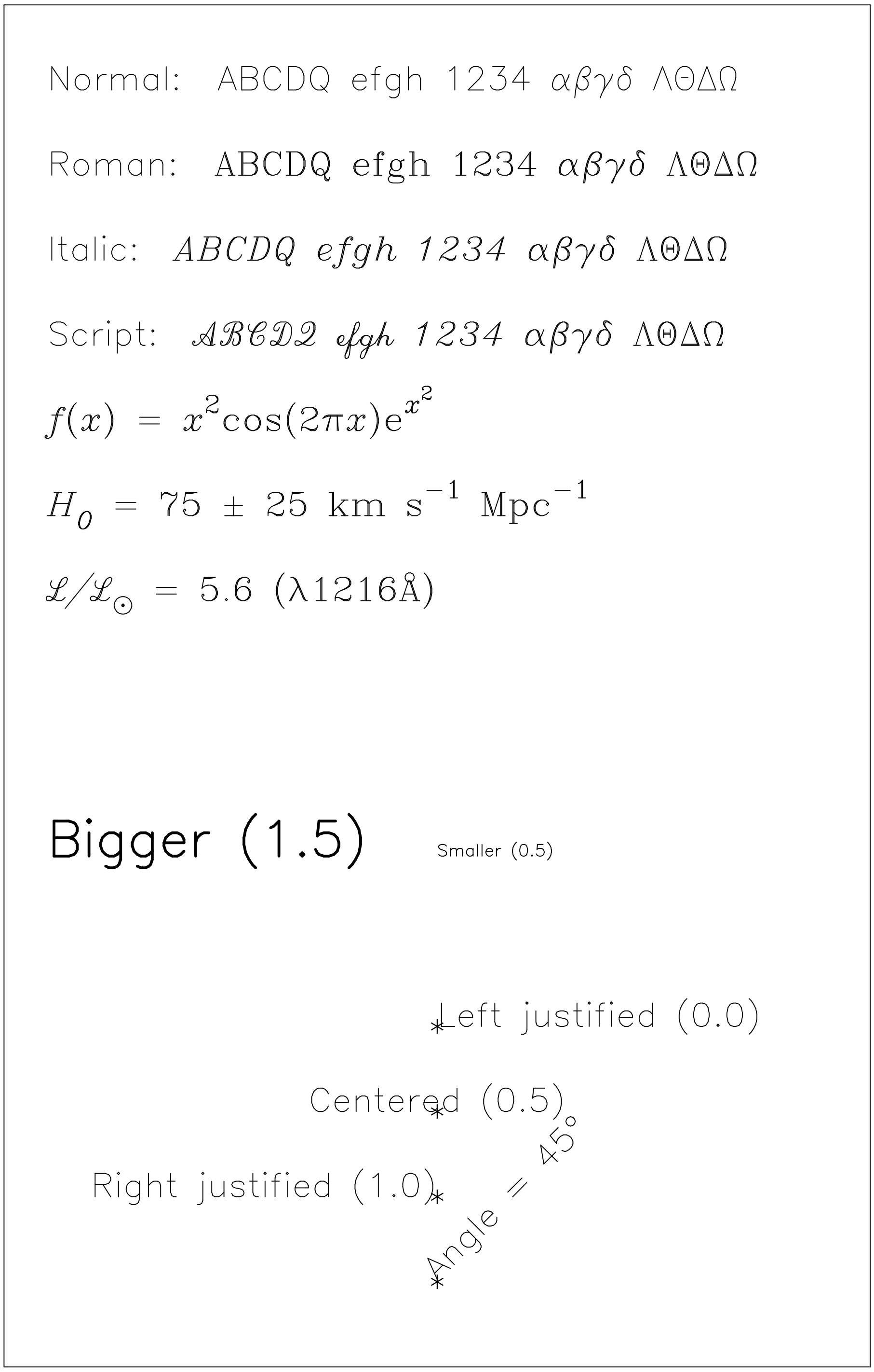

Figure E.3: A plot demonstrating various text capabilities.

E.4 Roman to Greek character mapping

```
expand 1.2
limits 0 10 0 10
move 5 8
putlabel 0.5 Roman to Greek Letter Key
move 5 7
putlabel 0.5 Examples:  \\gu = \gu    \\gU = \gU
move 3 6
putlabel 1.0 a b c d e f g h
move 5 6
putlabel 0.5 i k l m n o p q
move 7 6
putlabel 0.0 r s t u w x y z
move 3 5.5
putlabel 1.0 \ga \gb \gc \gd \ge \gf \gg \gh
move 5 5.5
putlabel 0.5 \gi \gk \gl \gm \gn \go \gp \gq
move 7 5.5
putlabel 0.0 \gr \gs \gt \gu \gw \gx \gy \gz
move 3 4.5
putlabel 1.0 A B C D E F G H
move 5 4.5
putlabel 0.5 I K L M N O P Q
move 7 4.5
putlabel 0.0 R S T U W X Y Z
move 3 4
putlabel 1.0 \gA \gB \gC \gD \gE \gF \gG \gH
move 5 4
putlabel 0.5 \gI \gK \gL \gM \gN \gO \gP \gQ
move 7 4
putlabel 0.0 \gR \gS \gT \gU \gW \gX \gY \gZ
```

Roman to Greek Letter Key

Examples: \gu = υ \gU = Υ

a	b	c	d	e	f	g	h	i	k	l	m	n	o	p	q	r	s	t	u	w	x	y	z
α	β	ξ	δ	ϵ	ϕ	γ	θ	ι	κ	λ	μ	ν	o	π	ψ	ρ	σ	τ	υ	ω	χ	η	ζ
A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	W	X	Y	Z
A	B	Ξ	Δ	E	Φ	Γ	Θ	I	K	Λ	M	N	O	Π	Ψ	P	Σ	T	Υ	Ω	X	H	Z

Figure E.4: A plot demonstrating how to generate Greek characters.

E.5 Macros and subdivision of a view surface

```
define mybox   # mybox $1=panel-# $2=xbox $3=ybox $4=lstyle $5=boxarg.
set \0 lstyle  # Save current line style.
panel 2 2 $1   # Select panel.
box $2 $3      # Draw box with input arguments.
lstyle 0$4     # Select a line style (0 if $4 is not present).
box g$5 g$5    # Draw a grid ($5 is l if logarithmic).
lstyle \0      # Reset previous line style.
end

viewport 0.2 0.9 0.3 0.8   # Set the viewport.
expand 1.1                 # Set the character size.
lstyle 1                   # Set line style to a solid line.
data data.dat              # Open data file.
limits -0.1 6.4 -1.3 1.3   # Set the window limits.
mybox 1 bcn bncv 4         # Draw lower left box.
ticksize 1 5 0.3 3         # Fix tick step size.
mybox 2 bcn bncv 2         # Draw lower right box.
ticksize 0 0 0 0           # Reset ticks to default value.
limits -1 1 -1 0.3         # Set up logarithm limits.
mybox 3 bcnl bncvl 4 l     # Draw upper left box.
xcolumn 1                  # Store the first column of numbers in X[].
logarithm x                # Take the log base 10 of the X[] data.
ycolumn 2                  # Store the second column of numbers in Y[].
logarithm y                # Take the log base 10 of the Y[] data.
symbol 16                  # Set current symbol to a filled square.
points                     # Draw the points.
mybox 4 bcnl bncvl 3 l     # Draw the upper right box.
```

