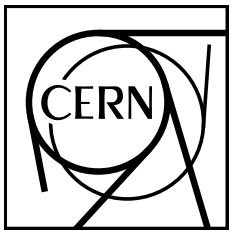


CERN Program Library Long Writeup Q121

PAW

Physics Analysis Workstation

An Introductory Tutorial

Application Software Group
Computing and Networks Division

CERN Geneva, Switzerland

Copyright Notice

PAW – Physics Analysis Workstation

CERN Program Library entry **Q121**

© Copyright CERN, Geneva 1995

Copyright and any other appropriate legal protection of these computer programs and associated documentation reserved in all countries of the world.

These programs or documentation may not be reproduced by any method without prior written consent of the Director-General of CERN or his delegate.

Permission for the usage of any programs described herein is granted apriori to those scientific institutes associated with the CERN experimental program or with whom CERN has concluded a scientific collaboration agreement.

Requests for information should be addressed to:

CERN Program Library Office
CERN-CN Division
CH-1211 Geneva 23
Switzerland
Tel. +41 22 767 4951
Fax. +41 22 767 8630
Email: cernlib@cern.ch

Trademark notice: All trademarks appearing in this guide are acknowledged as such.

Contact Person: Olivier Couet /CN (olivier.couet@cern.ch)

Technical Consultant: Michel Goossens /CN (michel.goossens@cern.ch)

Edition – February 1995

About this guide

Preliminary remarks

In this tutorial examples are in `monotype face` and strings to be input by the user are `underlined`. In the index the page where a command is defined is in **bold**, page numbers where a routine is referenced are in normal type.

Acknowledgements

The authors of PAW would like to thank all their colleagues who, by their continuous interest and encouragement, have given them the necessary input to provide a modern and easy to use data analysis and presentation system.

Vladimir Berezhnoi (IHEP, Serpukhov, USSR), the main author of the Fortran interpreter COMIS, provided one of the essential components of our system. Nicole Cremel has collaborated to the first versions of HPLOT. The PAW/HBOOK to MINUIT interface has been implemented in collaboration with Eliane Lessner (FNAL, USA) and Fred James. Jim Loken (Oxford, UK) has been our expert on VAX global sections. David Foster, Frederic Hemmer, Catherine Magnin and Ben Segal have contributed to the development of the PAW TCP/IP interface. Ben has also largely contibuted to the TELNETG and 3270G systems. Per Scharff-Hansen and Johannes Raab from the OPAL collaboration have made possible the interface with the OS9 system. Harald Johnstad (SSC, USA) and Lee Roberts (FNAL, USA) have contributed to the debugging phases of PAW in the DI3000 and DECGKS environments. Initial implementations of PAW on MVS/TSO, the Sun and the DEC Station 3100 were made by Alain Michalon (Strasbourg, France), François Marabelle (Saclay, France) and Walter Bruckner (Heidelberg, FRG), respectively. Lionel Cons has contributed to the implementation of the selection mechanisms for Ntuples. Isabelle Moulinier (Paris) has been working, as a summer student, on various improvements in the HIGZ/HPLOT packages.

Related Manuals

This document can be complemented by the following manuals:

- COMIS, Compilation and Interpretation System [1]
- HBOOK User Guide — Version 4 [2]
- HIGZ-HPLOT — High level Interface to Graphics and ZEBRA and HPLOT User Guide [3]
- KUIP — Kit for a User Interface Package [4]
- MINUIT — Function Minimization and Error Analysis [5]
- PAW — PAW Reference Guide [6]
- ZEBRA — Data Structure Management System [7]

This document has been produced using LaTeX [8] with the `cernman` style option, developed at CERN. All pictures shown are produced with PAW and are included in PostScript [9] format in the manual.

A gzipped PostScript file `paw.ps.gz`, containing a complete printable version of this manual, can be obtained by anonymous ftp as follows (commands to be typed by the user are underlined):

```
ftp asisftp.cern.ch
Connected to asis00.cern.ch.
220 asis00 FTP server (Version wu-2.4(2)...) ready.
Name (asisftp:username): ftp
Password: your_mailaddress
230 Guest login ok, access restrictions apply.
ftp> cd cernlib/doc/ps.dir
ftp> get paw.ps.gz    (type get paw.ps for the uncompressed version)
ftp> quit
```

Table of Contents

I PAW – Step by step **1**

1 A few words on PAW **3**

1.1 A short history 3
1.2 What is PAW? 3
1.3 What Can You Do with PAW? 3
1.4 A User's View of PAW 5
1.5 Fundamental Objects of PAW 7
1.6 The Component Subsystems of PAW 9
1.7 A PAW Glossary 12

2 General principles **15**
2.1 Access to PAW . . . 15
2.2 Initialising PAW . . . 17
2.3 PAW++ . . . 18
2.4 Command structure . . . 22
2.5 Getting help . . . 23
2.6 Special symbols for PAW . . . 25
2.7 PAW entities and their related commands . . . 25

3 PAW by Examples **28**
3.1 Basic Principles . . . 30
3.2 Starting the PAW Tutorial . . . 31
3.3 Vectors—Tutorial . . . 32
3.4 Vectors—Examples . . . 36
3.5 Function drawing—Examples . . . 56
3.6 Histograms—Tutorial . . . 68
3.7 Histograms—Examples . . . 80
3.8 Ntuples—Tutorial . . . 116
3.9 Ntuples—Examples . . . 122
3.10 SIGMA—Examples . . . 146
3.11 Pictures and PostScript . . . 152

II PAW - Commands and Concepts **167**

4 User interface - KUIP **169**
4.1 Command line syntax . . . 169
4.2 Aliases . . . 182
4.3 System functions . . . 185
4.4 Vectors . . . 192
4.5 Expressions . . . 193
4.6 Macros . . . 198
4.7 Motif mode . . . 214
4.8 Nitty-Gritty . . . 231

5 Vectors **236**
5.1 Vector creation and filling . . . 236
5.2 Vector addressing . . . 237
5.3 Vector arithmetic operations . . . 237
5.4 Vector arithmetic operations using SIGMA . . . 237
5.5 Using KUIP vectors in a COMIS routine . . . 238
5.6 Usage of vectors with other PAW objects . . . 238
5.7 Graphical output of vectors . . . 238
5.8 Fitting the contents of a vector . . . 238

6 SIGMA **239**
6.1 Access to SIGMA . . . 239
6.2 Vector arithmetic operations using SIGMA . . . 240
6.3 SIGMA functions . . . 241
6.4 Available library functions . . . 249

7 HBOOK **251**
7.1 Introduction . . . 251
7.2 Basic ideas . . . 252
7.3 HBOOK batch as the first step of the analysis . . . 254
7.4 Using PAW to analyse data . . . 258
7.5 Ntuples: A closer look . . . 260
7.6 Fitting with PAW/HBOOK/MINUIT . . . 270
7.7 Doing more with Minuit . . . 280

8 Graphics (HIGZ and HPLOT) **284**
8.1 HPLOT, HIGZ and local graphics package . . . 284
8.2 The metafiles . . . 285
8.3 The HIGZ pictures . . . 286
8.4 Setting attributes . . . 293
8.5 More on labels . . . 299
8.6 Colour, line width, and fill area in HPLOT . . . 300
8.7 Information about histograms . . . 304
8.8 Text drawing . . . 309
8.9 The HIGZ graphics editor . . . 320

9 Distributed PAW **321**
9.1 TELNETG and 3270G . . . 321
9.2 ZFTP . . . 324
9.3 Access to remote files from a PAW session . . . 324
9.4 Using PAW as a presenter on VMS systems (global section) . . . 326
9.5 Using PAW as a presenter on OS9 systems . . . 327

III PAW - Reference section **329**

10 KUIP **331**
10.1 ALIAS . . . 336
10.2 SET˙SHOW . . . 338

11 MACRO **348**
11.1 GLOBAL 350
11.2 SYNTAX 351

12 VECTOR **360**
12.1 OPERATIONS 366

13 HISTOGRAM **368**
13.1 2D˙PLOT 373
13.2 CREATE 375
13.3 HIO 378
13.4 OPERATIONS 380
13.5 GET˙VECT 385
13.6 PUT˙VECT 386
13.7 SET 387

14 FUNCTION **389**

15 NTUPLE **393**

16 GRAPHICS **406**
16.1 MISC 412
16.2 VIEWING 413
16.3 PRIMITIVES 415
16.4 ATTRIBUTES 429
16.5 HPLOT 430

17 PICTURE **434**

18 ZEBRA **439**
18.1 RZ 439
18.2 FZ 441
18.3 DZ 442

19 FORTRAN **444**

20 NETWORK **448**
20.1 PIAF 448

21 OBSOLETE **452**
21.1 GRAPHICS 452

A PAW tabular overview **454**

Bibliography **460**

Index **461**

Part I

PAW – Step by step

Chapter 1: A few words on PAW

1.1 A short history

Personal workstations equipped with a high resolution bitmap display, a speed of several tens of MIPS, with at least 20-30 Mbytes of main memory and 1 Gbyte of local disk space (e.g. DEC, HP-700, IBM RS6000, Sun Sparc and Silicon Graphics workstations) are now widely available at an affordable price for individual users. In order to exploit the full functionality of these workstations, at the beginning of 1986 the **P**hysics **A**nalysis **W**orkstation project **PAW** was launched at CERN. The first public release of the system was made at the beginning of 1988. At present the system runs on most of the computer systems used in the High Energy Physics (HEP) community (Mainframes, Workstations, PC's) but its full functionality is best exploited on personal workstations. In addition to its powerful data analysis, particular emphasis has been put on the quality of the user interface and of the graphical presentation.

1.2 What is PAW?

PAW is an interactive utility for visualizing experimental data on a computer graphics display. It may be run in batch mode if desired for very large and time consuming data analyses; typically, however, the user will decide on an analysis procedure interactively before running a batch job.

PAW combines a handful of CERN High Energy Physics Library systems that may also used individually in software that processes and displays data. The purpose of PAW is to provide many common analysis and display procedures that would be duplicated needlessly by individual programmers, to supply a flexible way to invoke these common procedures, and yet also to allow user customization where necessary.

Thus, PAW's strong point is that it provides quick access to many facilities in the CERN library. One of its limitations is that these libraries were not designed from scratch to work together, so that a PAW user must eventually become somewhat familiar with many dissimilar subsystems in order to make effective use of PAW's more complex capabilities. As PAW evolves in the direction of more sophisticated interactive graphics interfaces and object-oriented interaction styles, the hope is that such limitations will gradually become less visible to the user.

PAW is most effective when it is run on a powerful computer workstation with substantial memory, rapid access to a large amount of disk storage, and graphics support such as a large color screen and a three-button mouse. If the network traffic can be tolerated, PAW can be run remotely over the network from a large, multiuser client machine to more economical servers such as an X-terminal. In case such facilities are unavailable, substantial effort has been made to ensure that PAW can be used also in noninteractive or batch mode from mainframes or minicomputers using ASCII terminals.

1.3 What Can You Do with PAW?

PAW can do a wide variety of tasks relevant to analyzing and understanding physical data, which are typically statistical distributions of measured events. Below we list what are probably the most frequent and best-adapted applications of PAW; the list is not intended to be exhaustive, for it is obviously possible to use PAW's flexibility to do a huge number of things, some more difficult to achieve than others within the given structure.

Typical PAW Applications:

- **Plot a Vector of Data Fields for a List of Events.** A set of raw data is typically processed by the user's own software to give a set of physical quantities, such as momenta, energies, particle identities, and so on, for each event. When this digested data is saved on a file as an Ntuple, it may be read and manipulated directly from PAW. Options for plotting Ntuples include the following:

  - *One Variable.* If a plot of a one variable from the data set is requested, a histogram showing the statistical distribution of the values from all the events is automatically created. Individual events are not plotted, but appear only as a contribution to the corresponding histogram bin.
  - *Two or Three Variables.* If a plot of two or three variables from the data set is requested, no histogram is created, but a 2D or 3D scatter plot showing a point or marker for each distinct event is produced.
  - *Four Variables.* If a plot of four variables is requested, a 3D scatter plot of the first three variables is produced, and a color map is assigned to the fourth variable; the displayed color of the individual data points in the 3D scatter plot indicates the approximate value of the fourth variable.
  - *More than Four Variables.* More than four variables can be plotted but it is up to the user to customize the system in order to assign the additional variables to graphics attributes like the size or the shape (type) of the markers.
  - *Vector Functions of Variables.* PAW allows the user to define arbitrary vector functions of the original variables in an Ntuple, and to plot those instead of the bare variables. Thus one can easily plot something like $\sqrt{(P_x^2 + P_y^2)}$ if P_x and P_y are original variables in the data without having to add a new data field to the Ntuple at the time of its creation.
  - *Selection Functions (Cuts).* PAW does not require you to use every event in your data set. Several methods are provided to define Boolean functions of the variables themselves that pick out subsets of the events to be included in a plot.
  - *Plot presentation options.* The PAW user can set a variety of options to customize the format and appearance of the plots.

- **Histogram of a Vector of Variables for a List of Events.** Often one is more interested in the statistical distribution of a vector of variables (or vector functions of the variables) than in the variables themselves. PAW provides utilities for defining the desired limits and bin characteristics of a histogram and accumulating the bin counts by scanning through a list of events. The following are some of the features available for the creation of histograms:

  - *One Dimensional Histograms.* Any single variable can be analyzed using a one-dimensional histogram that shows how many events lie in each bin. This is basically equivalent to the single-variable data plotting application except that it is easier to specify personalized features of the display format. A variety of features allow the user to slice and project a 2D scatter plot and make a 1D histogram from the resulting projection.
  - *Two-Dimensional Histograms.* The distribution of any pair of variables for a set of events can be accumulated into a 2D histogram and plotted in a various of ways to show the resulting surface.
  - *Three-Dimensional Histograms.* Will be supported soon.

  - *Vector Functions of Variables.* User-defined functions of variables in each event can be used to define the histogram, just as for an Ntuple plot.
  - *Selection Functions (Cuts).* Events may also be included or excluded by invoking Boolean selection functions that are arbitrary functions of the variables of a given event.
  - *Event Weights.* PAW allows the user to include a multiplicative statistical bias for each event which is a scalar function of the available variables. This permits the user to correct for known statistical biases in the data when making histograms of event distributions.
  - *Histogram Presentation Options.* Virtually every aspect of the appearance of a histogram can be controlled by the user. Axis labels, tick marks, titles, colors, fonts, and so on, are specified by a large family of options. A particular set of options may be thought of as a "style" for presenting the data in a histogram; "styles" are in the process of becoming a formal part of PAW to aid the user in making graphics that have a standard pleasing appearance.

- **Fit a Function to a Histogram.** Once a histogram is defined, the user may fit the resulting shape with one of a family of standard functions, or with a custom-designed function. The parameters of the fit are returned in user-accessible form. Fitted functions of one variable may be attached to a 1D histogram and plotted with it. The capability of associating fits to higher dimensional histograms and overlaying their representations on the histogram is in the process of being added to PAW.

  The fitting process in PAW is normally carried out by the MINUIT library. To user this package effectively, users must typically supply data with reasonable numerical ranges and give reasonable initial conditions for the fit before passing the task to the automated procedure.

- **Annotate and Print Graphics.** A typical objective of a PAW user is to examine, manipulate, and display the properties of a body of experimental data, and then to prepare a graph of the results for use in a report, presentation, or publication. PAW includes for convenience a family of graphics primitives and procedures that may be used to annotate and customize graphics for such purposes. In addition, any graphics display presented on the screen can be converted to a PostScript file for black-and-white or color printing, or for direct inclusion in a manuscript.

1.4 A User's View of PAW

In order to take advantage of PAW, the user must first have an understanding of its basic structure. Below we explain the fundamental ways in which PAW and the user interact.

Initialization. PAW may be invoked in a variety of ways, depending on the user's specific computer system; these are described in the following chapter. As PAW starts, it prompts the user to select an interaction mode (or non-interactive mode) and window size and type (if interactive). The available window sizes and positions are specified in the user file `"higz_windows.dat"`. User-specific intializations are specified in the file `"pawlogon.kumac"`.

Command Mode Interface. The most basic interface is the **KUIP "command mode" interface.** KUIP provides a basic syntax for commands that are parsed and passed on to the PAW application routines to perform specific tasks. Among the basic features of KUIP with which the user interacts are the following:

- *Command Entry.* Any unique partially entered command is interpreted as a fully entered command. KUIP responds to an ambiguous command by listing the possible alternatives. On Unix systems,

individual command lines can be edited in place using individual control keystrokes similar to those of the emacs editor, or the `bash` or `tcsh` Unix command shells. On other systems, a command line that is in error can only be revised after it is entered, using the VAX/VMS editor "EDT" style text line editing language.

- *Parameters.* Parameters are entered after the basic command on the same line and are separated by spaces. If a parameter has embedded blanks, it must be it must be put between quotes. An exclamation point `(!)` can be used to keep the default parameters in a sequence when only a later parameter is being changed. If an underscore `(_)` is the last character on a line, the command may be continued on the next line; no spaces are allowed in the middle of continued parameter fields.
- *On-Line Assistance.* The `"usage"` and `"help"` commands can be used to get a short or verbose description of parameters and features of any command.
- *Command History.* A command history is kept both in memory for interactive inspection and on a disk file. The command history file can be recovered and used to reconstruct a set of actions carried out interactively.
- *Aliases.* Allow the abbreviation of partial or complete command sequences.
- *Macros.* A text file containing PAW commands and flow control statements.

KUIP/MOTIF Interface. If the user's workstation supports the OSF/Motif windowing system, PAW can be started in the KUIP/MOTIF mode: the executable module to be run in that case is called PAW++. However, a small text panel and a command history panel keep track of individual actions, and permit entry and recall of typed commands similar to the command mode interface.

The basic features of this interface are:

- *Pull-Down Menu "Commands".* Each PAW command (that can be given in input) has a corresponding item in a hierarchical pull-down menu (entry "Commands"). Commands that require arguments cause a parameter-entry dialog box to appear; when the arguments are entered and command execution requested (button "OK" or "Execute"), the command is executed as though typed from the command mode interface.
- *Action Panel(s).* A user may have a family of frequently executed macros or commands assigned to specific buttons on the action panel(s). These panels are totally user definable.
- *Object Browser.* All the objects known in PAW (Histograms, Ntuples, Vectors etc...) can be manipulated via icons and pull-down menus in the "Object Browser". This is in many ways similar to the well-known browsers in the PC/MAC utilities or the visual tools on some workstations.
- *Direct Graphics Interaction.* One can click in the graphics area and identify automatically which object has been selected. A pop-up menu appears with a list of possible actions on this object.

Graphics Output Window. The graphics image produced by PAW commands, regardless of the command interface, appears on a separate graphics output window. The actual size and position of this window on the screen is controlled by a list of numbers of the form `x-upper-left y-upper-left x-width y-height` in the user file `higz_windows.dat`. The width and height of the drawing area within this window are subject to additional user control, and the user can specify "zones," which are essentially ways of

dividing the window into panes to allow simultaneous display of more than one plot. Some facilities are available in the current version of PAW to use the mouse to retrieve data such as the height of a histogram bin.

1.5 Fundamental Objects of PAW

PAW is implicitly based on a family of fundamental objects (see figure 1.1). Each PAW command performs an action that either produces another object or produces a "side-effect" such as a printed message or graphics display that is not saved anywhere as a data structure. Some commands do both, and some may or may not produce a PAW data structure depending on the settings of global PAW parameters. In this section, we describe the basic objects that the user needs to keep in mind when dealing with PAW. The reader should perhaps note that the PAW commands themselves do not necessarily reflect the nature of PAW objects as clearly as they might, while the MOTIF interactive graphics interface in fact displays distinct icons for most of the object types listed below.

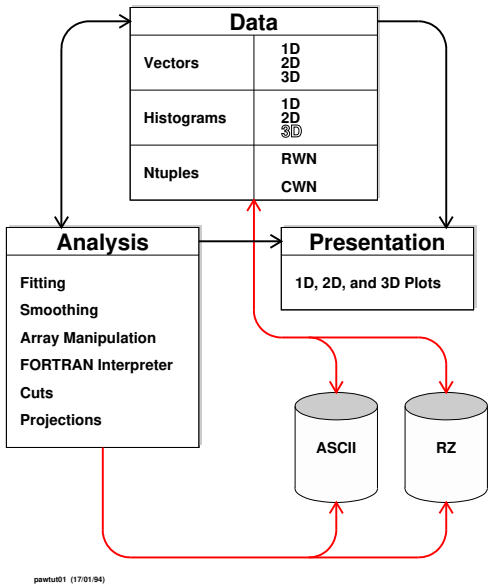


Figure 1.1: PAW's fundamental "data" objects

Objects:

- **1D Histograms.** A histogram is the basic statistical analysis tool of PAW. Histograms are created ("booked") by choosing the basic characteristics of their bins, variables, and perhaps customized display parameters; numbers are entered into the histogram bins from an Ntuple (the histogram is "filled") by selecting the desired events, weights, and variable transformations to be used while counts are accumulated in the bins. Functional forms are frequently fit to the resulting histograms and stored with them. Thus a fit as an object is normally associated directly with a histogram, although it may be considered separately.

- **2D Histograms.** 2D (and higher-dimensional) histograms are logical generalizations of 1D histograms. 2D histograms, for example, are viewable as the result of counting the points in a the sections of a rectangular grid overlaid on a scatter plot of two variables. Higher-dimensional histograms can also be fitted, and support for associating the results of a fit to a higher-dimensional histogram is currently being incorporated in PAW.

- **Ntuples.** An Ntuple is the basic type of data used in PAW. It consists of a list of identical data structures, one for each event. Typically, an Ntuple is made available to PAW by opening a ZEBRA file; this file, as created by HBOOK, contains one or more Ntuples and possibly also ZEBRA logical directories, which may store a hierarchy of Ntuples. A storage area for an Ntuple may be created directly using `ntuple/create`; data may then be stored in the allocated space using the `ntuple/loop` or `ntuple/read` commands. Other commands merge Ntuples into larger Ntuples, project vector functions of the Ntuple variables into histograms, and plot selected subsets of events.

- **Cuts.** A cut is a Boolean function of Ntuple variables. Cuts are used to select subsets of events in an Ntuple when creating histograms and ploting variables.

- **Masks.** Masks are separate files that are logically identical to a set of boolean variables added on the end of an Ntuple's data structure. A mask is constructed using the Boolean result of applying a cut to an event set. A mask is useful only for efficiency; the effect of a mask is identical to that of the cut that produced it.

- **Vectors.** PAW provides the facilities to store vectors of integer or real data. These vectors, or rather arrays with up to 3 index dimensions, can be manipulated with a set of dedicated commands. Furthermore they are interfaced to the array manipulation package SIGMA and to the Fortran interpreter COMIS. They provide a convenient and easy way to analyse small data sets stored in ASCII files.

- **Styles.** A "style" is a set of variables that control the appearance of PAW plots. Commands of the form `igset parameter value` determine fundamental characteristics of lines, axis format, text, and so on. Commands of the form `option attribute` choose particular plotting options such as logarithmic/linear, bar-chart/scatter-plot, and statistics display. Commands of the form `set parameter value` control a vast set of numerical format parameters used to control plotting. While the "style" object will eventually become a formal part of PAW, a "style" can be constructed by the user in the form of a macro file that resets all parameters back to their defaults and then sets the desired customizations.

- **Metafile.** In normal interactive usage, images created on the screen correspond to no persistent data structure. If one wishes to create a savable graphics object, the user establishes a *metafile*; as a graphics image is being drawn, each command is then saved in a text file in coded form that allows the image to be duplicated by other systems. PostScript format metafiles are especially useful because they can be directly printed on most printers; furthermore, the printed quality of graphics objects such as fonts can be of much higher quality than the original screen image.

- **Pictures.** Metafiles describing very complex graphics objects can be extremely lengthy, and therefore inefficient in terms of storage and the time needed to redraw the image. A *picture* is an exact copy of the screen image, and so its storage and redisplay time are independent of complexity. Pictures are also intensively used for object picking in the Motif version of PAW.

- **ZEBRA(RZ) Logical Directories.** In a single PAW session, the user may work simultaneously with many Ntuples, histograms, and hierarchies of Ntuple and histograms. However, this is not accomplished using the native operating system's file handler. Instead, the user works with a set of objects that are *similar* to a file system, but are instead managed by the ZEBRA RZ package. This can be somewhat confusing because a single operating system file created by RZ can contain an entire hierarchy of ZEBRA logical directories; furthermore, sections of internal memory can also be organized as ZEBRA logical directories to receive newly-created PAW objects that are not written to files. A set of commands `CDIR`, `LDIR`, and `MDIR` are the basic utilities for walking through a set of ZEBRA logical directories of PAW objects; Each set of directories contained in an actual file corresponds to a logical unit number, and the root of the tree is usually of the form `//LUNx`; the PAW objects and logical directories stored in internal memory have the root `//PAWC`. A macro is a set of command lines stored in a file, which can be created or modified with any text editor. In addition to all the PAW commands, special macro flow control statements are also available.
- **Operating System File Directories.** Many different ZEBRA files, some with logically equivalent Ntuples and histograms, can be arranged in the user's operating system file directories. Thus one must also keep clearly in mind the operating system file directories and their correspondence to the ZEBRA logical directories containing data that one wishes to work with. In many ways, the operating system file system is also a type of "object" that forms an essential part of the user's mental picture of the system.

1.6 The Component Subsystems of PAW

The PAW system combines different tools and packages, which can also be used independently and some of which have already a long history behind them (e.g. HBOOK and HPLOT, SIGMA, COMIS, MINUIT). Figure 1.2 shows the various components of PAW.

1.6.1 KUIP - The user interface package

The purpose of KUIP (**K**it for a **U**ser **I**nterface **P**ackage) is to handle the dialogue between the user and the application program (PAW in our case). It parses the commands input into the system, verifies them for correctness and then hands over control to the relevant action routines.

Commands are grouped in a tree structure and they can be **abbreviated** to their shortest unambiguous form. If an ambiguous command is typed, then KUIP responds by showing all the possibilities. **Aliases** allow the user to abbreviate part or the whole of commonly used command and parameters. A sequence of PAW commands can be stored in a text file and, combined with flow control statements, form a powerful **macro** facility. With the help of **parameters**, whose values can be passed to the macros, general and adaptable task solving procedures can be developed.

The user has the choice between different dialogue styles ranging from the conventional command line interface to a high-level windowed environment based on OSF/Motif . In order to save typing, **default values**, providing reasonable settings, can be used for most parameters of a command. A **history file**, containing the `n` most recently entered commands, is automatically kept by KUIP and can be inspected, copied or re-entered at any time. The history file of the last PAW session is also kept on disk.

1.6.2 HBOOK and HPLOT - The histograming and plotting packages

HBOOK and its graphics interface HPLOT are libraries of FORTRAN callable subroutines which have been in use for many years. They provide the following functionality:

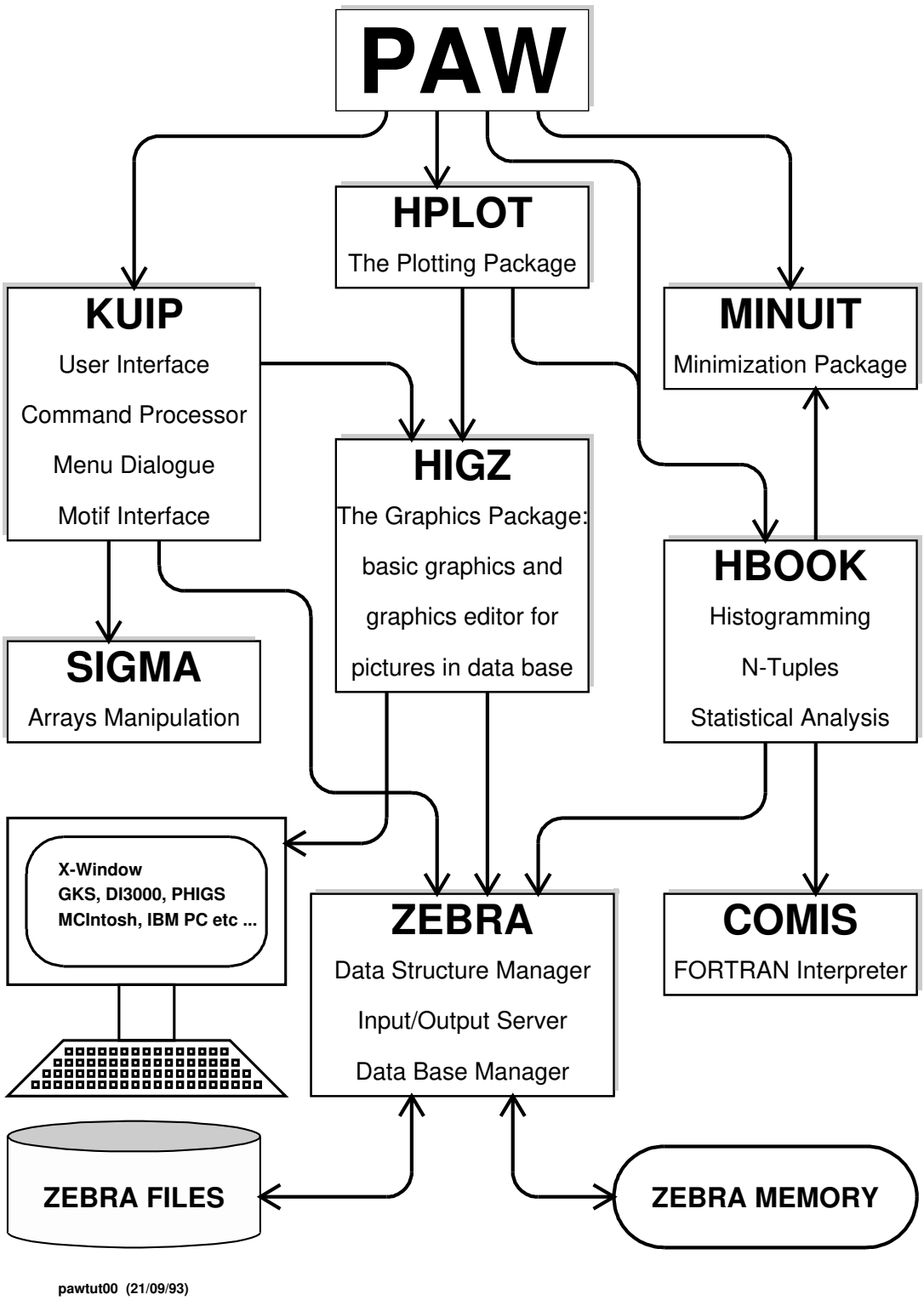


Figure 1.2: PAW and its components

- One- and two-dimensional histograms and Ntuples
- Projections and slices of two-dimensional histograms and Ntuples
- Complete control (input and output) of the histogram contents
- Operations and comparison of histograms
- Minimization and parameterization tools
- Random number generation
- Histograms and Ntuples structured in memory (directories)
- Histograms and Ntuples saved onto direct access ZEBRA files
- Wide range of graphics options:
  - Contour histograms, bar chart, shaded histograms, error bars, colour
  - Smoothed curves and surfaces
  - Scatter, lego, contour and surface plots
  - Automatic windowing
  - Graphics input

1.6.3 HIGZ - The graphics interface package

A **H**igh level **I**nterface to **G**raphics and **Z**EBRA (HIGZ) has been developed within the PAW project. This package is a layer between the application program (e.g. PAW/HPLOT) and the basic graphics package (e.g. X11) on a given system. Its basic aims are:

- Full transportability of the picture data base.
- Easy manipulation of the picture elements.
- Compactness of the data to be transported and accessibility of the pictures in direct access mode.
- Independence of the underlying basic graphics package. Presently HIGZ is interfaced with several GKS packages, X- Windows (X11), PHIGS, Mac, PC's graphic systems, GL (Silicon Graphics), GDDM (IBM), GPR (Apollo) as well as with the DI3000 system.

These requirements have been incorporated into HIGZ by exploiting the data management system ZEBRA.

HIGZ does not introduce new basic graphics features, but introduces some macroprimitives for frequently used functions (e.g. arcs, axes, boxes, pie-charts, tables). The system provides the following features:

- Basic graphics functions: basic primitives, attributes, space definition.
- Higher-level macroprimitives.
- Data structure management using an interface to the ZEBRA system.
- Interactive picture editing.

These features, which are available simultaneously, are particularly useful during an interactive session, as the user is able to "replay" and edit previously created pictures, without the need to re-run the application program. A direct interface to PostScript is also available.

1.6.4 ZEBRA - The data structure management system

The data structure management package ZEBRA was developed at CERN in order to overcome the lack of dynamic data structure facilities in FORTRAN, the favourite computer language in high energy physics. It implements the **dynamic creation and modification** of data structures at execution time and their transport to and from external media on the same or different computers, memory to memory, to disk or over the network, at an **insignificant cost** in terms of execution-time overheads.

ZEBRA manages any type of structure, but specifically supports linear structures (lists) and trees. ZEBRA input/output is either of a sequential or direct access type. Two data representations, **native** (no data conversion when transferred to/from the external medium) and **exchange** (a conversion to an interchange format is made), allow data to be transported between computers of the same and of different architectures. The direct access package **RZ** can be used to manage hierarchical data bases. In PAW this facility is exploited to store histograms, Ntuples and pictures in a hierarchical direct access directory structure.

1.6.5 MINUIT - Function minimization and error analysis

MINUIT is a tool to find the **minima of a multi-parameter function** and analyse the **shape around the minimum**. It can be used for **statistical analysis** of curve fitting, working on a χ^2 or log-likelihood function, to compute the **best fit** parameter values, their uncertainties and correlations. **Guidance** can be provided in order to find the correct solution, parameters can be kept fixed and data points can be easily added or removed from the fit. An interactive Motif based interface is in preparation.

1.6.6 COMIS - The FORTRAN interpreter

The COMIS interpreter allows the user to execute interactively a set of FORTRAN routines in interpretive mode. The interpreter implements a large subset of the complete FORTRAN language. It is an extremely important tool because it allows the user to specify his own complex data analysis procedures, for example selection criteria or a minimisation function.

1.6.7 SIGMA - The array manipulation language

A scientific computing programming language SIGMA (**S**ystem for **I**nteractive **G**raphical **M**athematical **A**pplications), which was designed essentially for mathematicians and theoretical physicists and has been in use at CERN for over 10 years, has been integrated into PAW. Its main characteristics are:

- The basic data units are scalars and one or more dimensional rectangular arrays, which are automatically handled.
- The computational operators resemble those of FORTRAN.

1.7 A PAW Glossary

Data Analysis Terminology

DST A "Data Summary Tape" is one basic form of output from a typical physics experiment. A DST is generally not used directly by PAW, but is analyzed by customized user programs to produce Ntuple files, which PAW can read directly.

Ntuple A list of identical data structures, each typically corresponding to a single experimental event. The data structures themselves frequently consist of a row of numbers, so that many Ntuples may be viewed as two-dimensional arrays of data variables, with one index of the array describing the position of the data structure in the list (i.e., the row or event number), and the other index referring to the position of the data variable in the row (i.e., the column or variable number). A meaningful name is customarily assigned to each column that describes the variable contained in that column for each event. However, the underlying utilities dealing with Ntuples are currently being generalized to allow the name of an element of the data structure to refer not only to a single number, but also to more general data types such as arrays, strings, and so on. Thus it is more general to view an Ntuple as a sequence of tree-structured data, with names assigned to the top-level roots of the tree for each event.

Event A single instance of a set of data or experimental measurements, usually consisting of a sequence of variables or structures of variables resulting from a partial analysis of the raw data. In PAW applications, one typically examines the statistical characteristics of large sequences of similar events.

Variable One of a user-defined set of named values associated with a single event in an Ntuple. For example, the (x, y, z) values of a momentum vector could each be variables for a given event. Variables are typically useful experimental quantities that are stored in an Ntuple; they are used in algebraic formulas to define boolean cut criteria or other dependent variables that are relevant to the analysis.

Cut A boolean-valued function of the variables of a given event. Such functions allow the user to specify that only events meeting certain criteria are to be included in a given distribution.

Mask A set of columns of zeros and ones that is identical in form to a new set of Ntuple variables. A mask is typically used to save the results of applying a set of cuts to a large set of events so that time-consuming selection computations are not repeated needlessly.

Function Sequence of one or more statements with a FORTRAN-like syntax entered on the command line or via an external file.

Statistical Analysis Terminology

Histogram A one- or two-dimensional array of data, generated by HBOOK in batch or in a PAW session. Histograms are (implicitly or explicitly) declared (booked); they can be filled by explicit entry of data or can be derived from other histograms. The information stored with a histogram includes a title, binning and packing definitions, bin contents and errors, statistic values, possibly an associated function vector, and output attributes. Some of these items are optional. The ensemble of this information constitutes an **histogram**.

Booking The operation of declaring (creating) an histogram.

Filling The operation of entering data values into a given histogram.

Fitting Least squares and maximum likelihood fits of parametric functions to histograms and vectors.

Projection The operation of projecting two-dimensional distributions onto either or both axes.

Band A band is a projection onto the X (or Y) axis restricted to an interval along the other Y (or X) axis.

Slice A slice is a projection onto the X (or Y) axis restricted to one bin along the other Y (or X) axis. Hence a slice is a special case of a band, with the interval limited to one bin.

Weight PAW allows the user to include a multiplicative statistical bias for each event which is a scalar function of the available variables. This permits the user to correct for known statistical biases in the data when making histograms of event distributions.

KUIP/ZEBRA User Environment Terminology

Macro A text file containing a set commands and logical constructs to control the flow of execution. Parameters can be supplied when calling a macro.

Vector The equivalent of a FORTRAN array supporting up to three dimensions. The elements of a vector can be stored using a real or an integer representation; they can be entered interactively on a terminal or read from an external file.

Logical Directory The ZEBRA data storage system resembles a file system organized as logical directories. PAW maintains a global variable corresponding to the "current directory" where PAW applications will look for PAW objects such as histograms. The ZEBRA directory structure is a tree, and user functions permit the "current directory" to be set anywhere in the current tree, as well as creating new "directories" where the results of PAW actions can be stored. A special directory called `//PAWC` corresponds to a memory-resident branch of this virtual file system. ZEBRA files may be written to the operating system file system, but entire hierarchies of ZEBRA directories typically are contained in a single binary operating system file.

Graphics Production Terminology

Metafile File containing graphical information stored in a device independent format, which can be replayed on various types of output devices. (e.g. the GKS Appendix E metafile and PostScript, both used at CERN).

Picture Graphics object composed of graphics primitives and attributes. Pictures are generated by the HIGZ graphics interface and they can be stored in a picture direct-access database, built with the RZ-package of the data structure manager ZEBRA.

PostScript High level page description language permitting the description of complex text and graphics using only text commands. Using PostScript representations of graphics makes it possible to create graphics files that can be exchanged with other users and printed on a wide variety of printers without regard to the computer system upon which the graphics were produced. Any graphics display produced by PAW can be expressed in terms of PostScript, written to a file, and printed.

Chapter 2: General principles

2.1 Access to PAW

At CERN the PAW program is interfaced on all systems via a command procedure which gives access to the three release levels of the CERN Program Library (PROduction, OLD and the NEW areas) and sets the proper environment if necessary. Users who are not at CERN or who are using non-central computer systems should contact their system administrator for help on PAW.

2.1.1 IBM/VM-CMS

There are three versions available:

GKS For any ASCII graphic terminal capable of emulating Tektronix or PG.

GDDM For IBM 3192G graphic terminals or its emulators (e.g. tn3270 on a Mac-II)

X11 For any X-window display connected to VM

You need a machine size of at least 11 Mb, that may be defined either temporarily for the current session (command DEFINE STORAGE 11M followed by an IPL CMS) or permanently for all subsequent sessions (command DIRM STOR 11M; you need to logoff once to make the definition effective).
An interface Rexx exec file PAW EXEC is located on the Q-disk and has the following interface:

```
PAW ( ver driver
```

The first parameter ver can have the values PRO, NEW and OLD and the second parameter driver the values GKS, GDDM or X11. The defaults are: PRO GKS. Help is available via FIND CMS PAW.

2.1.2 VAX/VMS

There are two versions available on VXCERN: GKS and X11. A command file CERN_ROOT:[EXE]PAW.COM is defined system-wide via the logical symbol PAW; its interface is:

```
PAW/ver/driver
```

(default is PRO GKS). You may set the initialization of PAW either as a PAWLOGON.KUMAC located in your home directory, or through the logical symbol DEFINE PAW$LOGON disk:[user.subdir]file.kumac to be defined usually in your LOGIN.COM. Help is available via HELP @CERNLIB PAW.

2.1.3 Unix systems

There are three versions available: GKS, GPR and X11. The driver shell script is located in the file /cern/pro/bin/paw. In order to access it automatically you could add the directory /cern/pro/bin to your command search path. The command syntax is:

```
paw -v ver -d driver
```

(default is -v PRO -d GKS). In the GKS case this shell script sets the proper GKS environment.

2.1.4 Note on the X11 version

The X11 version needs to know the X-host where graphics must be displayed; this can be specified on each system on the command line:

```
VM/CMS:    PAW ( X11 HOST yourhost
Vax/VMS:   PAW/X11/host=yourhost
Unix:      paw -d X11 -h yourhost
```

or at the "Workstation" prompt in PAW: `Workstation type (?=HELP) [CR]=1 :  1.yourhost`

On Vax/VMS the default X-window protocol is TCP/IP. If you want DECNET (e.g. when running from a Vaxstation) add the DECNET option to the command as follows:

```
PAW/X11/DECNET/host=yourhost
```

If yourhost is not specified, the output is redirected (like for all X11 applications) to the display defined via the environment variable DISPLAY.

The workstation type selects which type of workstation has to be opened. It corresponds to a line number in a file higz_windows.dat (HIGZWIN DATA on IBM/VM machines). PAW tries to open this file in your current working directory. If it does not succeed it tries in your HOME directory. If it doesn't succeed once more, it creates the file in your HOME directory as follows:

```
0000 0000 0600 0600
        .
        .
        .
0000 0000 0600 0600
```

where the lines define each of the workstation types (from 1 to 10) with the x-margin (left), y-margin (top), x-size (width) and y-size (height) of the corresponding window in pixels.