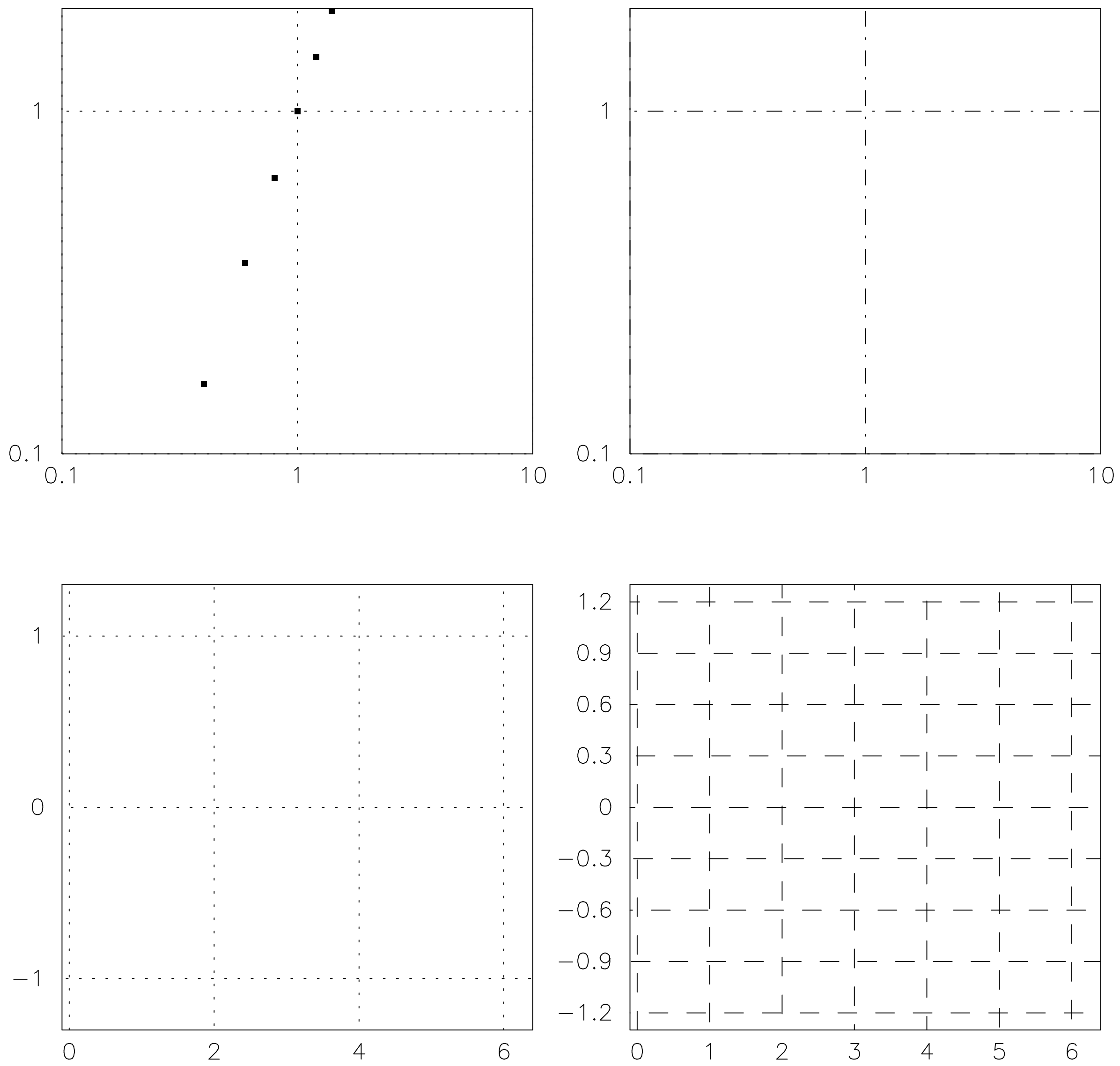

Figure E.5: A plot demonstrating different panels.

E.6 Generating data internally with macros

```
define generate   # Usage: loop N generate; $1=Counter; $2=N; uses \9.
set $1 $1 + 1                # Increment the counter.
set \9 4 * pi * (((2 * \10) - $2) / ($2 - 1))
set x[$1] \9                 # Set x[i] equal to 4*pi*(((2*i)-N)/(N-1)).
set \9 abs(\9)               # \9 now equals abs(x).
set y[$1] cos(\9) / (1 + \9) # Set y[i] equal to cos(|x|) / (1 + |x|).
end

viewport 0.25 0.85 0.3 0.8   # Set the viewport scale.
expand 1.2                   # Set the character size.
font 2                       # Set the font type.
set \10 0                    # Initialize the counter.
loop 100 generate \10 100    # Fill the x and y arrays.
set \0 4 * pi                # Save 4 * pi.
limits -\0 \0 -0.5 1.2       # Set the limits as -4pi, 4pi, -0.5, 1.2.
box bcnst bcnstv             # Draw the box.
connect                      # Draw the curve.
symbol 2                     # Select the '+' symbol.
points                       # Draw the points.
expand 1.5                   # Set the character size a bit bigger.
mtext T 1.0 0.5 0.5 y = cos(|x|) / (1 + |x|)
expand 1.2                   # Reset the character size.
xlabel (x)
ylabel (y)
```

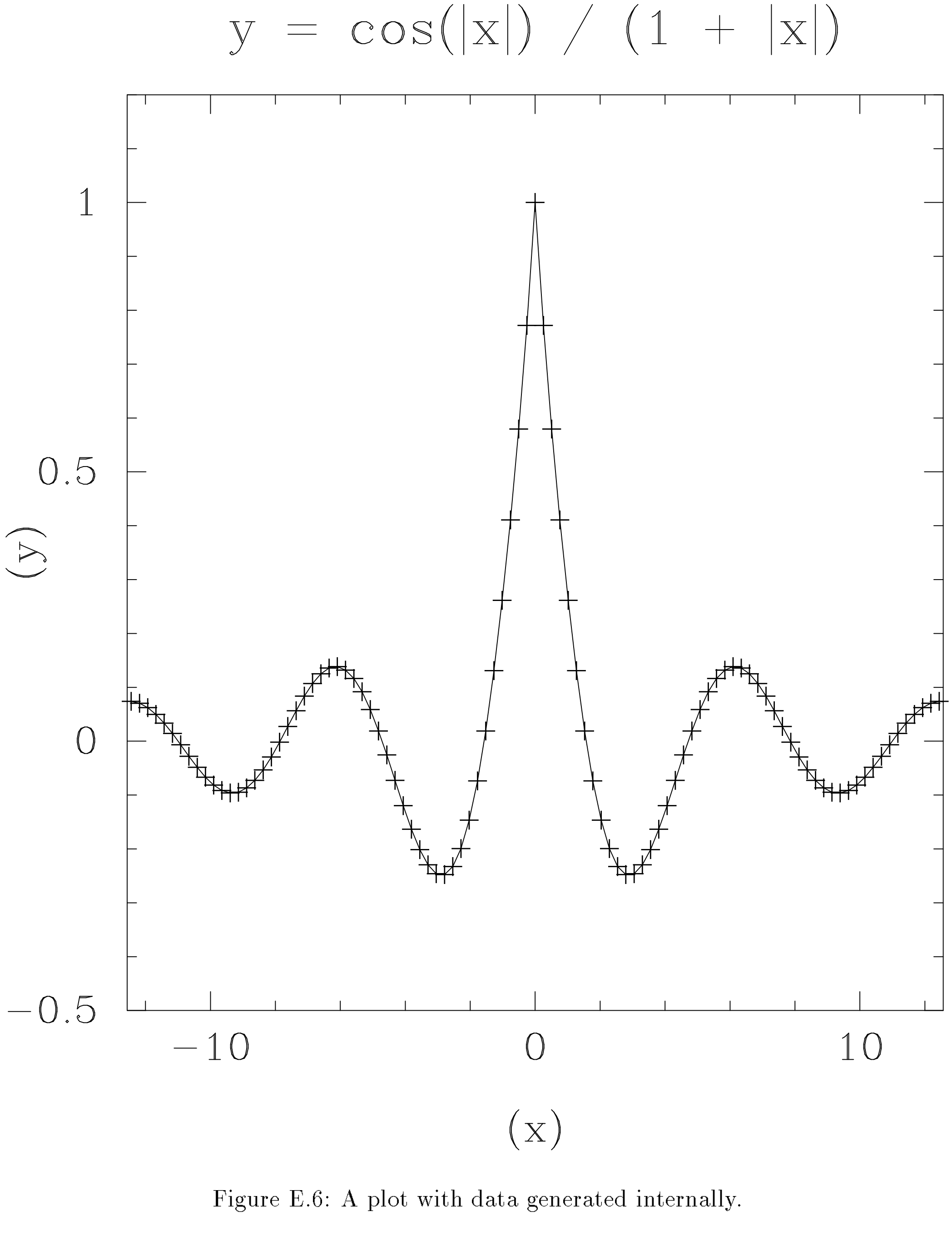

Figure E.6: A plot with data generated internally.

E.7 Symbols used to plot points

```
define dosymbol              # $1=Counter; $2=Nx; $3=Ny.
set $1 $1 + 1                # Increment the Counter.
panel -$2 -$3 -$1            # Set up the panel.
lwidth 1                     # Set the line width to narrow.
box bc bc                    # Draw the frame for this panel.
expand 0.5                   # Set a small character size.
set \0 $1 - 1                # Symbol number is $1 less one.
mtext t -1.1 0.97 1.0 \[\0]
lwidth 2                     # Get a thicker line.
expand 1.0                   # Reset the character size.
symbol \0                    # Chose this symbol.
dot 0 0                      # Draw the symbol at the panel center.
end

viewport 0.2 0.8 0.2 0.8     # Set the viewport scale.
winadj 0 4 0 8               # Set the aspect ratio.
limits -1 1 -1 1             # Set limits.
set nsig 0                   # Force display to only use integer format.
set \10 0                    # Initialize the loop counter.
loop 32 dosymbol \10 4 8     # Draw each symbol.
```

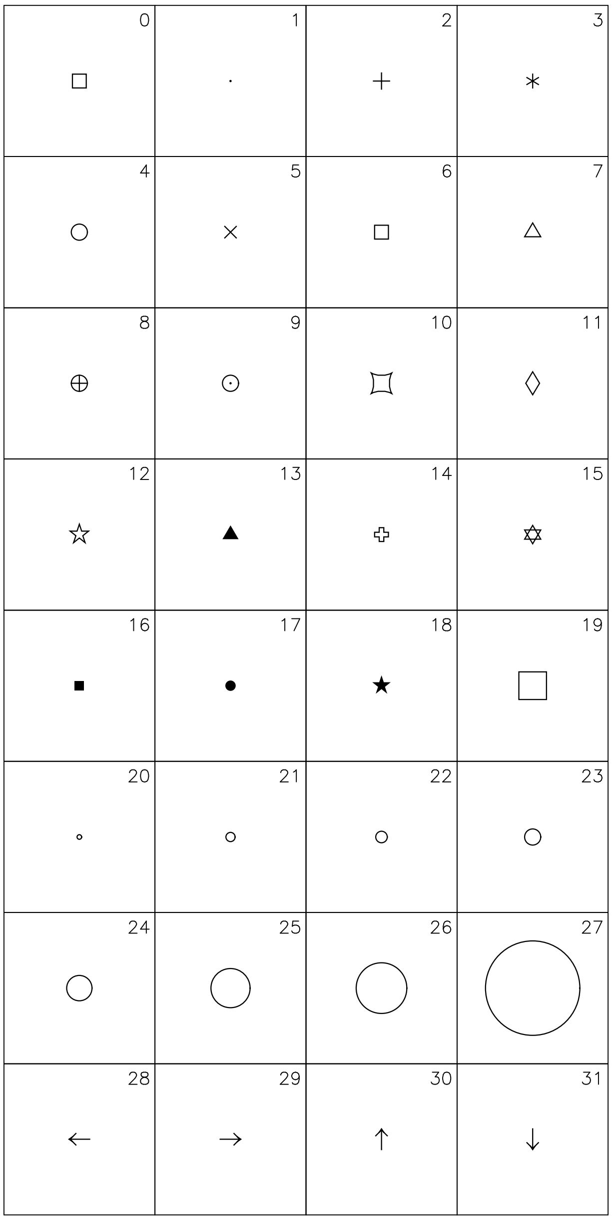

Figure E.7: A plot of the standard symbol markers.

E.8 Hershey Symbols

```
define dosymbol              # $1=Counter; $2=Nx; $3=Ny.
set \0 1 + ($1 % ($2 * $3))  # Make sure the index is in range.
panel -$2 -$3 -\0            # Set up the panel.
lwidth 1                     # Set the line width to narrow.
box bc bc                    # Draw the frame for this panel.
expand 0.5                   # Set a small character size.
mtext t -1.1 0.97 1.0 \[$1]
lwidth 2                     # Get a thicker line.
expand 1.0                   # Reset the character size.
symbol $1                    # Chose this symbol.
dot 0 0                      # Draw the symbol at the panel center.
set $1 $1 + 1                # Increment the Counter.
end

define doloop                # $1=Counter; $2=Nx; $3=Ny.
erase                        # Clear the screen or go to next page.
loop 256 dosymbol $1 $2 $3   # Draw this page and advance counter by 256.
end

viewport 0.2 0.8 0.2 0.8     # Set the viewport scale.
winadj 0 16 0 16             # Set the aspect ratio.
limits -1 1 -1 1             # Set limits.
set nsig 0                   # Force display to only use integer format.
ask 1                        # Make sure each "erase" is prompted.
set \11 0                    # Initialize the loop counter.
loop 4 doloop \11 16 16      # Do the first half of the job.
set \11 1792                 # Reinitialize the loop counter.
loop 5 doloop \11 16 16      # Do the second half of the job.
ask 0                        # Turn off the prompt.
```

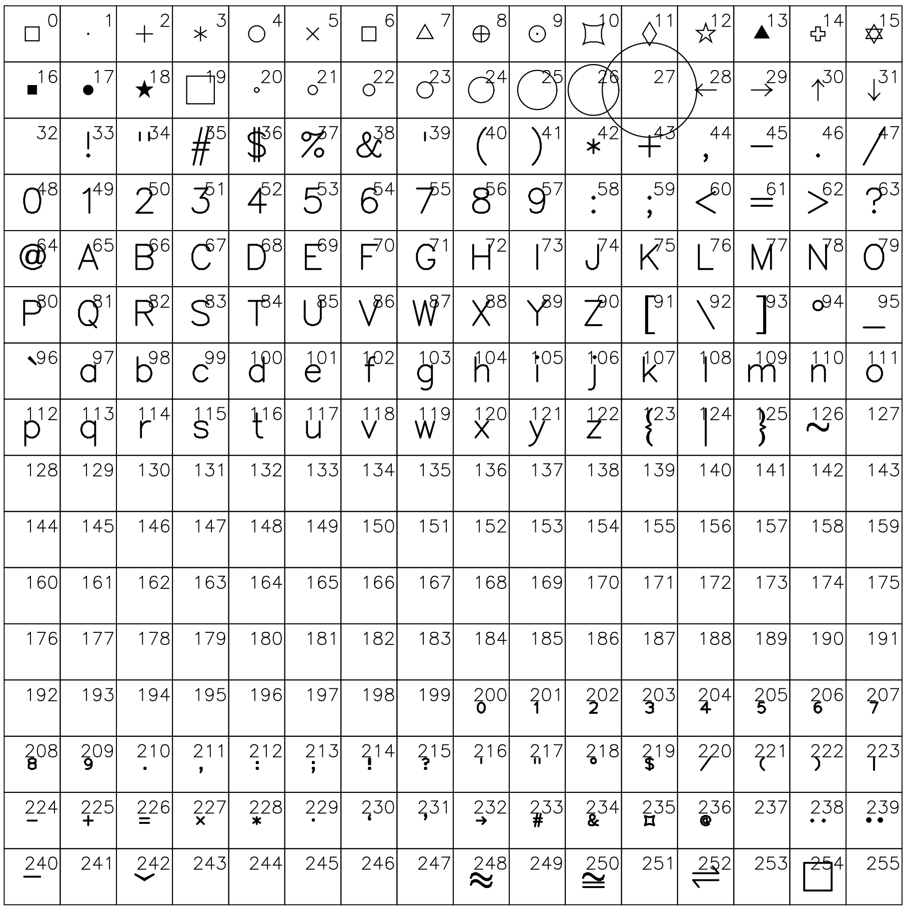

Figure E.8: A plot of a few Hershey symbols.