2.1.5 Different modes to start PAW

– A **batch** version of PAW is available (note that batch implies workstation type 0):

```
On Unix   do:  paw -b macroname
On VMS    do:  PAW/BATCH=macroname
On VM     do:  PAW (BATCH=macroname
```

– One can **disable** the automatic execution of the PAWLOGON macro:

```
On Unix   do:  paw -n
On VMS    do:  PAW/NOLOG
On VM     do:  PAW (NOLOG
```

2.2 Initialising PAW

When PAW is started, a **system** startup procedure is initiated, which indicates the current version of PAW and requests the **workstation type** of the terminal or workstation which you are using.

```
$ PAW
 ****************************************************
 *                                                  *
 *            W E L C O M E    to   P A W           *
 *                                                  *
 *        Version 2.03/12  17 September 1993        *
 *                                                  *
 ****************************************************
 Workstation type (?=HELP) <CR>=1 : ?

 List of valid workstation types:
       0:  Alphanumeric terminal
    1-10:  Describe in file higz_windows.dat
  n.host:  Open the display on host (1 < n < 10)
    7878:  FALCO terminal
    7879:  xterm
```

Note that if you specify 0, PAW will not open a graphics workstation. This may be appropriate if one wants to use PAW on an alphanumeric terminal.

Before passing control to the user, the system looks for a user-supplied file `pawlogon.kumac` or `PAWLOGON KUMAC (VM/CMS)`. The latter can contain commands which the user wants to be executed at PAW startup, e.g. declaration of files, creation of aliases, definition of HPLOT parameters. A simple version of this PAW initialisation file, displaying date and time, can be:

```
mess '****************************************************'
mess '*                                                  *'
mess '*   Starting PAW session on '//$date//' at '//$time//'     *'
mess '*                                                  *'
mess '****************************************************'
```

To ensure that only one version of this file is necessary, on VAX/VMS a **logical name** `PAW$LOGON` should be defined in the user's `LOGIN.COM`, as explained above. On a Unix workstation the file `pawlogon.kumac`, should be put into the directory. On IBM/VM-CMS the minidisk file search rule takes care of finding the file.

2.3 PAW++

paw++ is a new and powerful OSF/Motif based Graphical User Interface to the popular Physics Analysis Workstation PAW. The graphical user interface makes the full and rich command set of PAW available to even the naive user. Simple point and click operations are enough to execute commands that were previously accessable only to expert users. Figure 2.1 on the next page compares the functionalities of basic PAW with PAW++.

At present it is released on Unix workstations and VAX/VMS.

paw++ has, in addition to the conventional command line and macro types of interface, the following dialogue modes:

Pull Down menus They are useful to understand the command structure of the PAW system.

Command panels They give a "panel representation" of the commands.

Object Browser This is in many ways similar to the well-known browsers in the PC/MAC utilities or the visual tools on some workstations.

Direct graphics One can click in the graphics area and identify automatically which object has been selected. A pop-up menu appears with a list of possible actions on this object. For example, by clicking with the right mouse button on a histogram, one can make directly a gaussian fit, a smoothing etc. Pop-up menus are available by clicking on the **Graphics Window** to automatically produce PostScript, Encapsulated PostScript, LaTeX files or print the picture on your local printer.

Histogram Style Panel Buttons are available to change histogram attributes, colours, line styles, fonts, and axes representation. 2-D histograms can be rotated interactively. Zooming and rebinning can be performed interactivaly in real time.

Ntuple Viewer Just click on the Ntuple column name to histogram the column.

The new system is largely self-explanatory. Only a subset of PAW has been converted to this new user interface, but work is currently in progress to offer many new facilities in future releases.

On all the computers where the cmz is installed, just type paw++ to enter the program.

paw++ starts up with three windows on the screen:

The "paw++ Executive Window" includes a menu bar, a **Transcript Pad**, a current working directory indicator and an **Input Pad**.

The "paw++ Graphics 1" window displays the graphics output from higz/X11. Objects, e.g. histograms, displayed in the **Graphics Window** can be manipulated by pointing at them, pressing the right mouse button and selecting an operation from the popup menu. Pointing at the edge of the **Graphics Window** (between displayed object and window border) brings up a general popup menu. Up to 4 additional **Graphics Window** can be opened by selecting "Open New Window" from this menu.

The "paw++ Main Browser" displays all browsable classes and connected hbook files. Up to 4 additional browsers can be opened via the "View" menu of the "paw++ **Executive Window**" or via the "Clone" button on the browsers. For more information on the browsers see the "Help" menus.

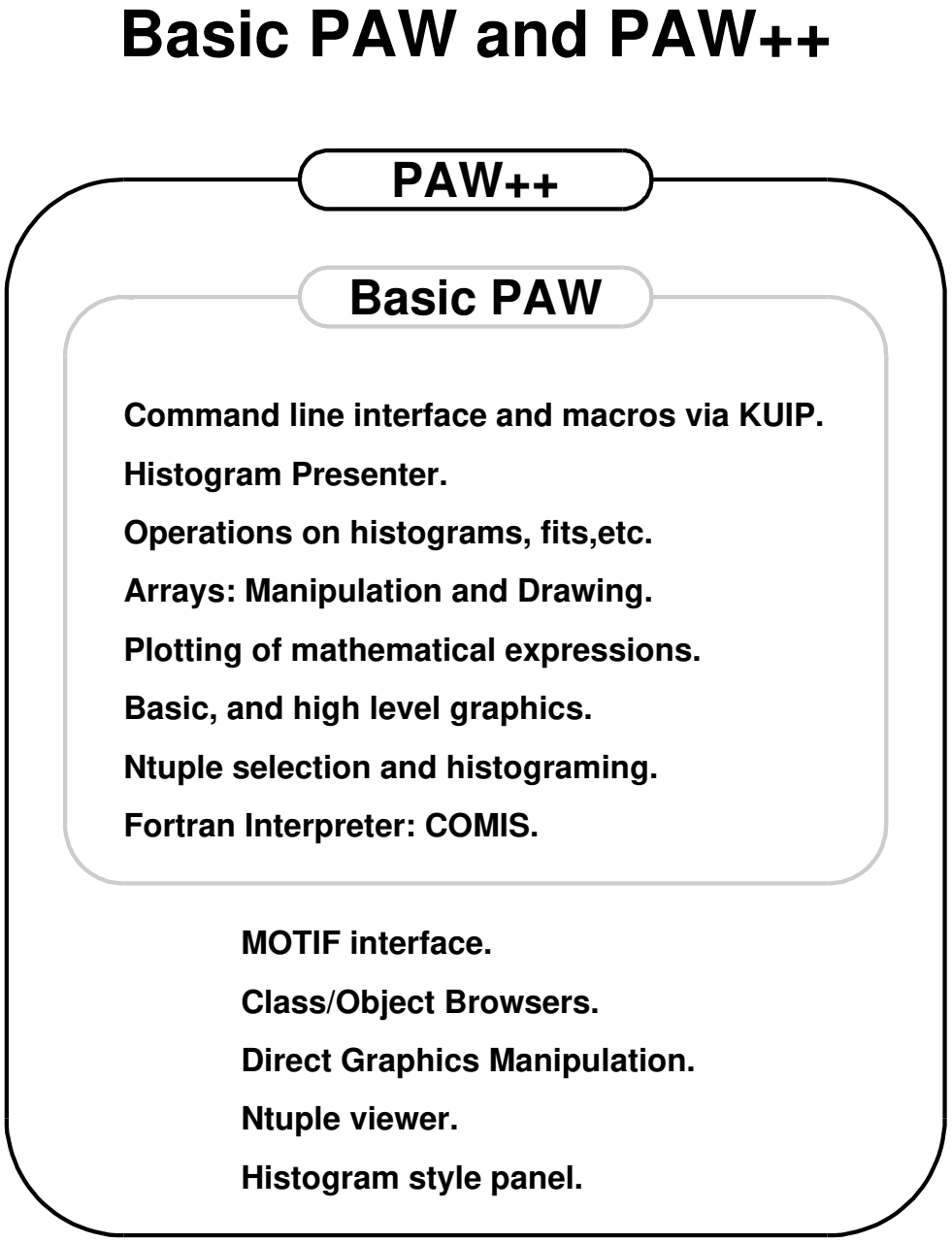
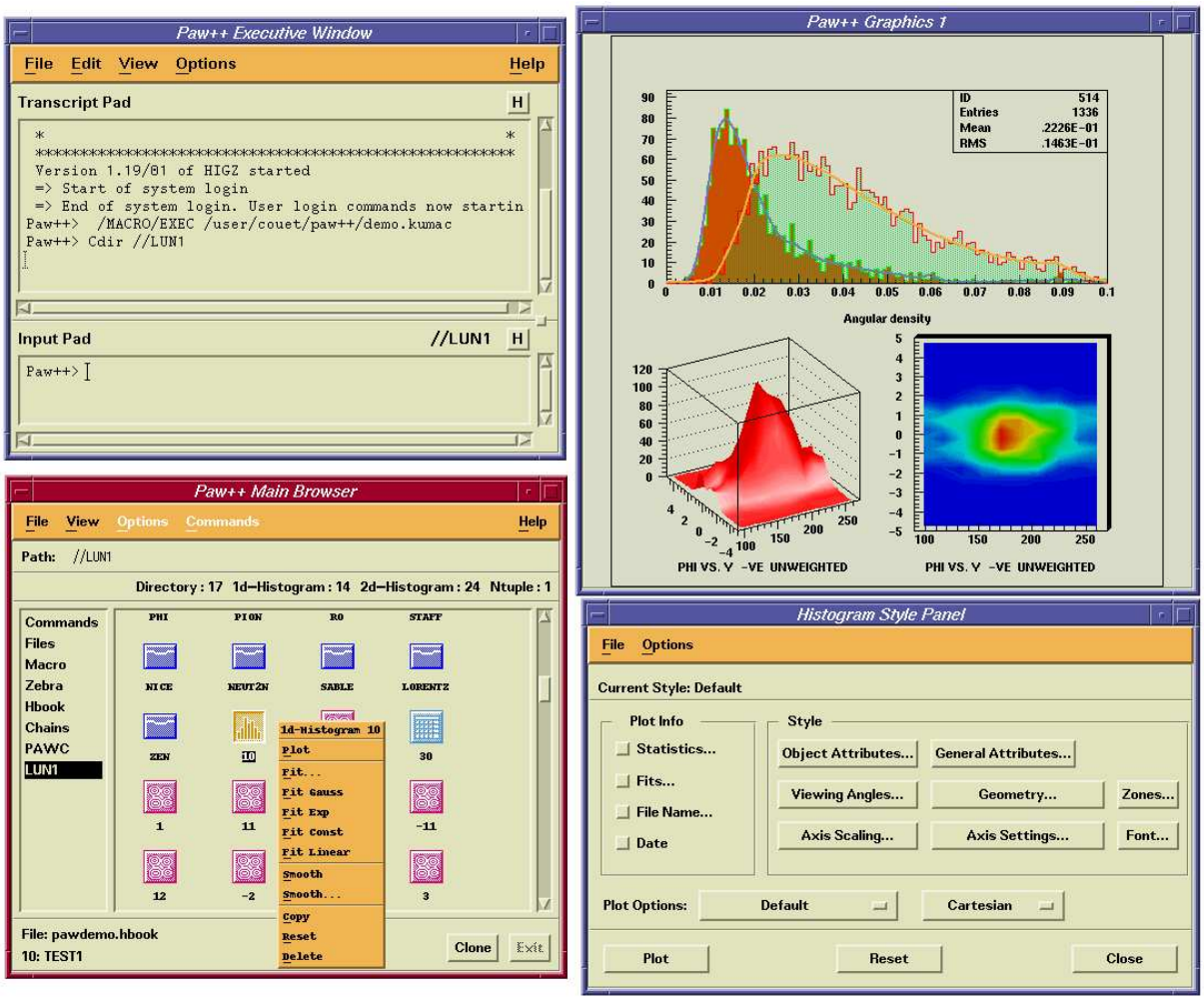
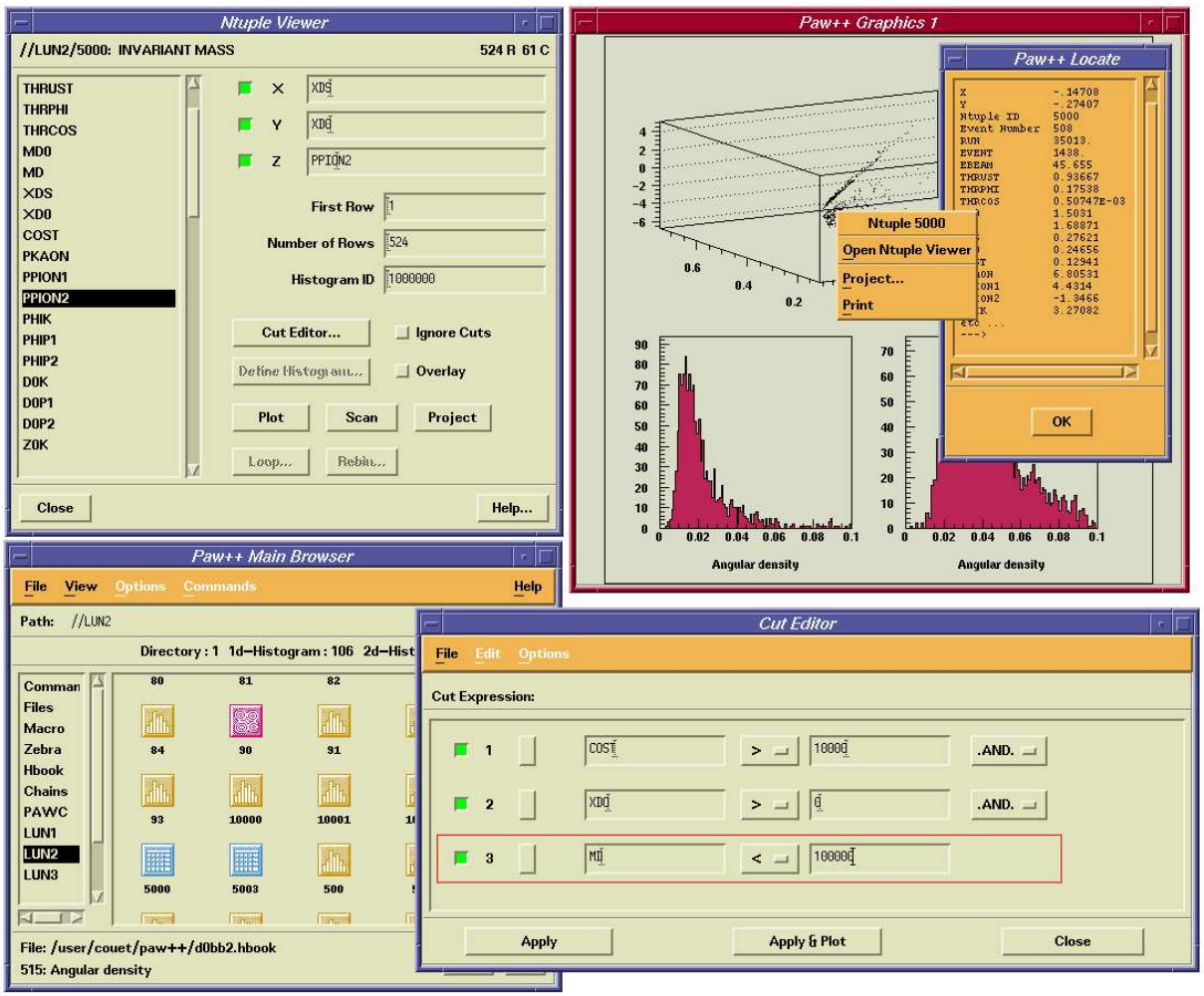


Figure 2.1: PAW and PAW++ compared

2.3.1 Overview of PAW++



– The upper left corner is the `paw++` **Executive Window**, with its **Input Pad** at the bottom and the **Transcript Pad** at the top.
– The `paw++` Browser, where the various entities (pictures, 1-D and 2-D histograms and Ntuples) are all defined with their own symbol, is shown bottom left. A "pop-up" menu has been activated for the chosen 1-D histogram. Several actions like `Plot`, `Smooth`, `Fit` etc... can be performed via this menu.
– The **Graphics Window** is seen top right. A 1-D view of the data points and two 2-D views (a Surface-plot and a colored contour plot) are shown. On the 1-D view, two 1-D histograms are superimposed. The results of a "smoothing" type of fit to the data points is also drawn. Information about the data and the fit can be found in the inserted window.
– The **Histogram Style Panel** at the lower right allows graphics attributes of the histogram to be controlled.



- The upper left corner shows the **Ntuple Viewer**. The left window shows the name of the various variables, characterizing the selected Ntuple. Other windows and press-buttons specify which combinations of the various variables and which events have to be treated (plotted, scanned, …).
- The lower left contains the `paw++` Browser, with this time an Ntuple selected. A double on a Ntuple icon open automatically the **Ntuple Viewer** on the active Ntuple.
- The **Graphics Window** is seen top right and shows a 3-D view of the combination of three variables, whose cuts are specified with the **Cut Editor** (see below).
- Direct graphics interactions with Ntuple data are possible. Just by clicking on a point in the **Graphics Window**, the event description is displayed in the **PAW++ Locate** window.
- The **Cut Editor** panel, shown at the lower right, allows various combinations of cuts to be specified and applied.

2.4 Command structure

PAW is based on the KUIP[4] User Interface package, which can provide different types of dialogue styles:

- Command mode, where the user enters a command line via the terminal keyboard.
- Alphanumeric menu mode, where the command is selected from a list.
- Graphics menu modes:
  - Pull-down menus, fixed layout reflecting the command structure;
  - Panels of function keys, interactive user definable multiple layouts.

It is possible to change interactively from one style to another.

The general format of a PAW command line is:

```
command parameters
```

The first part of the **command** has the format:

```
object/verb
```

where the **object** is the item on which the action is performed (e.g. `HISTOGRAM, VECTOR, NTUPLE`) and the `verb` is the action to be performed (e.g. `CREATE, DELETE, PLOT`). In some cases the object needs to be specified further (e.g. `GRAPHICS/PRIMITIVE`), while in other cases the verb's action needs to be clarified further (e.g. `CREATE/1D`). All components can be **abbreviated** to their shortest unambiguous form. For example the two following lines will have the same effect of creating a vector `A` with nine components:

```
VECTOR/CREATE A(9)
```

or

```
VE/CR A(9)
```

In the case that the form is ambiguous all possible interpretations for the given abbreviation are displayed.

The second part of a command are its **parameters** and their meaning is determined by their **position**. Some of these can be **mandatory** with the remaining ones **optional**. If all mandatory parameters are not provided on the command line, PAW will prompt the user to specify them, indicating the default values if defined. If the user wants to assign the default value to a parameter from the command line he can use the **place-holder** character **exclamation mark (!)** to signify this to PAW. In the case of optional parameters, the user **must** provide them in the correct sequence if he wants to **change** their values, otherwise the corresponding defaults are taken. Parameters containing blanks must be enclosed within single quotes.

In the example below we create a one-dimensional histogram, providing the parameters one by one answering the PAW query:

```
PAW > histogram/create/1dhisto
Histogram Identifier (<CR>= ): 10
Histogram title (<CR>= ): title1
Number of channels (<CR>=100): <CR>
Low edge (<CR>=0): 10.
Upper edge (<CR>=100): 20.
```

For the command below we provide all parameters on the command line, including an optional one (`1000.`), which by default has the value `0`. Note that this parameter **must** be specified explicitly, since PAW **does not** prompt for it, as seen in the previous example. Note also the use of the exclamation mark to take the default for the number of channels (`100`).

```
PAW > hi/cr/1d 20 title2 ! 10. 20. 1000.
```

2.5 Getting help

Once inside PAW, one can start entering commands. An interesting first try would be the `HELP` command, which displays a list of items, preceded by a number and followed by one line of explanation. In the next example we search for a command to create a one-dimensional histogram.

```
      PAW > help

From  /...

 1:   KUIP          Command Processor commands.
 2:   MACRO         Macro Processor commands.
 3:   VECTOR        Vector Processor commands.
 4:   HISTOGRAM     Manipulation of histograms, Ntuples.
 5:   FUNCTION      Operations with Functions. Creation and plotting.
 6:   NTUPLE        Ntuple creation and related operations.
 7:   GRAPHICS      Interface to the graphics packages HPLOT and HIGZ.
 8:   PICTURE       Creation and manipulation of HIGZ pictures.
 9:   ZEBRA         Interfaces to the ZEBRA RZ, FZ and DZ packages.
10:   FORTRAN       Interface to MINUIT, COMIS, SIGMA and FORTRAN
                    Input/Output.
11:   NETWORK       To access files on remote computers.
12:   OBSOLETE      Obsolete commands.

Enter a number ('Q'=command mode): 4

   /HISTOGRAM

   Manipulation of histograms, Ntuples.  Interface to the HBOOK package.


From  /HISTOGRAM/...

 1: * FILE          Open an HBOOK direct access file.
 2: * LIST          List histograms and Ntuples in the current directory.
 3: * DELETE        Delete histogram/Ntuple ID in Current Directory
                    (memory).
 4: * PLOT          Plot a single histogram or a 2-Dim projection.
 5: * ZOOM          Plot a single histogram between channels ICMIN and
                    ICMAX.
 6: * MANY_PLOTS    Plot one or several histograms into the same plot.
 7: * PROJECT       Fill all booked projections of a 2-Dim histogram.
 8: * COPY          Copy a histogram (not Ntuple) onto another one.
 9: * FIT           Fit a user defined (and parameter dependent) function
                    to a histogram ID (1-Dim or 2-Dim) in the specified
                    range.
10:   2D_PLOT       Plotting of 2-Dim histograms in various formats.
11:   CREATE        Creation ("booking") of HBOOK objects in memory.
12:   HIO           Input/Output operations of histograms.
13:   OPERATIONS    Histogram operations and comparisons.
14:   GET_VECT      Fill a vector from values stored in HBOOK objects.
15:   PUT_VECT      Replace histogram contents with values in a vector.
16:   SET           Set histogram attributes.
```

```
Enter a number ('=one level back, 'Q'=command mode): 11

   /HISTOGRAM/CREATE

   Creation ("booking") of HBOOK objects in memory.


From  /HISTOGRAM/CREATE/...

 1: * 1DHISTO       Create a one dimensional histogram.
 2: * PROFILE       Create a profile histogram.
 3: * BINS          Create a histogram with variable size bins.
 4: * 2DHISTO       Create a two dimensional histogram.
 5: * PROX          Create the projection onto the x axis.
 6: * PROY          Create the projection onto the y axis.
 7: * SLIX          Create projections onto the x axis, in y-slices.
 8: * SLIY          Create projections onto the y axis, in x-slices.
 9: * BANX          Create a projection onto the x axis, in a band of y.
10: * BANY          Create a projection onto the y axis, in a band of x.
11: * TITLE_GLOBAL  Set the global title.

Enter a number ('=one level back, 'Q'=command mode): 1

 * /HISTOGRAM/CREATE/1DHISTO  ID TITLE NCX XMIN XMAX [  VALMAX ]

   ID         C 'Histogram Identifier' Loop
   TITLE      C 'Histogram title' D=' '
   NCX        I 'Number of channels' D=100
   XMIN       R 'Low edge' D=0.
   XMAX       R 'Upper edge' D=100.
   VALMAX     R 'Maximum bin content' D=0.

   Create a one dimensional histogram.  The contents are set to zero.  If
   VALMAX=0, then a full word is allocated per channel, else VALMAX is used
   as the maximum bin content allowing several channels to be stored into
   the same machine word.

<CR>=continue, 'Q'=command mode, 'X'=execute: q
```

The meaning of the notation used in the text displayed by the HELP command is explained on page III. Moreover an item preceded by a **star** indicates a **terminal leaf** in the command tree, i.e. an **executable** command.

One can also inquire about **creating a one-dimensional histogram** by typing simply:

```
      HELP histogram/create/1dhisto
or
      HELP his/cre/1d
or even
      HELP 1
```

The system will then display the following information:

```
  * /HISTOGRAM/CREATE/1DHISTO  ID TITLE NCX XMIN XMAX [  VALMAX ]

   ID         C 'Histogram Identifier' Loop
   TITLE      C 'Histogram title' D=' '
   NCX        I 'Number of channels' D=100
   XMIN       R 'Low edge' D=0.
   XMAX       R 'Upper edge' D=100.
   VALMAX     R 'Maximum bin content' D=0.
```

```
Create a one dimensional histogram.  The contents are set to zero.  If
VALMAX=0, then a full word is allocated per channel, else VALMAX is used
as the maximum bin content allowing several channels to be stored into
the same machine word.
```

2.5.1 Usage

Very often a single line description of the usage of a command is sufficient as a reminder. This can be obtained by the USAGE command, e.g.:

```
PAW > USAGE 1d

* /HISTOGRAM/CREATE/1DHISTO  ID TITLE NCX XMIN XMAX [  VALMAX ]
```

2.6 Special symbols for PAW

One should pay attention to the fact that, in addition to their common arithmetic meaning, the symbols in table 2.1 have a special connotation when working with PAW .

Symbol	Meaning
`blank`	Separator between command and parameter and between different parameters
`/`	Separator between command elements
	Comment line (if first character of the command line)
`|`	Inline comments
`'`	String delimiter
`_`	Line continuation in KUIP commands
`@`	Escape character to be put in front of `|` and `'` to interpret them as literal
`!`	Place-holder for command parameter (i.e. default value is taken)
	At beginning of command line: Unix C shell-like **history**
	(e.g. `!!, !number, !-number, !string`)
`[]`	Macro argument delimiters
`#`	Separator between macro file and macro member
`( )`	Vector subscript delimiters
`:`	Vector subscript range
`,`	Multi-dimensional vector subscript dimensions delimiter
Note: These special characters loose their effect when imbedded in single quotes.	

Table 2.1: Special symbols

2.7 PAW entities and their related commands

Relations which exist between various PAW entities as described in section 1.6 on page 9 and the operations which can be performed upon them have been schematically represented in figure 2.2. All commands shown in the picture next to the lines connecting the objects have been abbreviated in a way that

they are unambiguous and can be typed to PAW, which will then detail the various parameters to be supplied.

There are three main input/output formats, namely a simple text file (e.g. with data points or commands), a direct access ZEBRA RZ file (used by HBOOK and HIGZ for storing histograms and pictures on a given machine) and a ZEBRA FZ sequential file, which can be used to transfer structured ZEBRA data between various computers. The RZ and FZ representations can be transformed into each other using the TOALFA and FRALFA commands.

The three main PAW objects, Ntuples, histograms and vectors, can be **printed** on an alphanumeric screen (`PRINT` commands) or they can be plotted on a graphics screen (`PLOT` commands). The picture can be transformed into a ZEBRA data structure and stored in a HIGZ database for later reference (e.g. editing by the HIGZ editor), or an external presentation can be obtained via the creation of a **metafile**. This "metafile" can for instance consist of GKS or PostScript commands, which can then be interpreted by the relative drivers and printed on an output device, if so desired.

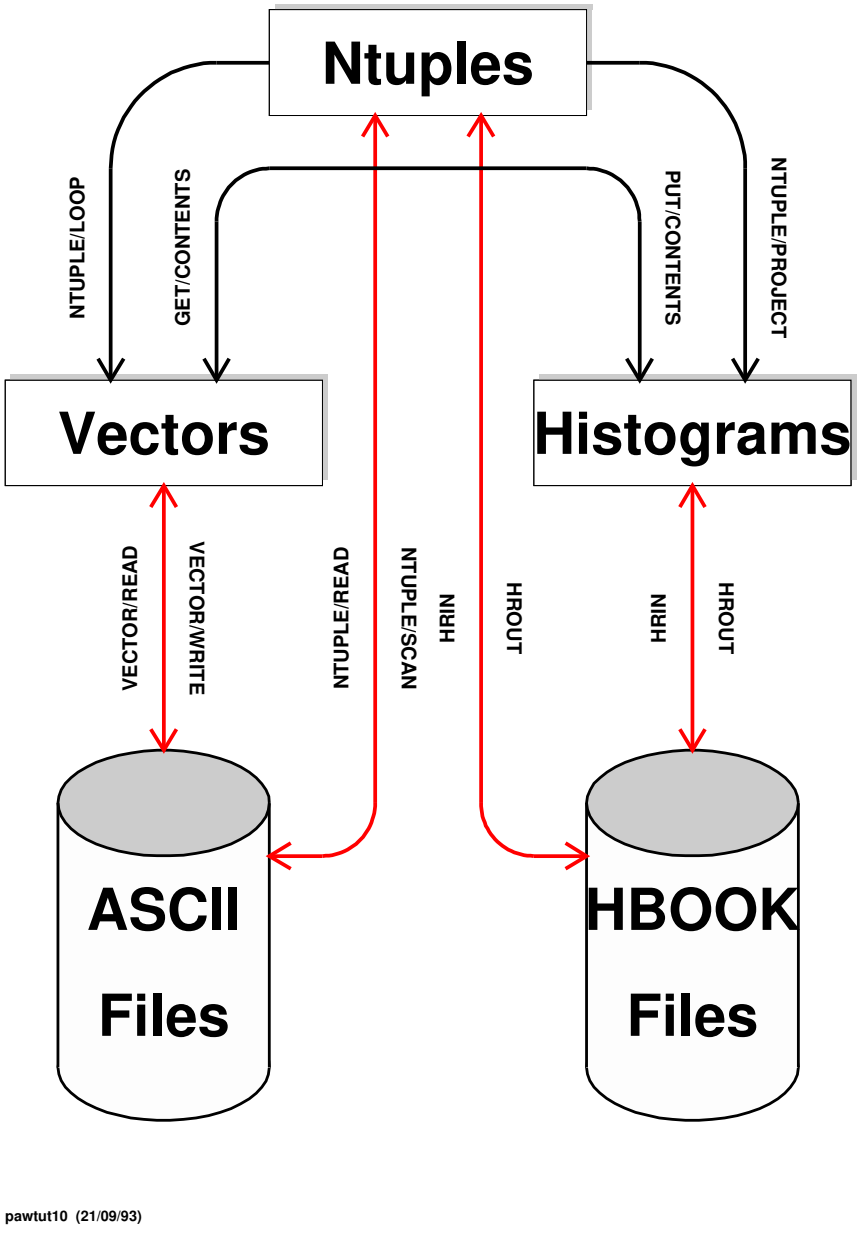


Figure 2.2: PAW entities and their related commands

Chapter 3: PAW by Examples

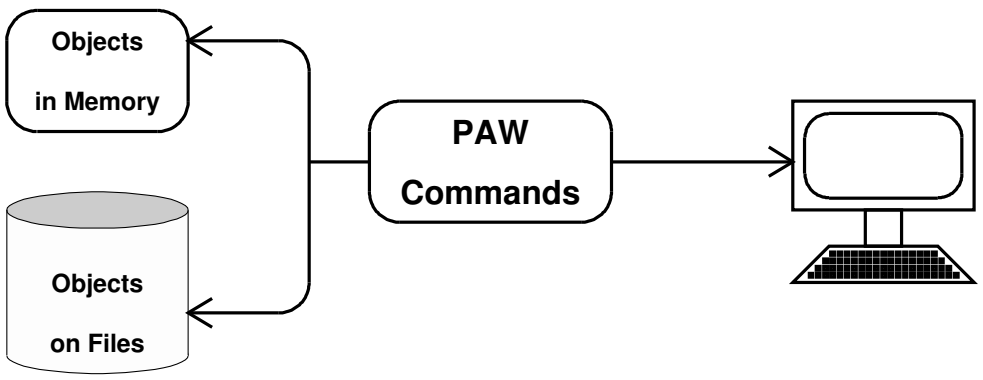
Contents

3.1 Basic Principles . . . 30
3.2 Starting the PAW Tutorial . . . 31
3.3 Vectors—Tutorial . . . 32
3.4 Vectors—Examples . . . 36
3.4.1 Starting with vectors . . . 36
3.4.2 Some more vector commands . . . 38
3.4.3 The VECTOR/DRAW options . . . 40
3.4.4 Vectors and Histograms . . . 42
3.4.5 Vector operations . . . 44
3.4.6 Simple macro, with a loop and a VECTOR fit . . . 46
3.4.7 Macros flow control . . . 48
3.4.8 More on fits . . . 50
3.4.9 VECTOR/READ using MATCH . . . 54
3.5 Function drawing—Examples . . . 56
3.5.1 Plot a few one-dimensional functions . . . 56
3.5.2 Plot a one-dimensional function and loop . . . 58
3.5.3 More on macro input parameters . . . 60
3.5.4 Plot two-dimensional functions . . . 62
3.5.5 The Mandelbrot distribution . . . 64
3.5.6 Three-dimensional functions drawing . . . 66
3.6 Histograms—Tutorial . . . 68
3.7 Histograms—Examples . . . 80
3.7.1 Histograms creation . . . 80
3.7.2 Read histograms from file and plot . . . 84
3.7.3 Histogram archiving . . . 88
3.7.4 Multiple fits on histograms . . . 90
3.7.5 Histogram operations . . . 92
3.7.6 Keep and update histograms . . . 96
3.7.7 Playing with dice . . . 98
3.7.8 Two-dimensional histograms representations . . . 100
3.7.9 Non equidistant contour plots . . . 102
3.7.10 Coordinate systems . . . 104
3.7.11 Logarithmic scales on lego plots . . . 106
3.7.12 Subranges in histogram identifiers . . . 108
3.7.13 Stacked Lego plots . . . 110
3.7.14 A more complex example . . . 112
3.8 Ntuples—Tutorial . . . 116
3.9 Ntuples—Examples . . . 122
3.9.1 Ntuple creation . . . 122

3.9.2 Automatic and user binning . . . 128
3.9.3 Simple selection criteria on Ntuple . . . 130
3.9.4 Use of Ntuple masks and loops . . . 134
3.9.5 The use of Ntuple Cuts . . . 136
3.9.6 Ntuple and 2D histograms . . . 138
3.9.7 Profile histograms and Ntuples . . . 140
3.9.8 Copy a Ntuple variable into a Vector . . . 142
3.9.9 Chain of Ntuples . . . 144
3.10 SIGMA—Examples . . . 146
3.10.1 Examples of the SIGMA processor (1) . . . 146
3.10.2 Examples of the SIGMA processor (2) . . . 150
3.11 Pictures and PostScript . . . 152
3.11.1 Merge pictures onto one plot . . . 152
3.11.2 Pie charts . . . 154
3.11.3 Feynman diagrams . . . 156
3.11.4 Making a complex graph with PAW . . . 158
3.11.5 Making slides . . . 161
3.11.6 How to use PostScript files . . . 164

3.1 Basic Principles

- paw (Physics Analysis Workstation) is an *interactive system* designed for data analysis and data presentation.
- paw provides a set of *commands* acting on specific objects. The main objects or data type are: *vectors*, *histograms*, and *ntuples*. The aim of the examples is to explain how to work with these objects.
- The paw commands are organized in a *tree*, whose general structure is: OBJECT/ACTION.
  Examples: NTUPLE/PLOT, HISTOGRAM/PROJECT, VECTOR/DRAW
- The usual user interface is a "command line interface": commands are typed on keyboard and executed after <CR>. Commands parameters are separated with blank.
- Command editing and retrieving is also possible. It is controlled via the command RECALL_STYLE.
- Commands can be grouped into "Macros". Macros are files with the extension .kumac containing paw commands and flow control operators like "do loop", "if endif", etc .. . To execute a macro it is enough to type EXEC macroname.
- online help can be obtained with the commands:
  - HELP to have the full description of a command.
  - USAGE to have only the command syntax.
- A printable version of the reference manual can be obtained with the command MANUAL.
- paw++ provides a Motif based User Interface to paw.
- paw and paw++ have the SAME basic functionality.



3.2 Starting the PAW Tutorial

This tutorial present the basic principles of `paw` using a set of examples (`paw` macros). It tries to cover the most frequently used basic functions of `paw`. In the examples, highlighted points are written in UPPERCASE with a reference in the left margin. This reference point to a comment after the listing of the macro. If the example produce a graphics output, it is given on the page behind the example. Under each figure, the name of the corresponding macro is given.

Starting the PAW Tutorial

```
MACRO PAWLOGON
Mess '************************************************************'
Mess '*                                                          *'
Mess '*                  Starting PAW examples                   *'
Mess '*                                                          *'
Mess '*                     29-30 June 1993                      *'
Mess '*                                                          *'
Mess '************************************************************'
```

This example shows what could be the `MACRO PAWLOGON` (in the file `PAWLOGON.KUMAC`) which is automatically executed (if it exists) at the beginning of each `paw` session.

It is assumed that the macro `ALDDEF` is executed before each example.

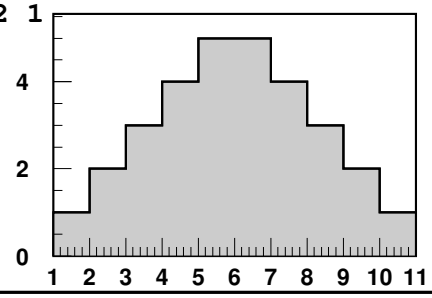
alldef.kumac

```
MACRO ALLDEF
Size 18 24
Next
Set * ; Option * ; Igset *
Size 18 24
Histogram/Delete * ; Vector/Delete *
Title_global ' '
Title_global ' ' U
Option NBOX
Option NGRI
Set *WID 1
Set CSIZ 0.25 ; Set VSIZ 0.25 ; Set TSIZ 0.32
Set XMGL 1.2  ; Set XMGR 1.2  ; Set YMGU 0.5 ; Set YMGL 1.5
Set GSIZ 0.1
Set YHTI 0.7
Set KSIZ 0.15
Set MTYP 1
Zone 1 1
Next
Return
```

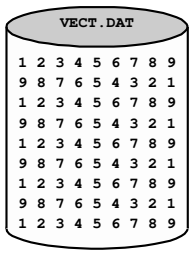
3.3 Vectors—Tutorial

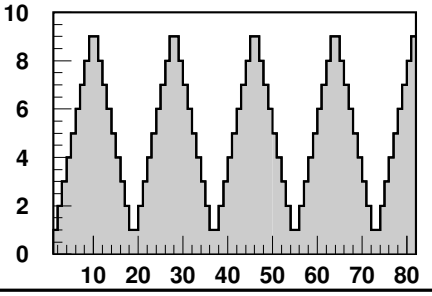
Vector Creation

```
PAW > V/CRE X(10) R 1 2 3 4 5 5 4 3 2 1

PAW > V/WRITE X ! 5(F3.1,1X)

1.0 2.0 3.0 4.0 5.0

5.0 4.0 3.0 2.0 1.0
```

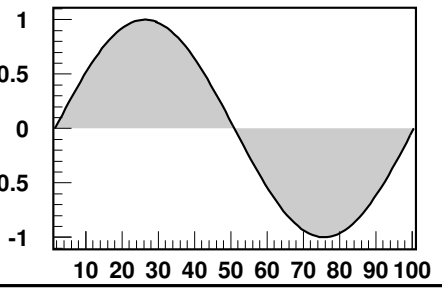



```
PAW > V/READ X VECT.DAT
```

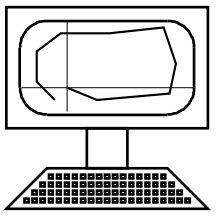


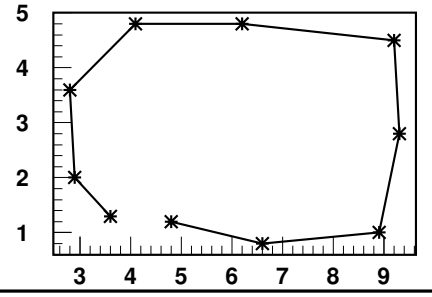



```
PAW > SIGMA X=SIN(ARRAY(100,0#2*PI))

PAW > V/PRINT X

X (    1 ) =    .0000000E+00

X (    2 ) =    .6342392E-01

etc ...
```




```
PAW > VLOCATE X Y
```

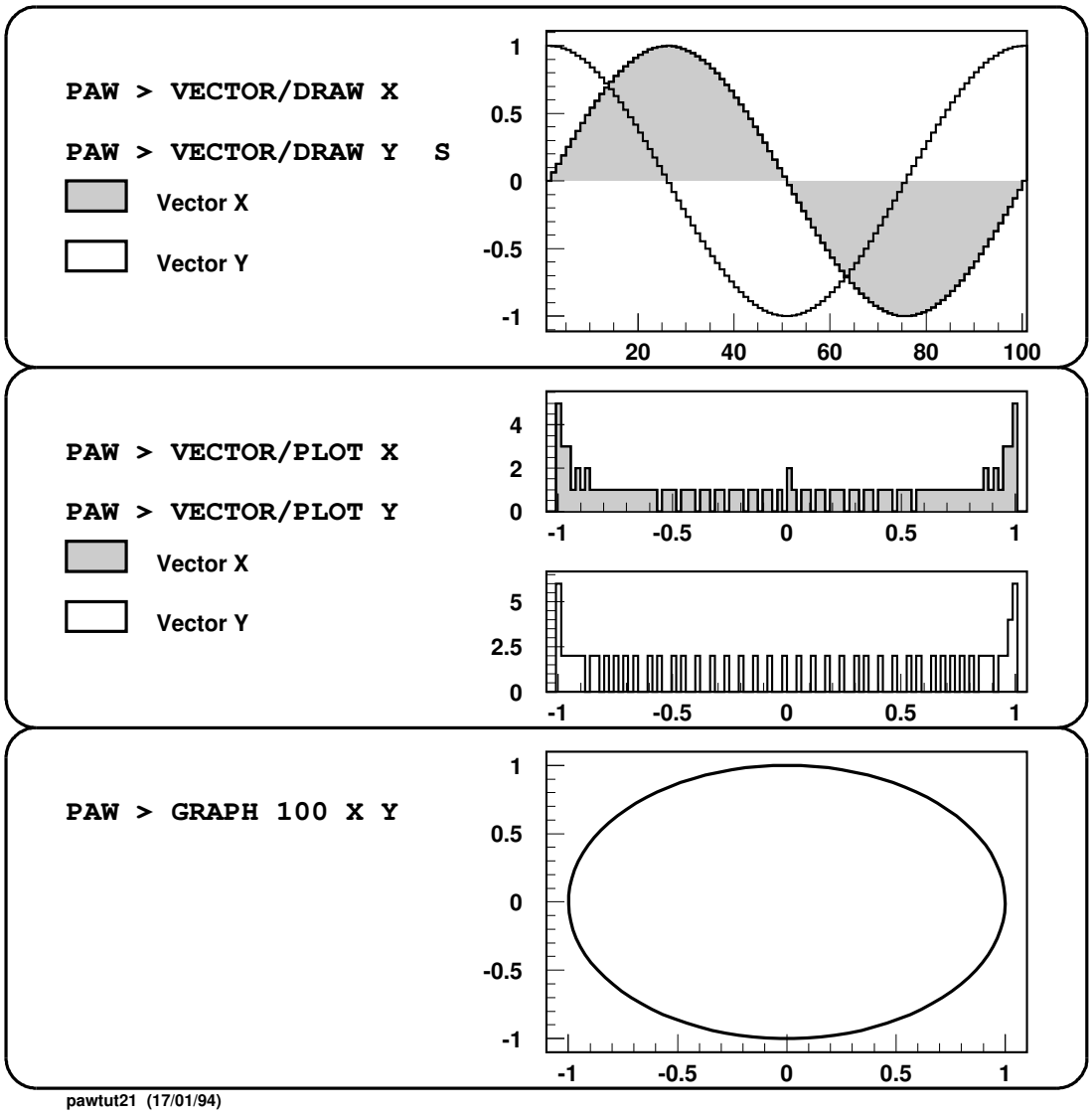




pawtut20 (21/09/93)

Vector Drawing

```
PAW > SIGMA X = SIN(ARRAY(100,0#2*PI))
PAW > SIGMA Y = COS(ARRAY(100,0#2*PI))
```



pawtut21 (17/01/94)

Vectors and COMIS

The declaration VECTOR may be used inside a COMIS routine to address a KUIP vector. If the vector does not exist, it is created with the specifications provided by the declared dimension.

```
PAW > VECTOR/CREATE x(10) R 1 2 3 4 5 6 7 8 9 10
PAW > CALL VECT.F
PAW > VECTOR/WRITE x ! 10(1x,f3.0)
  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.
PAW > VECTOR/WRITE y ! 10(1x,f4.0)
   1.   4.   9.  16.  25.  36.  49.  64.  81. 100.
```

```
SUBROUTINE VECT
VECTOR X,Y(10)
DO I=1,10
   Y(I) = X(I)*X(I)
ENDDO
END
```

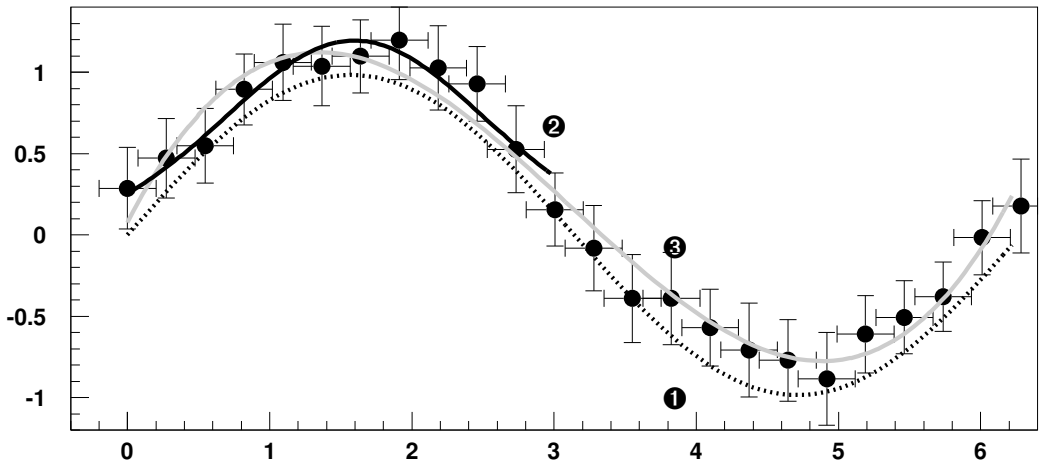
Fitting Vectors - Errors

```
PAW > APPLICATION SIGMA
SIGMA > alpha = array(24,0#2*PI)
SIGMA > sina  = sin(alpha)+rndm(alpha)*0.3
SIGMA > errx  = array(24,0.2#0.2)
SIGMA > erry  = errx+rndm(errx)*0.1
SIGMA > EXIT
❶ PAW > VECTOR/FIT alpha sina erry P3
❷ PAW > VECTOR/FIT alpha(1:12) sina erry G S
  PAW > VECTOR/CREATE par(1) r 10
❸ PAW > VECTOR/FIT alpha sina erry SINFIT.F S 1 par
  PAW > HPLOT/ERRORS alpha sina errx erry 24
```

```
function sinfit(x)
common /pawpar/ par(1)
sinfit=par(1)*sin(x)
end
```

EXT NO.	PARAMETER NAME	VALUE	ERROR	STEP SIZE	FIRST DERIVATIVE
1	P1	1.0309	.75837E-01	.95058E-02	.25594E-01

CHISQUARE = .4981E+00 NPFIT = 24



pawtut29 (21/09/93)

3.4 Vectors—Examples

3.4.1 Starting with vectors

Starting with vectors

```
❻     * Starting with vectors
❻     VECTOR/CREATE VECT1(10)  | Create a vector of length 10
❶     VECTOR/INPUT VECT1 10 8 6 4 2 3 5 7 9 11
❼❶    VECTOR/CRE VX(20) R 1. 2. 3. 4. 5. 6. 7. 8. 9. _
      10. 11. 12. 13. 14. 15. 16. 17. 18. 19. 20.
      v/cr vy(20) r 1.1 3.2 5.3 7.4 7.5 6.6 4.3 2.1 6.6 _
      11.1 16.2 18.3 19.0 17.8 16.0 12.1 9.1 6.1 3.1 6.6
❽     ZON 1 2
❷     VECTOR/DRAW VECT1
❹❷    GRAPH 20 VX VY
❹❺    graph 20 VX VY *
❺     gra 20 VX VY C
❸     VECT/DEL *
```

❶ Here we see two ways to fill a vector:

(a) `V/CREATE`: create a vector and, optionally, fill it.

(b) `V/INPUT`: allows to fill an existing vector.

We will see other ways later.

❷ Graphic representations of vectors : `VECTOR/DRAW` and `GRAPH`.

❸ `VECT/DELETE` allows to delete a vector from memory. "*" means delete all vectors in memory. Very often in `paw` a command acting on a specific kind of objects (vectors, histogram, pictures) can access the complete object set with "*".

Note also:

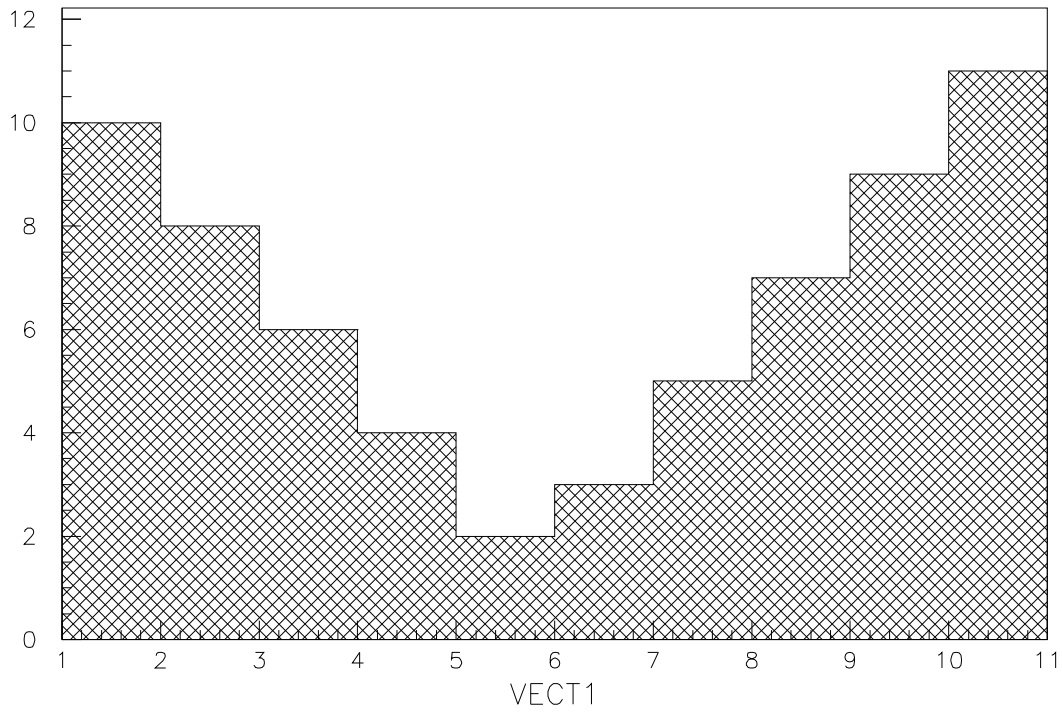
❹ The `paw` commands are case insensitive.

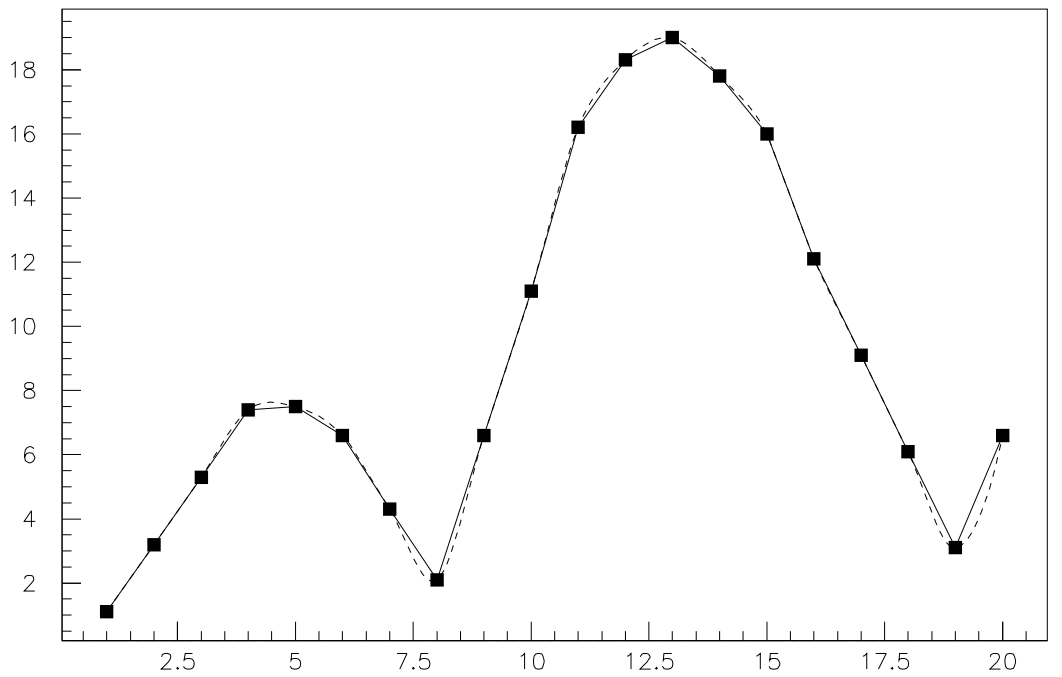
❺ Command abbreviations are permitted.

❻ The character "*" and "|" are used for comments.

❼ The character "'" is used to indicate a continuation line.

❽ The command `ZONE` subdivides the graphical area.


12
10
8
6
4
2
0
1
2
3
4
5
6
7
8
9
10
11
VECT1


18
16
14
12
10
8
6
4
2
2.5
5
7.5
10
12.5
15
17.5
20

3.4.2 Some more vector commands

Some more vector commands

```
      vector/create VECT(10,3) R _
      1. 2. 3. 4. 5. 6. 7. 8. 9. 10. _
      9.1 8.1 7.1 6.1 5.1 4.1 3.1 2.1 1.1 0.1 _
      6.2 4.2 3.2 2.2 1.2 1.2 2.2 3.2 4.2 5.2
      vector/create VECT1(10) R _
      1.1 2.2 3.3 4.4 5.5 6.6 5.5 4.4 3.3 2.2
❺     SET HTYP 244 ; VE/DR VECT(1:10,3)
❶ ❷   VECTOR/DRAW VECT(1:10,3) ! SC
❹ ❻   VECTOR/DRAW VECT1 ! L*S
      ve/list
❸     VE/WRITE VECT 'vector.data' '(3(10f5.0,/))'
```

❶ A vector can have up to three dimensions. Dimensions which are not specified are taken as 1, for example `VEC(10)` → `VEC(10,1,1)` and `VEC` → `VEC(1,1,1)`.

❷ It is possible to access a subrange of a vector, for example: `V(2:3)`, `V(3:)` or `V(:5)`.

❸ The command `VECT/WRITE` creates the file `vector.data` as follows:

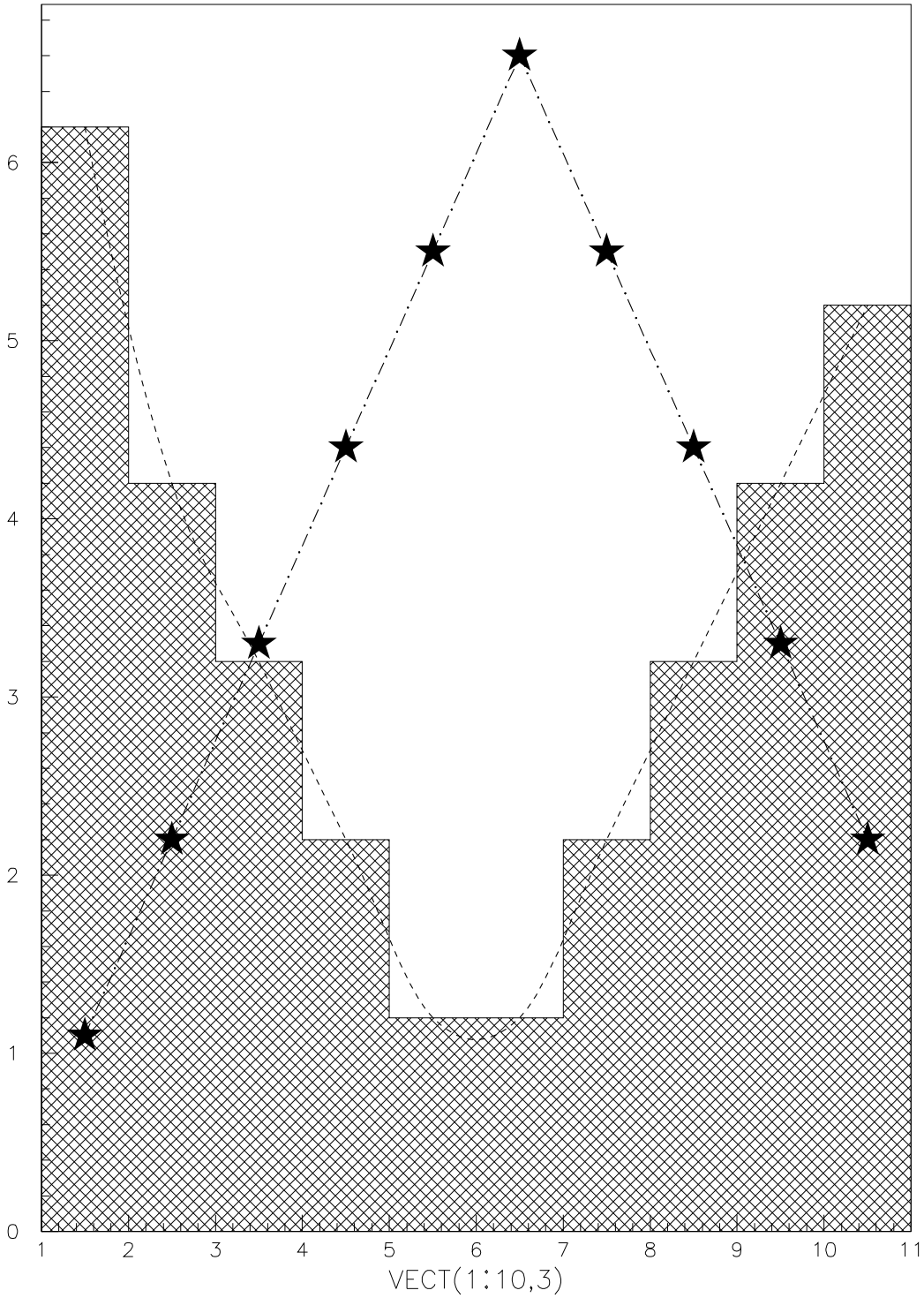
```
 1.   2.   3.   4.   5.   6.   7.   8.   9.  10.
 9.   8.   7.   6.   5.   4.   3.   2.   1.   0.
 6.   4.   3.   2.   1.   1.   2.   3.   4.   5.
```

Note also:

❹ The character "!" means default value of a parameter.

❺ It is possible to have several commands, separated with ";", on the same line.

❻ Many commands have a parameter which defines options. Such parameters (often called `CHOPT` or `OPTION`) have the attribute "Option" (see the help). Each option is a character string. It is possible to mix several options, e.g. "`SC`" or "`L*S`".


0
1
2
3
4
5
6
1
2
3
4
5
6
7
8
9
10
11
VECT(1:10,3)

3.4.3 The VECTOR/DRAW options

Some possible data representations with VECTOR/DRAW

```
      zone 2 3
      ve/create v(10) R 5 1 3 2 4 1 3 1 8 6
❶     SET HTYP 244
      ve/draw v
      ve/draw v ! b
      ve/draw v ! l
❸     VE/DRAW V CHOPT=L*
      ve/draw v ! bl*
❷     IGSET MTYP 21
      ve/draw v ! e
      ve/de V
❹     RETURN
```

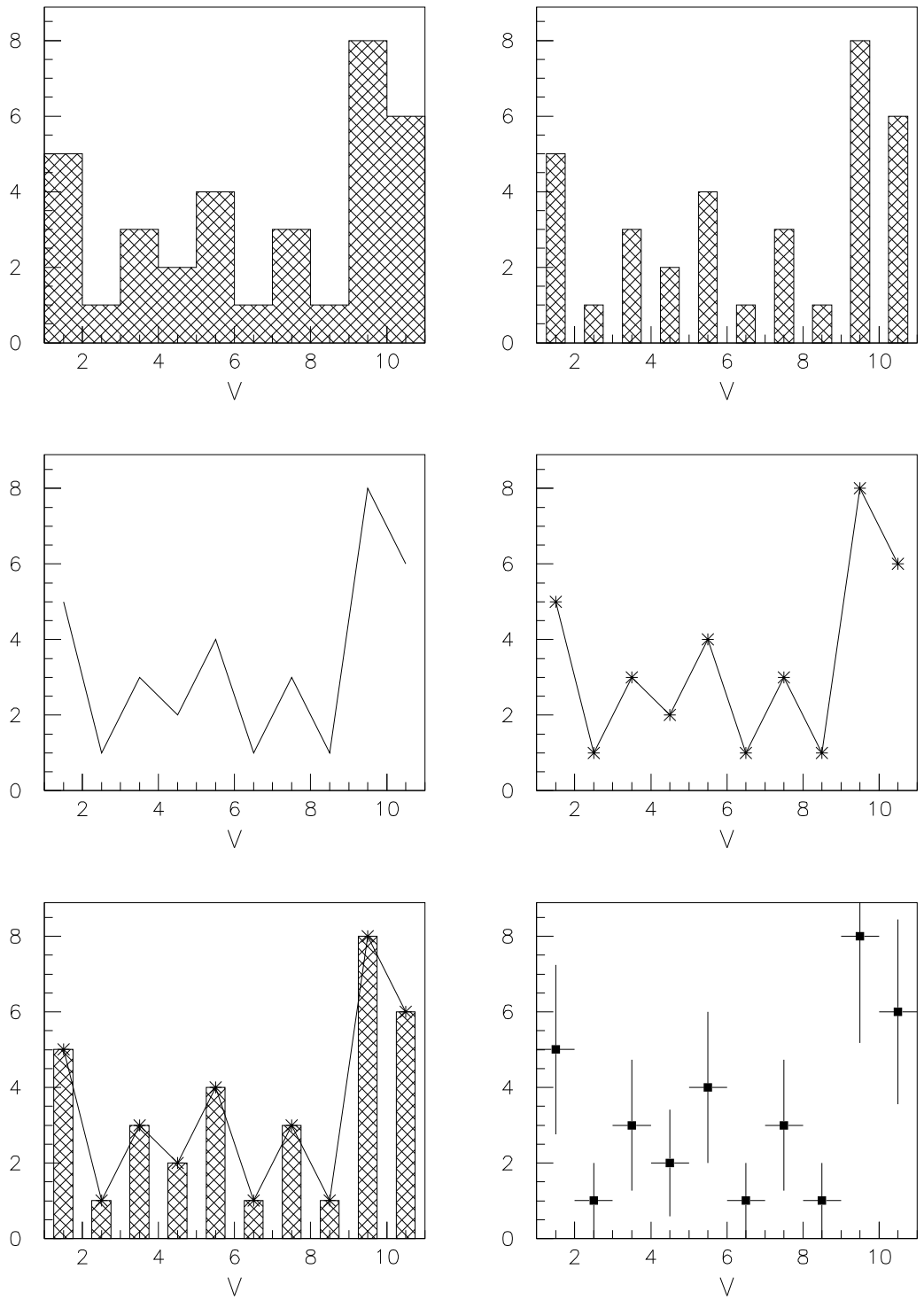
❶ The command `SET` defines some high level graphics attributes for commands like `VECT/DRAW` or `HIST/PLOT`. Here the `HTYP` (Histogram hatch TYPe) is defined.

❷ `IGSET` is used to define basic graphics attributes like line width, marker type etc ... . Here the marker type is defined. It is possible to type always `SET` instead of `IGSET` i.e. if a `IGSET` parameter is invoke with the `SET` command, the command `IGSET` is automatically invoked.

❸ By default the parameters of a command are positional but it is possible to assign values by name, i.e. `PARAMETER=value`. For example we have here `CHOPT=L*`. In this case the "!" can be suppressed.

Note also:

❹ The statement `RETURN` is not mandatory in a macro except if there are several macros in the same file. In this case, a macro within a file can be executed by: `EXEC FILENAME#MACRONAME`.


0
2
4
6
8
2
4
6
8
10
V

3.4.4 Vectors and Histograms

Functionality of VECT/DRAW, VECT/PLOT, VECT/HFILL and PUT/CONT

```
      zone 2 2
      ve/create VECT1(10) R 1 2 3 4 5 5 4 3 2 1
      *
      ve/draw VECT1
❶     VE/PLOT VECT1
      *
❹     CREATE/1DHISTO 100 'test vector/hfill' 5 1. 6.
      max 100 2.5
❷     VE/HFILL VECT1 100
      histo/plot 100 b
      hi/de 100
      *
      create/1dhisto 100 'test put/contents' 10 1. 11.
❺     MAX 100 5.5
❺     MIN 100 0.5
❸     PUT/CONTENTS 100 VECT1
      histo/plot 100
```

❶ `VECT/PLOT` draws the statistic of the given vector.

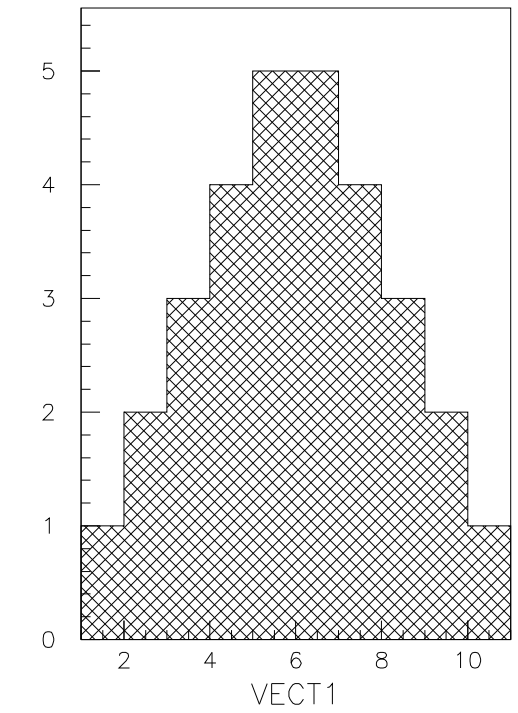
❷ `VECT/HFILL` fills an existing histogram (create with `1DHIST`) with the values taken from a vector. Note that the command `VECTOR/PLOT` can automatically book an histogram and fill it with the vector content.

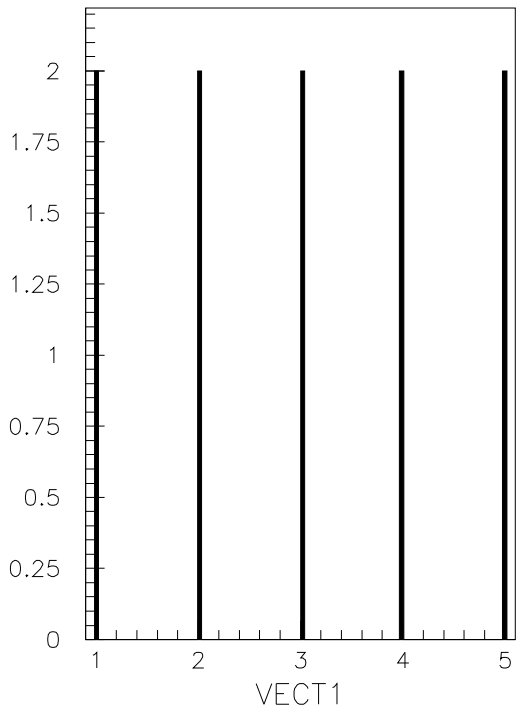
❸ `PUT/CONT` replaces the content of an histogram with the values of a vector.

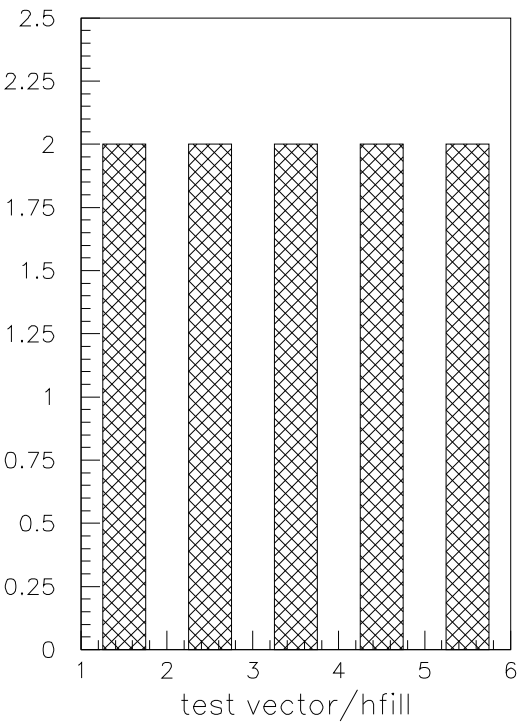
Note also:

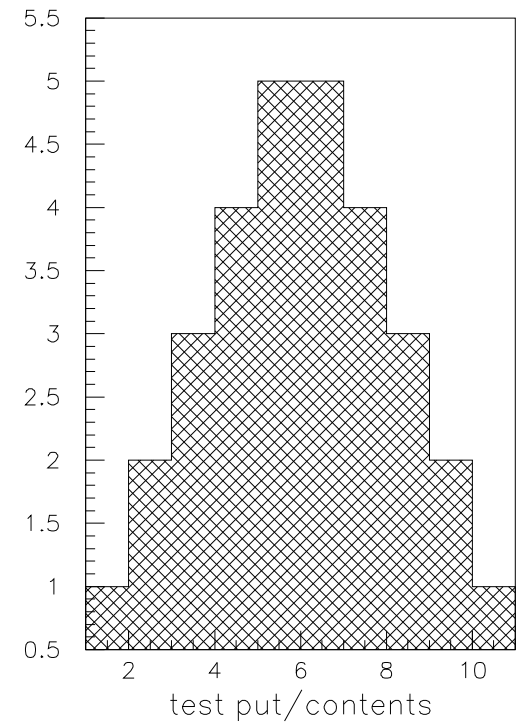
❹ Histograms are `hbook` objects. They can be created, like here, interactively in `paw` or in a batch `hbook` program.They can be stored in direct access files (we will see examples later).

❺ `MIN` and `MAX` define the minimum and maximum of an histogram. By default they are computed automatically.


5
4
3
2
1
0
2
4
6
8
10
VECT1


2
1.75
1.5
1.25
1
0.75
0.5
0.25
0
1
2
3
4
5
VECT1


2.5
2.25
2
1.75
1.5
1.25
1
0.75
0.5
0.25
0
1
2
3
4
5
6
test vector/hfill


5.5
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
2
4
6
8
10
test put/contents

3.4.5 Vector operations

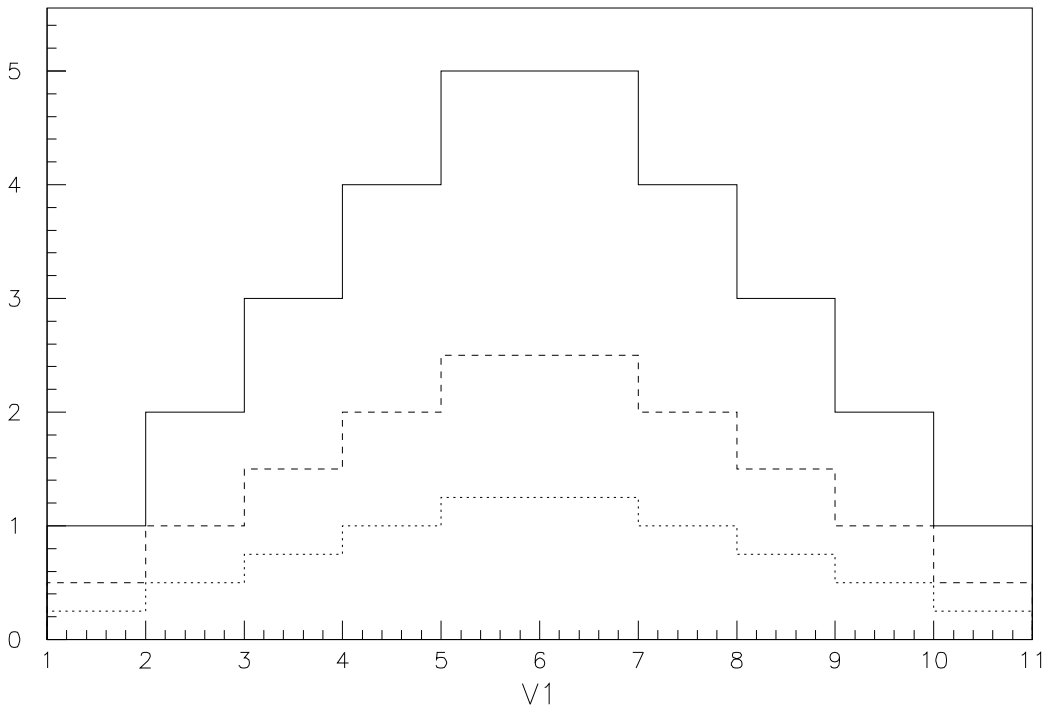
Vector operations

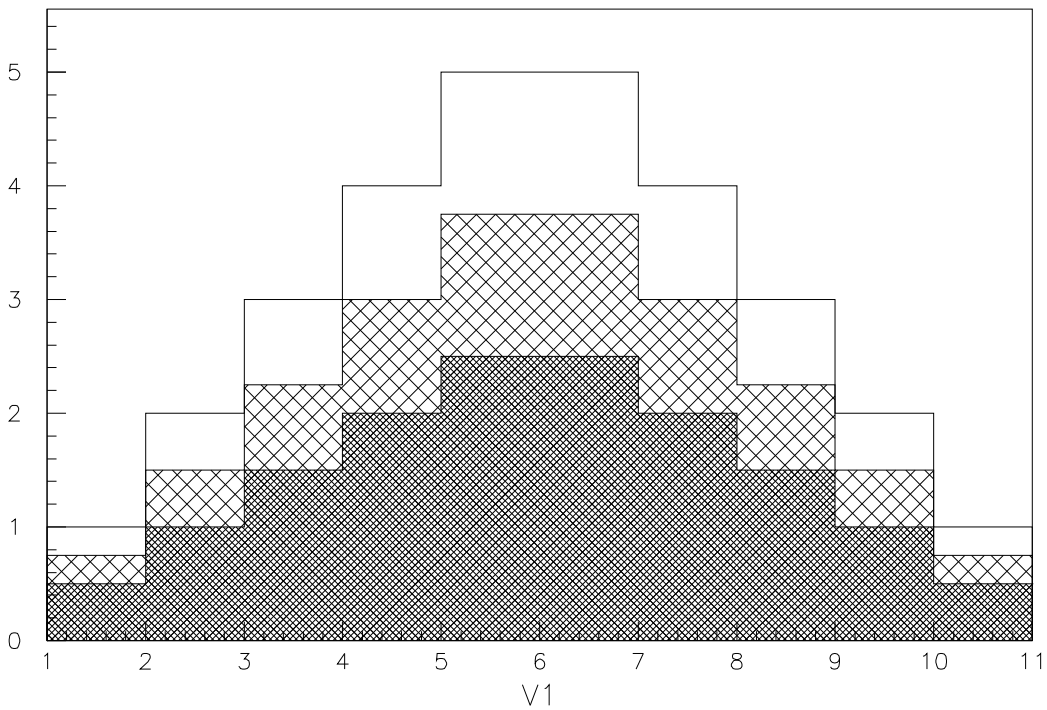
```
     zone 1 2
     ve/create V1(10) R 1 2 3 4 5 5 4 3 2 1
     vector/operations/vscale V1 0.5   V12
❶    VE/OP/VSCALE             V1 0.25  V14
     ve/dr V1
     ve/dr V12 ! S
     ve/dr V14 ! S
❶    VSUB                      V1 V14 V14M
     ve/dr V1
     set htyp 344
     ve/dr V14M ! S
     set htyp 144
     ve/dr V12  ! S
```

❶ Some simple operations are possible on vectors:

```
VBIAS     : Y(i) = a + X(i)
VSCALE    : Y(i) = a * X(i)
VADD      : Z(i) = X(i) + Y(i)
VMULTIPLY : Z(i) = Z(i) * Y(i)
VSUBSTRACT: Z(i) = X(i) - Y(i)
VDIVIDE   : Z(i) = X(i) / Y(i)
```

In these operations the resulting vectors are created automatically. Note that for more complicate operations like `SQRT` or trigonometric functions etc... , `sigma` must be used (we will see examples later).


0
1
2
3
4
5
1
2
3
4
5
6
7
8
9
10
11
V1


0
1
2
3
4
5
1
2
3
4
5
6
7
8
9
10
11
V1

3.4.6 Simple macro, with a loop and a VECTOR fit

Simple macro, with a loop and a VECTOR fit

```
      ve/create VECT(10,3)
❶     VE/READ VECT 'vector.data'
      *
      ve/print VECT(1:10,3)
      vbias vect(1:10,1) 0.5 vect(1:10,1)
      zon 1 2
      *
❹     DO IP = 2,3
        ve/draw vect(1:10,[ip])
❷ ❸     ORDER = [IP] - 1
❺       VECT/FIT VECT(1:10,1) VECT(1:10,[IP]) ! P[order] WS
❹     ENDDO
      ve/delete VECT
```

❶ The file `vector.data` previously created is read again in this example via the command `VECT/READ`. Note that it is not necessary to specify the format.

❷ This example shows the usage of variables in the macros (`IP`). The content of a variable can be accessed via:

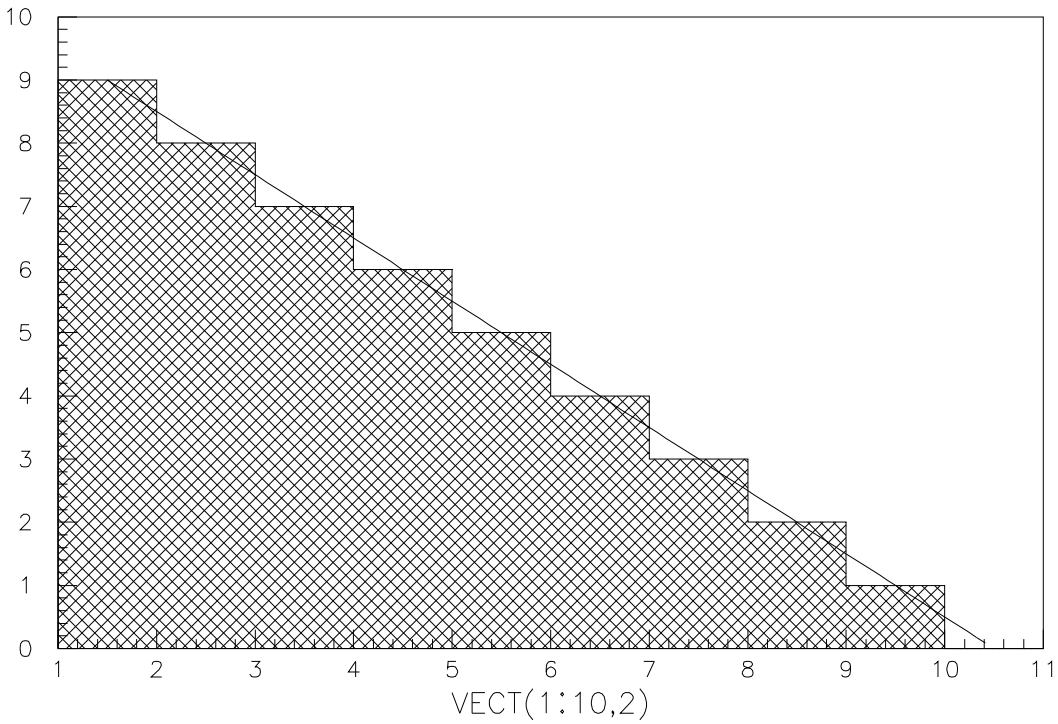
```
[variable]
```

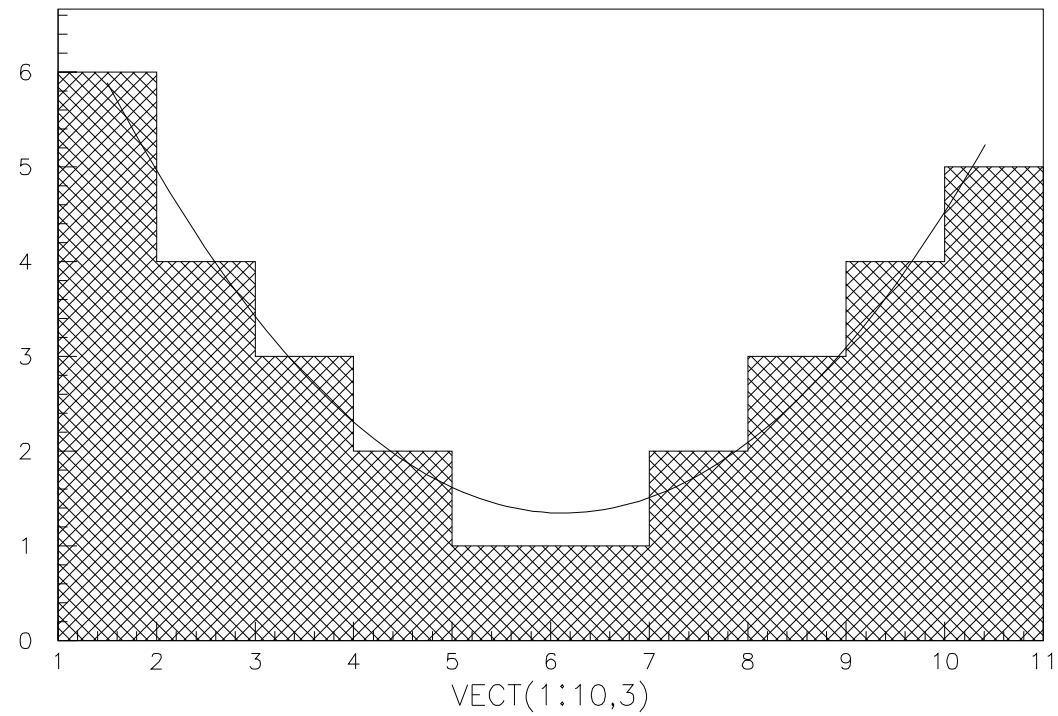
Note that the name of a variable in not case sensitive.

❸ Simple computations on variables are possible, like `i=[i]+1` or `a=[b]+2`. However it is not possible to do complex operations on variables. For this kind of computation vectors and `sigma` (or `comis`) must be used.

❹ Some controls statements are available in macros (see the complete list in the next example).

❺ It is possible to fit the vectors with functions. Here the function used for the fit is a polynome. The fitting mechanisms are very complete in `paw` and simple to use. All the details useful to use the commands `HIST/FIT` and `VECT/FIT` are given in the `paw` manual.


10
9
8
7
6
5
4
3
2
1
0
1 2 3 4 5 6 7 8 9 10 11
VECT(1:10,2)


6
5
4
3
2
1
0
1 2 3 4 5 6 7 8 9 10 11
VECT(1:10,3)

3.4.7 Macros flow control

There are several constructs available for controlling the flow execution, which include conditional statement blocks, several looping constructs and variable assignation.