E.9 Fitting

```
define display     # $1 = panel number; $2 = data file; $3 = Fit type.
color 1            # Reset color index.
panel 1 -3 -$1     # Set the selected panel.
data $2            # Open the data file.
xcolumn 2
ycolumn 3
ecolumn 4
ylabel \fr $3 Fitting
end

viewport 0.2 0.8 0.2 0.8 # Set the viewport scale.
font 2             # Choose a nice font.
expand 1.2         # Choose a nice expansion.
display 1 gauss.dat Gaussian # Read data containing two gaussians.
limits             # Autoscale the limits.
connect            # Draw the data.
box bcst bncstv    # Draw the box (no X numbers).
fit gauss 2  4.5 2.2 2.8  2.5 4.9 2.8 # Fit 2 gaussians.
lstyle 4           # Select a dashed line style.
color 3            # Set color to green.
plotfit x1 x2 0.5  # Plot the fit at high resolution.
color 4            # Set color to blue.
plotfit x1 x2 2.0  # Plot the fit at coarse resolution.
lstyle 1           # Reset line style to solid.
range 6 10         # Limit the range in the X direction.
fit gauss 2  4.5 2.2 2.8  2.5 4.9 2.8 # Fit 2 gaussians.
range 0 0          # Reset the range to the full X-axis.
color 2            # Set color to red.
plotfit 6 10       # Plot the fit for the limited range.

display 2 line.dat Linear # Read data scattered linearly.
limits x1 x2 0 0   # Retain X limits; autoscale Y limits.
symbol 2           # Select the '+' symbol.
points             # Display data as points.
box bcst bncstv    # Draw the box (no X numbers).
fit lsqfit         # Fit via Least Squares method.
color 3            # Set color to green.
plotfit            # Plot the lsqfit solution.
move 0 6.5         # Draw a line for...
draw 1 6.5         # ... the fit legend.
label \fr lsqfit (Least Absolute Deviations)
fit medfit         # Fit via Least Absolute Deviations.
color 4            # Set color to blue.
lstyle 4           # Select a dashed line style.
plotfit            # Plot the medfit solution.
```

```
move 0 6.2          # Draw a line for...
draw 1 6.2          # ... the fit legend.
label \fr medfit (Least Squares Method)
lstyle 1            # Reset line style to solid.
color 1             # Reset color index.

display 3 poly.dat Polynomial # Read data containing a polynomial.
limits x1 x2 0 10   # Retain X limits; autoscale Y limits.
connect             # Draw the data.
box bcnst bncstv    # Draw the box (with X numbers).
fit polynomial 5    # Fit a simple polynomial of order 4.
color 3             # Set color to green.
plotfit             # Plot the polynomial solution.
move 0 8            # Draw a line for...
draw 1 8            # ... the fit legend.
label \fr Polynomial
fit legendre 5      # Fit a Legendre polynomial of order 4.
color 4             # Set color to blue.
lstyle 4            # Select a dashed line style.
plotfit             # Plot the Legendre solution.
move 0 7            # Draw a line for...
draw 1 7            # ... the fit legend.
label \fr Legendre Polynomial
lstyle 1            # Reset line style to solid.
color 1             # Reset color index.
```

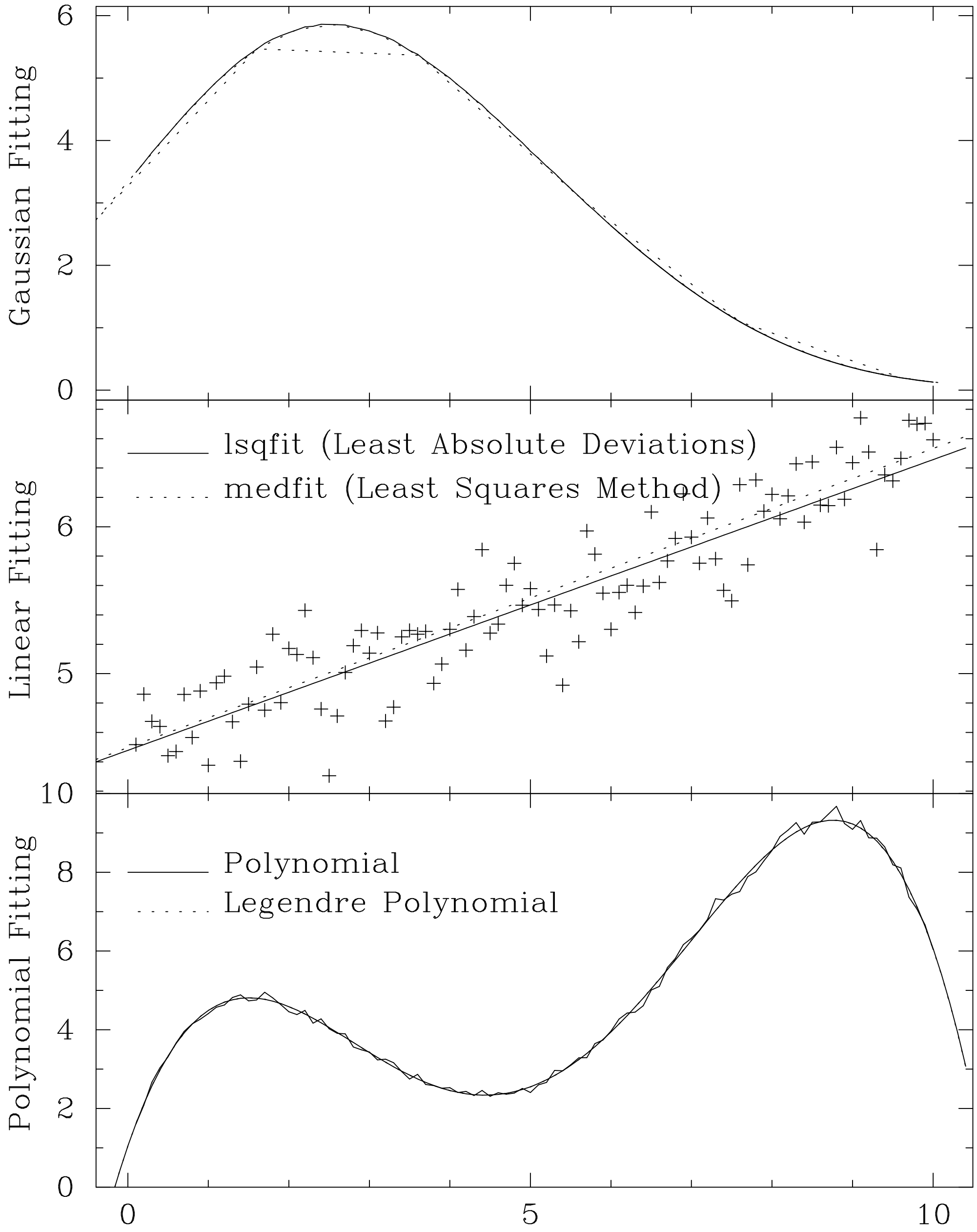

Figure E.9: A plot showing different ways to fit data.

E.10 A simple image

```
viewport 0.25 0.75 0.2 0.9 # Set viewport size.
expand 1.1                 # Set the character size.
font 2                     # Set the font type.
image orion.fits 1         # Read in plane #1.
winadj 0 nx 0 ny           # Set the x/y scale to be the same.
header rd                  # Read header and set limits.
halftone 0 20              # Draw the halftone.
box bcnsthz bcnstvdyzf     # Draw a box.
expand 1.5                 # Make the top label a bit bigger.
mtext T -2 0.15 0 T\dmax
expand 1.1                 # Reset character size.
xlabel \ga (1950)
mtext L 3.5 0.5 0.5 \gd (1950)
wedge RI 0.5 3.0 0 20 BCMV
```

Contents of the image header file

```
SIMPLE  =                    T  /
BITPIX  =                  -32  /
NAXIS   =                    2  /
NAXIS1  =                  201  /
NAXIS2  =                  321  /
BSCALE  =    1.00000000000E+00  /
BZERO   =    0.00000000000E+00  /
CDELT1  =   -1.24999992549E-02  /
CROTA1  =    0.00000000000E+00  /
CRPIX1  =    1.61000000000E+02  /
CRVAL1  =    8.32499982901E+01  /
CTYPE1  =           'RA---SIN'  /
CDELT2  =    1.24999992549E-02  /
CROTA2  =    0.00000000000E+00  /
CRPIX2  =    2.41000000000E+02  /
CRVAL2  =   -5.49999969802E+00  /
CTYPE2  =           'DEC--SIN'  /
DATAMIN =        -1.715574E-07  /
DATAMAX =             48.60251  /
END
```