Macro Statements

STATEMENT	DESCRIPTION
`MACRO mname par1=val1 ...`	begin macro `mname`
`EXEC mname par1 par2=val2 ...`	execute macro `mname`
`RETURN`	end of a macro
`READ par`	read macro parameter `par` from keyboard
`SHIFT`	control parameters list
`label:`	label (must terminate with a colon)
`GOTO label`	jump to `label`
`ON ERROR GOTO label`	resume at `label` on error condition
`OF ERROR`	temporarily deactivate the `ON ERROR GOTO` handling
`ON ERROR`	reactivate the latest `ON ERROR GOTO` handling
`IF logical_expression GOTO label`	conditional statement
`IF-THEN, ELSEIF, ELSE, ENDIF`	Macro flow control
`CASE, ENDCASE`	Macro flow control
`WHILE-DO, ENDWHILE`	Macro flow control
`REPEAT, UNTIL`	Macro flow control
`DO, ENDDO`	Macro flow control
`FOR, ENDFOR`	Macro flow control
`BREAKL`	Macro flow control
`EXITM`	Macro termination
par = arithmetic˙expression	assignment statement

Conditional statement

```
MACRO DOC1
  A = 10
  NN = 1.5
  TOT = [A]+[NN]
  IF [TOT] > 11 THEN
    MESSAGE Sum of [A] and [NN] is [TOT]
    AOK = correct
  ELSE
    AOK = wrong
  ENDIF
  MESSAGE KUIP arithmetic is [AOK].
RETURN
```

```
PAW > EXEC DOC1
Sum of 10 and 1.5 is 11.5
KUIP arithmetic is correct.
```

Unassigned variables cannot be substituted by their values.

```
MACRO DOC2
   A = 10
   NN = 1.5
   TOT = [A]+[XX]
   MESSAGE Result of sum is [TOT]
RETURN
```

```
PAW > EXEC DOC2
Result of sum is 10+[XX]
```

3.4.8 More on fits

Fit the function *sin* between 0 and 2π

```
❹    APPLICATION SIGMA
❹       alpha=array(100,0#2*PI)
❹       sina=sin(alpha)+rndm(alpha)*0.1
❹       err=array(100,0.1#0.1)
❹    EXIT
     zone 2 2
❶    V/FIT ALPHA(1:50) SINA(1:50) ERR(1:50) G
❶    V/FIT ALPHA SINA ERR P3
❶    V/FIT ALPHA SINA ERR P5
     v/create par(1) r 10.
❷    V/FIT ALPHA SINA ERR SINFIT.F ! 1 PAR
❸    V/PRI PAR
```

❶ In this macro two different types of predefined fits are used: Gaussian, Polynomial. As we will see later, the histograms fitting command `HISTO/FIT` has exactly the same syntax except that the 3 vectors are replaced by an unique parameter: The histogram identifier. On histograms some other minimization mechanisms are available via the commands `SPLINE`, `SMOOTH`, etc.. .

❷ It is also possible to defined specific functions. Here the function `SINFIT` is defined as follow:

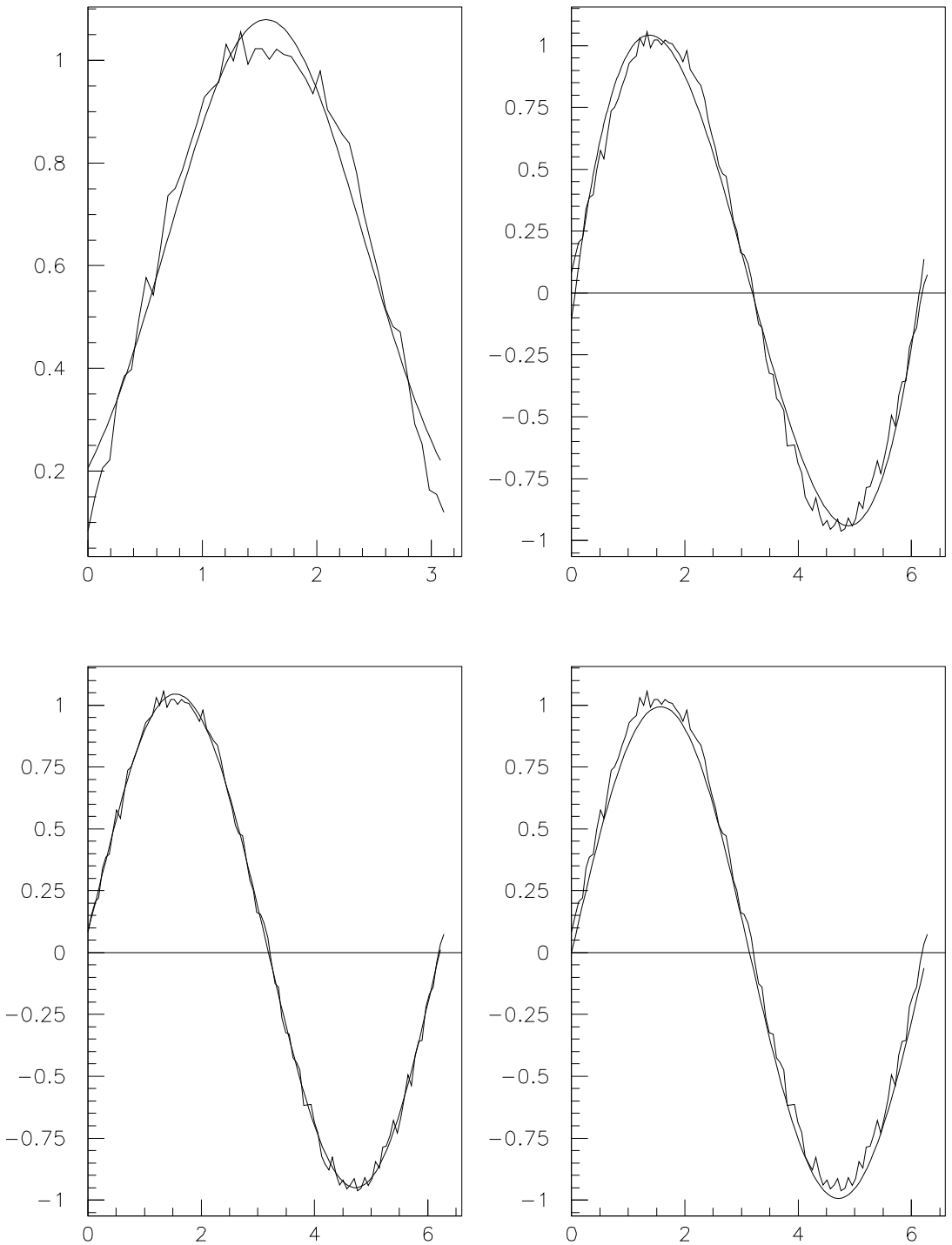
The function `SINFIT`

```
function sinfit(x)
common /pawpar/ par(1)
sinfit=par(1)*sin(x)
end
```

❸ This `VECT/PRI` shows that now `PAR(1)` is close to 1.

```
PAR(1) = 0.994221
```

❹ Vector initialization with `sigma`. We will see other `sigma` examples later.


1
0.8
0.6
0.4
0.2
0
1
2
3
1
0.75
0.5
0.25
0
−0.25
−0.5
−0.75
−1
0
2
4
6
1
0.75
0.5
0.25
0
−0.25
−0.5
−0.75
−1
0
2
4
6
1
0.75
0.5
0.25
0
−0.25
−0.5
−0.75
−1
0
2
4
6

Output of the Gaussian fit

```
    ********************************************
    *                                          *
    * Function minimization by SUBROUTINE HFITV  *
    * Variable-metric method                   *
    * ID =          0  CHOPT =                 *
    *                                          *
    ********************************************
Convergence when estimated distance to minimum (EDM) .LT.   .10E-03

FCN=   2221.676     FROM MIGRAD    STATUS=CONVERGED    239 CALLS      240 TOTAL

                    EDM=   .85E-05    STRATEGY= 1      ERROR MATRIX ACCURATE

 EXT PARAMETER                                   STEP         FIRST
 NO.   NAME        VALUE          ERROR          SIZE      DERIVATIVE
  1      P1        1.1316        .24808E-01    .64412E-03    .15289
  2      P2        1.5419        .21417E-01    .62018E-03    .42301E-01
  3      P3       -.76813        .17032E-01    .43531E-03   -.25527
```

Output of the Polynomial fit (P3)

```
CHISQUARE =  .2290E+02  NPFIT =  100
    ********************************************
    *                                          *
    * Function minimization by SUBROUTINE HFITV  *
    * Variable-metric method                   *
    * ID =          0  CHOPT =                 *
    *                                          *
    ********************************************
Convergence when estimated distance to minimum (EDM) .LT.   .10E-03

FCN=   49.31862     FROM MIGRAD    STATUS=FAILED        90 CALLS       91 TOTAL
                    EDM=   .79E-01  STRATEGY=1  ERROR MATRIX UNCERTAINTY= 70.2%

 EXT PARAMETER                APPROXIMATE        STEP         FIRST
 NO.   NAME        VALUE          ERROR          SIZE      DERIVATIVE
  1      P1       -.13523        .34965E-02    .00000E+00    5.6896
  2      P2        1.8729        .53793E-02    .00000E+00   -6.8643
  3      P3       -.86391        .32623E-03    .00000E+00    94.054
  4      P4        .91424E-01    .23105E-03    .00000E+00    6.6564

CHISQUARE =  .5137E+00  NPFIT =  100
```

Output of the Polynomial fit (P5)

```
     *******************************************
     *                                         *
     * Function minimization by SUBROUTINE HFITV  *
     * Variable-metric method                  *
     * ID =          0  CHOPT =                *
     *                                         *
     *******************************************
 Convergence when estimated distance to minimum (EDM) .LT.   .10E-03

 FCN=   7.164283     FROM MIGRAD    STATUS=FAILED       240 CALLS      241 TOTAL
                     EDM=   .19E+01    STRATEGY= 1      ERR MATRIX NOT POS-DEF

  EXT PARAMETER                APPROXIMATE        STEP         FIRST
  NO.   NAME        VALUE          ERROR          SIZE      DERIVATIVE
   1      P1        .46785E-01    .20704E-03    .68172E-07    32.993
   2      P2        .93224        .10038E-02    .74579E-06   -551.05
   3      P3        .20962        .33827E-03    .16770E-06   -3073.1
   4      P4       -.36899        .32674E-03    .29519E-06   -1084.4
   5      P5        .82836E-01    .19712E-04    .66269E-07    821.80
   6      P6       -.52834E-02    .12561E-05    .42267E-08   -5204.8

 CHISQUARE =  .7622E-01  NPFIT =  100
```

Output of the "`comis`" fit

```
     *******************************************
     *                                         *
     * Function minimization by SUBROUTINE HFITV  *
     * Variable-metric method                  *
     * ID =          0  CHOPT =                *
     *                                         *
     *******************************************
 Convergence when estimated distance to minimum (EDM) .LT.   .10E-03

 FCN=   32.13273     FROM MIGRAD    STATUS=CONVERGED     21 CALLS       22 TOTAL
                     EDM=   .92E-05    STRATEGY= 1      ERROR MATRIX ACCURATE

  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME        VALUE          ERROR          SIZE      DERIVATIVE
   1      P1        .99811        .13752E-01    .51510E-04   -.31172

 CHISQUARE =  .3246E+00  NPFIT =  100
```

3.4.9 VECTOR/READ using MATCH

*** VECTOR/READ VLIST FNAME [FORMAT OPT MATCH]**

```
❶    V/READ X,Y,Z match.dat 6x,3(F4.1) ! /Data/
     v/draw X
     v/draw Y ! S
     v/draw Z ! S
```

match.dat

```
❷    Title: File used for tests of the MATCH parameter in V/READ
     Data : 1.0 2.0 3.0
     Data : 2.0 3.0 4.0
     Data : 3.0 4.0 5.0
     Data : 4.0 5.0 6.0
❷    This line will be ignored by a V/READ with MATCH
     Data : 5.0 6.0 7.0
     Data : 6.0 7.0 8.0
     Data : 7.0 8.0 9.0
     Data : 8.0 9.0 1.0
     Data : 9.0 1.0 2.0
❷    End
```

This example shows how the MATCH parameter can be used in order to read only a subset of a file. MATCH is used to specify a pattern string, restricting the vector filling only to the records in the file which verify the pattern. Example of patterns:

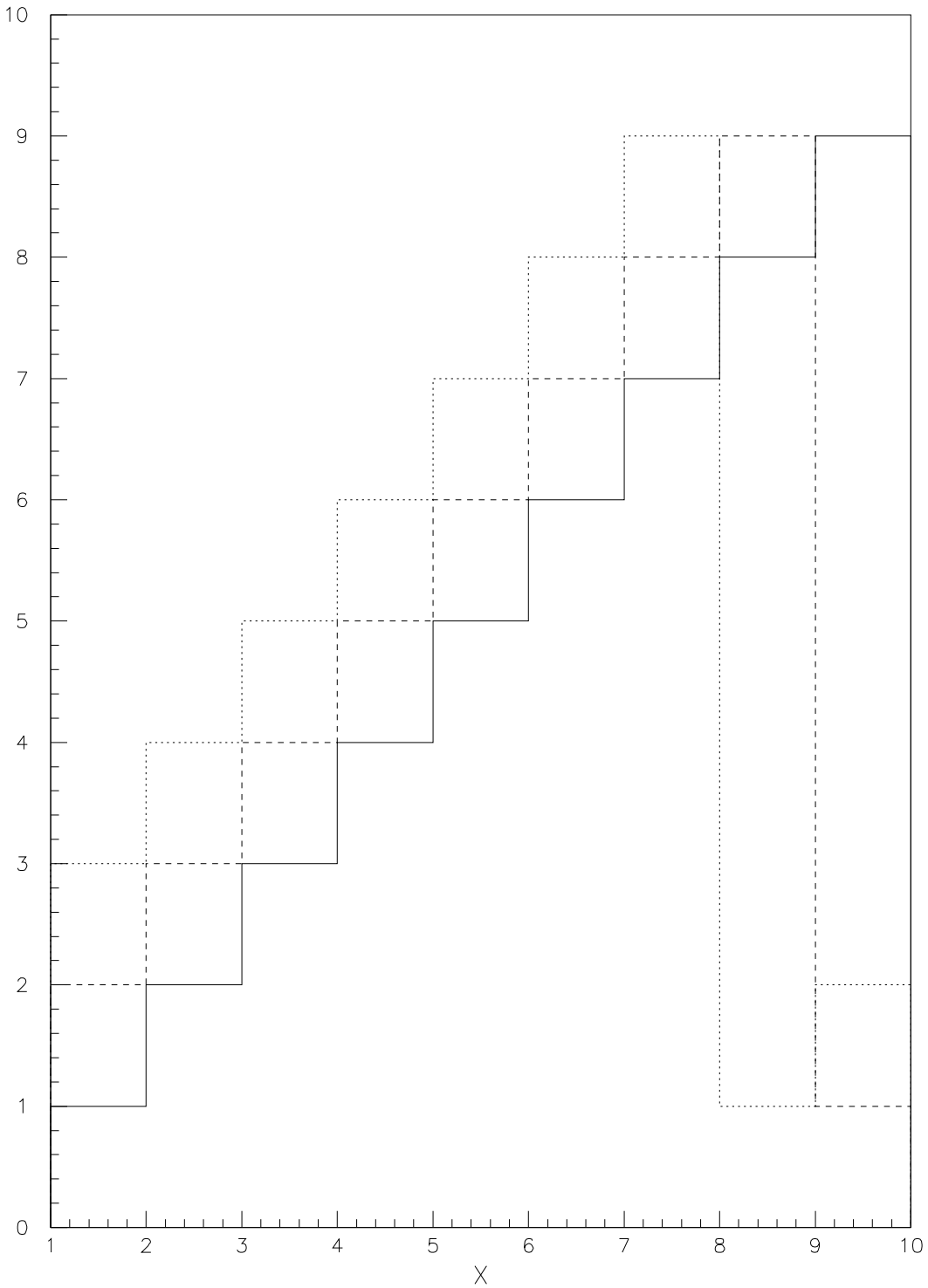
- /string/ match a string (starting in column 1)
- -/string/ do not match a string (starting in column 1)
- /string/(n) match a string, starting in column n
- /string/(*) match a string, starting at any column

❶ When the MATCH parameter is used, the command V/READ reads the file in two passes:

(a) to find how many lines should be read in order to create vectors with the proper length.

(b) to read the lines where the MATCH parameter is found.

❷ these lines are skipped during the reading pass.


10
9
8
7
6
5
4
3
2
1
0
1
2
3
4
5
6
7
8
9
10
X

3.5 Function drawing—Examples

3.5.1 Plot a few one-dimensional functions

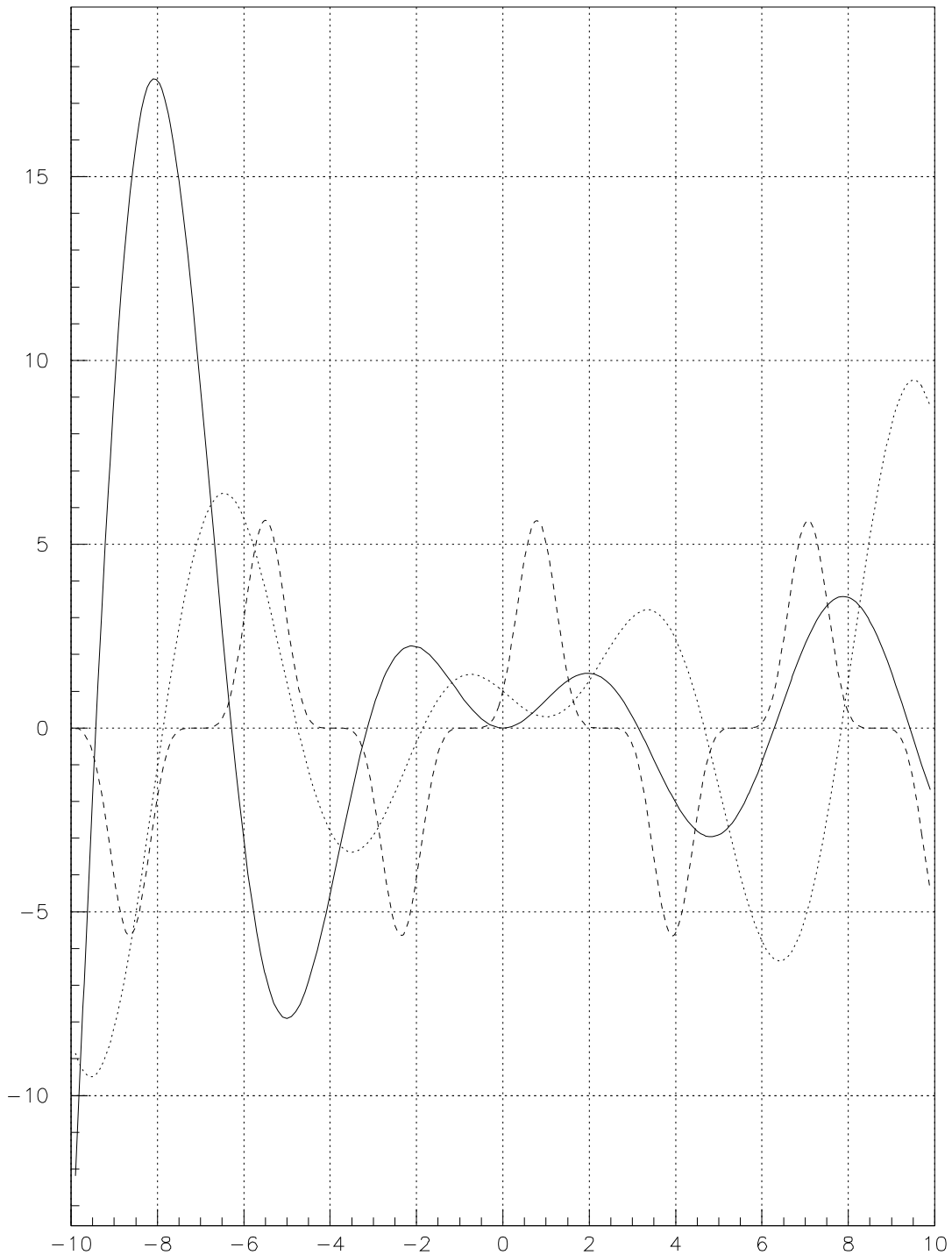
```
* FUNCTION/PLOT UFUNC XLOW XUP [ CHOPT ]

❸    OPT GRID
❶    FUNC/PLOT X*SIN(X)*EXP(-0.1*X)  -10. 10.
❷    SET DMOD 2
     func/plot (sin(x)+cos(x))**5    -10. 10. s
     set dmod 3
     func/plot (sin(x)/(x)-x*cos(x)) -10. 10. s
```

❶ `FUN/PLOT` allows to plot 2D functions. The character "`x`" or "`X`" is used as the variable name. The command `FUN1` is analog to `FUNC/PLOT` but it produces also an histogram with the value of the function. The number of steps used to compute the function along the X axis can be defined via the command `POINTS`.

Note also:

❷ `SET DMOD` allows to define the line type for the drawing the function. Note that `IGSET LTYP` cannot be used is this case because in the command `FUN/PLOT` many different lines are drawn (axes, boxes, etc ..). So a specific attribute must be used (`DMOD`) for the line type of a function or an histogram.

❸ `OPTION GRID` allows to have a grid on the subsequent plots.


15
10
5
0
−5
−10
−10
−8
−6
−4
−2
0
2
4
6
8
10

3.5.2 Plot a one-dimensional function and loop

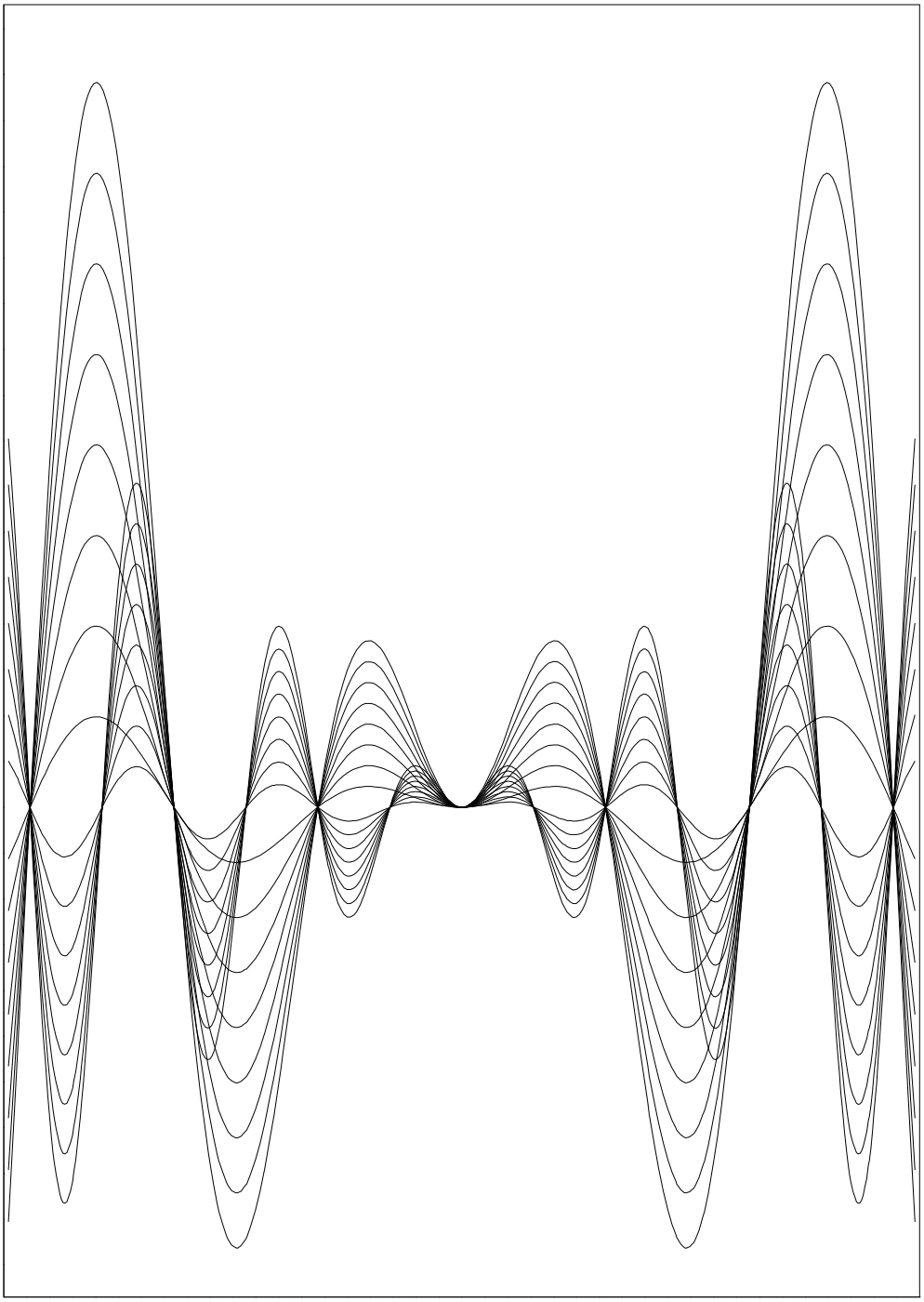
Plot a one-dimensional function and loop

```
❶ ❷  MACRO PLOT  1=8
     * The Macro parameter is the number of plots to be drawn.
     * the defaults is 8.
     set dmod 1
❸    SET XTIC 0.0001
❸    SET YTIC 0.0001
     set xval 100.
     set yval 100.
     opt utit
     fun/plot x*sin(x) -10 10
     fun/plot x*cos(x)*sin(x) -10 10 s
     a=[1]-1
     do i=[a],1,-1
       fun/plot x*sin(x)*[i]/[1] -10 10 s
       fun/plot x*cos(x)*sin(x)*[i]/[1] -10 10 s
     enddo
```

❶ In this example we can see that macros can have input parameters. These parameters can be positional, and they can be accessed in the macro via `[n]`, where `n` is the parameter number in the input list, or they can be specified by name and they are accessed like variables. The next example gives more details on the input parameters management.

❷ If one parameter (positional or not) needs to have a default value, the value can be specified on the `MACRO` line. At execution time this default value is taken if no value is given. Note that for parameters given by name, the default value on the line `MACRO` is mandatory.

❸ It is possible to define the geometry of a picture via the `SET` parameters described on the figure 8.3,. In this example the size of the tick marks is set to 0 (`XTIC` and `YTIC`). But it is not possible to specify: `SET XTIC 0` as, for the `SET` command, 0 means default value.



3.5.3 More on macro input parameters

Access to the parameter list

```
  MACRO P1                                        PAW > exe p1 23 9
  i = 10                                                23 squared is 529
❶ FOR p IN [*] [i] 1 2                                  9 squared is 81
    sq = [p] * [p]                                      10 squared is 100
    message [p] squared is [sq]                         1 squared is 1
  ENDFOR                                                2 squared is 4
```

Indexed positional parameters

```
  MACRO P2                                        PAW > exe p2 23 9 48
❷ DO i = 1, [#]                                         parameter 1 is 23
❸   message parameter [i] is [%i]                       parameter 2 is 9
  ENDDO                                                 parameter 3 is 48
```

❶ The * sign allows to access the list of input parameters.

❷ The # sign allows to access the number of input parameters.

❸ % allows to have indexed positional parameters.

3.5.4 Plot two-dimensional functions

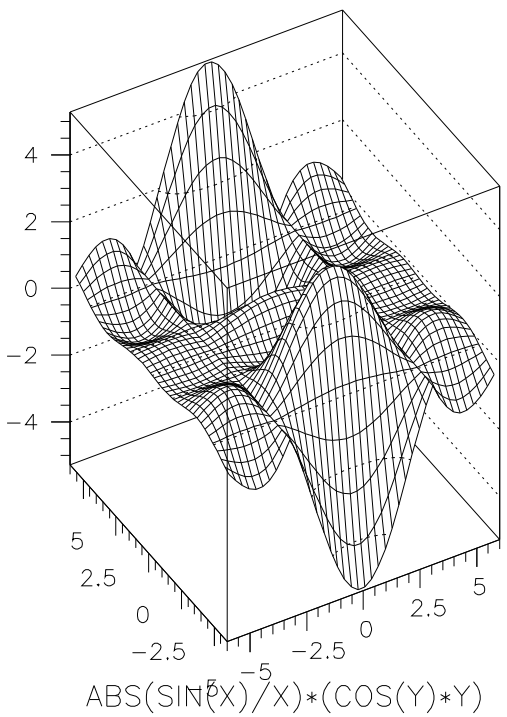
*** FUNCTION/FUN2 ID UFUNC NCX XMIN XMAX NCY YMIN YMAX [CHOPT]**

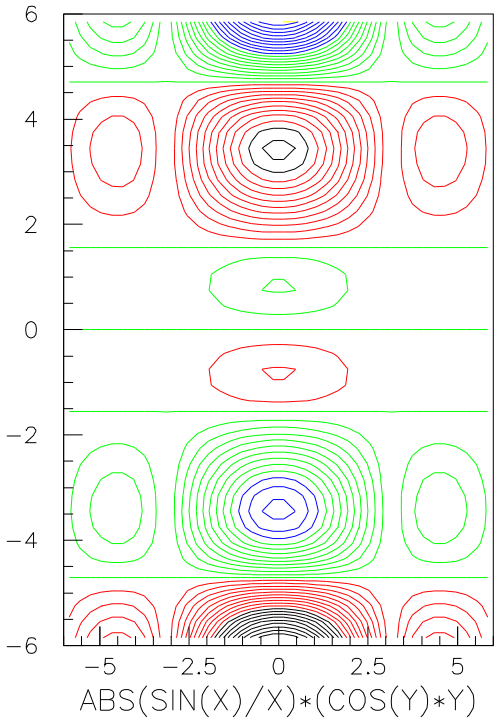
```
zone 2 2
❶ FUN2 10 ABS(SIN(X)/X)*(COS(Y)*Y) 40 -6 6 40 -6 6
contour 10 40 0
hi/de 10
fun2 10 x*sin(x)*y*sin(y) 40 -10. 10. 40 -10. 10. C
h/pl 10 surf4
```

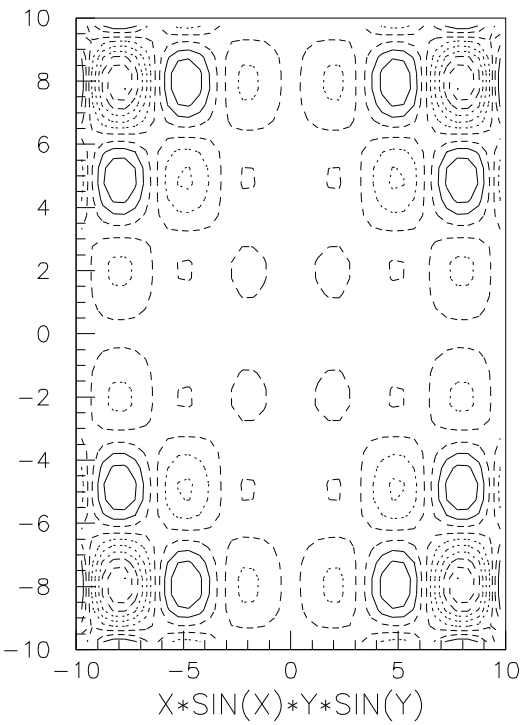
❶ The command `FUN2` allows to plot 2D functions and fill an histogram. The variables names are `X` and `Y`.

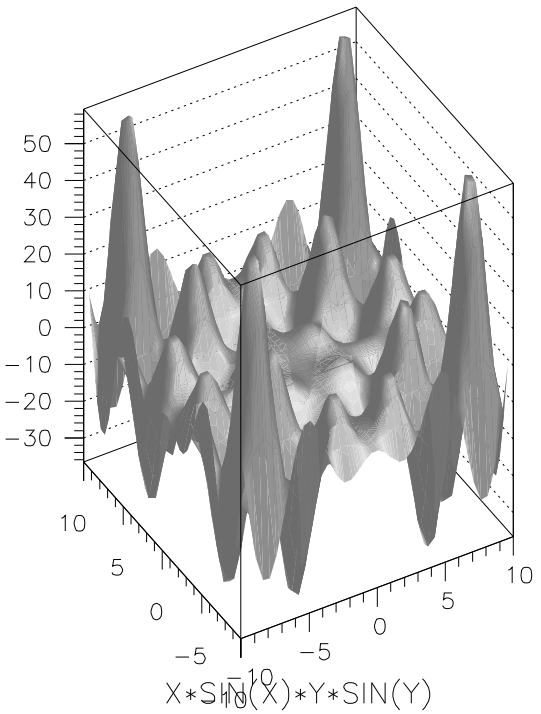
❷ It is possible to represent a 2D histogram in several ways :

(a) As a scatter plot.
(b) With proportional boxes.
(c) With a color table.
(d) As a surface plot.
(e) As a surface with color levels.
(f) As a surface with a contour plot on top.
(g) As a surface with Gouraud shading.
(h) As a lego plot.
(i) As a lego plot with colours or shading.
(j) As a line contour plot.
(k) As a table.
(l) As an arrows plot.


4
2
0
−2
−4
5
2.5
0
−2.5
−5
−5
−2.5
0
2.5
5
ABS(SIN(X)/X)*(COS(Y)*Y)


6
4
2
0
−2
−4
−6
−5
−2.5
0
2.5
5
ABS(SIN(X)/X)*(COS(Y)*Y)


10
8
6
4
2
0
−2
−4
−6
−8
−10
−10
−5
0
5
10
X*SIN(X)*Y*SIN(Y)


50
40
30
20
10
0
−10
−20
−30
10
5
0
−5
−10
−5
0
5
10
X*SIN(X)*Y*SIN(Y)

3.5.5 The Mandelbrot distribution

Calculate and plot (BOX option) the Mandelbrot distribution

```
❶    FUN2 10 mandel.f [1] -2.4 .8 [1] -1.2 1.2 ' '
❷    HI/PL 10 BOX
```

FORTRAN Routine MANDEL

```
      real function mandel(xp)
      dimension xp(2)
      data nmax/30/
      x=xp(1)
      y=xp(2)
      xx=0.
      yy=0.
      do n=1,nmax
         tt=xx*xx-yy*yy+x
         yy=2.*xx*yy+y
         xx=tt
         if (4..lt.xx*xx+yy*yy) go to 20
      enddo
   20 mandel=float(n)/float(nmax)
      end
```

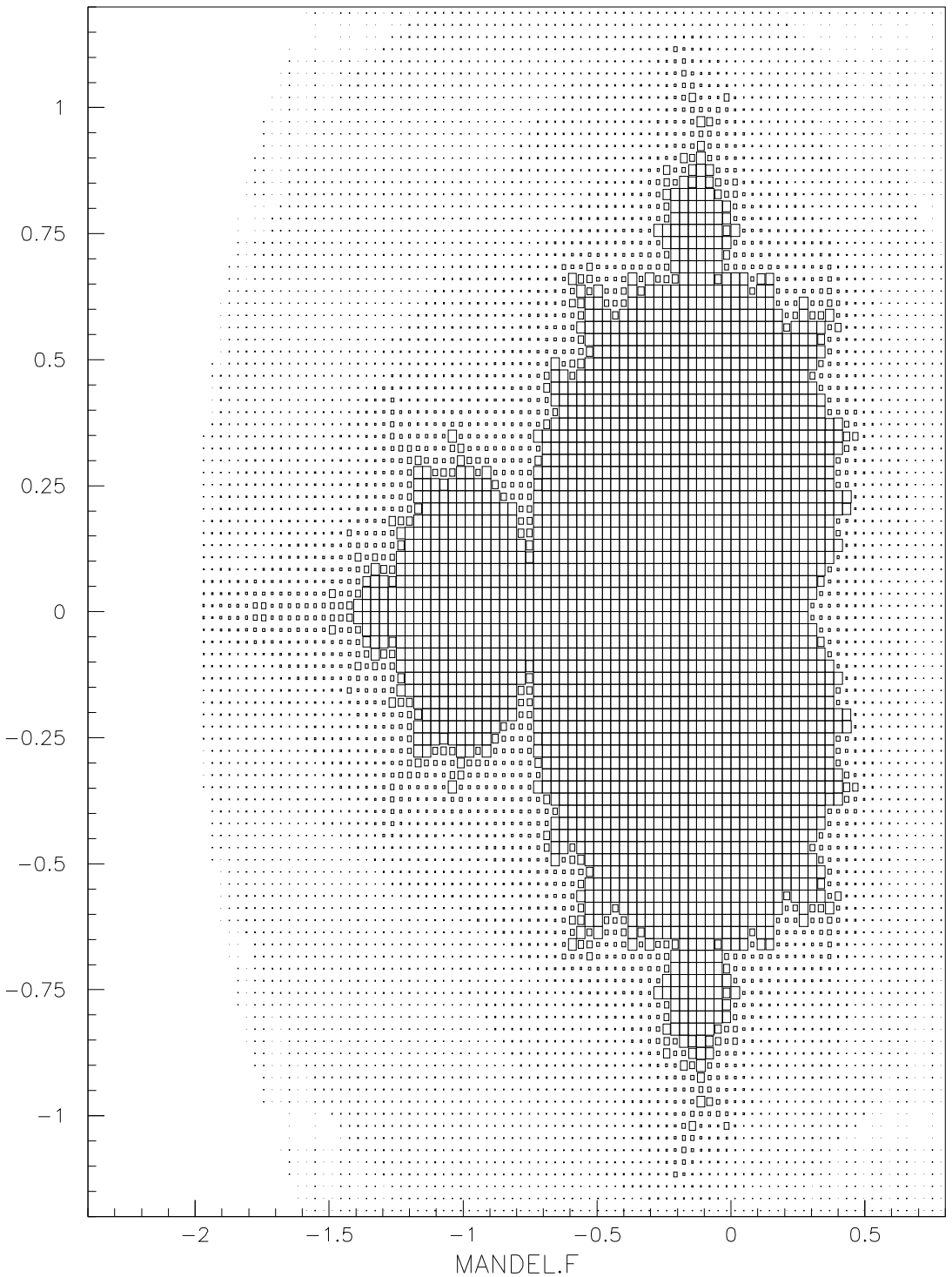
❶ This example shows one of the usages of `comis`. In this case, the name of the function to be plotted by `FUN2` is replaced by a `comis` FORTRAN function.

❷ `CHOPT=' '` in the command `FUN2` means to fill only the histogram without producing the plot which is by default a surface. The plot is produced by the command `HIST/PLOT`.

❸ The vector `XP` is an input parameter given by `FUN2`, for each cell, to the FORTRAN program. `XP` contains the `X` and `Y` coordinates of each cell. You can try to insert:

```
print*, XP
```

in `mandel.f` to see the values changing (in this case it is better to set the input parameter of the macro to 10).


1
0.75
0.5
0.25
0
−0.25
−0.5
−0.75
−1
−2
−1.5
−1
−0.5
0
0.5
MANDEL.F

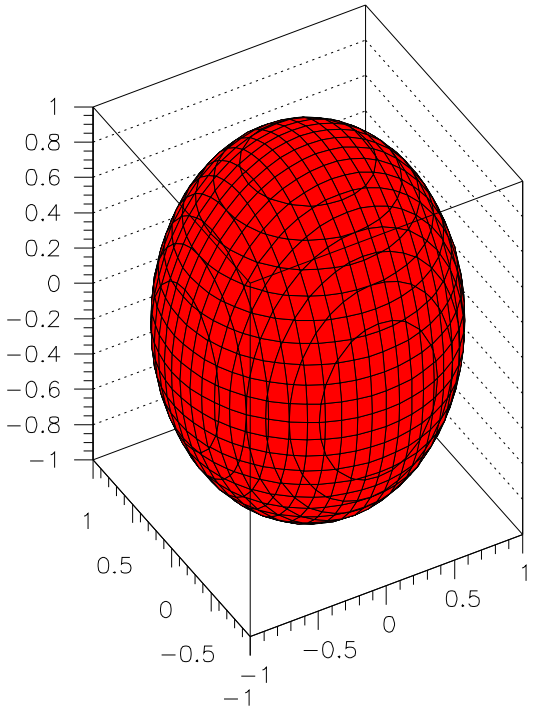
3.5.6 Three-dimensional functions drawing

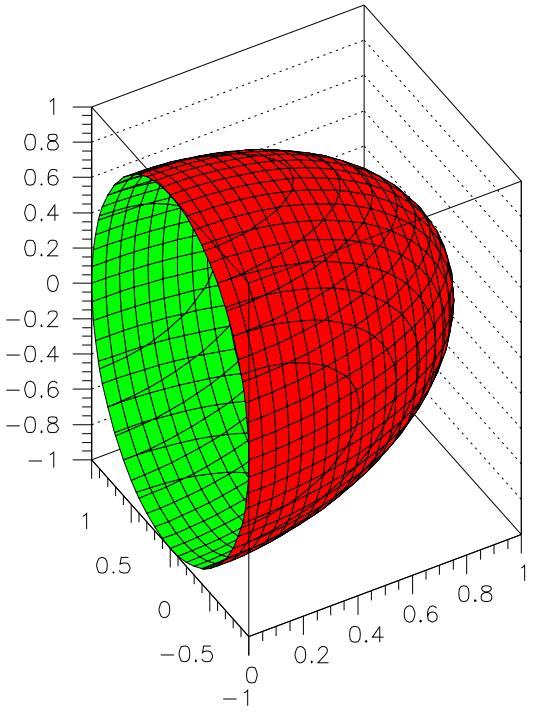
FUNCTION/DRAW **and** RANGE

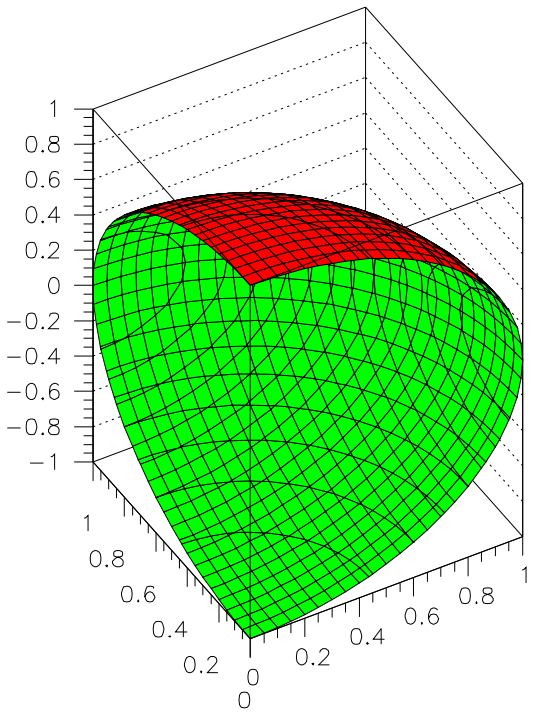
```
      zon 2 2
❶     FUN/DRAW X**2+Y**2+Z**2=1
❷     RANGE 0 1
❶     FUN/DRAW X**2+Y**2+Z**2=1
❷     RANGE 0 1 0 1
❶     FUN/DRAW X**2+Y**2+Z**2=1
❷     RANGE 0 1 0 1 0 1
❶     FUN/DRAW X**2+Y**2+Z**2=1
```

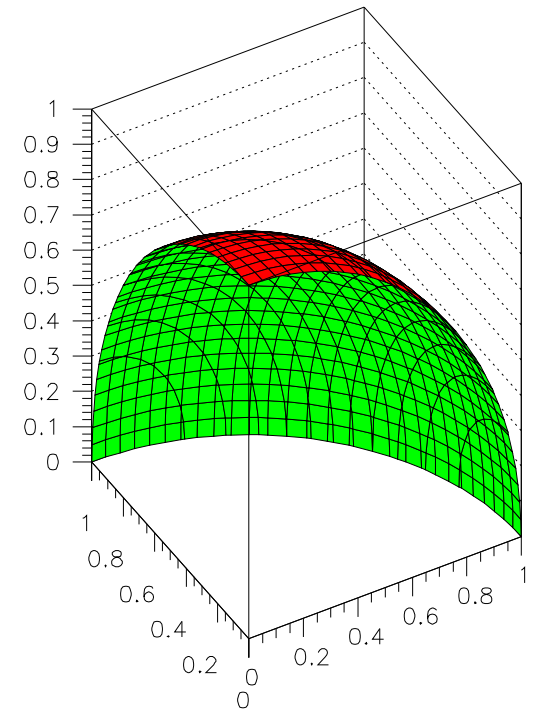
❶ This command draws a sphere of radius 1. The function can be also a `comis` program.

❷ The command `RANGE` modify the X, Y and Z range in which the function is drawn.

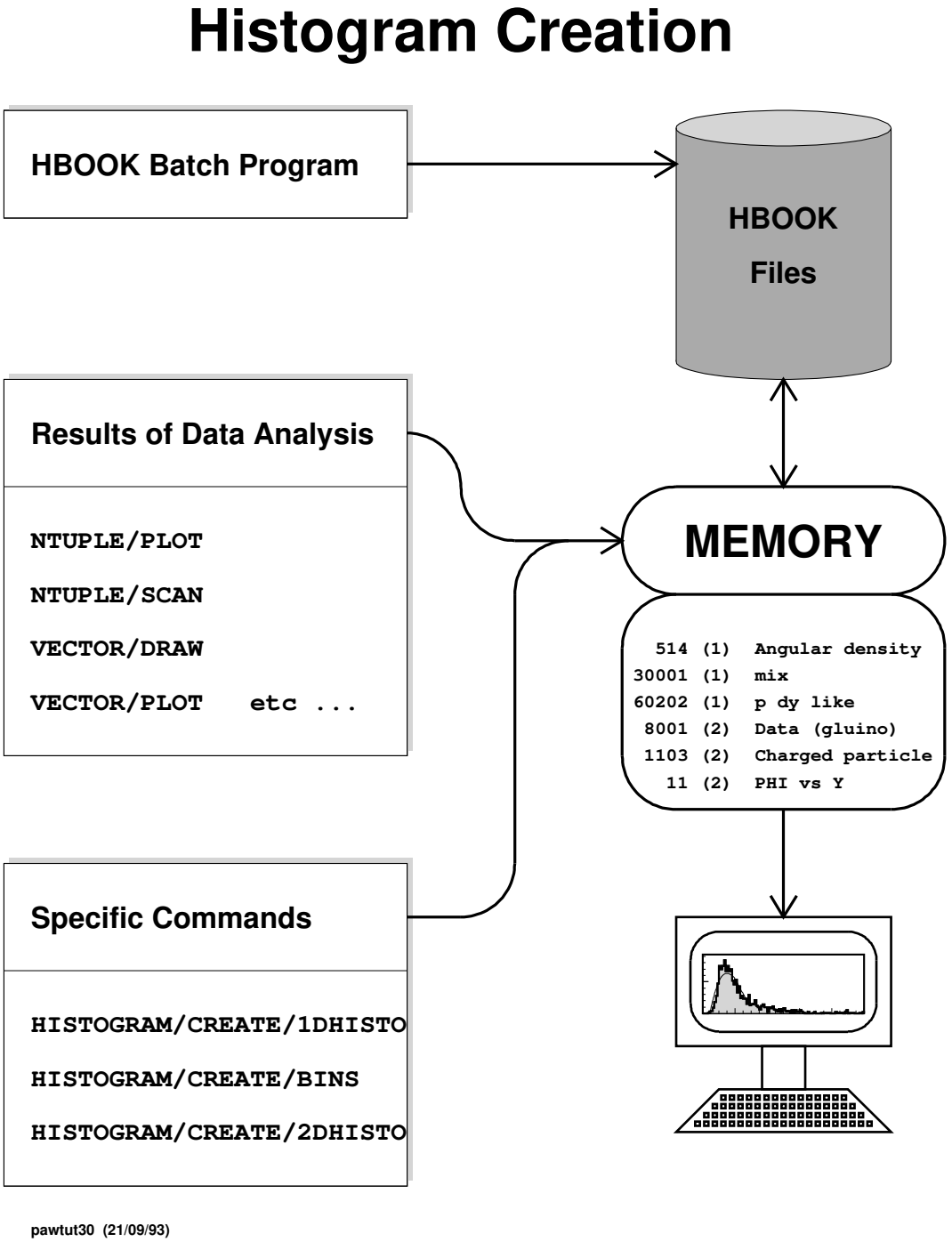

1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
1
0.5
0
−0.5
−1
−1
−0.5
0
0.5
1


1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
1
0.5
0
−0.5
−1
0
0.2
0.4
0.6
0.8
1

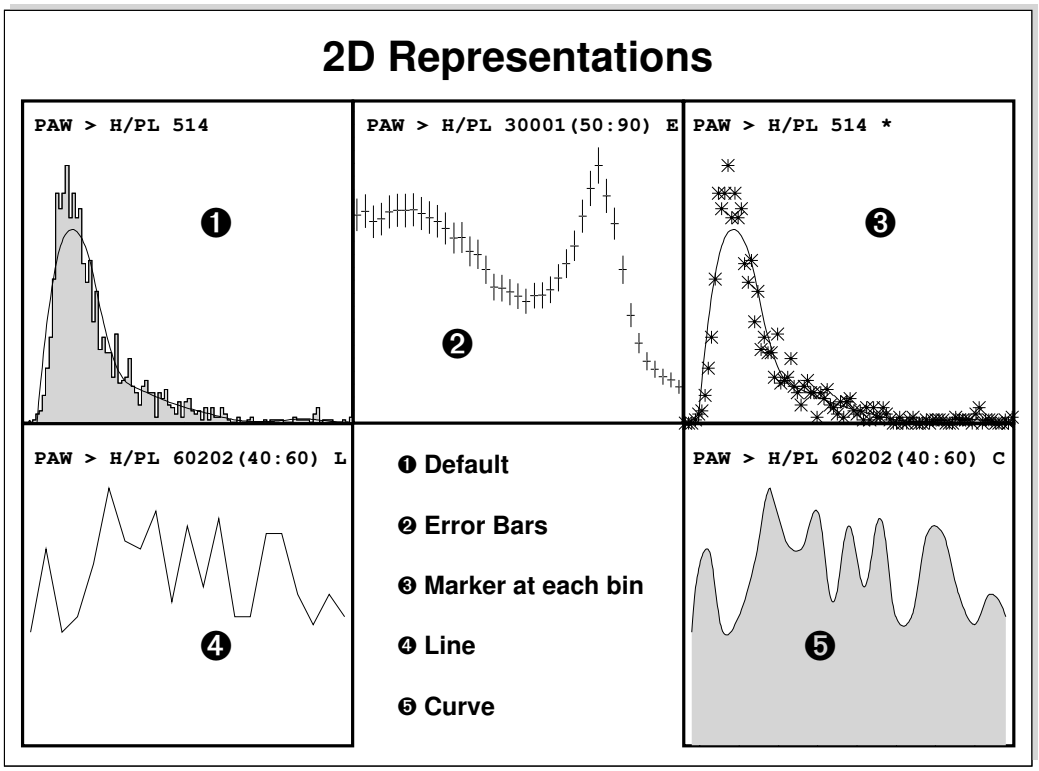

1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1

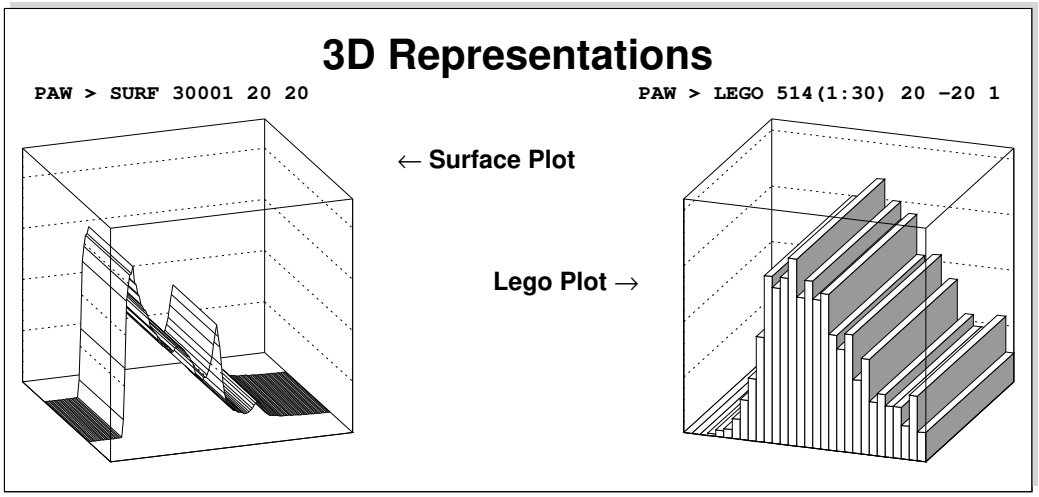

1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1

3.6 Histograms—Tutorial



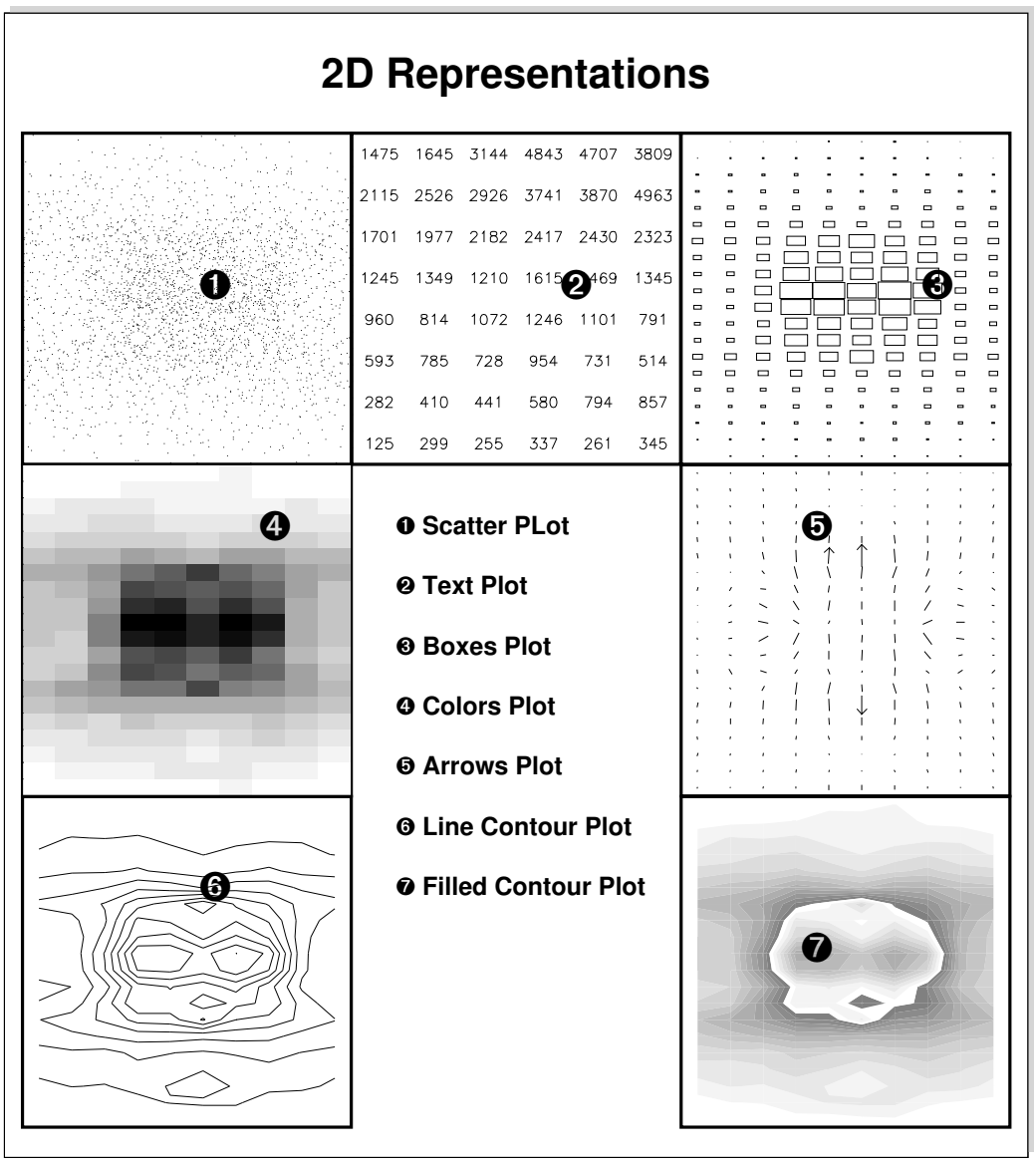
1D Histogram Drawing





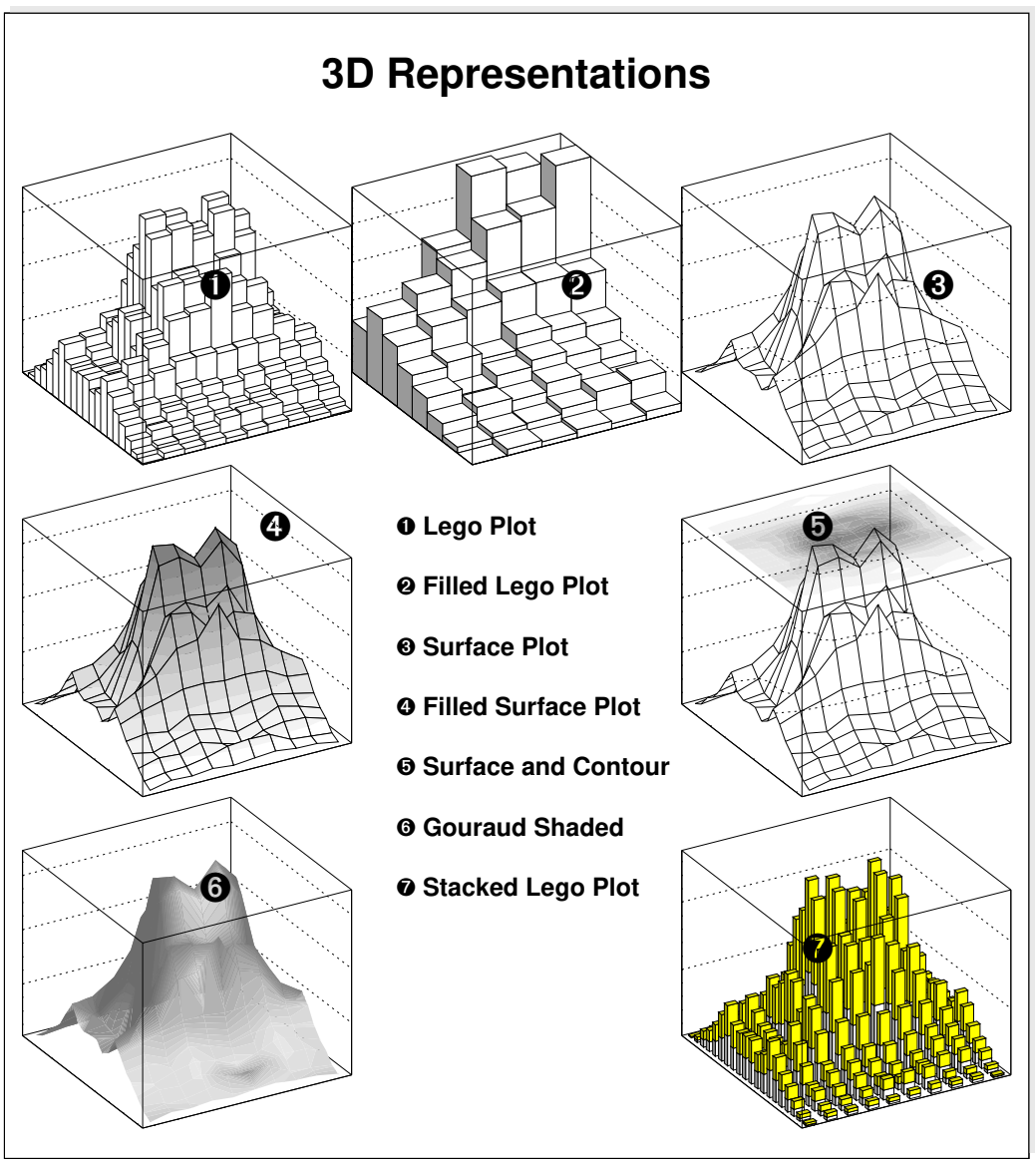
pawtut31 (21/09/93)

2D Histogram Drawing (1)

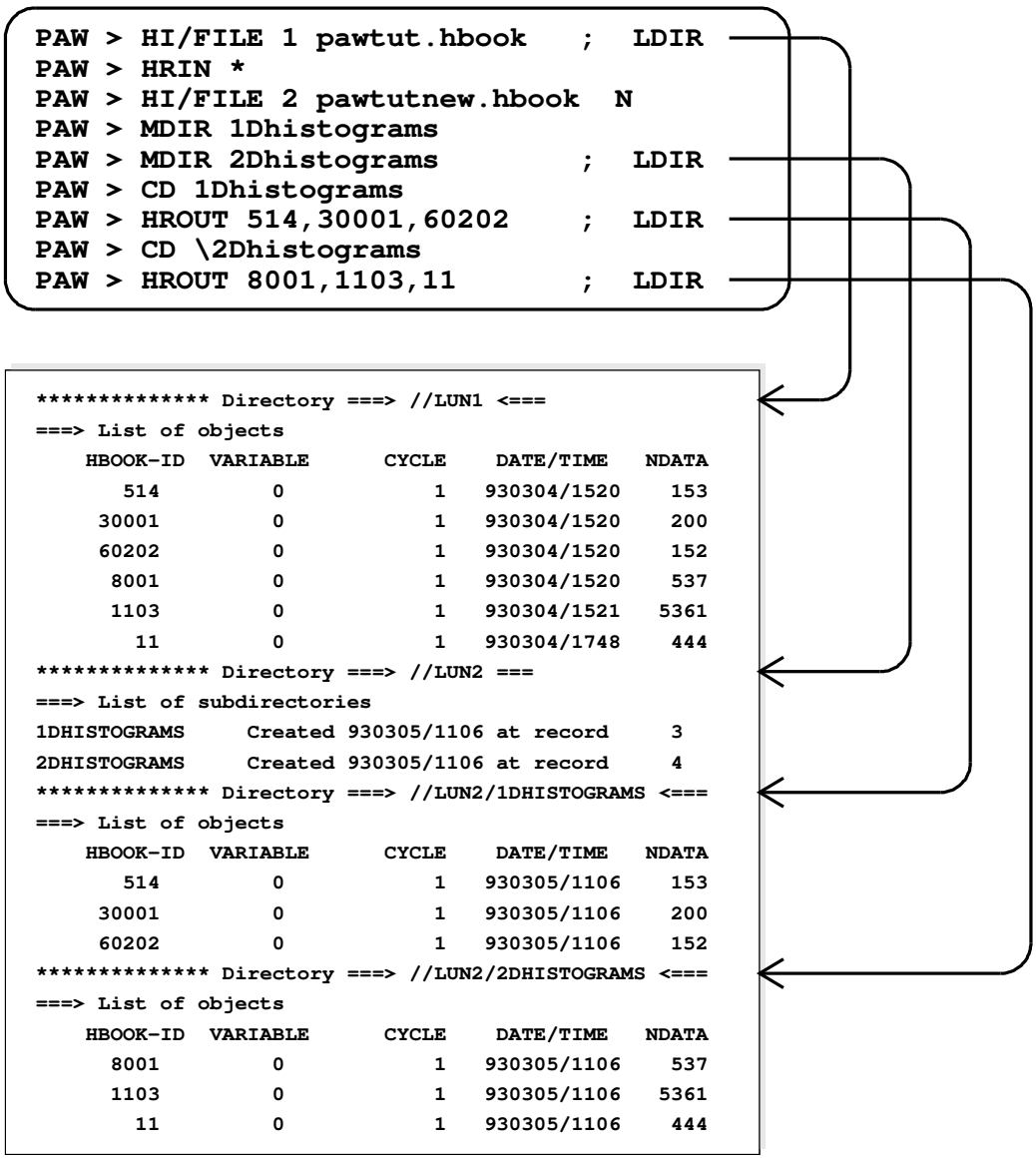


pawtut32 (21/09/93)

2D Histogram Drawing (2)



Histogram Archiving



pawtut34 (21/09/93)

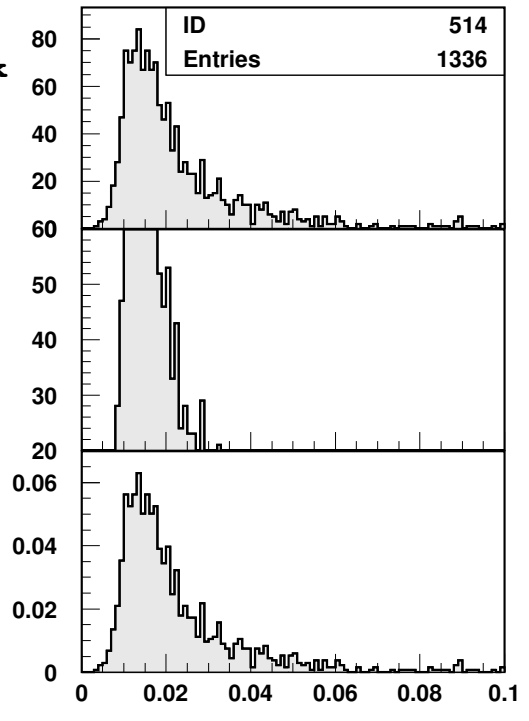
Histogram Operations

Basic Operations

`ADD ID1 ID2 ID3 [ C1 C2 OPTION ]`	Add histograms: ID3 = C1*ID1 + C2*ID2.
`SUBTRACT ID1 ID2 ID3 [ C1 C2 OPTION ]`	Subtract histograms: ID3 = C1*ID1 - C2*ID2.
`MULTIPLY ID1 ID2 ID3 [ C1 C2 OPTION ]`	Multiply histogram contents: ID3 = C1*ID1 * C2*ID2.
`DIVIDE ID1 ID2 ID3 [ C1 C2 OPTION ]`	Divide histograms: ID3 = C1*ID1 / C2*ID2.

MIN, MAX and NORMALIZE

```
PAW > H/FILE 1 pawtut.hbook
PAW > HISTO/PLOT 514
PAW > MIN 514 20
PAW > MAX 514 60
PAW > HISTO/PLOT 514
PAW > HISTO/del 514
PAW > HRIN 514
PAW > CD //pawc
PAW > NORMALIZE 514 1
PAW > HISTO/PLOT 514
```



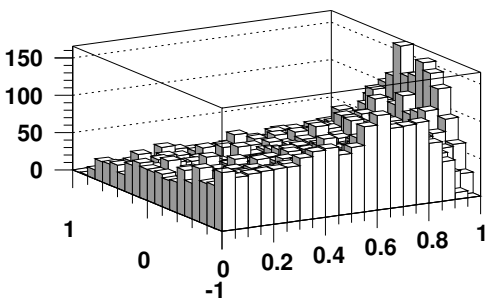
pawtut35 (21/09/93)

Histogram Projections

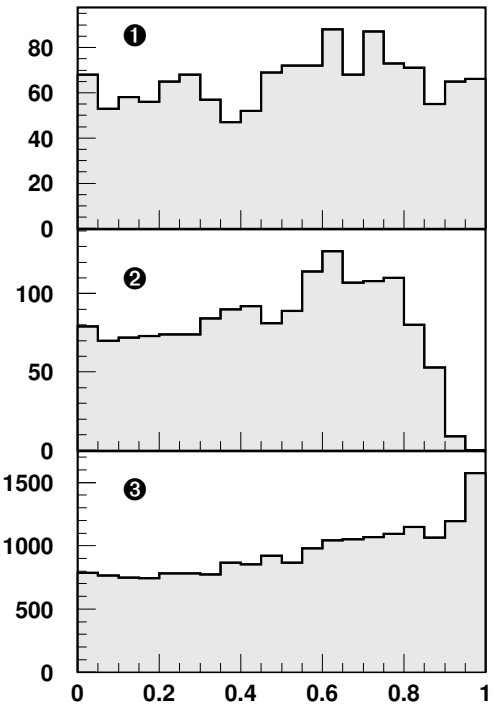
Basic Operations

`HISTOGRAM/CREATE/BANX ID YMIN YMAX`	Create a projection onto the x axis, in a band of y.
`HISTOGRAM/CREATE/SLIX ID NSLICES`	Create projections onto the x axis, in y-slices.
`HISTOGRAM/CREATE/PROX ID`	Create the projection onto the x axis.
`HISTOGRAM/PROJECT ID`	Fill all booked projections of a 2-Dim histogram.

Note that a BANY, SLIY, and PROY are also available




```
PAW > BANX 8001 0.5 1
PAW > SLIX 8001 20
PAW > PROX 8001
PAW > H/PROJECT 8001
PAW > H/PLOT 8001.banx    ❶
PAW > H/PLOT 8001.slix.1  ❷
PAW > H/PLOT 8001.prox    ❸
```

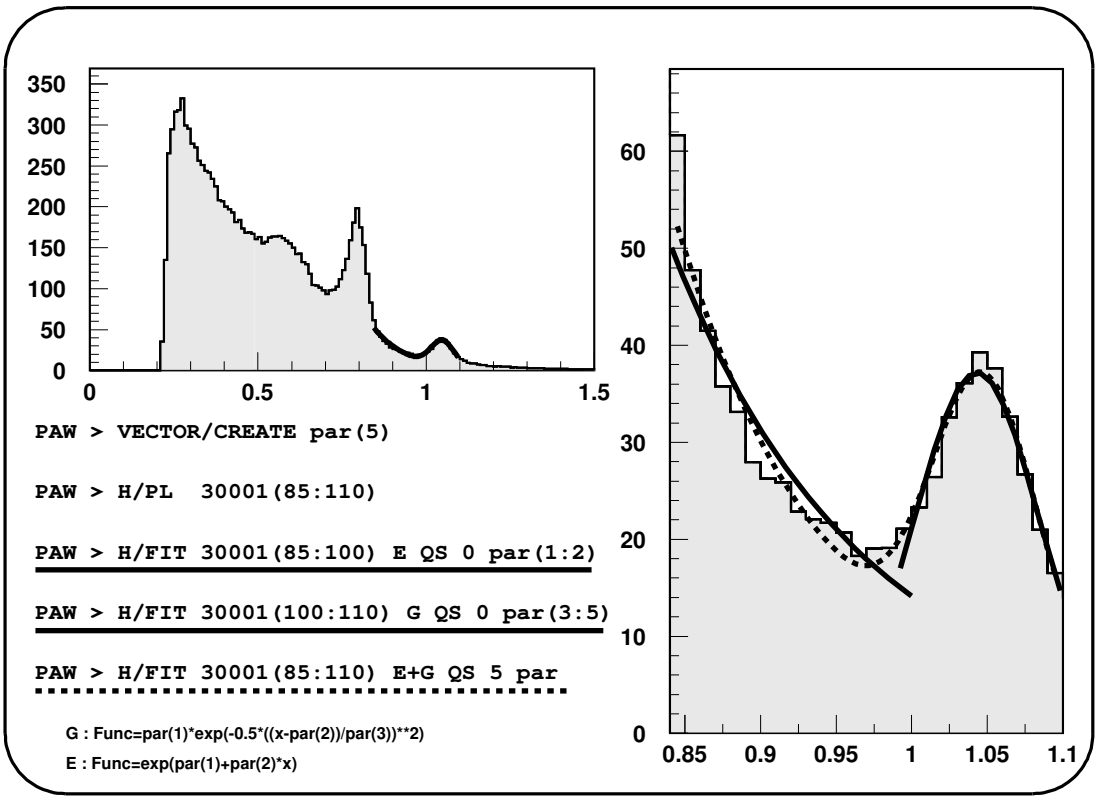


pawtut36 (21/09/93)

Histogram Fitting

The HISTOGRAM/FIT command

```
HISTOGRAM/FIT ID FUNC [ CHOPT NP PAR STEP PMIN PMAX ERRPAR ]
ID          Histogram Identifier
FUNC        Function name
CHOPT       Options
NP          Number of parameters
PAR         Vector of parameters
STEP        Vector of steps size
PMIN        Vector of lower bounds
PMAX        Vector of upper bounds
ERRPAR      Vector of errors on parameters
```



pawtut37 (21/09/93)

Histogram Smoothing (1)

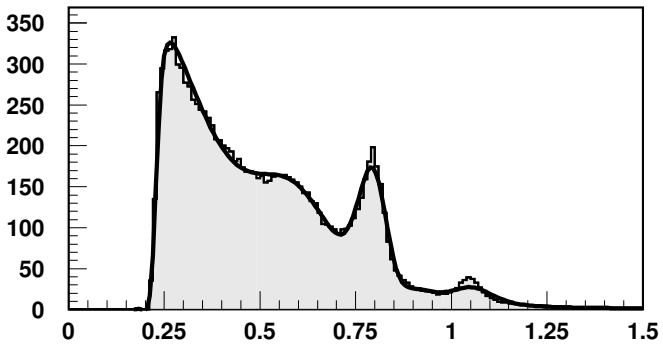
The HISTOGRAM/OPERATIONS/SMOOTH command

```
HISTOGRAM/OPERATIONS/SMOOTH ID [ OPTION SENSIT SMOOTH ]
ID          Histogram or Ntuple Identifier
OPTION      Options
SENSIT      Sensitivity parameter
SMOOTH      Smoothness parameter
```

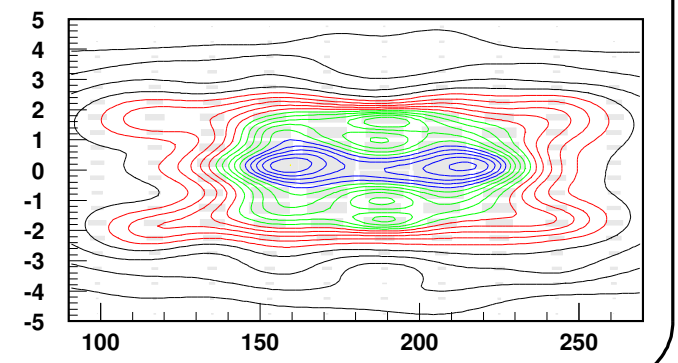
For multiquadric smoothing, SENSIT controls the sensitivity to statistical fluctuations. SMOOTH controls the (radius of) curvature of the multiquadric basis functions.

For spline smoothing, SENSIT and SMOOTH control the no. of knots (= 10 * SENSIT) and degree of splines (= SMOOTH + 2) (thus if SENSIT and SMOOTH are at their default values a 10-knot cubic spline is used).

```
PAW > SMOOTH 30001
```




```
PAW > SMOOTH 11
```



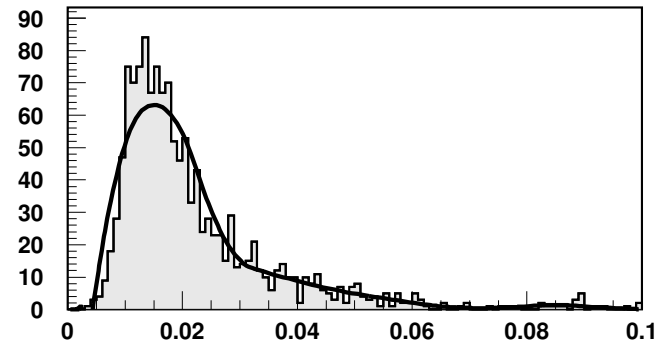
pawtut38 (21/09/93)

Histogram Smoothing (2)

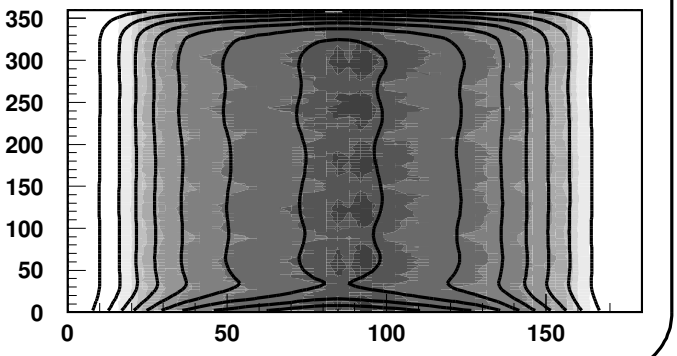
The HISTOGRAM/OPERATIONS/SPLINE command

```
HISTOGRAM/OPERATIONS/ID [ ISEL KNOTX KX ]
ID          Histogram or Ntuple Identifier
ISEL        Option flag
KNOTX       Number of knots
KX          Degree of the spline
```

```
PAW > SPLINE 514

PAW > H/PLOT 514
```




```
PAW > CONTOUR 1103 ! 3

PAW > SPLINE 1103

PAW > CONTOUR 1103 ! 2S
```

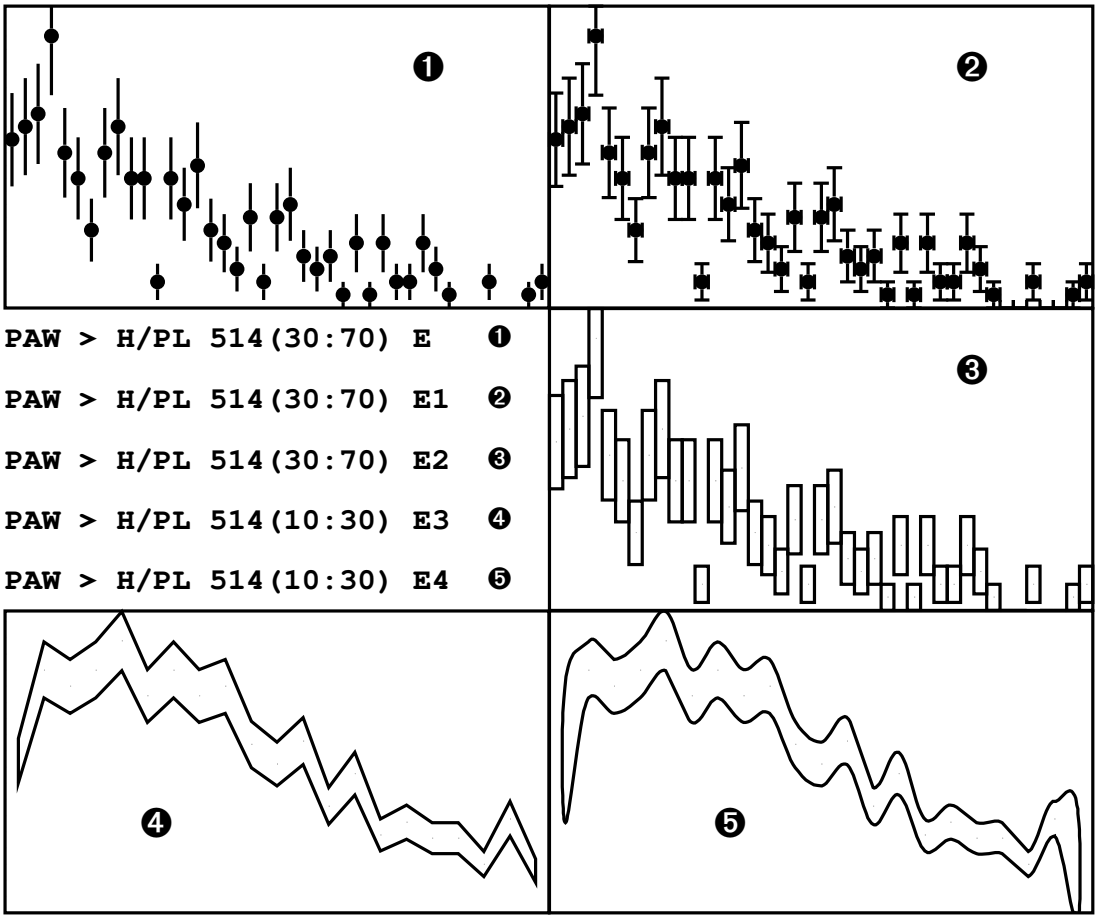


pawtut39 (21/09/93)

Error bars Drawing (1)

The command HISTOGRAM/PLOT provides five different options in order do draw histograms with error bars:

- ❶ **Simple error bars and current marker.**
- ❷ **Like ❶ plus small lines at the end of the error bars.**
- ❸ **Error rectangles.**
- ❹ **A filled area through the end points of the vertical error bars.**
- ❺ **A smoothed filled area through the end points of the vertical error bars.**



pawtut65 (21/09/93)

Error bars Drawing (2)

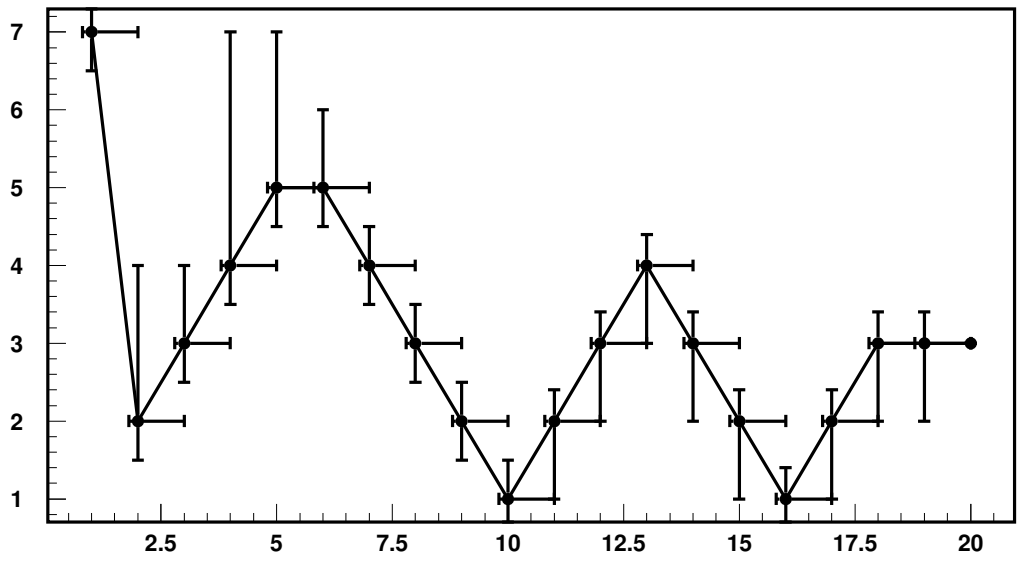
Two commands are provided to draw error bars from data inside vectors:

❶ GRAPHICS/HPLOT/ERRORS X Y EX EY N [ISYMB SSIZE CHOPT]

❷ GRAPHICS/HPLOT/AERRORS X Y EXL EXU EYL EYU N [ISYMB SSIZE CHOPT]

The first one allows to draw symmetric error bars on X and Y directions.
The second one is more general, it allows to define asymmetric errors
both on X and Y directions.

```
040PAW > V/CR X(20) R 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
PAW > V/CR Y(20) R 7 2 3 4 5 5 4 3 2  1 2 3 4 3 2 1 2 3 3 3
PAW > V/CR EXL(20) R 19*0.2 0.
PAW > V/CR EXU(20) R 19*1 0.
PAW > V/CR EYL(20) R 10*0.5 9*1 0.
PAW > V/CR EYU(20) R 3 2 1 3 2 1 .5 .5 .5 .5 9*.4 0.
PAW > GRAPH 20 X Y
PAW > AERROR X Y EXL EXU EYL EYU 20 20 .2 1
```



pawtut66 (21/09/93)

3.7 Histograms—Examples

3.7.1 Histograms creation

Creation of one and two dimensional histograms

```
      zon 1 2
      function/fun1 100 htfun1.f 100. 0. 1.
      1dh  110 'Test 1-dim Histo' 100 0. 1. 1000.
❶     CALL UROUT.F(5000)
❻     FUN/FUN2 200 HTFUN2 25. 0. 1. 25. 0. 1. C
      hi/li
❸ ❺   HISTOGRAM/FILE 1 PAWHISTS.HBOOK 1024 N
❹     HROUT 0
```

The FORTRAN Routine HTFUN1

```
      function htfun1(x)
      data c1,c2,xm1,xm2,xs1,xs2
     +/1.,0.5,0.3,0.7,0.07,0.12/
      a1=-0.5*((x-xm1)/xs1)**2
      a2=-0.5*((x-xm2)/xs2)**2
      x1=c1
      x2=c2
      if(abs(a1).gt.0.0001)x1=c1*exp(a1)
      if(abs(a2).gt.0.0001)x2=c2*exp(a2)
      htfun1=x1+x2
      end
```

The FORTRAN Routine HTFUN2

```
❻     function htfun2(x,y)
      htfun2=100*htfun1(x)*htfun1(y)
      end
```

The FORTRAN Routine UROUT

```
      subroutine urout(nev)
      do i=1,nev
❷        x=HRNDM1(100,I)
❷        CALL HFILL(110,X,0.,1.)
      enddo
      end
```

❶ In this example `comis` is used in the simplest way, via the command `CALL` (`CALL UROUT.F`). This command just calls the FORTRAN routine given as parameter and executes it.

❷ It is possible to call several routines of the CERN library. `HELP CALL` gives the list of available routines (see next page). Here the routines `HRNDM1` and `HFILL` (to fill an histogram) are called by `UROUT`.