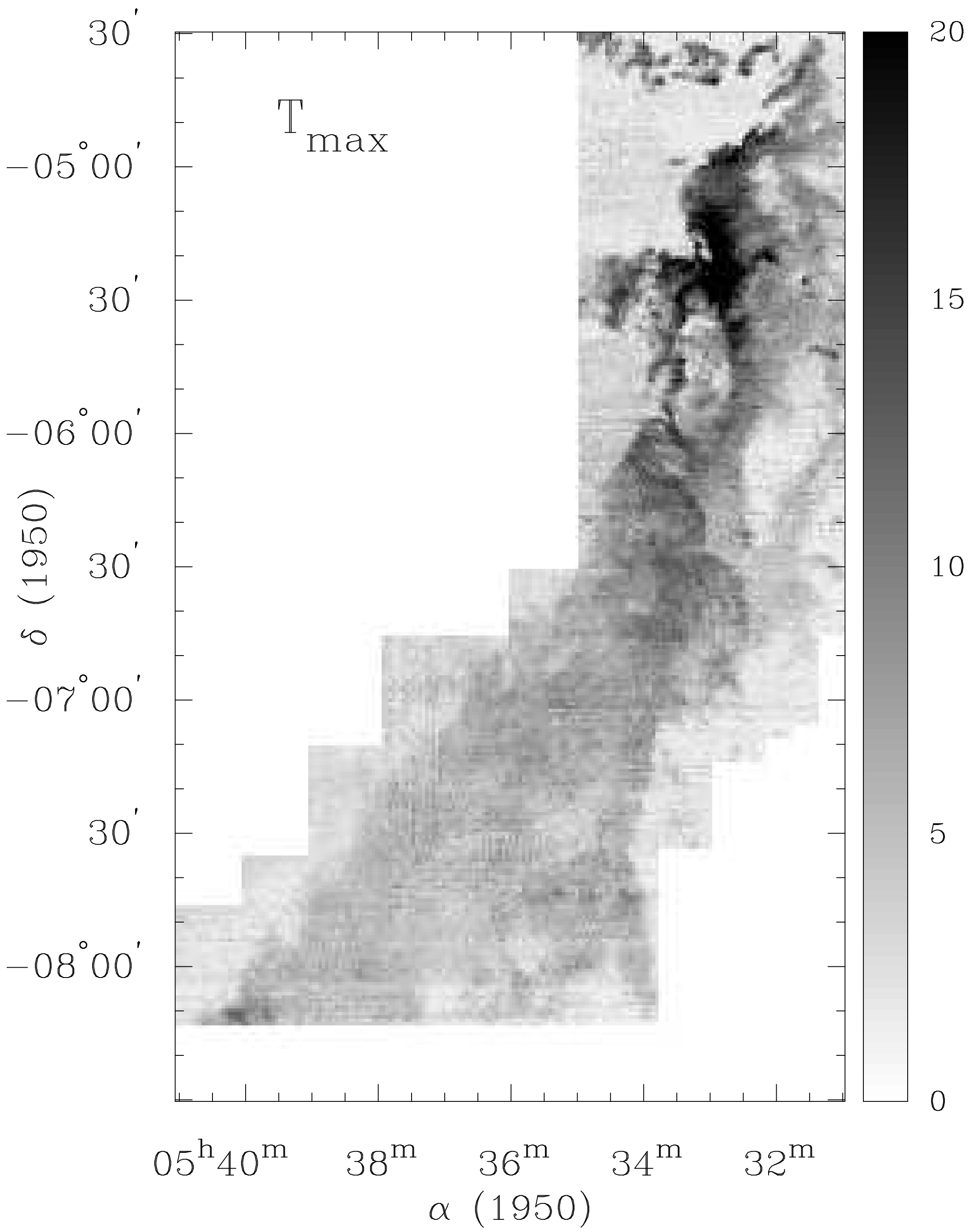

Figure E.10: A basic image, halftone plot, and wedge.

E.11 Multiple image layout and more macros

```
define xytovp             # $1=x $2=y : Sets ($3, $4) to viewport coords.
set $3 ((vx2 - vx1) * ($1 - x1) / (x2 - x1)) + vx1
set $4 ((vy2 - vy1) * ($2 - y1) / (y2 - y1)) + vy1
end

define vptoxy             # $1=vx $2=vy : Sets ($3, $4) to xy coords.
set $3 ((x2 - x1) * ($1 - vx1) / (vx2 - vx1)) + x1
set $4 ((y2 - y1) * ($2 - vy1) / (vy2 - vy1)) + y1
end

define dobox              # $1=blcx $2=trcx $3=blcy $4=trcy
move $1 $3                # Move to lower left corner.
draw $1 $4                # Draw to upper left corner.
draw $2 $4                # Draw to upper right corner.
draw $2 $3                # Draw to lower right corner.
draw $1 $3                # Close box by drawing to lower left corner.
end

define outline            # $1-$4=box, $5=lstyle; sets \11-\14.
new tmpls                 # Create a dummy lstyle variable.
set tmpls lstyle          # Save current lstyle value.
lstyle $5                 # Set to new input lstyle value.
dobox $1 $2 $3 $4         # Draw outline of box.
xytovp $1 $3 \11 \12      # Convert to viewport values and save them.
xytovp $2 $4 \13 \14      # Convert to viewport values and save them.
lstyle tmpls              # Reset lstyle value.
free tmpls                # Release the dummy lstyle variable.
end

define doconnect          # $1=lstyle; sets \1-\4 and uses \11-\14.
new tmpls xp yp           # Create a dummy lstyle and position variables.
set tmpls lstyle          # Save current lstyle value.
xytovp x1 y1 \1 \2        # Convert lower left corner to viewport values.
xytovp x2 y2 \3 \4        # Do the same for the upper right corner.
viewport 0 1 0 1          # Make entire screen accessable.
lstyle $1                 # Set to new input lstyle value.
vptoxy \11 \12 xp yp      # Convert point to world coordinates.
move xp yp                # Move to this position.
vptoxy \1 \2 xp yp        # Convert point to world coordinates.
draw xp yp                # Draw to this position.
vptoxy \13 \14 xp yp      # Convert point to world coordinates.
move xp yp                # Move to this position.
vptoxy \3 \4 xp yp        # Convert point to world coordinates.
draw xp yp                # Draw to this position.
lstyle tmpls              # Reset lstyle value.
```

```
free tmpls xp yp          # Release the variables.
end

ticksize 240 4 3600 4     # Force the tick size spacing.
viewport 0.2 0.55 0.2 1.0 # Select the left side.
lstyle 1                  # Set the line style.
expand 1.15               # Set the character size.
font 2                    # Set the font type.
image orion.fits 1        # Read in plane #1.
winadj 0 nx 0 ny          # Set the x/y scale to be the same.
header rd                 # Read header and set limits.
halftone 0 20             # Draw the halftone.
box bcnsthz bcnstvdyz     # Draw a box.
xlabel \ga (1950)
mtext L 3.5 0.5 0.5 \gd (1950)
expand 1.5                # Make the top label a bit bigger.
mtext T -2 0.1 0 T\dmax
expand 1.15               # Reset character size.
limits 1 nx 1 ny          # Reset the limits to pixels.
set \1 nx - 101           # Set the left edge subimage.
set \2 nx                 # Set the right edge subimage.
set \3 ny - 161           # Set the bottom edge subimage.
set \4 ny                 # Set the top edge subimage.
outline \1 \2 \3 \4  4    # Draw an outline around selected region.
viewport 0.7 0.95 vy1 vy2 # Select the right side.
subimage \1 \2 \3 \4      # Select the subimage.
set \5 \2 - \1 + 1        # When setting the aspect ratio, add 1 to
set \6 \4 - \3 + 1        # each axis to account for the pixel size.
winadj 0 \5 0 \6          # Scale so RA = DEC.
header rd                 # Reset the limits to RA, DEC.
levels 10 20 30 40        # Set the contour levels.
contour                   # Draw the contour.
ticksize 120 4 1800 3     # Force a smaller tick size spacing.
box bcnsthz bcnstvdyzf    # Draw a box.
doconnect 2               # Connect image with subimage outline.
```