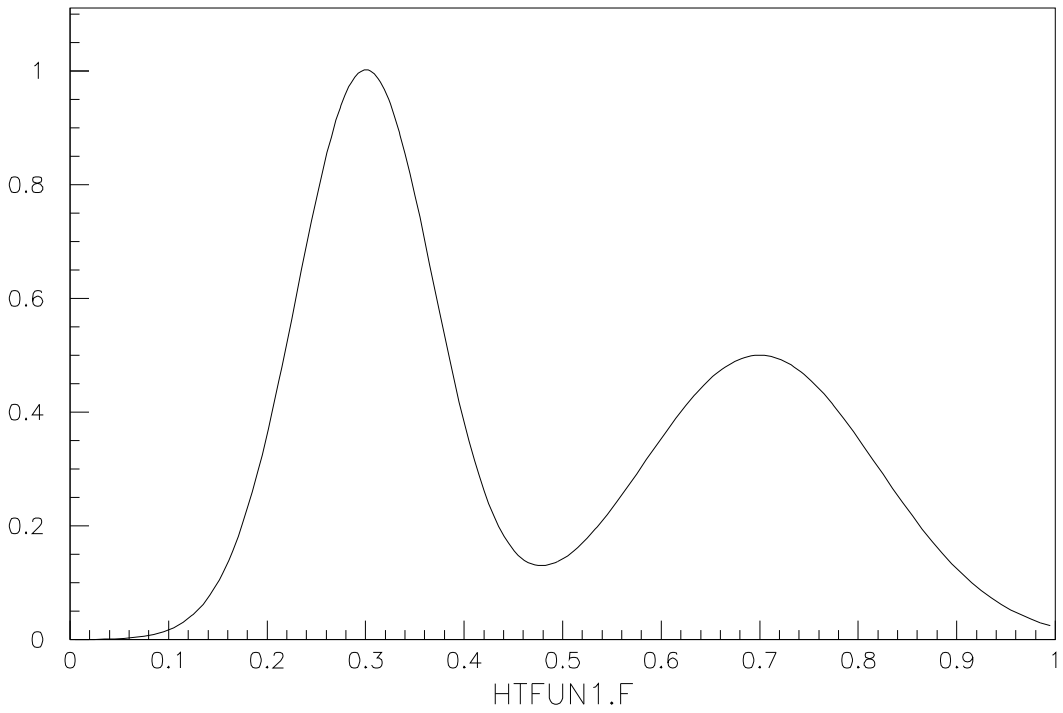
❸ It is possible to store the histograms in memory into a direct access file opened via the command `HIST/FILE`. Here `CHOPT=N` means: "create a New `hbook` file". If the first parameter (`LUN`) is 0 the next free logical unit will be used.

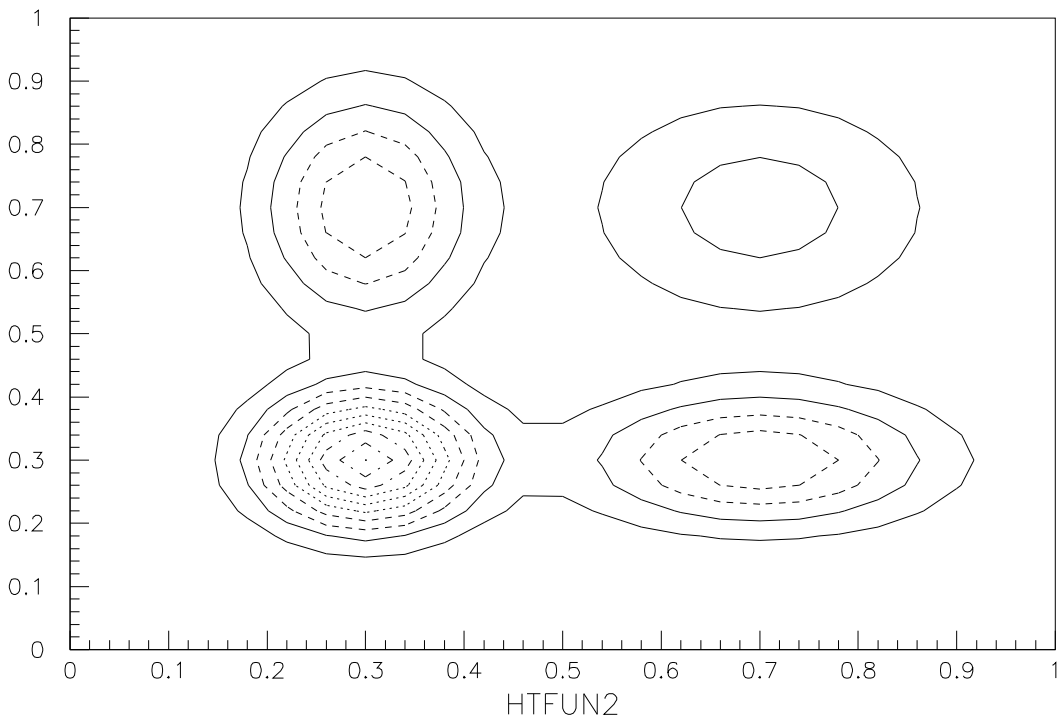
❹ To store an histogram in a file it is enough to execute the command `HROUT`. `HROUT 0` (or `HROUT *`) stores all the histograms currently in memory.

❺ Several files can be attached via `HIST/FILE` during a `paw` session. To change the current file it is enough to execute `CD //LUNn` where "`n`" is the first parameter given to `HI/FILE`. Note that the command `LD //` gives the list of all the files currently attached. Each attached direct access file is similar to a directory (cf UNIX).

❻ `HTFUN2` is in the file `htfun1.f`. That is why it can be invoked without the extension `.f` because it has been compiled during the `CALL` to `htfun1`.

Most of the time, the histograms are created and filled outside `paw` in batch programs calling `hbook` directly, and after interactively analyzed with `paw`.


1
0.8
0.6
0.4
0.2
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
HTFUN1.F


1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
HTFUN2

The following routines from the CERN Program Library can be called:

From HBOOK

```
HBOOK1,HBOOK2,HBOOKN,HFILL,HF1,HPRINT,HDELET,HRESET
HFITGA,HFITPO,HFITEX,HPROJ1,HPROJ2,HFN,HGFIT
HROPEN,PAOPEN,PACLOS,PAREAD,PAWRIT,HCDIR,HGIVEN
HTITLE,HBFUN1,HBFUN2,HRNDM1,HRNDM2,HBARX,HBARY
HPAK,HPAKE,HUNPAK,HGIVE,HGN,HGNF,HGNPAR,HF2,HFF1,HFF2
HRIN,HROUT,HI,HIE,HIX,HIJ,HIF,HIDALL,HNOENT,HX,HXY
HTITLE,HCOPY,HSTATI,HBPROF,HOPERA,HIDOPT,HDERIV
HMAXIM,HMINIM,HMAX,HMIN,HSUM,HNORMA,HREND
HEXIST,HRGET,HRPUT,HSCR,HFIND,HCX,HCXY,HLABEL
HBPROX,HBPROY,HBANDX,HBANDY,HBSLIX,HBSLIY
HBOOKB,HBSTAT,HDIFF,HUNPKE,HREBIN,HERROR
HOUTPU,HERMES,HISTDO,HFUNC,HIJXY,HXYIJ,HLPOS,HFC1
HSPLI1,HSPLI2,HMDIR,HLDIR,HLOCAT,HFITH,HFITV,HFINAM
HBNT,HBNAME,HBNAMC,HFNT,HFNTB,HGNT,HGNTF,HGNTV,HBSET
```

From HPLOT

```
HPLOT,HPLSYM,HPLERR,HPLEGO,HPLNT,HPLSUR,HPLSOF
HPLABL,HPLSET,HPLGIV,HPLOC,HPLTOC,HPLNEW,HPLOPT
```

From ZEBRA

```
FZIN,FZOUT,FZFILE,FZENDI,FZENDO
RZCDIR,RZLDIR,RZFILE,RZEND,RZIN,RZOUT,RZVIN,RZVOUT
RZOPEN,RZIODO
```

From KUIP

```
KUGETV,KUDPAR,KUVECT,KILEXP,KUTIME,KUEXEL,KUPROS
KUNWG,KUCMD,KUGUID,KUNDPV,KUPAR,KUPVAL,KUACT
```

From HIGZ

```
IPL,IPM,IFA,IGTEXT,IGBOX,IGAXIS,IGPIE,IGRAPH,IGHIST
IGARC,IGLBL,IGRNG,IGMETA,IGSA,IGSET,IRQLC,IRQST,ISCR
ISELNT,ISFAIS,ISFASI,ISLN,ISMK,ISVP,ISWN,ITX,ICLRWK
IGPAVE,IGTERM
```

From KERNLIB

```
VZERO,UCOPY,RNDM,RANNOR,LENOCC,SBIT0,SBIT1,SBYT
JBIT,JBYT,UCTOH,UHTOC,CLTOU,CUTOL
ERF,ERFC,FREQ,PROB
```

The following common blocks may be referenced

```
/PAWC/, /QUEST/, /KCWORK/, /PAWPAR/, /PAWIDN/
/HCFITS/, /HCFITD/
```

3.7.2 Read histograms from file and plot

Read histograms from file and plot

```
❶    HISTOGRAM/FILE 1 PAWHISTS.HBOOK
❷    HRIN  *
❸    LDIR
❸    HI/LIST
❺    ZON 2 2
     hi/pl 100
     set htyp 244
     hi/pl 110
❺    ZONE 1 2 2 'S'
     hi/plot 200
❹    CLOSE 1
```

❶ In this example the existing file `PAWHISTS.HBOOK` is attached in `READ-ONLY` mode.

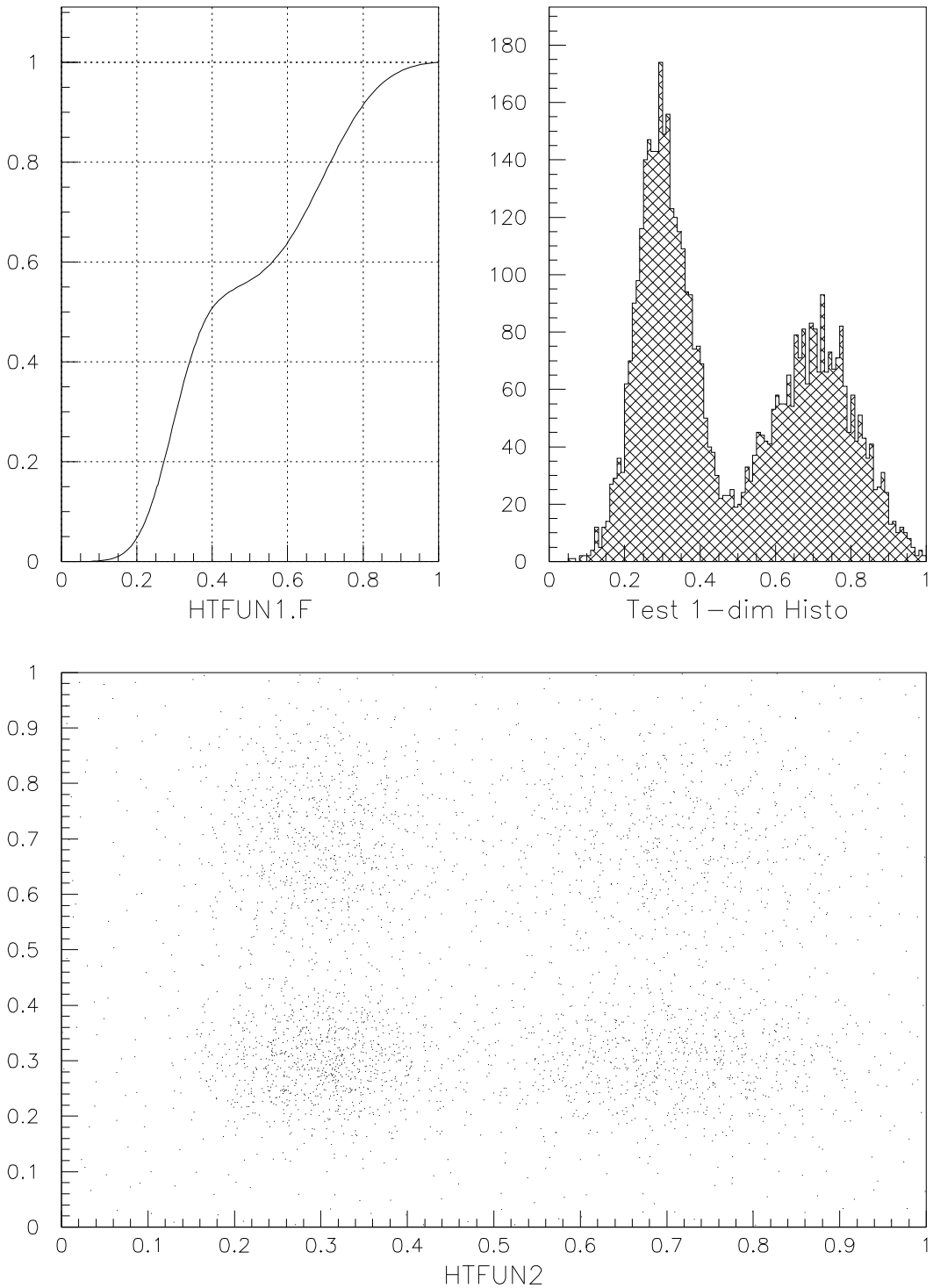
❷ The command `HRIN *` (or `HRIN 0`) gets all the histograms from the file `PAWHISTS.HBOOK` into the memory. Note that commands like `HIST/PLOT` take automatically the histogram from the file if it is not already in memory.

❸ Both `LDIR` and `HI/LIST` give the list of the histograms. `LDIR` is the generic command to list the content of a `zebra` file. It has no knowledge about the objects stored in the file that's why it cannot retrieve the histogram names. The `hbook` specific command `HIST/LIST` is able to find informations on the histogram like the histogram title and the histogram type. On the next page is given the output of these two commands.

❹ To release an histogram file it is enough to do `CLOSE n` where "`n`" is the logical unit number used by the command `HIST/FILE` (the first parameter of this command).

Note also:

❺ The usage of the command `ZONE`. It is used two times to define zones with different sizes.


HTFUN1.F
Test 1−dim Histo
HTFUN2

Output of LDIR

```
************** Directory ===> //LUN1 <===

                Created 911128/1154  Modified 911128/1154


===> List of objects
    HBOOK-ID  CYCLE   DATE/TIME   NDATA   OFFSET    REC1    REC2
       100       1   911128/1154    152       1       3
       110       1   911128/1154     85     153       3
       200       1   911128/1154    778     238       3

 Number of records =    2  Number of megawords =  0 +  2039 words
 Per cent of directory quota used =    .050
 Per cent of file used            =    .050
 Blocking factor                  =  49.561
```

Output of HIST/LIST

```
===> Directory :
       100 (1)   htfun1.f
       110 (1)   Test 1-dim Histo
       200 (2)   htfun2
```

3.7.3 Histogram archiving

In this example, the histograms in an existing `hbook` file are moved in a new `hbook` file in two separated directories according to their type.

Histogram archiving and directories into `hbook` files

```
❶    HISTOGRAM/FILE 0 pawtut.hbook
     hi/li
     hrin  *
     close 1
❷    HISTOGRAM/FILE 0 pawtutnew.hbook ! N
❸    MDIR 1Dhistograms
❸    MDIR 2Dhistograms
     ldir
     cd 1Dhistograms
❹    HROUT 514,30001,60202
     ldir
     cd //LUN1/2Dhistograms
❹    HROUT 8001,1103,11,12
     ldir
     close 1
```

❶ Attach an existing `hbook` file.

❷ Create a new `hbook` file.

❸ Create two subdirectories in the file `pawtutnew.hbook`.

❹ Store the 1d and 2d histograms in two separated directories. Some commands like `HROUT` are able to loop on parameters if a list is given. Such parameters have the attribute “Loop” when a help is performed on the command.

Output of LDIR

```
===> Directory :
        10 (N)   CERN Population
       514 (1)   Angular density
     30001 (1)   mix
     60202 (1)   p dy like
      8001 (2)   Data (gluino)
      1103 (2)   Charged particle theta vs. phi
        11 (2)   PHI VS. Y  +VE    WEIGHTED
        12 (2)   PHI VS. Y  +VE    WEIGHTED

************** Directory ===> //LUN1 <===

                 Created 930602/1428  Modified 930602/1428


===> List of subdirectories
1DHISTOGRAMS     Created 930602/1428 at record     3
2DHISTOGRAMS     Created 930602/1428 at record     4

===> List of objects
    HBOOK-ID  VARIABLE                             CYCLE    DATE/TIME   NDATA

************** Directory ===> //LUN1/1DHISTOGRAMS <===

                 Created 930602/1428  Modified 930602/1428


===> List of objects
    HBOOK-ID  VARIABLE                             CYCLE    DATE/TIME   NDATA
       514         0                                   1   930602/1428    153
     30001         0                                   1   930602/1428    200
     60202         0                                   1   930602/1428    152

************** Directory ===> //LUN1/2DHISTOGRAMS <===

                 Created 930602/1428  Modified 930602/1428


===> List of objects
    HBOOK-ID  VARIABLE                             CYCLE    DATE/TIME   NDATA
      8001         0                                   1   930602/1428    537
      1103         0                                   1   930602/1428   5361
        11         0                                   1   930602/1428    444
        12         0                                   1   930602/1428  13114
```

3.7.4 Multiple fits on histograms

Fit of the histogram 110 with two Gaussians

```
      histogram/File 1 pawhists.hbook
      hrin  *
❶     VECT/CREATE PAR(6)
      histo/plot 110
❺     SET FWID 6
❺     SET DMOD 2
❷     HISTO/FIT 110(1:50) G QS 0 PAR(1:3)
❸     HISTO/FIT 110(50:100) G QS 0 PAR(4:6)
❺     SET DMOD 1
❹     HISTO/FIT 110 G+G QS 6 PAR
```

❶ The vector `PAR` will be used to get the initial values of the fit parameters.

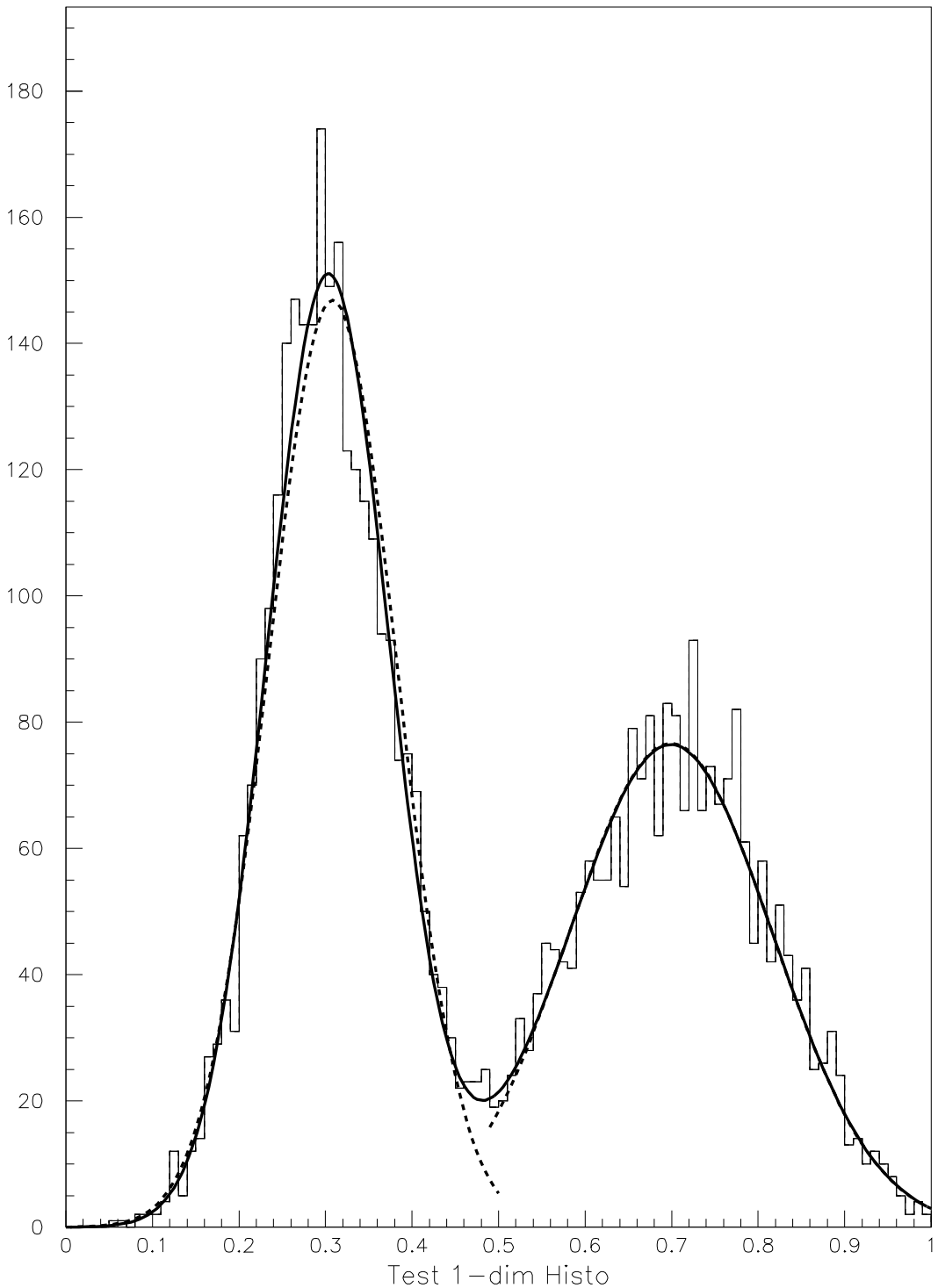
❷ Compute a gaussian fit on the first 50 channels. After this command the gaussian parameters are stored in `PAR(1:3)`.

❸ Compute a gaussian fit on the last 50 channels. After this command the gaussian parameters are stored in `PAR(4:6)`.

❹ Compute the global fit using `PAR` for initial values.

Note also:

❺ The first two gaussian fits are drawn with dashed lines and the third one with a solid line.


180
160
140
120
100
80
60
40
20
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Test 1−dim Histo

3.7.5 Histogram operations

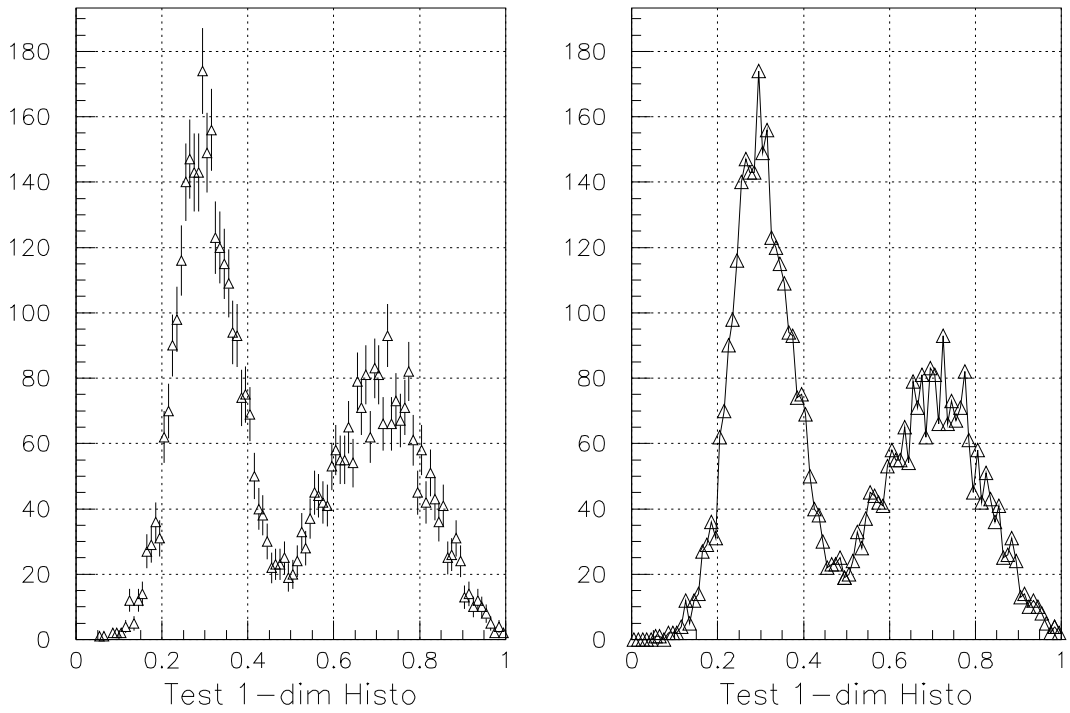
Perform operations on histograms read from file and save results

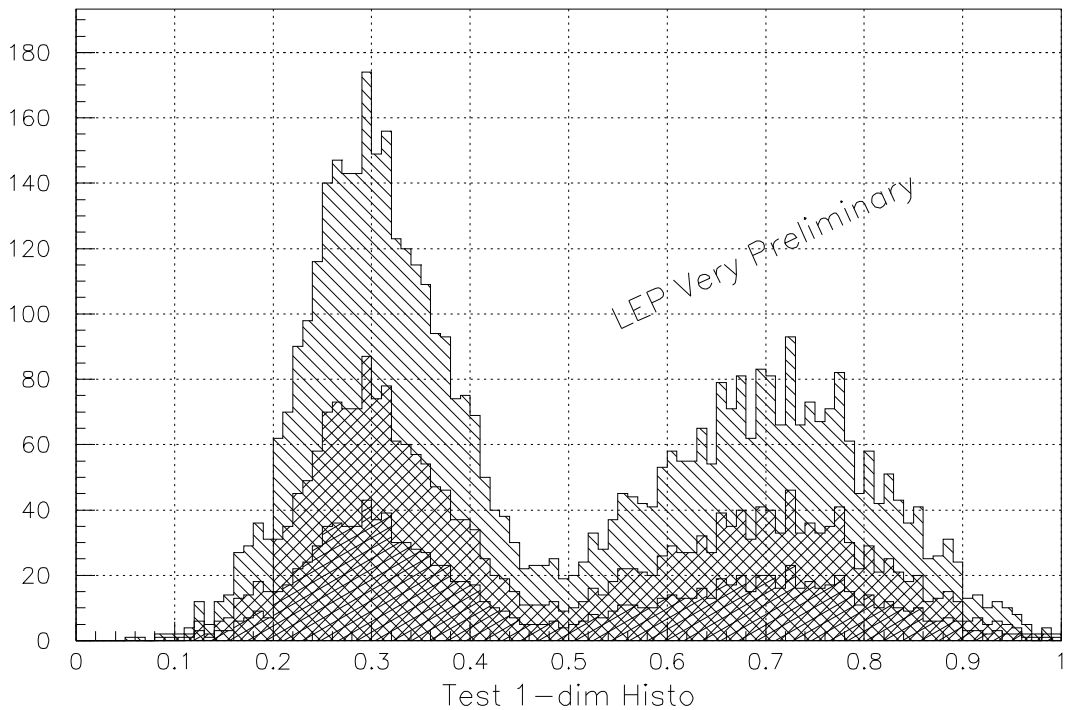
```
❶   HISTOGRAM/FILE 1 PAWHISTS.HBOOK ! U
    hrin  *
    zon 2 2
    opt grid
    igset mtyp 26
    hi/pl 110 e
    hi/pl 110 pl
    zon 1 2 2 s
❷   HI/OP/ADD 110 110 120 0.5 0.
    hi/op/add 110 110 130 0.25 0.
    set htyp 245
    hi/pl 110
    set htyp 254
❸   HI/PL //PAWC/120 s
    set htyp 253
    hi/pl //PAWC/130 s
    text 0.55 95. 'LEP Very Preliminary' 0.35 25.
    hrout 0
```

❶ The option "U" (for Update) in the command HIST/FILE, is used when the user wants to change the content of an existing histogram file by adding a new histogram (HROUT p 166) or deleting an histogram (HSCRATCH p 166).

❷ It is possible to perform operations between histograms like addition with the commands in the menu HISTOGRAM/OPERATIONS.

❸ The memory, like the attached files, can be considered as a directory. This is the current directory by default and //PAWC is its name. The command HI/PL //PAWC/id plots the histogram "id" in memory while the current directory is //LUN1.


180
160
140
120
100
80
60
40
20
0
0 0.2 0.4 0.6 0.8 1
Test 1−dim Histo
180
160
140
120
100
80
60
40
20
0
0 0.2 0.4 0.6 0.8 1
Test 1−dim Histo


180
160
140
120
100
80
60
40
20
0
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
Test 1−dim Histo
LEP Very Preliminary

How to embellish the graphical ouputs

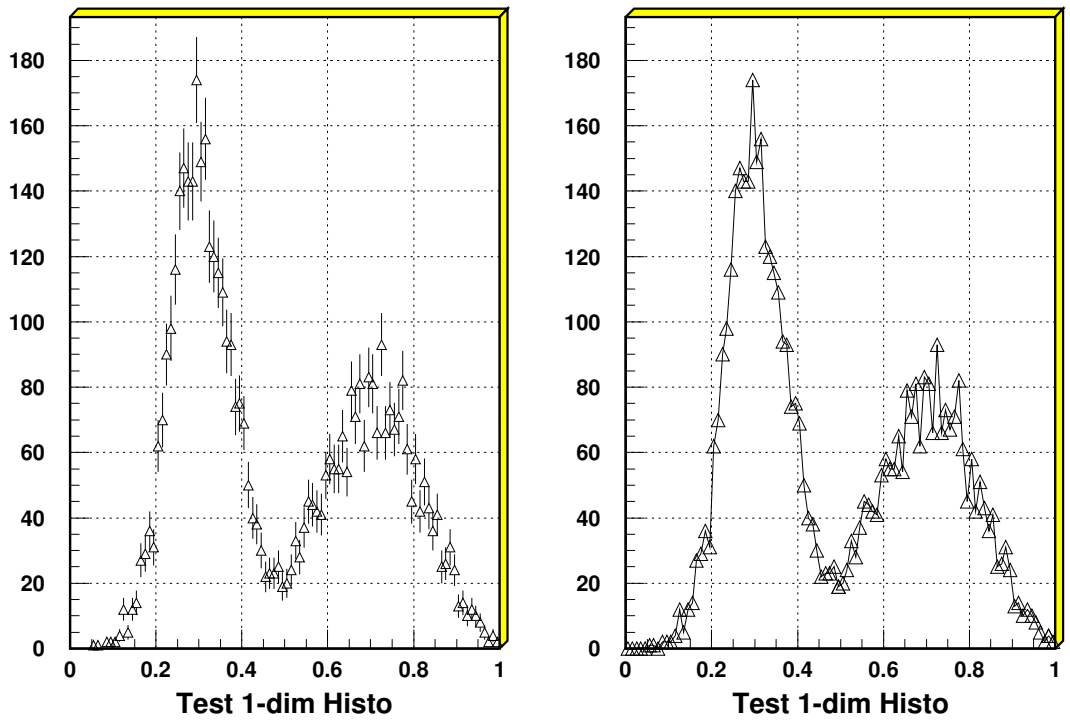
```
      histogram/file 1 pawhists.hbook ! u
      hrin  0
      zon 2 2
      opt grid
❶     SET *FON -60
❷     SET BWID 4
❸     SET BCOL 1.5
      igset mtyp 26
      hi/pl 110 e
      hi/pl 110 pl
      zon 1 2 2 s
      hi/op/add 110 110 120 0.5 0.
      hi/op/add 110 110 130 0.25 0.
      set htyp 245
      hi/pl 110
      set htyp 254
      hi/pl //pawc/120 s
      set htyp 253
      hi/pl //PAWC/130 s
❹     IGSET CHHE .35
❹     IGSET TANG 25.
❹     ITX 0.55 95. 'LEP Very Preliminary'
      hrout 0
```

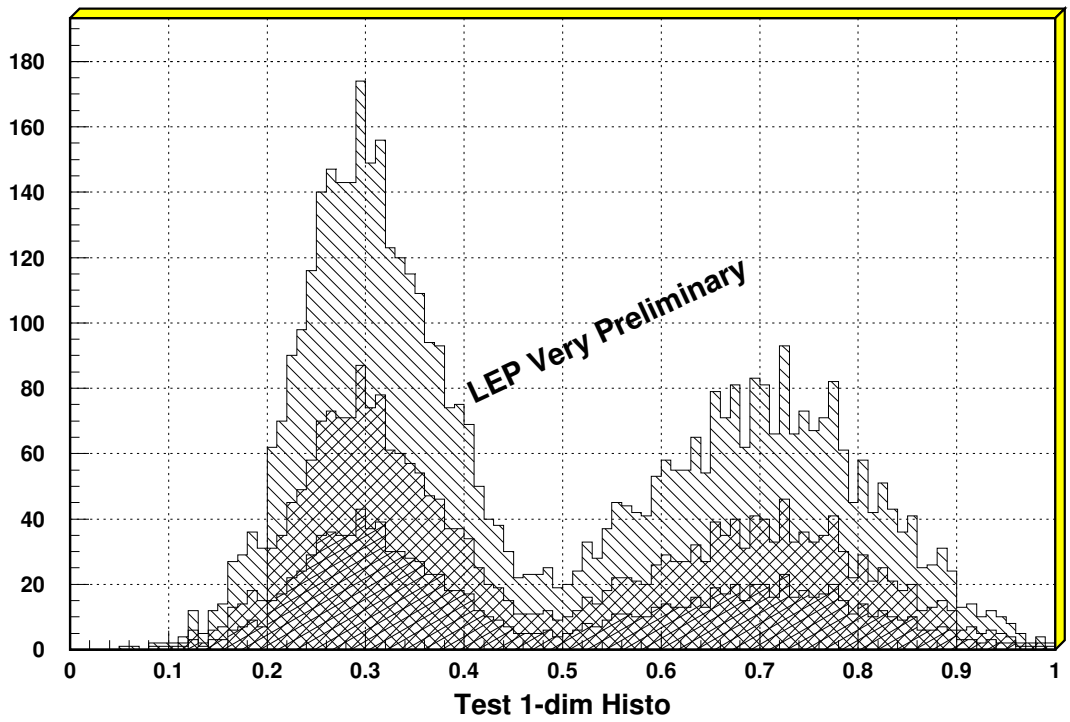
❶ All the text fonts used for `HISTO/PLOT` are set to -60.

❷ The line width for the boxes around the histograms is set to 4 pixels. Like for the fonts it is possible to do `SET *WID` to set all the width available in the `SET` command.

❸ The color of the shadow around the histograms is set to 5 (Yellow), it appears grey on black and white PostScript printers.

❹ To access hardware fonts (ie PostScript fonts) the command `ITX` and its related attributes should be used.


180
160
140
120
100
80
60
40
20
0
0 0.2 0.4 0.6 0.8 1
Test 1-dim Histo
180
160
140
120
100
80
60
40
20
0
0 0.2 0.4 0.6 0.8 1
Test 1-dim Histo


LEP Very Preliminary
180
160
140
120
100
80
60
40
20
0
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
Test 1-dim Histo

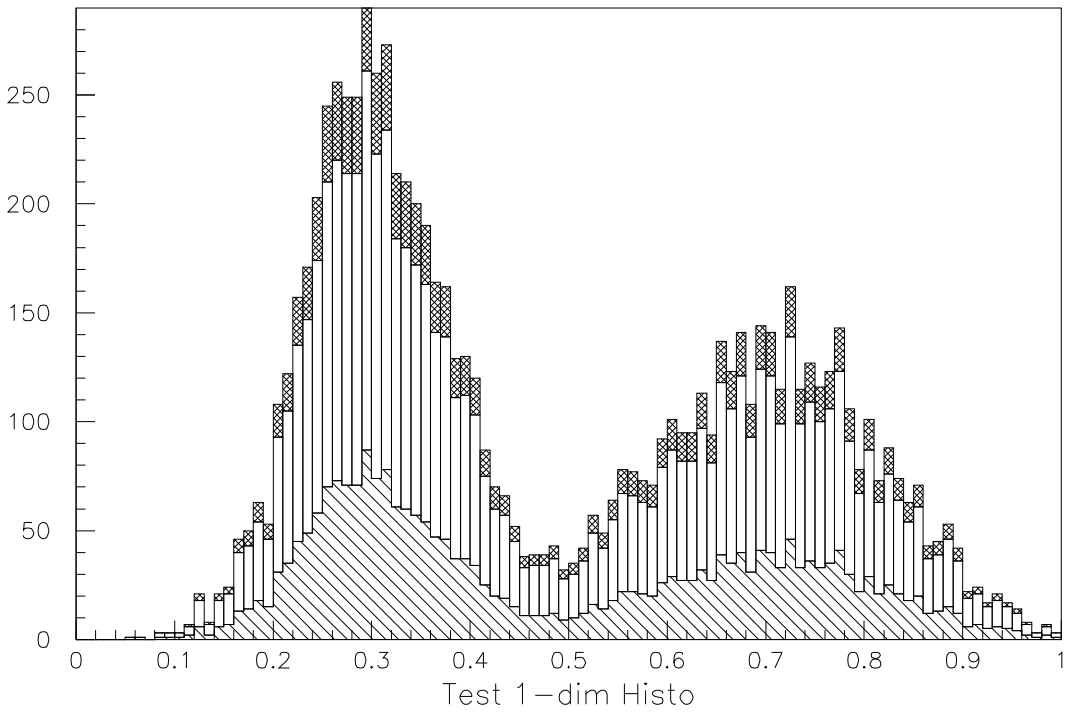
3.7.6 Keep and update histograms

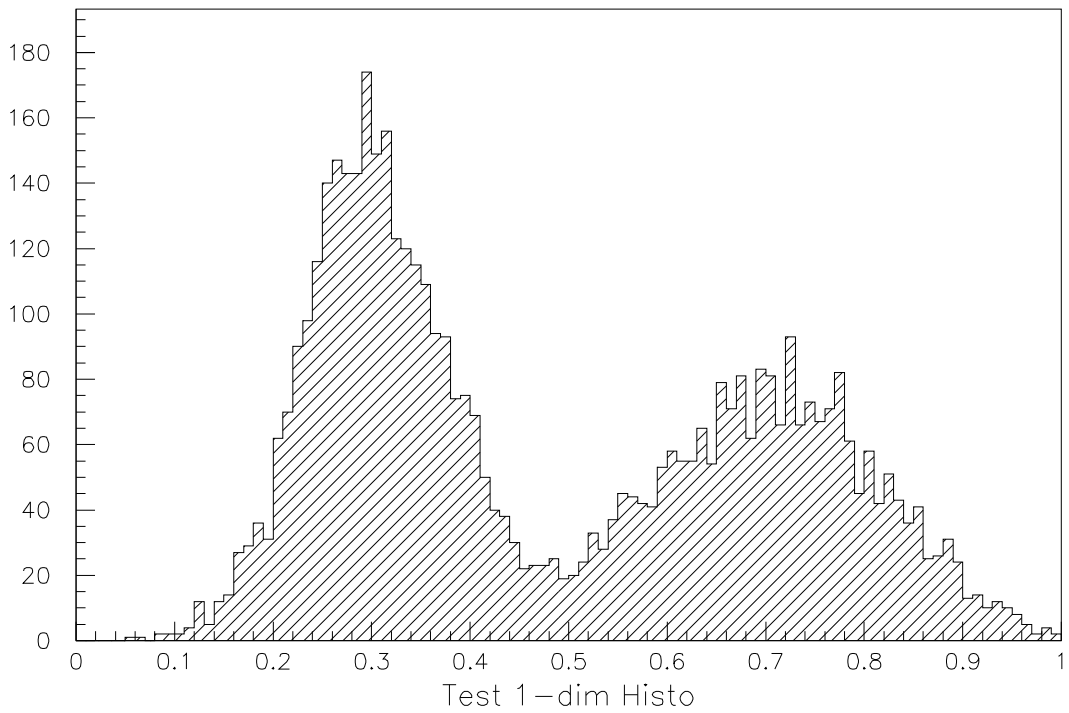
Graphical operations on histograms (Keep and Add)

```
      histogram/file 1 pawhists.hbook
      zone 1 2
      set htyp 245
❶     H/PL 120 K
      set htyp 254
❶     H/PL 110
      set htyp
❷     H/PL 110 +
      set htyp 144
      hi/pl 130 +
```

❶ The option "K" in the command HIST/PLOT keep the histogram in memory at the graphics level to allow updating. If no zone is defined, the option "K" is not necessary.

❷ If an histogram is kept in memory (automatically or via option "K") it is possible to add the content of an other histogram with option "+".


250
200
150
100
50
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Test 1−dim Histo


180
160
140
120
100
80
60
40
20
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Test 1−dim Histo

3.7.7 Playing with dice

Graphical operations on histograms (Keep and Update)

```
      MACRO DICE 1=50
      set hcol 1001
      set ndvx -11.05
❻     OPT STAT
❶     CALL DICE.F([1])
      hi/fit 3 g
```

FORTRAN routine dice

```
      subroutine dice(n)
      ifirst=1
❷       CALL HBOOK1(3,'Playing with two dice',11,2.,13.,0.)
        do 3 j=1,n
          ix1=6.*rndm(.01234)+1
          ix2=6.*rndm(.56789)+1
❸         CALL HFILL(3,FLOAT(IX1+IX2),0.,1.)
          if (ifirst.eq.1) then
❹           CALL HPLOT(3,'BK',' ',0)
            ifirst=0
          else
❺           CALL HPLOT(3,'BU',' ',0)
          endif
        enddo
      end
```

❶ This macro call a `comis` routine only to be faster. The `comis` routine can be replaced by a macro, in particular the options "`K`" and "`U`" are also available in command `HIST/PLOT` (try `HELP H/PL`).

❷ The histogram is also booked in the FORTRAN program. The corresponding `paw` command is `1DHISTO`.

❸ Two random numbers between 1 and 6 are generated and the histogram is filled with the sum of this numbers to simulate dice playing.

❹ The first time the histogram is plotted the option "`K`" is used to keep in memory a copy of the histogram in order to update it later.

❺ With the "`U`" option, `paw` looks at the current kept histogram contents and update the plot with the new contribution without redrawing everything. This mechanism is used in data acquisition.

❻ The statistics are also updated.