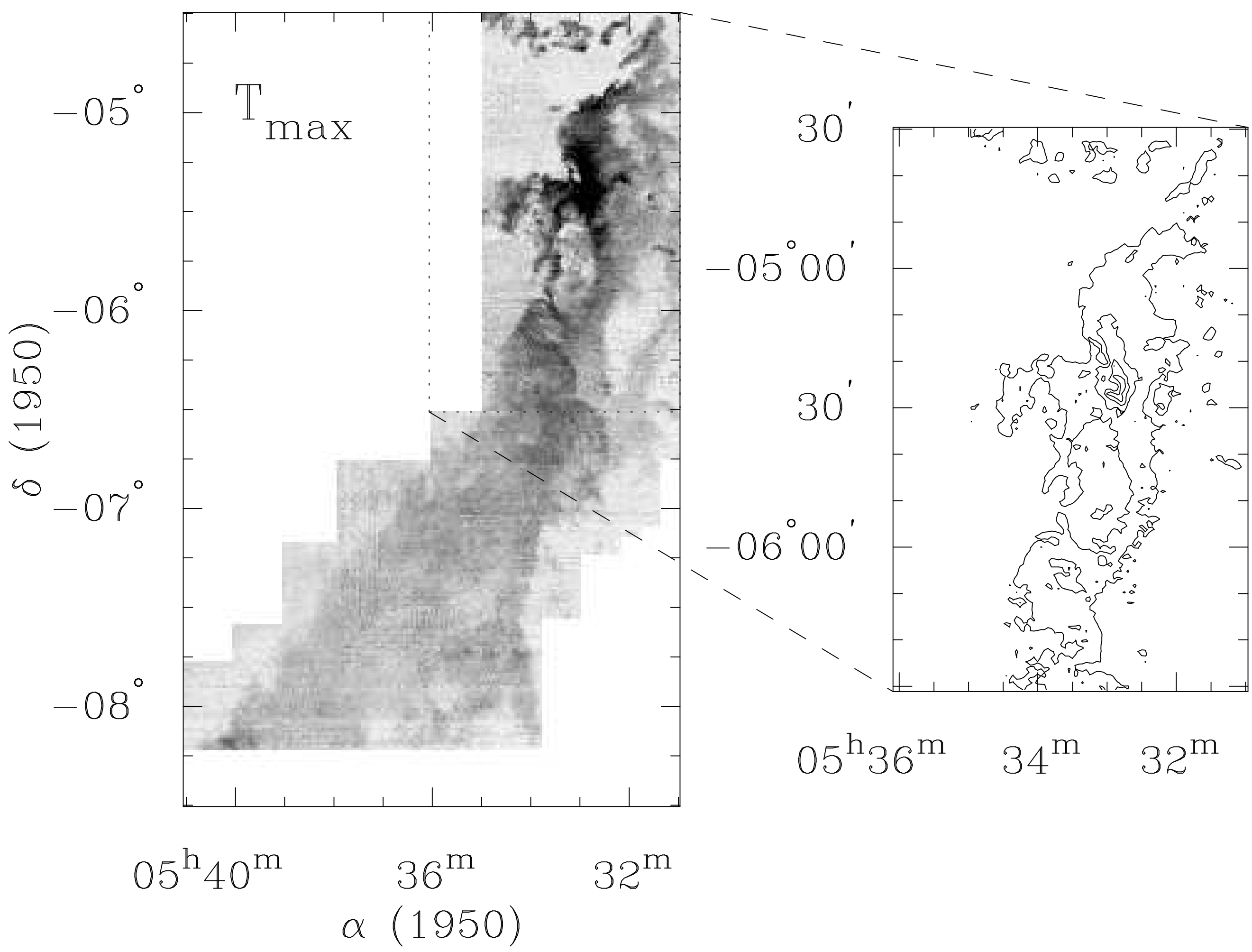

Figure E.11: A plot demonstrating the use of subimages.

E.12 Overlay of multiple images

```
viewport 0.25 0.75 0.2 0.9 # Set viewport size.
expand 1.1                 # Set the character size.
font 2                     # Set the font type.
image orion.fits 1         # Read in plane #1.
winadj 0 nx 0 ny           # Set the x/y scale to be the same.
header rd                  # Read header and set limits.
set \1 x1                  # Save the limits of this rectangle.
set \2 x2
set \3 y1
set \4 y2
halftone 0 20              # Draw the halftone.
image iras60.fits          # Read the second image.
header rd                  # Get the coordinate transfer function.
set \11 x1                 # Save the limits of ...
set \12 x2                 # ...this smaller rectangle.
set \13 y1
set \14 y2
limits \1 \2 \3 \4         # Override 2nd image limits with 1st.
levels 40 60 80 100        # Set the contour levels.
contour                    # Draw the contours.
move \11 \13               # Draw a box around...
draw \12 \13               # ...the limits of...
draw \12 \14               # ...the smaller box.
draw \11 \14
draw \11 \13
box bcnsthz bcnstvdyzf     # Draw a box.
expand 1.5                 # Make the next labels a bit bigger.
mtext T -2 0.15 0 \u12\dCO T\dmax
mtext T -12 0.15 0 60\gmm
expand 1.1                 # Reset the character size.
xlabel \ga (1950)
mtext L 3.5 0.5 0.5 \gd (1950)
```

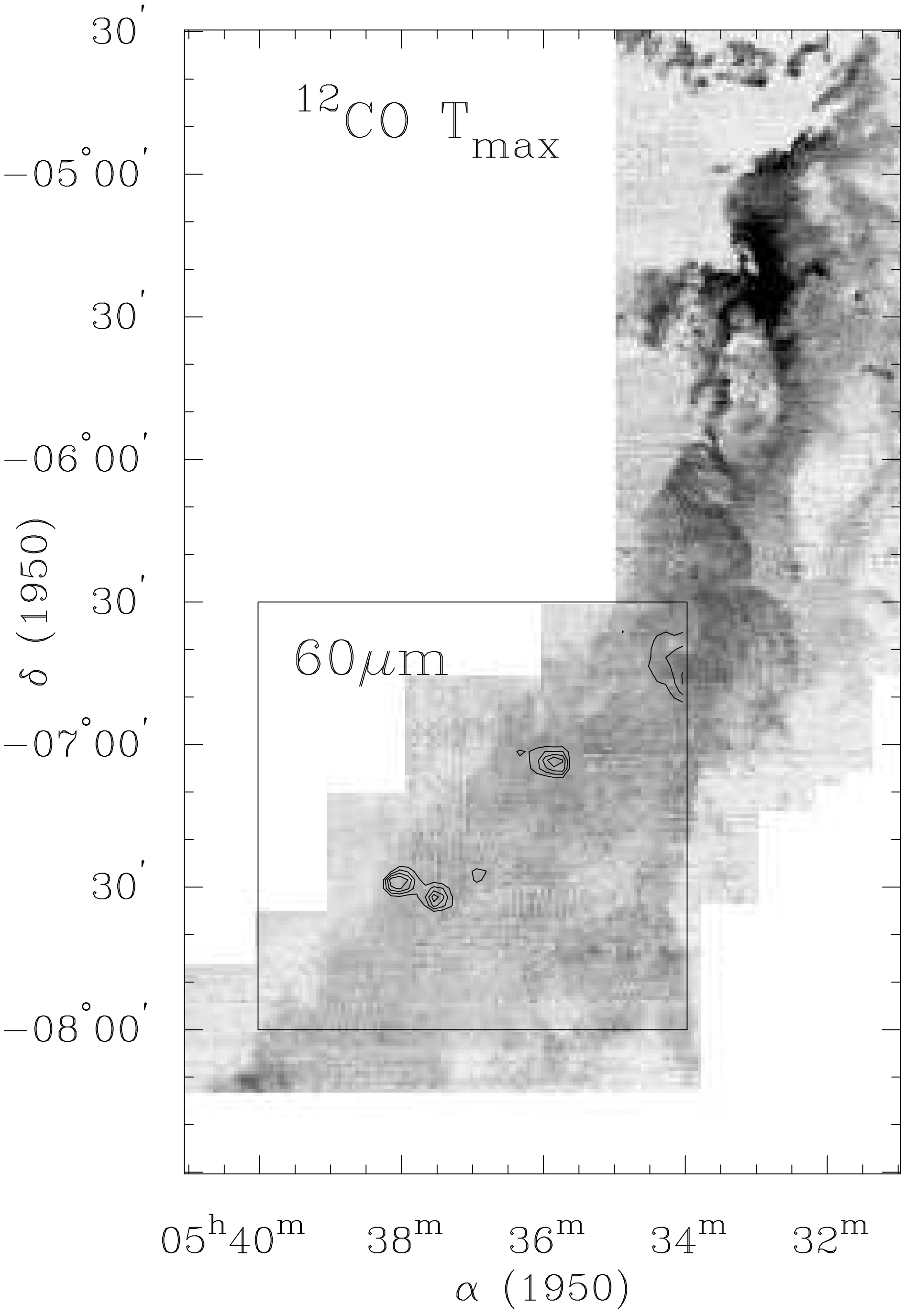

Figure E.12: A plot demonstrating how to overlay different images.