3.7.8 Two-dimensional histograms representations

Different representations of two-dimensional histograms

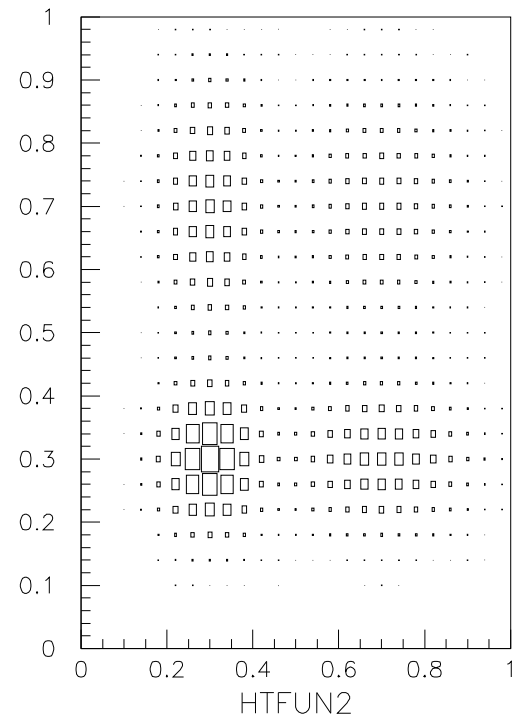
```
      histogram/file 1 pawhists.hbook
      zon 2 2
❶     HI/PL 200 BOX
❶     CONTOUR 200 20 0
❶     LEGO 200
❶     SURFACE 200
      hi/del *
```

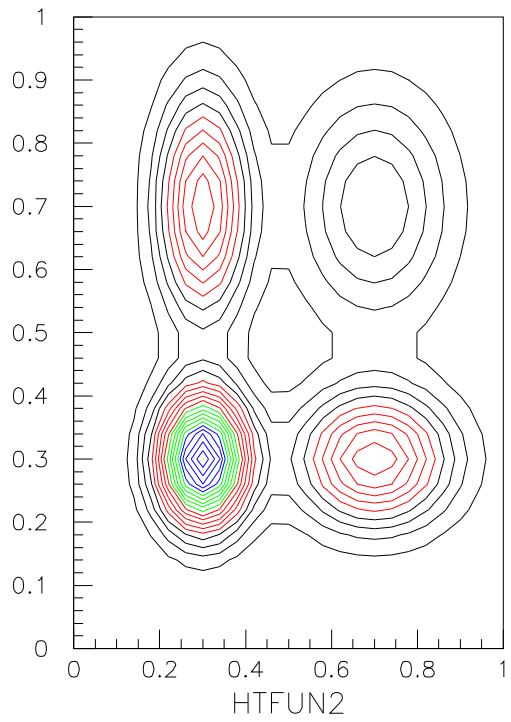
❶ As we have already seen, the command H/PL allows to draw 2D histograms in different ways. Three additional commands are also available:

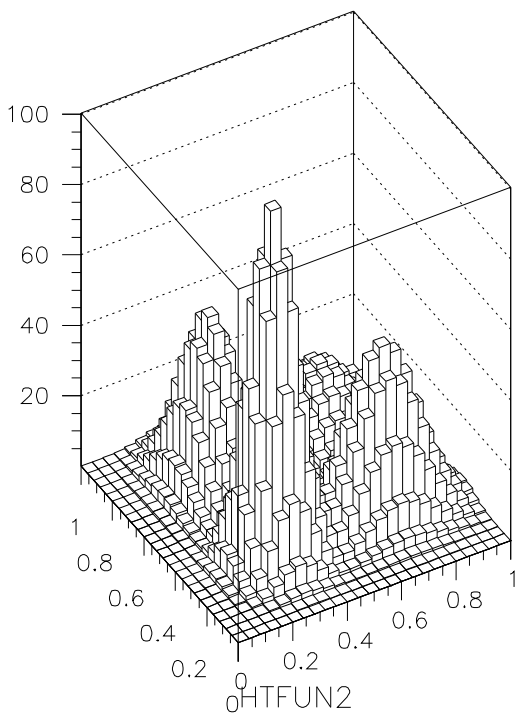
```
  * /HISTOGRAM/2D_PLOT/CONTOUR  [ ID NLEVEL CHOPT PARAM ]
  * /HISTOGRAM/2D_PLOT/SURFACE  [ ID THETA PHI CHOPT ]
  * /HISTOGRAM/2D_PLOT/LEGO     [ ID THETA PHI CHOPT ]
\index{histogram!contour}
\index{histogram!surface}
\index{histogram!lego}
```

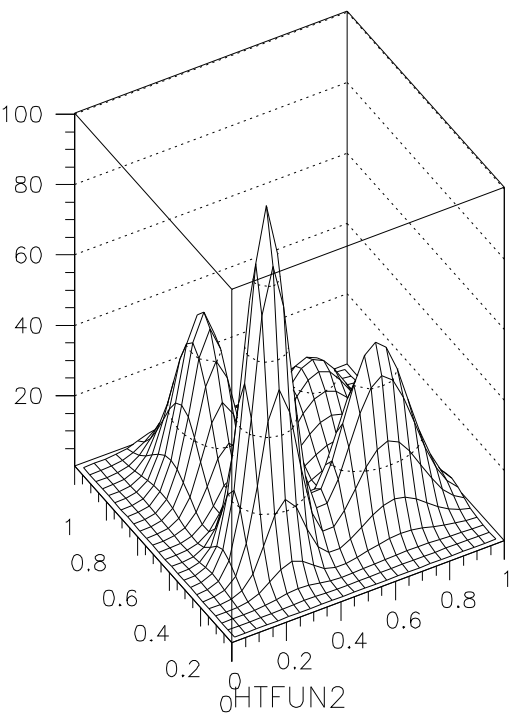
These commands have more parameters than `HIS/PLOT`. For example `CONTOUR` allows to specify a set of levels to be drawn via the parameter `PARAM` (see next example).

❷ Note that it is also possible to have 1D histograms represented as lego or surface plots. For example you can do: `HI/PLOT 110 LEGO`.


1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 0.2 0.4 0.6 0.8 1
HTFUN2


1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 0.2 0.4 0.6 0.8 1
HTFUN2


100
80
60
40
20
1
0.8
0.6
0.4
0.2
0
0.2 0.4 0.6 0.8 1
HTFUN2


100
80
60
40
20
1
0.8
0.6
0.4
0.2
0
0.2 0.4 0.6 0.8 1
HTFUN2

3.7.9 Non equidistant contour plots

User defined non equidistant contour plots

```
      histogram/file 1 pawhists.hbook
❶     VECTOR/CREATE LEVEL(8) R  10 11 12 13 14 15 90 99
      zone 1 2
❷     CONTOUR 200 8 2 LEVEL
      arrow .8 .75 .5 .54 .2
❺     ARROW .8 .75 .5 .44 .2
❹     SET CHHE .28
❸     ITX .81 .5 '10.0'
      Arrow .5 .32 .1 .28 .2
      Itx .51 .1 '100.0'
      option LOGY
      h/plot 200 BOX
❻     ARROW .5 .32 .1 .28 .2
❻     ITX .51 .1 '100.0'
      close 1
```

The command `CONTOUR` allows to draw user defined levels.

❶ The vector `LEVEL` contains the list of 8 levels to be drawn.

❷ Only the levels specified in the the vector `LEVEL` are drawn.

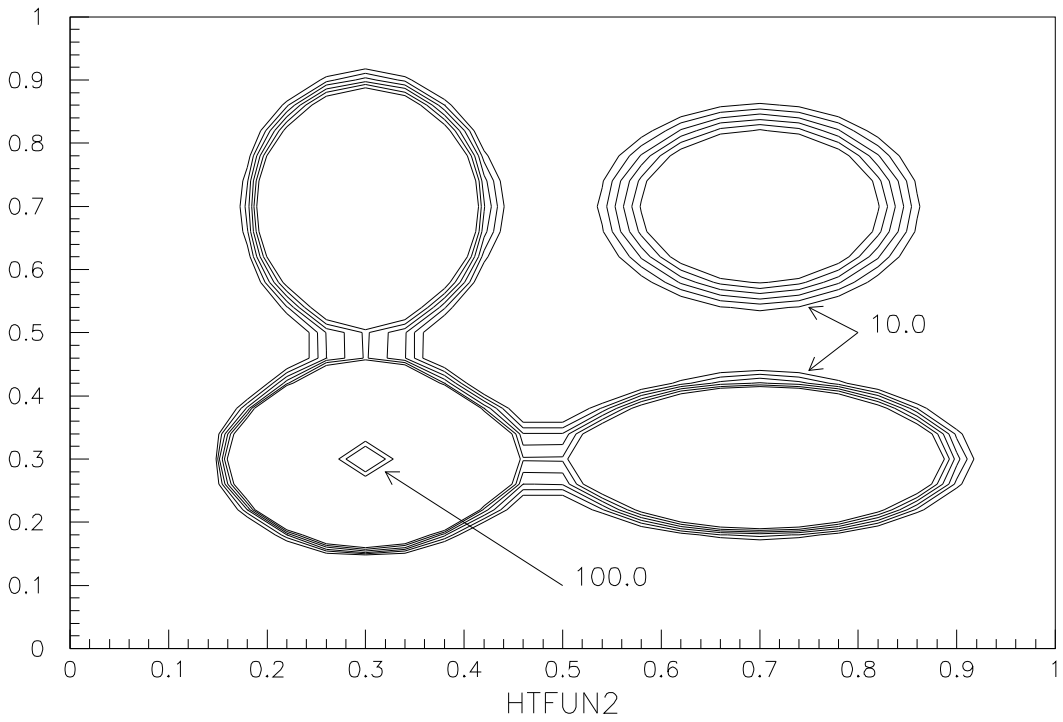
Note also:

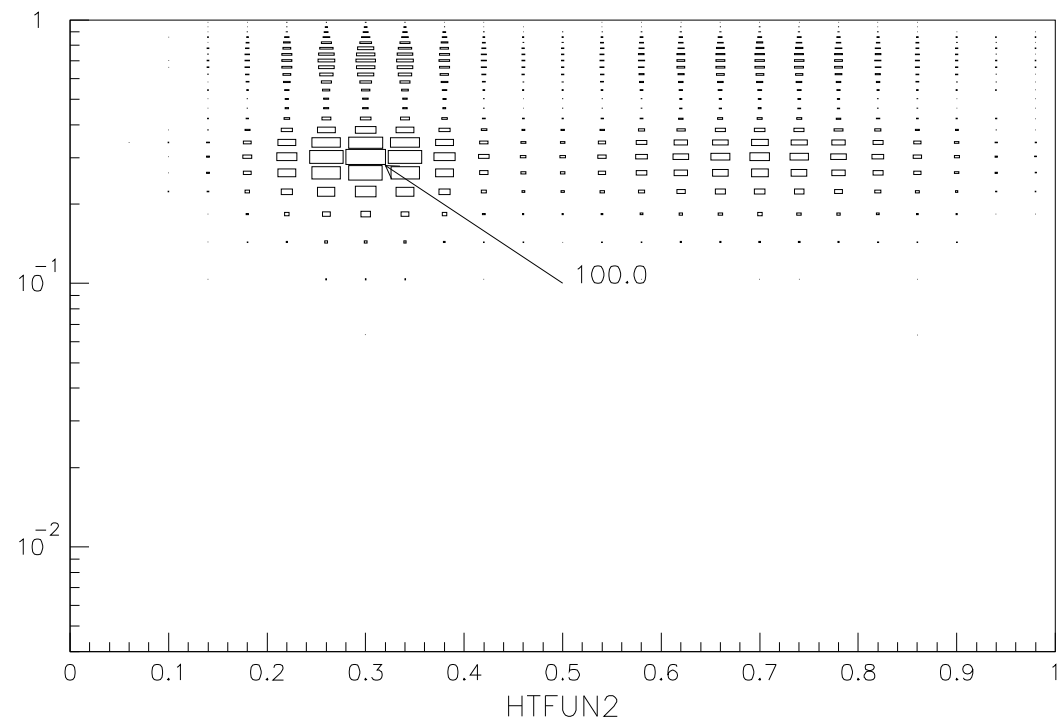
❸ Some comments can be drawn with the command `ITX`.

❹ The size of the text is in centimeters even if the position is in histogram coordinates (current normalization transformation).

❺ The position of the arrow is in the current normalization transformation (here histogram coordinates), but its size is in centimeters (last parameter. Here 0.2).

❻ Arrows and text can be drawn in logarithmic coordinates. For lines the logarithm should be computed with `sigma`.


1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
10.0
100.0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
HTFUN2


1
10^{-1}
10^{-2}
100.0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
HTFUN2

3.7.10 Coordinate systems

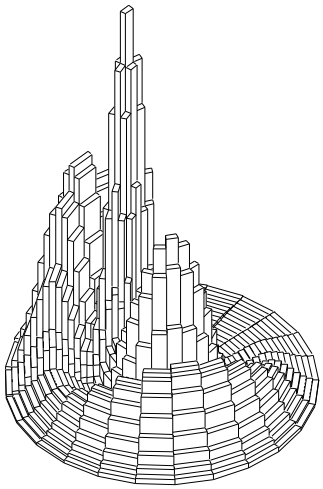
Coordinate systems with legos and surfaces

```
      histogram/file 1 pawhists.hbook
      zon 2 2
❷     OPT UTIT
❷     TITLE 'Polar coordinates' U
❶     HI/PL 200 LEGO,POL
      title 'Cylindrical coordinates' U
❶     HI/PL 200 LEGO,CYL
      title 'Spherical coordinates' U
❶     HI/PL 200 LEGO,SPH
      title 'Pseudo rapidy coordinates' U
❶     HI/PL 200 LEGO,PSD
      close 1
```

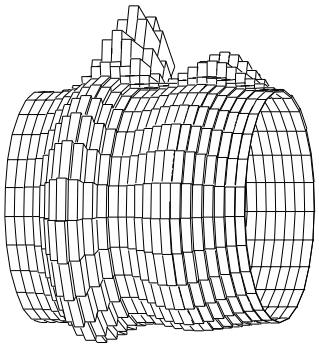
❶ Legos and Surfaces plot can be drawn in Polar, Cylindrical, Spherical and Pseudo rapidity coordinates.

Note also:

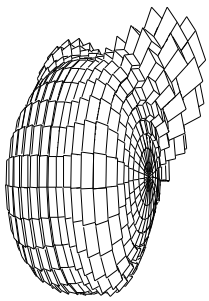
❷ The option `UTIT` allows to use "user title" on histogram. To define the title itself, the command `TITLE` should be used with the option `U`. Without this option `TITLE` define the global title.



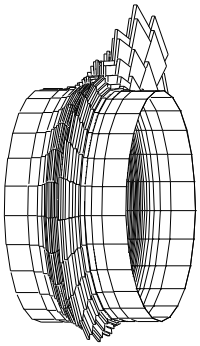
Polar coordinates



Cylindrical coordinates



Spherical coordinates



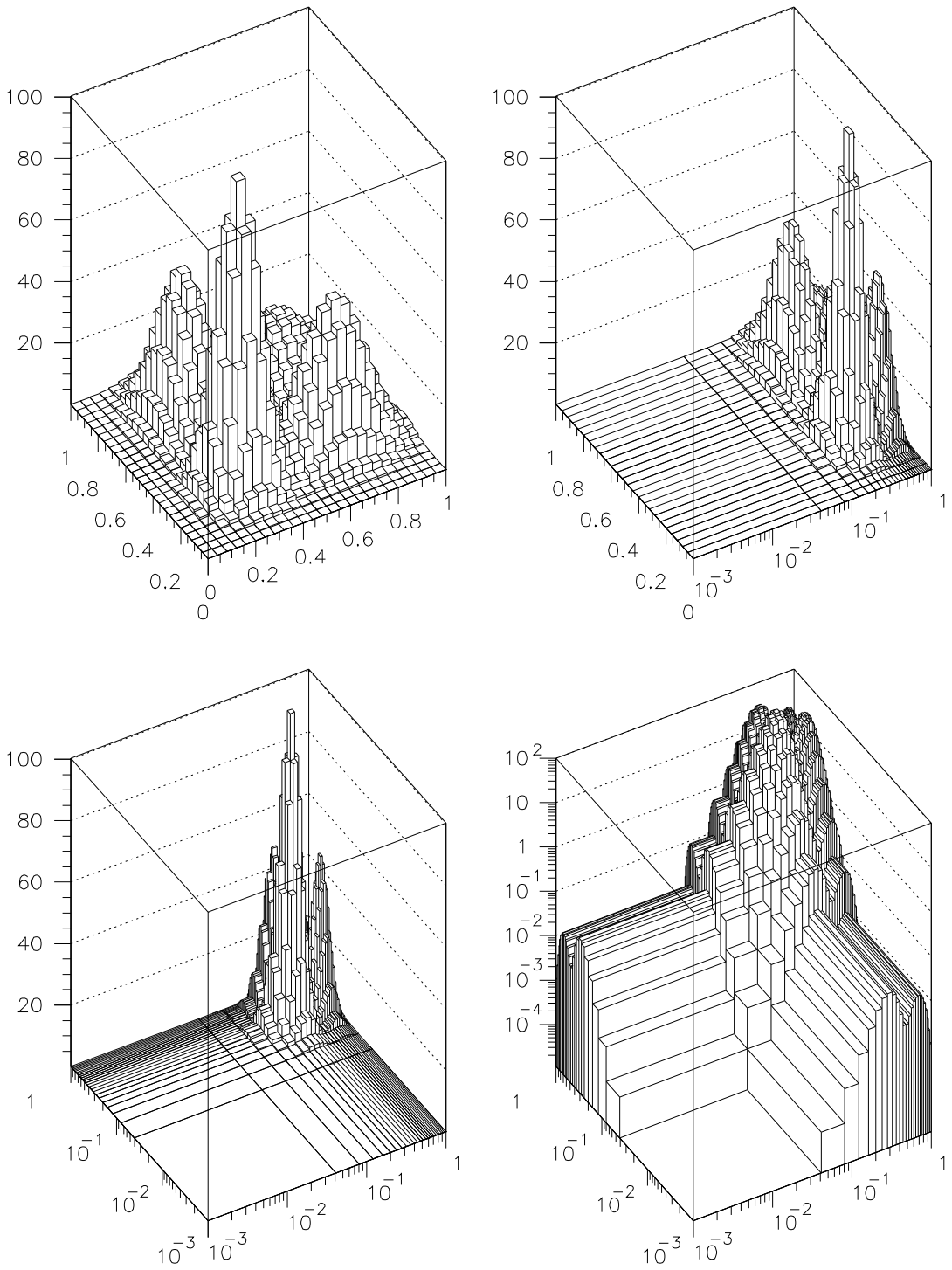
Pseudo rapidy coordinates

3.7.11 Logarithmic scales on lego plots

Logarithmic scales on lego plots and surfaces plot

```
histogram/file 1 pawhists.hbook
zon 2 2
opt utit
hi/pl 200 lego
OPT LOGX                    ❶
hi/pl 200 lego
OPT LOGY                    ❶
hi/pl 200 lego
OPT LOGZ                    ❶
hi/pl 200 lego
close 1
```

❶ Logarithmic are possible on Legos and Surfaces plot. It works also in Polar, Cylindrical, Spherical and Pseudo rapidity coordinates.


100
80
60
40
20
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1
10^{-3}
10^{-2}
10^{-1}
10^{2}
10
10^{-4}

3.7.12 Subranges in histogram identifiers

Usage of subranges in histogram identifiers

```
      histogram/file 1 pawhists.hbook
      hrin  0
      close 1
❹     TRACE ON
      zon 2 2
❶     HI/PL 110(56:95) E
❹     * Comments are not printed in TRACE mode
      hi/pl 200(8:8,)      box
❸     HI/PL 200(3:15,3:15) CONT
❹     TRACE OFF
❷     hi/pl 200(0.:12,0.1:0.5) LEGO
```

❶ This example shows how to plot subranges of 1D or 2D histograms. The different possibility to give the range are the following:

(a) `id(n1:n2)` with `n1` $\geq$ `n2`.

(b) `id(n1:)` in this case `n2` $=$ `number of bins`.

(c) `id(:n2)` in this case `n1` $=$ `1`.

❷ If `n1` or `n2` are integer they are consider as bin numbers. But if they are real they are consider axis values. Note that bin values and axis values can be mixed inside the same range definition.

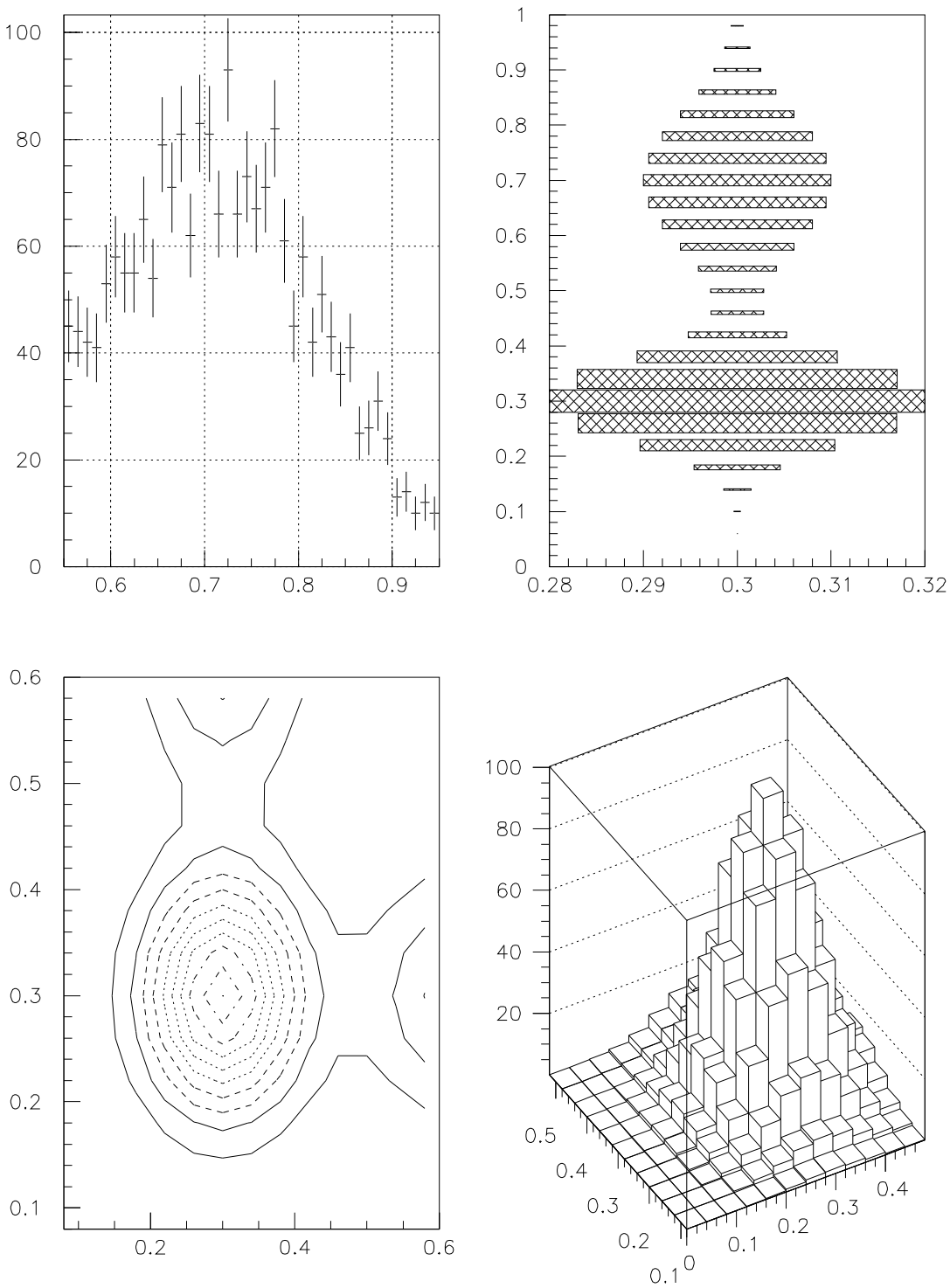
❸ In case of 2D histograms, the two ranges are separate with ",".

Note also:

❹ The `TRACE` command sets `ON` or `OFF` the trace mode. When this mode is on, all the command executed inside macros are displayed on the standard output.

Ouput of the TRACE mode

```
>>>>> zon 2 2
>>>>> HI/PL 110(56:95) E
>>>>> hi/pl 200(8:8,)      box
>>>>> HI/PL 200(3:15,3:15) CONT
>>>>> TRACE OFF
```


100
80
60
40
20
0
0.6
0.7
0.8
0.9
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.28
0.29
0.3
0.31
0.32
0.6
0.5
0.4
0.3
0.2
0.1
0.2
0.4
0.6
100
80
60
40
20
0.5
0.4
0.3
0.2
0.1
0
0.1
0.2
0.3
0.4

3.7.13 Stacked Lego plots

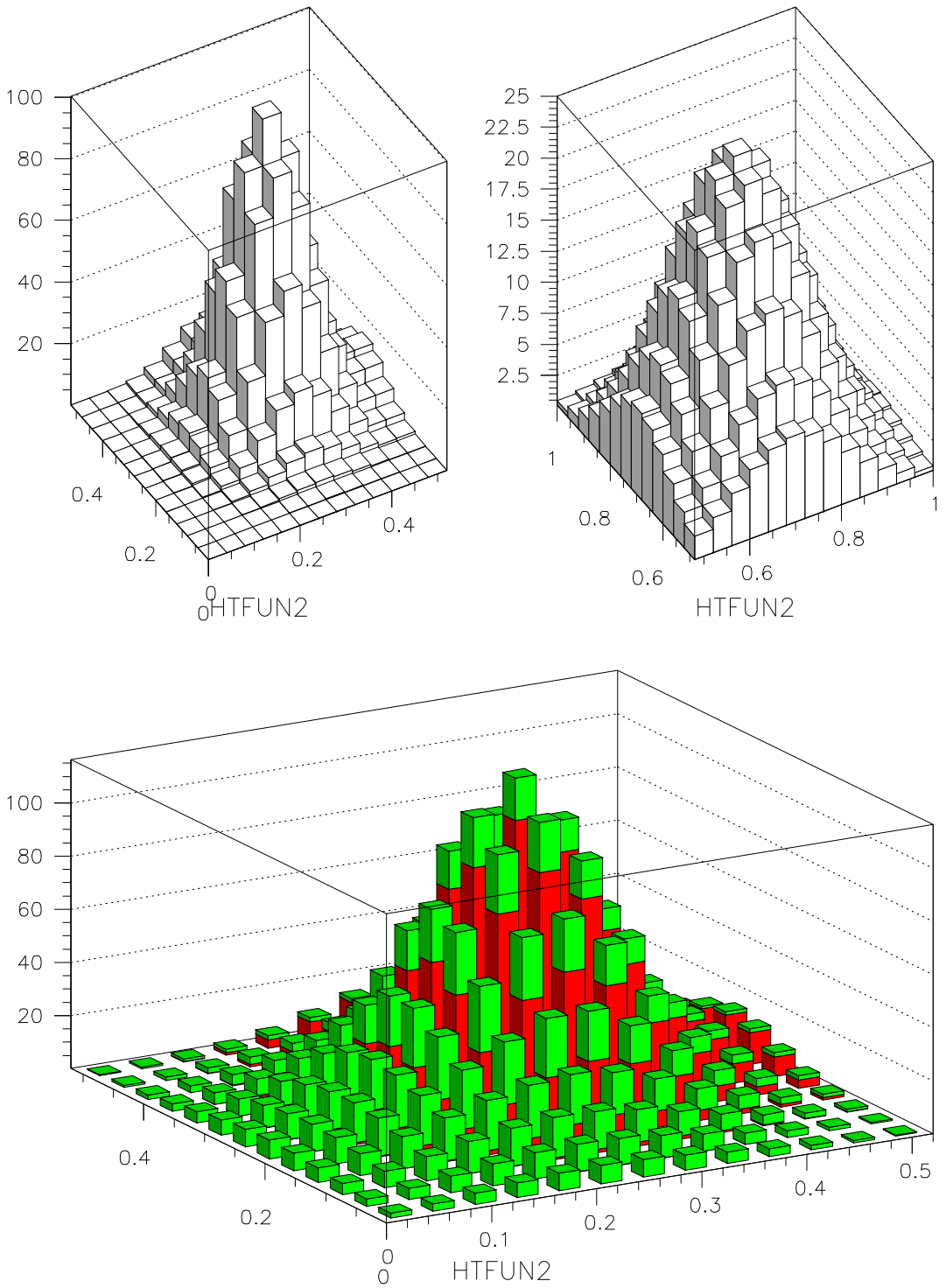
Stacked Lego plots and subranges

```
     hi/file 1 pawhists.hbook
     zon 2 2
❶    HIST/PLOT 200(0.:0.5,0.:0.5) LEGO1
❶    HIST/PLOT 200(0.5:1.,0.5:1.) LEGO1
     zon 1 2 2 s
❸    OPTION BAR
❷    HIST/PLOT 200(0.:0.5,0.:0.5)+200(0.5:1.,0.5:1.) LEGO1
     close 1
```

❶ The two commands draw submatrices of the histogram 200 as Lego plots.

❷ The submatrices previously drawn are now stacked.

❸ The option `BAR` is active on Lego plots.


100
80
60
40
20
0.4
0.2
0
0
0.2
0.4
HTFUN2
25
22.5
20
17.5
15
12.5
10
7.5
5
2.5
1
0.8
0.6
0.6
0.8
1
HTFUN2
100
80
60
40
20
0.4
0.2
0
0
0.1
0.2
0.3
0.4
0.5
HTFUN2

3.7.14 A more complex example

Fit the background with a P3

```
      Macro PAWEX15A ID=30001 IP1=40 IP2=111 IZ1=33 IZ2=150 LOOP=20
      *
      Igset 2BUF 1
      Hi/file 1 pawtut.hbook ; Hrin [ID]
      Set FWID 6 ; Set DMOD 1
❶     CALL hinfo.f([ID])
      NBIN = hid(1)
      Vector/Create  FUNC([NBIN])
      Vector/Create  Y([NBIN])
      Vector/Create  S([NBIN])
      Vector/Create  X([NBIN],[LOOP])
      Histogram/Copy [ID] 1
      Histogram/Copy [ID] 2
      *
❷     Do i=1,[LOOP]
         Histogram/Plot 1
❸        Histogram/Fit  1([IZ1]:[IZ2]) P3 0q
❸        Get/Func 1 FUNC ; Put/Cont 2 FUNC
❸        Sub 1 2 3
❹        Histogram/Fit  3([IP1]:[IP2]) G 0q
         Histogram/Plot 3([IP1]:[IP2]) FUNCS
❹        Get/Func 3 FUNC ; Put/Cont 2 FUNC
❹        Sub 1 2 1
         Get/Func 3 X(1:[NBIN],[i])
         Call igterm
      Enddo
      *
      Get/Func 1 FUNC ; Put/Cont 2 FUNC
      Sub [id] 2 3
      Zone 1 2
      Histogram/Plot [ID]
      Histogram/Plot 1 FUNCS
❺     Do i=1,[LOOP]
         Vector/Copy X(1:[NBIN],[i]) Y
         SIGMA S = S + Y
         SIGMA Y = Y+FUNC
         Put/Cont 2 Y
         Histogram/Plot 2([IP1]:[IP2]) SL
      Enddo
      Histogram/Plot 3([IP1]:[IP2]) HIST
      Put/Cont 2 S
      Histogram/plot 2([IP1]:[IP2]) Sl
      *
      Close 1
      V/Del FUNC,X,Y,S ; H/Del 1,2,3
```

The routine hinfo.f

```
subroutine hinfo(id)
character*32 chtitl
vector hid(6)
call hgive(id,chtitl,ncx,xmin,xmax,ncy,ymin,ymax,nwt,loc)
hid(1) = ncx
hid(2) = xmin
hid(3) = xmax
hid(4) = ncy
hid(5) = ymin
hid(6) = ymax
end
```

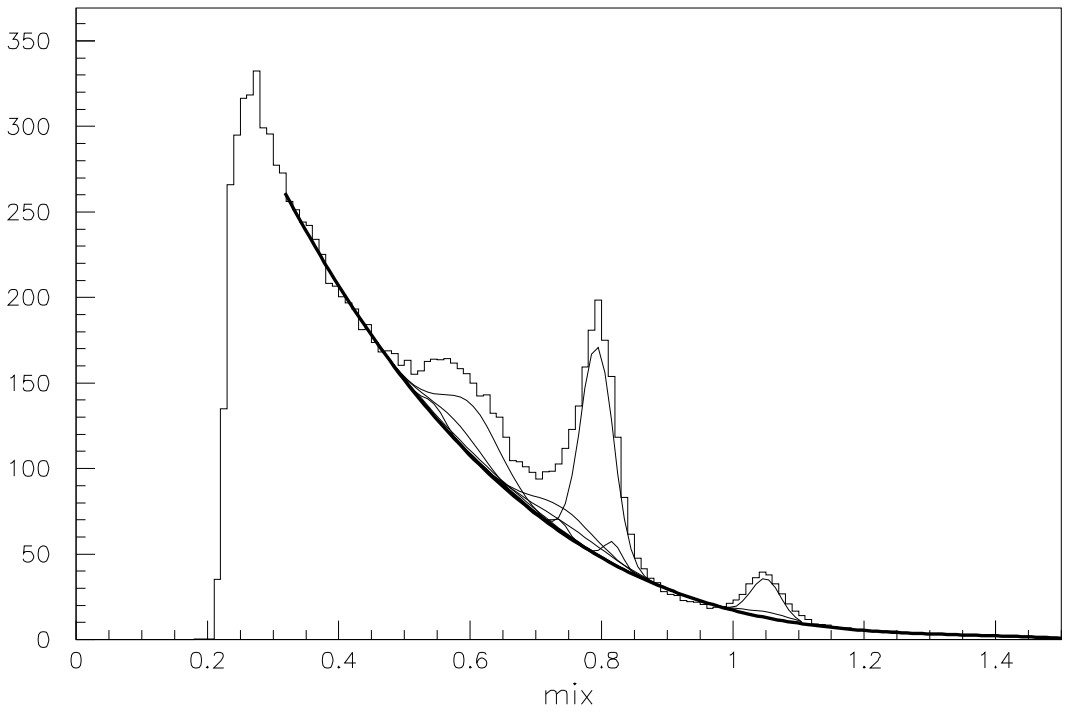
❶ This routine allows to have informations on an histogram.

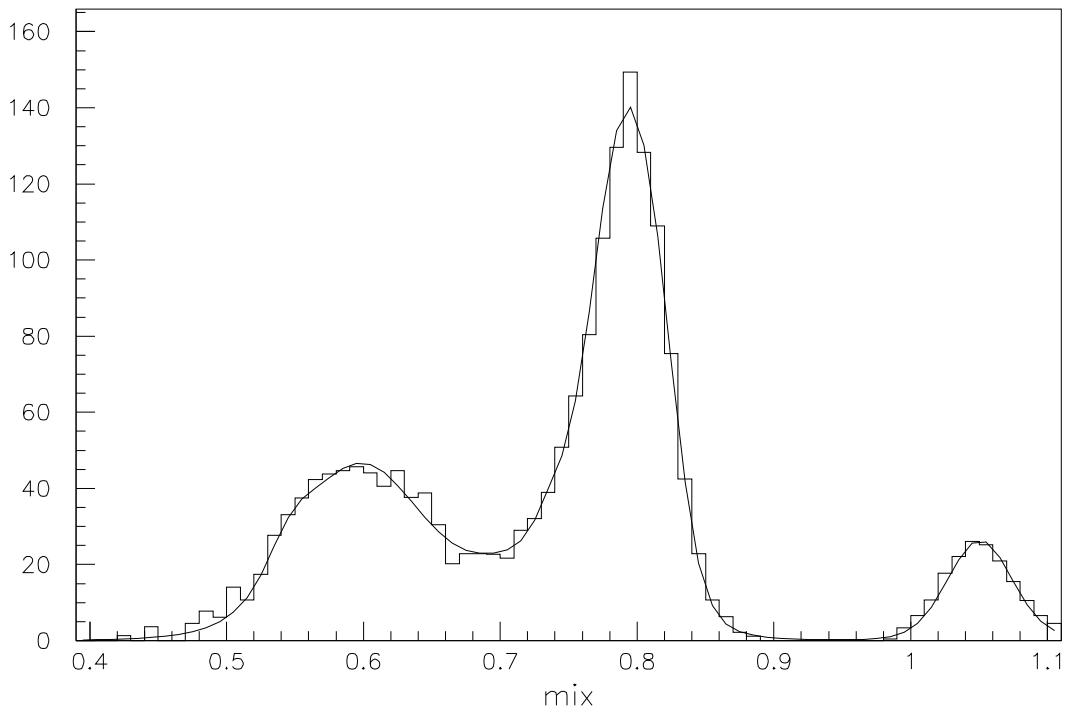
❷ This loop try to find a `P3` background.

❸ After a `P3` fit, a new histogram is booked with the fit value at each channel. This new histogram is consider as an approximation of the background and is removed from the original histogram.

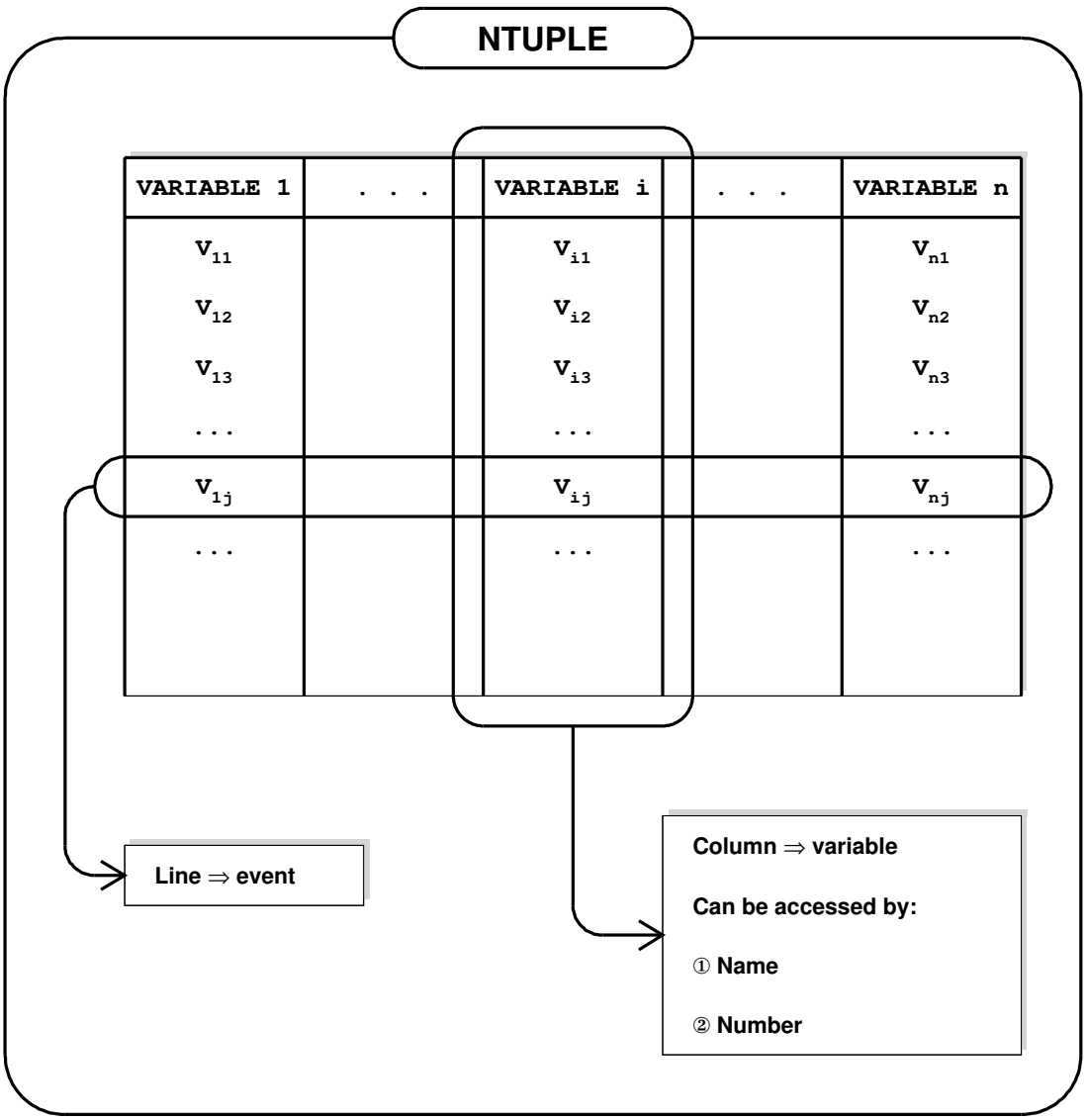
❹ A gaussian fit allows to remove the pick.

❺ This loop produce the two final plots.