Appendix F

Frequently Asked Questions

This Appendix presents many frequently asked questions about WIP and WIP commands. These questions have come from users from their usage of WIP. Hopefully, this list will address most of the questions that novice users may encounter and present solutions to problems that others may find useful as they become more experienced and attempt more complicated plots. Each topic will first present the problem or question and then a solution will begin following the '◇' character. This list of questions should also be the most volatile of the entire manual as more users submit questions and solutions to various problems.

Lack of Printer Output

When I run my plot file on my screen, everything looks great. But when I send it to the printer, nothing shows up. Here are the commands I execute:

```
% wip
WIP>  input myfile.wip                          # Plots okay on the screen.
WIP>  device myfile.ps/vps                      # Get a printing device.
WIP>  input myfile.wip                          # Play through the commands again.
WIP>  end                                       # All finished.
% lpr myfile.ps
```

The light on the printer flashes for a few minutes and then stops and does not produce a plot. What's wrong?

◇ Whenever you send a plot to a printing device, you should terminate the plot by using either the `hardcopy` command or another call to the `device` command. These commands inform the plot that there is not going to be any further information coming and it should insert device specific closing commands.

In the case of the printing device '/vps' shown above, the commands that are inserted instruct the printer to print and eject the page it has been generating. Without these inserted commands, the plot file is (internally) read by the printer and then ignored because it is never told to eject the page.

An alternative to the example above is to use the `phard` command. This command initiates a (temporarily) new, user specified device and then plays back all the commands issued and then performs the `hardcopy` command to close the plot file. The final operation of the `phard` command is to reinstate the device that was active when this command was called. To illustrate, the commands shown previously, can be replaced with the following:

```
% wip
WIP>  input myfile.wip                  # Plots okay on the screen.
WIP>  phard myfile.ps/vps               # Spool it to the printing device.
WIP>  end                               # All finished.
```

If the `print` user variable has its default value, the plot will automatically be spooled to the printer.

Finally, this plot could also be sent by using the single command line:

```
% wip -x -d myfile.ps/vps myfile.wip
```

See Appendix B for an explanation of the command line options and their meanings.

`erase` Command Troubles

I have a macro defined as follows:

```
WIP>  define doloop                     # $1=Passed counter.
DEFINE>  erase                          # Erase the screen.
DEFINE>  expand 0.5                     # Initialize the expansion.
DEFINE>  set \0 -1                      # Initialize a counter.
DEFINE>  loop 16 mymacro \0 $1          # Call another macro.
DEFINE>  end                            # End of macro definition.
```

which I use to plot multi-page panel plots (`panel` commands appear in `mymacro` macro). When I call this macro as follows:

```
WIP>  device myfile.ps/vps        # Set up the printer file.
WIP>  doloop 1                    # Do the first page.
WIP>  hardcopy                    # Send it to the printer.
WIP>  doloop 2                    # Do the second page.
WIP>  hardcopy                    # Send it to the printer.
WIP>  doloop 3                    # Do the third page.
WIP>  hardcopy                    # Send it to the printer.
```

I get the following error messages:

```
Trouble setting expand value.
Saving expand as 1.000000.
Trouble setting expand value.
Saving expand as 1.000000.
```

I am calling `expand` with a proper argument; what is wrong?

◇ The error message is not so much to do with the macro, per say, as to what it is going on in the macro. Every time the command `hardcopy` is called, the device that was used to do the drawing is closed. However, the next command to be executed is to enter the macro again and the first "graphical command" is to set the expansion size. WIP checks that the requested expand size matches the returned value from PGPLOT. If the two do not match, a warning message is sent (which is the message that was displayed). Now, PGPLOT interprets `erase` commands as a new page request. For printing devices, this is basically the same as the `hardcopy` command except that it does not close the device. Because the macro doloop above has the `erase` command at the top of it, a suggested alternative to the commands above would be (the macros may stay the same):

```
WIP>  device myfile.ps/vps        # Set up the printer file.
WIP>  doloop 1                    # Do the first page.
WIP>  doloop 2                    # Do the second page.
WIP>  doloop 3                    # Do the third page.
WIP>  hardcopy                    # Send it to the printer.
```

The erase command will automatically switch to a new page of paper for hardcopy (printing) devices (and all drawing will appear in the same file) or erase the screen for interactive devices.

Offset Positions

I often need an offset to move away from my current position. Is there (or can there be) a command like "shift" that moves the pointer relative to the previous move or draw end position?

◇ A general offset command would be quite difficult to create due to the possible range and direction of world coordinates. If you want a macro that would do it for a particular orientation and scale (*e.g.* RA/dec plots), consider the following macro definition:

```
WIP>  define doshift                               # $1=delx; $2=dely.
DEFINE>  new offx offy                             # Creates offx and offy.
DEFINE>  set offx cx + $1                          # Set up x-offset.
DEFINE>  set offy cy + $2                          # Set up y-offset.
DEFINE>  move offx offy                            # Move to the offset position.
DEFINE>  free offx offy                            # Release offx and offy.
DEFINE>  end
```

and commands:

```
WIP>  move 5000 50000                              # Some command that sets position.
WIP>  doshift 30 300                               # Shift the position.
WIP>  draw 5100 49000                              # Another set position command.
WIP>  doshift 30 500                               # Shift the position.
```

The `move` or `draw` commands could be folded into the `doshift` macro by adding another 3 arguments, as in:

```
WIP>  define mdshift                               # $1/2=delx/y; $3/4=x/y; $5=m/d.
DEFINE>  new offx offy                             # Creates offx and offy.
DEFINE>  if (0$5 == 0) move $3 $4                  # Move if ($5 == 0).
DEFINE>  if (0$5 == 1) draw $3 $4                  # Draw if ($5 == 1).
DEFINE>  set offx cx + $1                          # Set up x-offset.
DEFINE>  set offy cy + $2                          # Set up y-offset.
DEFINE>  move offx offy                            # Move to the offset position.
DEFINE>  free offx offy                            # Release offx and offy.
DEFINE>  end
```

Then, you can use it as:

```
WIP>  mdshift 30 300 5000 50000 0
WIP>  mdshift 30 500 5100 49000 1
```

Writing Numbers as Text

I have been using WIP to make a macro for doing multi-panel plots with velocity labeling all automatically grabbed from the image header, or at least attempting to do so. I have made good progress but I am having trouble dealing with changing numbers into characters and having the various text writing command believe in text variables. For example if I calculate the velocity of a channel in a macro, how do I load that value into a text variable and get it printed to the screen without doing it one symbol at a time?

◇ This is illustrated in section E.7.

Box Label is Incorrect

I am trying to display an image with Right Ascension and Declination coordinates. When I use the command

```
WIP>  box bcnstz bcnstvz
```

with my image, the y-coordinate values appear incorrect. The `header` command appears to be getting the parameters okay (using the '`show i`' command).

◇ What is happening is that the second argument to the `box` command is missing the `y` character. This character tells the time labeling routine not to break up the label at the 24 hour mark (*i.e.* do not do any day formatting). To correctly display declination, you must include both the `y` and the `z` character in the second argument to `box`. Another alternative is to include in the second argument the `d` option (and the `h` in the first argument) which will add unit superscripts. The `d` option implicitly implies the `y` option. For more details about these and other characters and what duties they perform, see Table 3.4.

FITS Images

I have loaded a FITS image, retrieved the header, and displayed and boxed the image in WIP as follows:

```
WIP>  image myimage.fits
WIP>  header rd
WIP>  halftone
WIP>  box bcnsthz bcnstvdzy
```

No halftone is present and the box is label incorrectly. I have displayed this image in IRAF before making a FITS file with no problems; why can't WIP display it?

◇ Some FITS image writers omit keywords. Make sure that the image header keywords required by WIP are present in the FITS image. In order to properly read header information and use it to display the coordinates with the `box` command, the optional header keywords should also be present. See the FITS image description in Section 5.1 for a list of required and requested keywords. Also see Section E.10 for a sample FITS header.

Fortran Callable

Is it possible to call various WIP routines from a Fortran program?

◇ No. WIP is a collection of routines designed to act as a user interface to the PGPLOT graphics routines. Fortran programmers should call the individual PGPLOT routines directly.

Panel Movements with Images

I am using the `panel` command to display multiple images side by side. I want to have the individual boxes butt up against each other. However, every time I call `panel` with negative arguments (even after calling `submargin 0 0`), there is still a gap between the boxes.

◇ This is dealt with in Section 5.3.

User Interaction with Prompts

This problem concerns writing instructions to users inside the WIP command window. For example, a macro allows the user to select where in a frame to place a label using the cursor. But, if this macro is to be generally usable, it really should write to the command window a message telling the user what the macro is expecting at each cursor prompt. Can this be done simply in WIP?

◇ This is quite easy to do with the `echo` command. The following simple macro shows how to prompt just before a cursor call.

```
WIP>    define doprompt                               # $1-$0 used for the label.
DEFINE> echo "Find the desired position"              # Prompt the user.
DEFINE> echo "and then hit any key."
DEFINE> cursor                                        # Move to the desired position.
DEFINE>                                               # Put the label at the cursor position.
DEFINE> label $1 $2 $3 $4 $5 $6 $7 $8 $9 $0
DEFINE> end                                           # End of macro definition.
```

Overlapping Y-Axis Labels

When I chose vertical tick labels along the y-axis of the box, the y-axis text label overlaps them (especially if the numbers at a major tick interval is 3 or more digits).

◇ Vertical labeling is a problem for large value coordinates (like declination). The default `ylabel` command is actually an alias for the `mtext` command. In fact, the following two commands are identical:

```
DEFINE> mtext L 2.2 0.5 0.5 Yaxis label string
DEFINE> ylabel Yaxis label string
```

To use labels along the y-axis without overlapping the tick labels, use the `mtext` command with a larger displacement (*i.e.* use 3 or 4 instead of 2.2). Also be careful to make sure that the left viewport value is large enough that it does not clip the tick labels.

Logarithmic Error Bars

I want to plot error bars on a log plot. Is there a simple way to do this? I've tried using the `logarithm err` command but can't seem to get it to work. Any suggestions?

◇ Logarithmic plots with error bars is a bit tricky because the values in the **ERR** array are offset values rather than the actual error values. To solve this, either modify the data outside of WIP or write a macro to do the math internally. Examples of both are shown below. In both examples, it is assumed that the **X**, **Y**, and **ERR** data in `data.lis` are found in columns 1, 2, and 3, respectively. For UNIX users, the `awk` command:

```
% awk '{print $0,($2+$3)/$2,$2/($2-$3)}' data.lis > d2.lis
```

will generate a new file called `d2.lis` which has the same contents as the old file plus new values in the fourth and fifth columns. Using `d2.lis`, the following WIP commands should work:

```
WIP>  data d2.lis
WIP>  xcolumn 1
WIP>  ycolumn 2
WIP>  ecolumn 4
WIP>  logarithm y
WIP>  logarithm err
WIP>  points
WIP>  errorbar 2
WIP>  ecolumn 5
WIP>  logarithm err
WIP>  errorbar 4
```

or, within WIP, one can use:

```
WIP>  define eplus
DEFINE>  set $1 $1 + 1
DEFINE>  set err[$1] log((y[$1] + err[$1]) / y[$1])
DEFINE>  end
WIP>  define eminus
DEFINE>  set $1 $1 + 1
DEFINE>  set err[$1] log(y[$1] / (y[$1] - err[$1]))
DEFINE>  end
WIP>  data data.lis
WIP>  xcolumn 1
WIP>  ycolumn 2
WIP>  ecolumn 3
WIP>  set \0 0
```

```
WIP> loop npts(err) eplus \0
WIP> logarithm y
WIP> points
WIP> errorbar 2
WIP> ycolumn 2
WIP> ecolumn 3
WIP> set \0 0
WIP> loop npts(err) eminus \0
WIP> logarithm y
WIP> points
WIP> errorbar 4
```

Aspect Ratio Without Images

I have tried to plot the spatial distribution of stars (*i.e.* dots on Right Ascension – Declination plot) but I can't get the aspect ratio of the map right. The star positions are read in from a text file. I know that `winadj` works for images but when I use it for a simple plot with individual points, it didn't seem to understand that the Right Ascension unit is 15 times smaller than the Declination unit. Am I missing something?

◇ In order to get the spatial distribution correct, you will have to help WIP out. You are correct that WIP doesn't know that Right Ascension is about 15 times smaller than the Declination axis. This is because the world coordinate system has no units associated with it. The best solution is use `winadj` with the spatial extent arguments provided in the same units. For example, if your plot runs in Right Ascension from 5^h37^m to 5^h29^m and Declination from $-5°$ to $-8°$, you might use the following:

```
WIP> set \1 (5 * 3600) + (37 * 60)
WIP> set \2 (5 * 3600) + (29 * 60)
WIP> set \3 -8 * 3600
WIP> set \4 -5 * 3600
WIP> set \5 (\1 - \2) * 15
WIP> set \6 \4 - \3
WIP> winadj 0 \5 0 \6
WIP> limits \1 \2 \3 \4
WIP> box bcnsthz bcnstvdzy
```

This will generate the proper 2:3 aspect ratio and also label the coordinates correctly.

Strange Hardcopy Output

Suppose I have two WIP plotting command files and do the following:

```
WIP>  input first.wip         # Plots out OK on the workstation.
WIP>  phard /ps               # Plots out OK on the PostScript printer.
WIP>  input second.wip        # Plots out OK on the workstation.
WIP>  phard /ps               # Plots both first and second figures!
```

Why does the second PostScript file contain both figures?

◇ What you are actually getting out in the second PostScript file is the combination of both `input` commands. This is because the `input` command stays on the playback stack and the `phard` command plays back all commands on the stack. Hence, the second `phard` command will call `input first.wip` followed by `input second.wip`.

Changing the Number of Digits Displayed

I wish to do something quite simple: have a number like 1.234 (which is generated within a WIP macro) be part of a plot label. Is this possible? If so, there a way to control the number of significant figures for this purpose?

◇ The answer to both questions is yes. Any label with the construct `\[`*x*`]` will actually be plotted as if the current value of the user variable x was used in the command. See Section 3.6 and Table 3.5 for more details on how to insert user variables in a text command.

To control the number of significant digits, change the value of the user variable `nsig`. By default, it is set to 3; specifying a larger value will cause more digits to be displayed.

Leaving Loops Early

Is it possible to break out of a `loop` command early (*i.e.* before the total number of iterations are reached) based on some condition within that macro?

◇ The only way to do this is (not very elegant) to add a conditional to the macro and have the macro exit when the condition is met. This will still mean the macro is called the requested number of times, but each time it is called with the counter greater than the conditional value, it will just exit. A trivial example follows that will only print out the first ten numbers even though the loop is being executed twenty times:

```
WIP>  define dummy                # $1=counter.
DEFINE>  set $1 $1 + 1            # Increment the counter.
DEFINE>  if ($1 > 10) end         # Exit if condition is true.
DEFINE>  echo $1                  # Write out the current value.
DEFINE>  end                      # End the macro definition.
WIP>  set \0 0                    # Initialize the counter.
WIP>  loop 20 dummy \0            # Loop through the macro.
```

Index