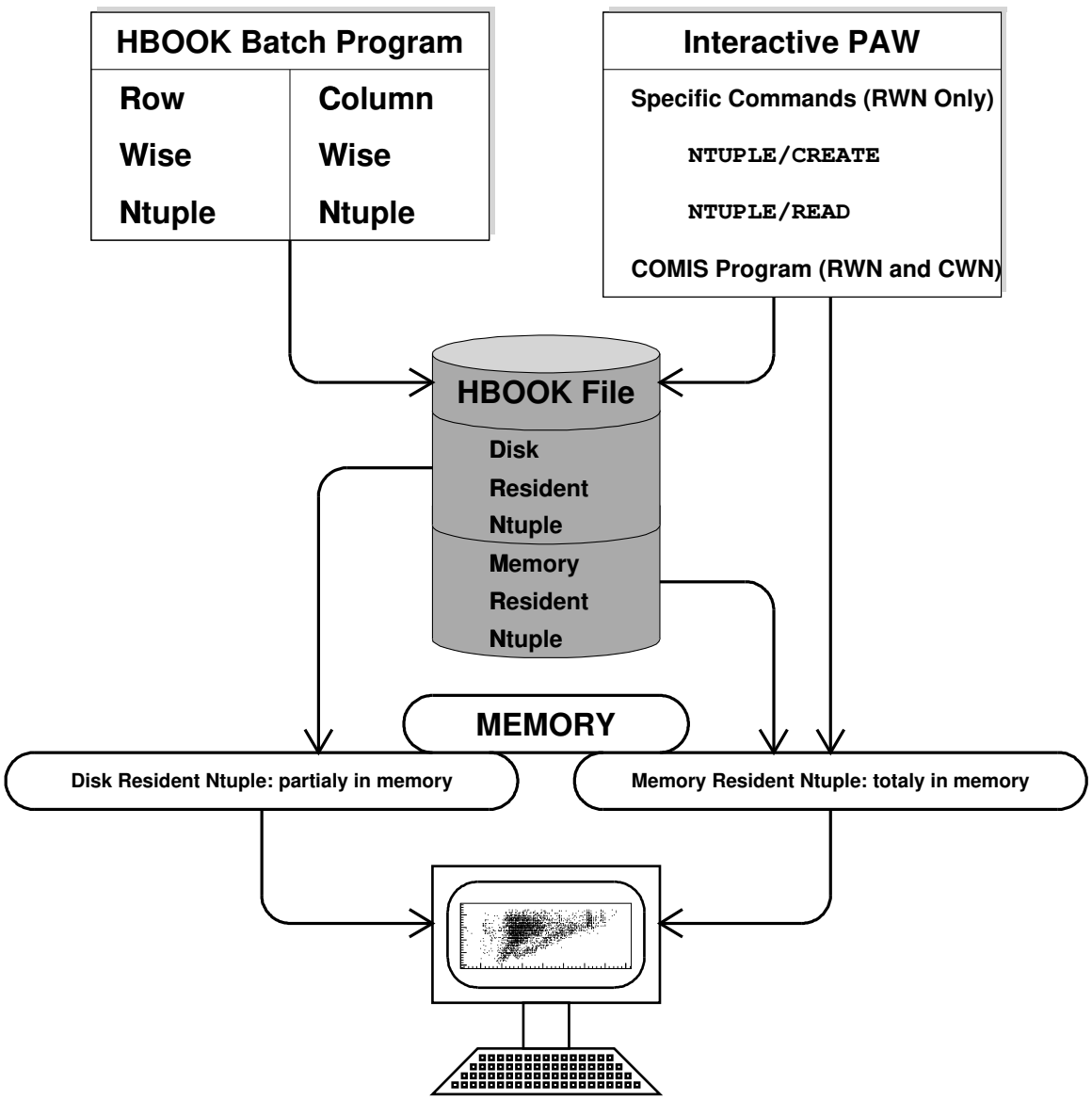

350
300
250
200
150
100
50
0
0
0.2
0.4
0.6
0.8
1
1.2
1.4
mix


160
140
120
100
80
60
40
20
0
0.4
0.5
0.6
0.7
0.8
0.9
1
1.1
mix

3.8 Ntuples—Tutorial



Ntuple Creation



pawtut41 (21/09/93)

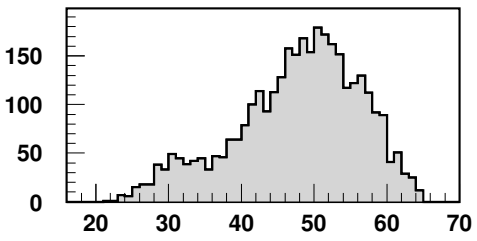
Ntuple cuts definition

The NTUPLE/CUTS command

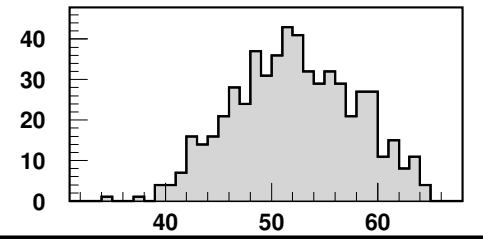
```
NTUPLE/CUTS CUTID [ OPTION FNAME ]
CUTID       Cut identifier
OPTION      Options
FNAME       File name
```

Define the CUTID with the format $nn. nn is an integer between 1 and 99.
This cut can then be used in subsequent commands NTUPLE/PLOT, PROJECT.

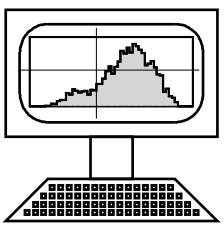
```
PAW > HI/FILE 1 pawtut.hbook
PAW > NTUPLE/PLOT 10.age
```

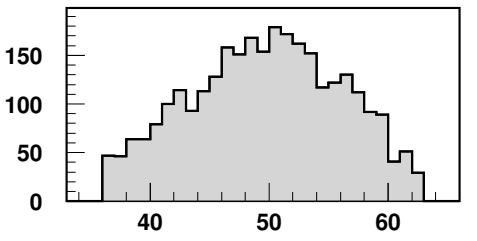



```
PAW > CUT $1 grade>10
PAW > NTUPLE/PLOT 10.age $1
```




```
PAW > CUT $6 G
```






```
PAW > NTUPLE/PLOT 10.age $6
```

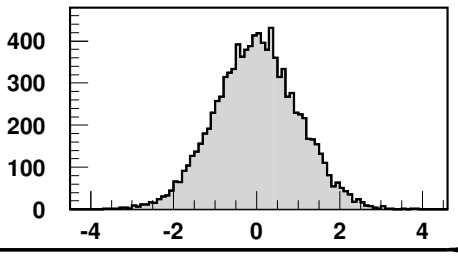
pawtut47 (21/09/93)

Ntuple Drawing

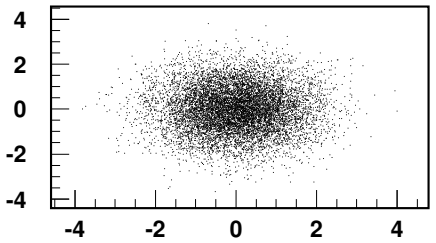
The NTUPLE/PLOT command

```
NTUPLE/PLOT IDN [ UWFUNC NEVENT IFIRST NUPD OPTION IDH ]
IDN         Ntuple Identifier
UWFUNC      Selection function
NEVENT      Number of events
IFIRST      First event
NUPD        Frequency to update histogram
OPTION      Options
IDH         Identifier of histogram to fill
```

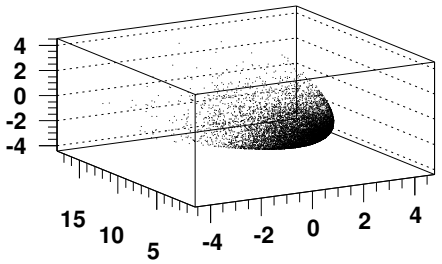
```
PAW > HI/FILE 1 hrztest.hbook
PAW > NTUPLE/PLOT 30.x
```




```
PAW > NTUPLE/PLOT 30.x%y
```




```
PAW > NTUPLE/PLOT 30.x%z%y
```



pawtut49 (21/09/93)

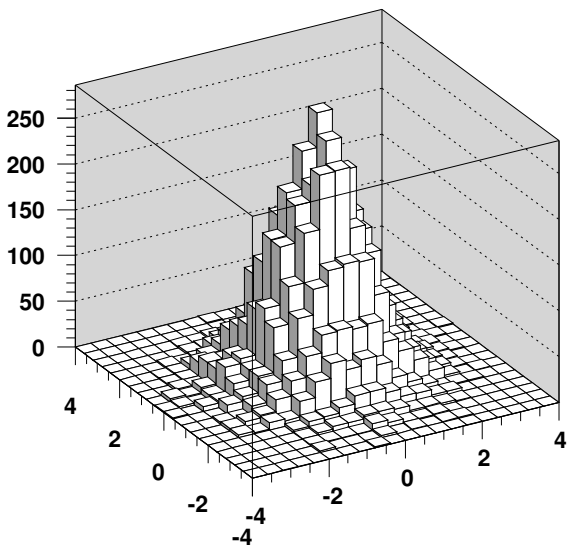
Ntuple Projection

The NTUPLE/PROJECT command

```
NTUPLE/PROJECT IDH IDN [ UWFUNC NEVENT IFIRST ]
IDH         Identifier of histogram to fill
IDN         Identifier of Ntuple
UWFUNC      Selection function or cut identifier
NEVENT      Number of events
IFIRST      First event
```

Project an Ntuple onto a 1-Dim or 2-Dim histogram, possibly using a selection function or predefined cuts.

```
PAW > 2DHISTO 2 'X vs Y' 20 -4 4 20 -4 4

PAW > NTUPLE/PROJECT 2 30.x%y

PAW > HISTO/PLOT 2 LEGO1
```



pawtut50 (21/09/93)

Loop on Ntuple Events

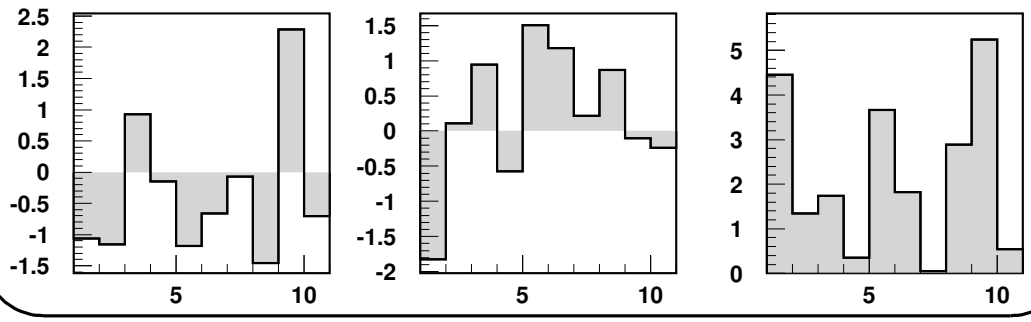
The NTUPLE/LOOP command

```
NTUPLE/LOOP IDN UWFUNC [ NEVENT IFIRST ]
IDN         Identifier of Ntuple
UWFUNC      Selection function or cut identifier
NEVENT      Number of events
IFIRST      First event
```

Invoke the selection function UWFUNC for each event starting at event IFIRST.

```
PAW > HISTO/FILE 1 hrztest.dat
PAW > NTUPLE/LOOP 30 copy.f
PAW > V/DR vx(1:10) ; V/DR vy(1:10) ; V/DR vz(1:10)
```

```
      REAL FUNCTION COPY(XDUMMY)
      COMMON/PAWIDN/IDNEVT,VIDN1,VIDN2,VIDN3,VIDN(10),
     +   X    ,   Y    ,   Z
*
      VECTOR VX(10000), VY(10000), VZ(10000)
*
      VX(IDNEVT) = X
      VY(IDNEVT) = Y
      VZ(IDNEVT) = Z
      END
```



pawtut51 (21/09/93)

3.9 Ntuples—Examples

3.9.1 Ntuple creation

Creation of an Row-Wise Ntuple (RWN) and first look at its contents

```
❶   NTUPLE/CREATE 10 'CERN Population' 11 ' ' 3500  _
    Category Division Flag Age Service Children Grade _
    Step Nation Hrweek Cost
    *
❷   NTUPLE/READ 10 APTUPLE.DAT
❸   HISTO/FILE 1 RWN_APTUPLE.HBOOK 1024 N
❸   HROUT 10
❹   NTUPLE/PRINT 10
    zone 1 2
❻   OPT STAT
❻   SET STAT 110
❹   NTUPLE/PLOT 10.Age
    ntuple/plot 10.Division
```

❶ `/NTUPLE/CREATE IDN TITLE NVAR CHRZPA NPRIME VARLIST`: Creates an Ntuple, i.e., a matrix of n columns. Each line of the matrix is often called an "event". Internally there is two different way to access the data: by rows (Row-Wise Ntuple) or by columns (Column-Wise Ntuple). The Ntuple may be created either in memory or, if necessary, using an automatic overflow to an histogram file.

❷ `NT/READ` allows to fill an RW/Ntuple with numeric values read from an existing `ASCII` file.

❸ Like histograms, Ntuples are `hbook` objects and can be stored into histogram files opened via the command `HIST/FILE`.

❹ The command `NT/PRINT` gives the description of the Ntuple (see next page).

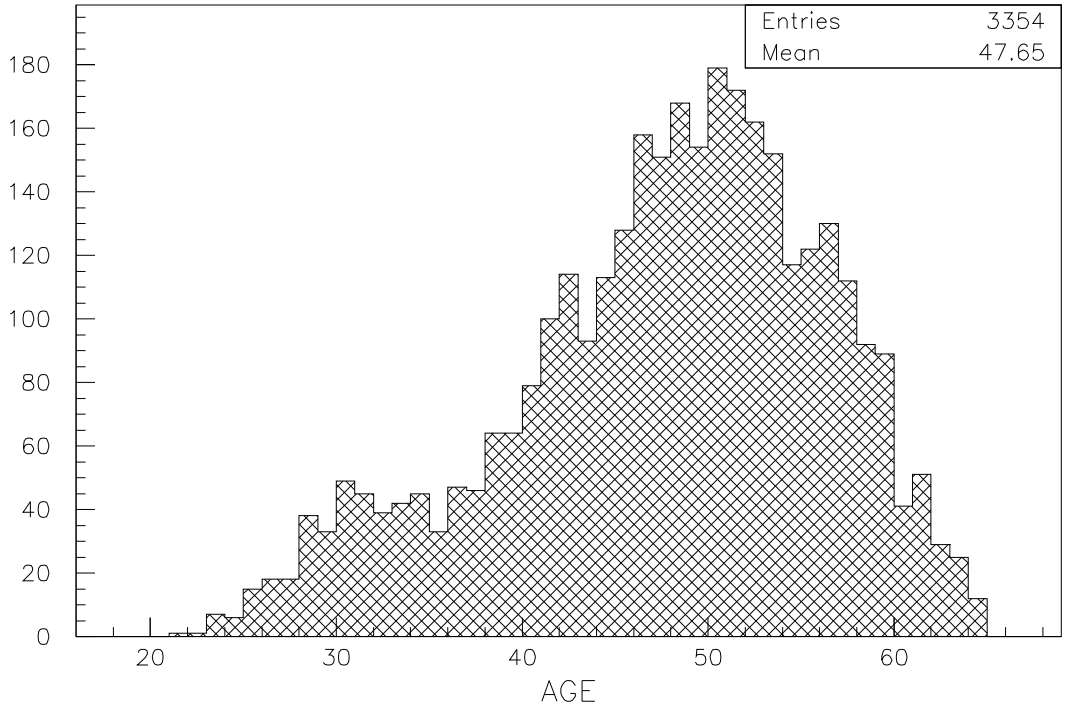
❺ `NT/PLOT` allows to plot an Ntuple. The syntax is:

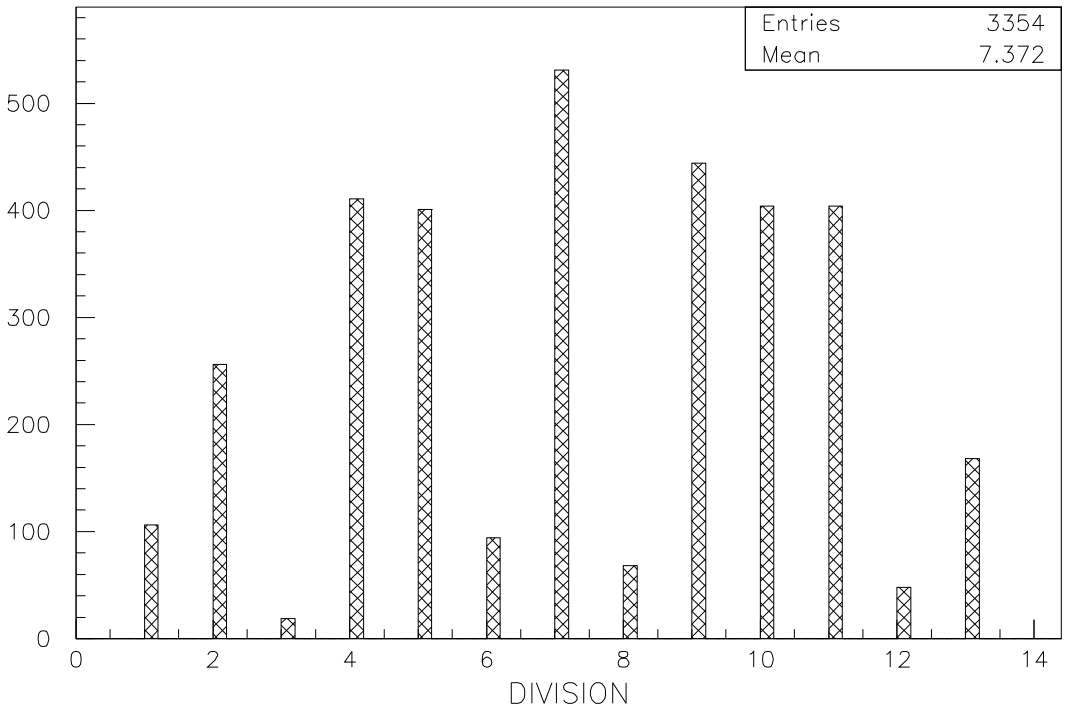
```
NT/PLOT nid.n ......
```

where "`nid`" is the Ntuple identifier (a number) and "`n`" is the number or the name of one of the variable in the Ntuple. By default, if "`n`" is not specified, the first variable of the Ntuple is ploted.

Note also:

❻ `OPT STAT` and `SET STAT` are used to plot some statistical informations.


Entries 3354
Mean 47.65
180
160
140
120
100
80
60
40
20
0
20
30
40
50
60
AGE


Entries 3354
Mean 7.372
500
400
300
200
100
0
0
2
4
6
8
10
12
14
DIVISION

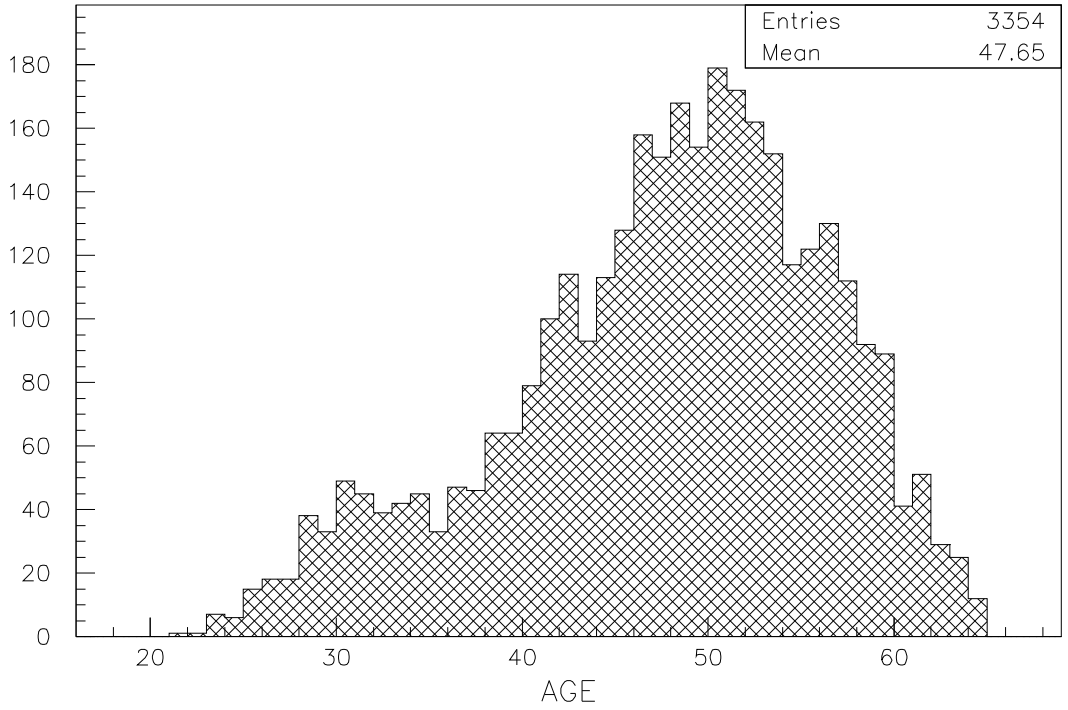
Creation of Column-Wise Ntuple (CWN)

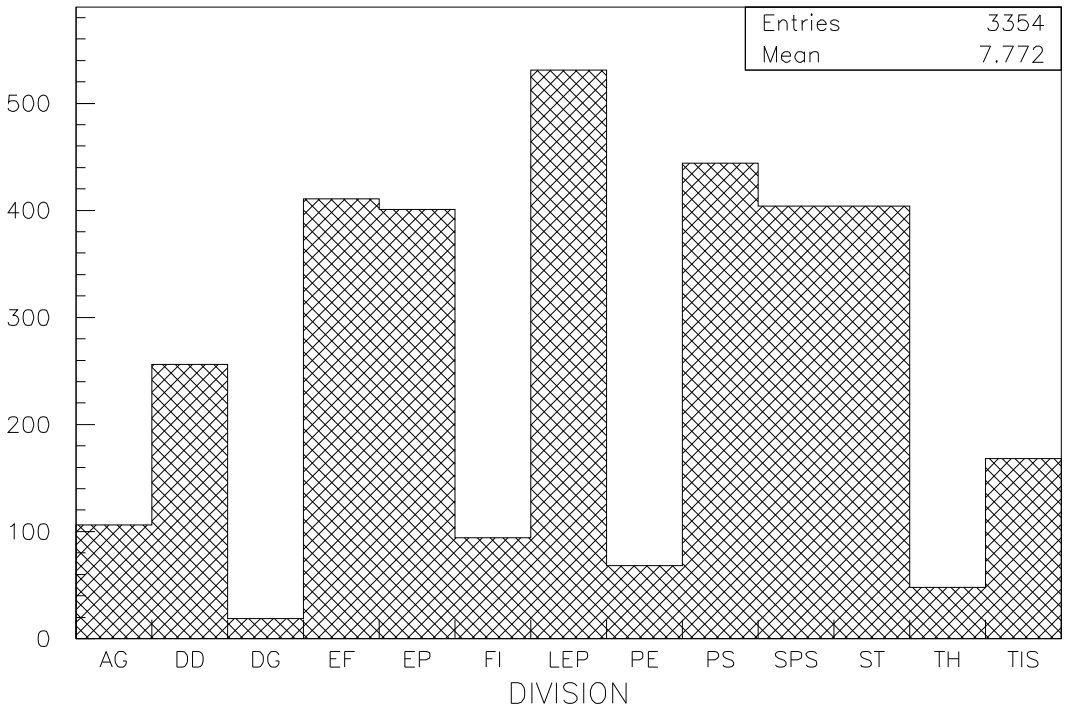
```
❶    HISTO/FILE 1 CWN_APTUPLE.HBOOK 1024 N
❷    CALL CERNPOP.F
     hrout 1
     ntuple/print 1
     zone 1 2
     opt stat
     set stat 110
     ntuple/plot 1.Age
❸    NTUPLE/PLOT 1.Division
```

❶ A new `hbook` file is open. If the Ntuple created after doesn't fit in memory, it will be automatically write on this file.

❷ This command create and read a CW/Ntuple. It is the equivalent of the `/NTUPLE/CREATE` and `/NTUPLE/READ` commands in the previous example (for the time being these commands work only with the RWN format). For more details on the CW/Ntuples management see the `hbook` manual.

❸ The axis are directly drawn with character labels.


Entries 3354
Mean 47.65
180
160
140
120
100
80
60
40
20
0
20
30
40
50
60
AGE


Entries 3354
Mean 7.772
500
400
300
200
100
0
AG
DD
DG
EF
EP
FI
LEP
PE
PS
SPS
ST
TH
TIS
DIVISION

COMIS routine used to create a CW/Ntuple

```
      Subroutine cernpop
*
      integer category, flag, age, service, children, grade, step,
     +        hrweek, cost
      common /cern/ category, flag, age, service, children, grade,
     +              step, hrweek, cost
      character*4   division, nation
      common /cernc/ division, nation
*
      character*132 chform
      dimension    rdata(11)
      character*4 divs(13), nats(15)
      data divs /'AG', 'DD', 'DG', 'EF', 'EP', 'FI', 'LEP', 'PE',
     +          'PS', 'SPS', 'ST', 'TH', 'TIS'/
      data nats /'AT', 'BE', 'CH', 'DE', 'DK', 'ES', 'FR', 'GB',
     +          'GR', 'IT', 'NL', 'NO', 'PT', 'SE', 'ZZ'/
*
      open(unit=41,file='aptuple.dat',status='old')
*
      call hbnt(1,'CERN Population (CWN)',' ')
      chform = ' CATEGORY[100,600]:I, FLAG:U:4, AGE[1,100]:I,'//
     +         ' SERVICE[0,60]:I, CHILDREN[0,10]:I, GRADE[3,14]:I,'//
     +         ' STEP[0,15]:I, HRWEEK:I, COST:I'
      call hbname(1, 'CERN', category, chform)
      chform = 'DIVISION:C,           NATION:C'
      call hbnamc(1, 'CERN', division, chform)
*
10    read(41, '(10F4.0, F7.0)', end=20) rdata
      category = rdata(1)
      division = divs(int(rdata(2)))
      flag     = rdata(3)
      age      = rdata(4)
      service  = rdata(5)
      children = rdata(6)
      grade    = rdata(7)
      step     = rdata(8)
      nation   = nats(int(rdata(9)))
      hrweek   = rdata(10)
      cost     = rdata(11)
      call hfnt(1)
      goto 10
*
  20  close (41)
      end
```

RWN NT/PRINT output

```
*********************************************************
* NTUPLE ID=   10  ENTRIES=   3354   CERN Population   *
*********************************************************
*  Var numb  *   Name    *    Lower     *    Upper     *
*********************************************************
*      1     * CATEGORY  * 0.102000E+03 * 0.567000E+03 *
*      2     * DIVISION  * 0.100000E+01 * 0.130000E+02 *
*      3     * FLAG      * 0.000000E+00 * 0.310000E+02 *
*      4     * AGE       * 0.210000E+02 * 0.640000E+02 *
*      5     * SERVICE   * 0.000000E+00 * 0.350000E+02 *
*      6     * CHILDREN  * 0.000000E+00 * 0.600000E+01 *
*      7     * GRADE     * 0.300000E+01 * 0.140000E+02 *
*      8     * STEP      * 0.000000E+00 * 0.150000E+02 *
*      9     * NATION    * 0.100000E+01 * 0.150000E+02 *
*     10     * HRWEEK    * 0.200000E+01 * 0.440000E+02 *
*     11     * COST      * 0.686000E+03 * 0.188530E+05 *
*********************************************************
```

CWN NT/PRINT output

```
*****************************************************************
* Ntuple ID = 1      Entries = 3354      CERN Population (CWN)  *
*****************************************************************
* Var numb * Type * Packing *    Range     *  Block   *  Name    *
*****************************************************************
*      1   * I*4  *    11   * [100,600]    * CERN     * CATEGORY
*      2   * U*4  *    4    *              * CERN     * FLAG
*      3   * I*4  *    8    * [1,100]      * CERN     * AGE
*      4   * I*4  *    7    * [0,60]       * CERN     * SERVICE
*      5   * I*4  *    5    * [0,10]       * CERN     * CHILDREN
*      6   * I*4  *    5    * [3,14]       * CERN     * GRADE
*      7   * I*4  *    5    * [0,15]       * CERN     * STEP
*      8   * I*4  *         *              * CERN     * HRWEEK
*      9   * I*4  *         *              * CERN     * COST
*     10   * C*4  *         *              * CERN     * DIVISION
*     11   * C*4  *         *              * CERN     * NATION
*****************************************************************
*  Block   * Unpacked Bytes * Packed Bytes *   Packing Factor    *
*****************************************************************
* CERN     *    44          *    22        *    2.000           *
* Total    *    44          *    22        *    2.000           *
*****************************************************************
* Number of blocks = 1     Number of columns = 11               *
*****************************************************************
```

3.9.2 Automatic and user binning

Read an Ntuple from a histogram file. Automatic and user binning

```
      hi/file 2 'rwn_aptuple.hbook'
      zon 2 2
      ntuple/pl 10.age
      1dhisto 11 'Age - User binning' 45 20. 65.
❷     SET NDVX -509
❶     NTUPLE/PROJECT 11 10.AGE
      hi/plot 11
      1dhisto 12 'Cost - User binning' 50 0. 20000.
❷     SET NDVX
      ntuple/plot 10.cost
      set ndvx -504
      ntuple/pl 10.Cost ! -12
```

❶ `NT/PROJECT` Project an Ntuple onto a 1-Dim or 2-Dim histogram. The histogram is not reset before the projection. This allows several PROJECTs from different Ntuples.

❷ By default the labeling on the axis is automatic. It possible to change the number of division via the commands `SET NDVX`, `SET NDVY` and `SET NDVY`. The number of divisions (`NDIV`) is calculated according to the following convention:

```
(NDIV = N1 + 100*N2 + 10000*N3)
```

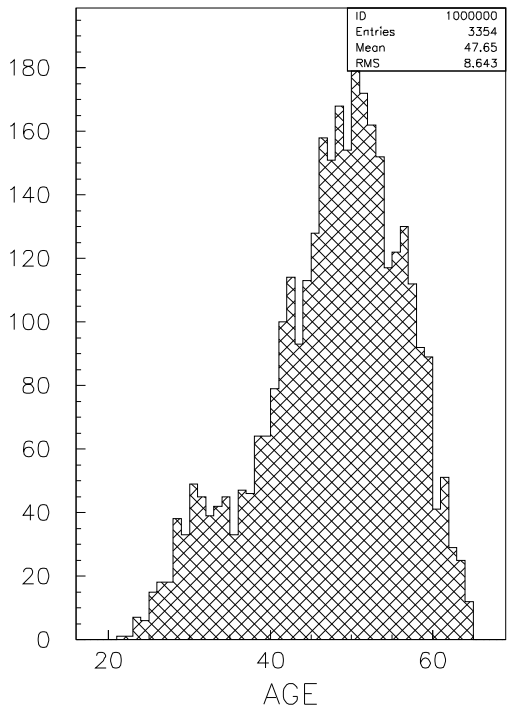
Where `N1` is the number of primary divisions, `N2` is the number of second order divisions and `N3` is the number of third order divisions.

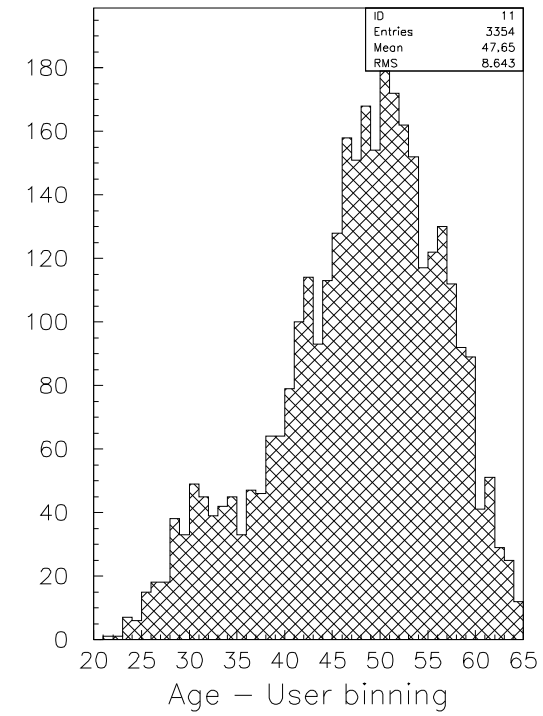
The sign of `NDIV` is also used to control the labeling:

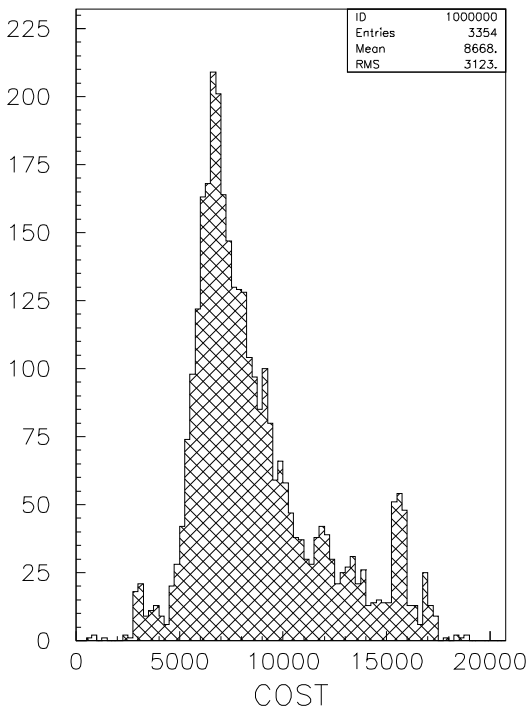
(a) If `NDIV` is positive, it is taken as a maximum number and the binning is optimized.

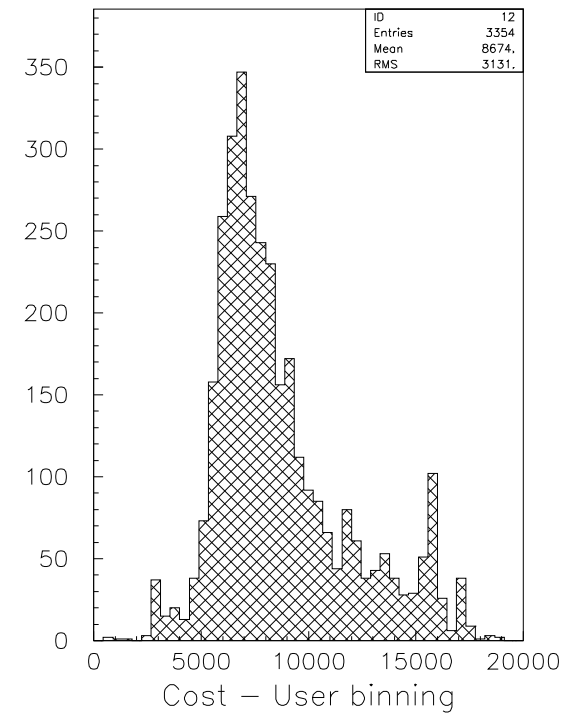
(b) If `NDIV` is negative, its absolute value is taken as the exact number of division without optimization.

(c) If `NDIV` equal zero is given the default (510. i.e. 10 primary divisions and 5 secondary) is taken.


ID 1000000
Entries 3354
Mean 47.65
RMS 8.643
AGE


ID 11
Entries 3354
Mean 47.65
RMS 8.643
Age – User binning


ID 1000000
Entries 3354
Mean 8668.
RMS 3123.
COST


ID 12
Entries 3354
Mean 8674.
RMS 3131.
Cost – User binning

3.9.3 Simple selection criteria on Ntuple

Ntuple SCAN and the use of simple selection criteria

```
      hi/file 2 'rwn_aptuple.hbook'
❹     ALIAS/CREATE DIVEP 5
      alias/create NATFR 7
      cd //pawc
      *
❶ ❷   NT/SCAN //LUN2/10 nation=NATFR.and.division=DIVEP _
      ! ! 5 age service children grade step
      *
      hi/cr/1d 200 'Number of years at CERN' 35 0. 35.
      max 200 250
      set ndvx 507
      set htyp 235
❸     NT/PL //LUN2/10.SERVICE ! -200
❺     ATITLE 'Years at CERN' 'Number of staff'
      set htyp 253
❷ ❸   NT/PL //LUN2/10.SERVICE NATION=NATFR -200 ! ! S
      set htyp 250
      nt/pl //LUN2/10.Service division=DIVEP.and.nation=NATFR -200 ! ! S
```

❶ `NT/SCAN` prints in an alphanumeric way the content of an Ntuple. On the next page is given the output of this command.

❷ In the commands `NT/PLOT` and `NT/SCAN`, the second parameter is the selection criteria. Only the events satisfying this selection are taken into account.

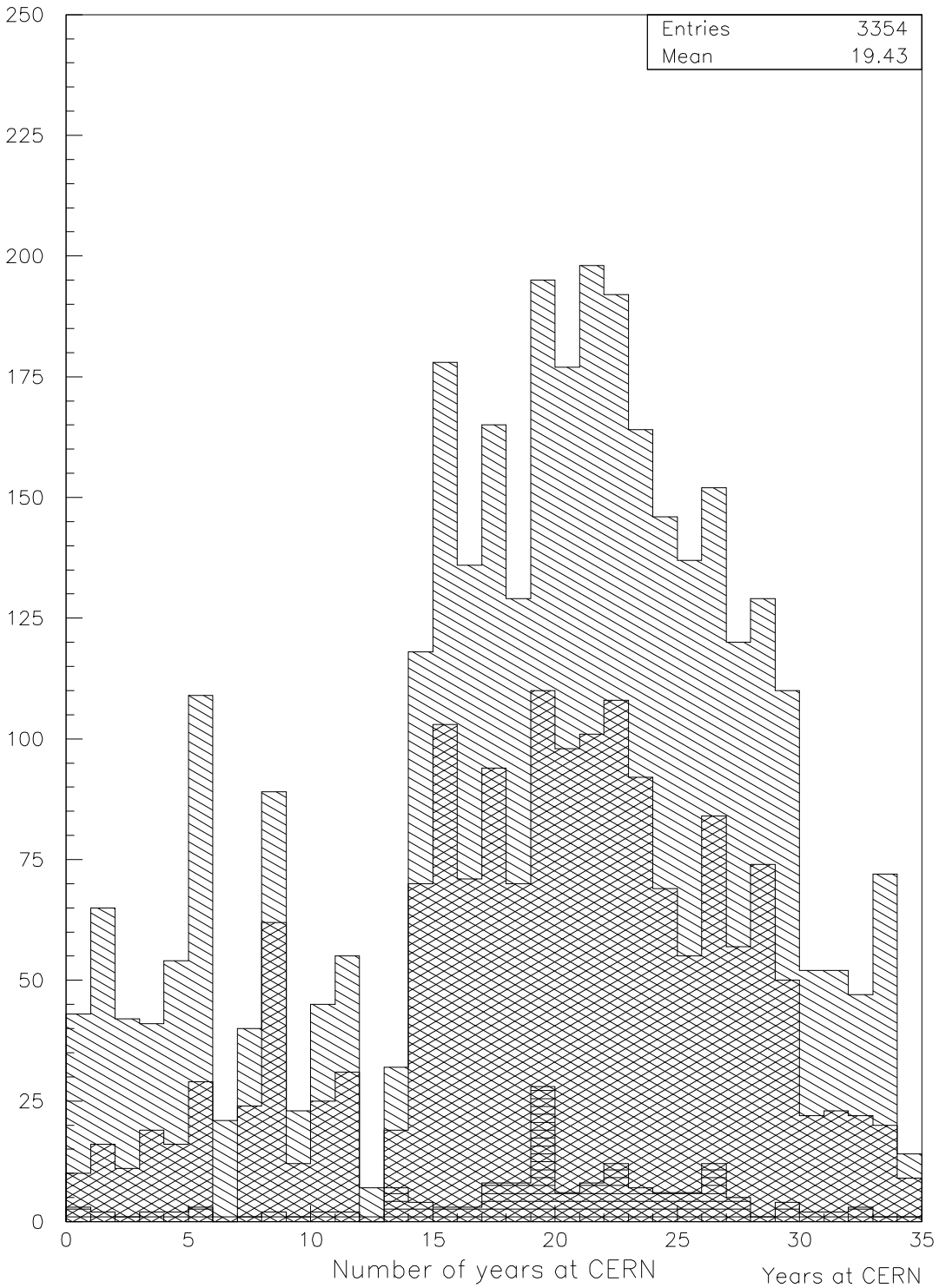
❸ By default `NT/PLOT` fill an histogram with the indentifier 1000000. The next invocation of this command will overwrite the content of this histogram. If either `NEVENT` or `IFIRST` or `NUPD` are negative, then the identifier of the histogram being filled will be taken as `IDF=-NEVENT` or `IDF=-IFIRST` or `IDF=-NUPD`. IDF may have been created with `H/CREATE`. Before filling `IDF`, the contents of `IDF` are reset if `IDF` already exists. Note that `IDF` not equal to 1000000 is a convenient way to force user binning. This is used here.

We'll see later another way to fill an histogram with data read in an Ntuple.

Note also:

❹ The aliases allow to define shortcut abbreviations. The aliases are known globally e.g. in all macros and in command mode.

❺ `ATITLE` allows to define the title on the axis.


Entries 3354
Mean 19.43
250
225
200
175
150
125
100
75
50
25
0
0
5
10
15
20
25
30
35
Number of years at CERN
Years at CERN

NT/SCAN output

```
*************************************************************************
* ENTRY *   AGE       *   SERVICE   *   CHILDREN  *   GRADE     *   STEP      *
*************************************************************************
!    48 !   56.000    !   34.000    !   .00000E+00!   7.0000    !   8.0000    !
!   194 !   62.000    !   27.000    !   .00000E+00!   7.0000    !   13.000    !
!   213 !   56.000    !   26.000    !   .00000E+00!   6.0000    !   13.000    !
!   214 !   45.000    !   26.000    !   .00000E+00!   6.0000    !   12.000    !
!   216 !   56.000    !   19.000    !   .00000E+00!   5.0000    !   13.000    !
!   266 !   63.000    !   26.000    !   .00000E+00!   13.000    !   10.000    !
!   267 !   59.000    !   32.000    !   .00000E+00!   13.000    !   10.000    !
!   273 !   55.000    !   26.000    !   1.0000    !   12.000    !   13.000    !
!   275 !   53.000    !   26.000    !   1.0000    !   11.000    !   13.000    !
!   279 !   51.000    !   30.000    !   .00000E+00!   6.0000    !   13.000    !
!   315 !   56.000    !   25.000    !   .00000E+00!   8.0000    !   6.0000    !
!   318 !   64.000    !   26.000    !   .00000E+00!   6.0000    !   13.000    !
!   320 !   49.000    !   26.000    !   .00000E+00!   6.0000    !   13.000    !
!   327 !   59.000    !   19.000    !   .00000E+00!   5.0000    !   13.000    !
!   328 !   51.000    !   25.000    !   .00000E+00!   5.0000    !   13.000    !
More...? ( <CR>/N/G ): n
==>     15 events have been scanned
```

3.9.4 Use of Ntuple masks and loops

Use of Ntuple masks and loops

```
      hi/file 2 'rwn_aptuple.hbook'
      1dhisto 20 'Distribution by grade' 12 3 15
      max 20 700
      ntuple/plot 10.grade ! -20
❶     NT/MASK STMASK n 3500
❷     NT/LOOP 10.GRADE STEP=15>>STMASK(1)
      nt/loop 10.grade grade>4.and.step=13>>stmask(2)
      nt/loop 10.grade _
      (grade=13.and.step=10).or.(grade=14.and.step=7)>>stmask(3)
      NT/PLOT 10.GRADE _
      STMASK(1).OR.STMASK(2).OR.STMASK(3)>>STMASK(4) -20 ! ! s
❸     NT/MASK STMASK P
❹     NT/MASK STMASK C
```

❶ NT/MASK perform operations with masks. A mask is a direct-access file with the name MNAME.MASK (here STMASK.MASK). It must contain as many 32 bit words as there are events in the associated Ntuple. Masks are interesting when only a few events of a Ntuple are selected with a time consuming selection algorithm.

❷ The symbol "≫" in NT/LOOP and NT/PLOT allows to fill the mask according to the selection function.

❸ This command allows to print the definition of the mask.

Output of the command NT/MASK STMASK P

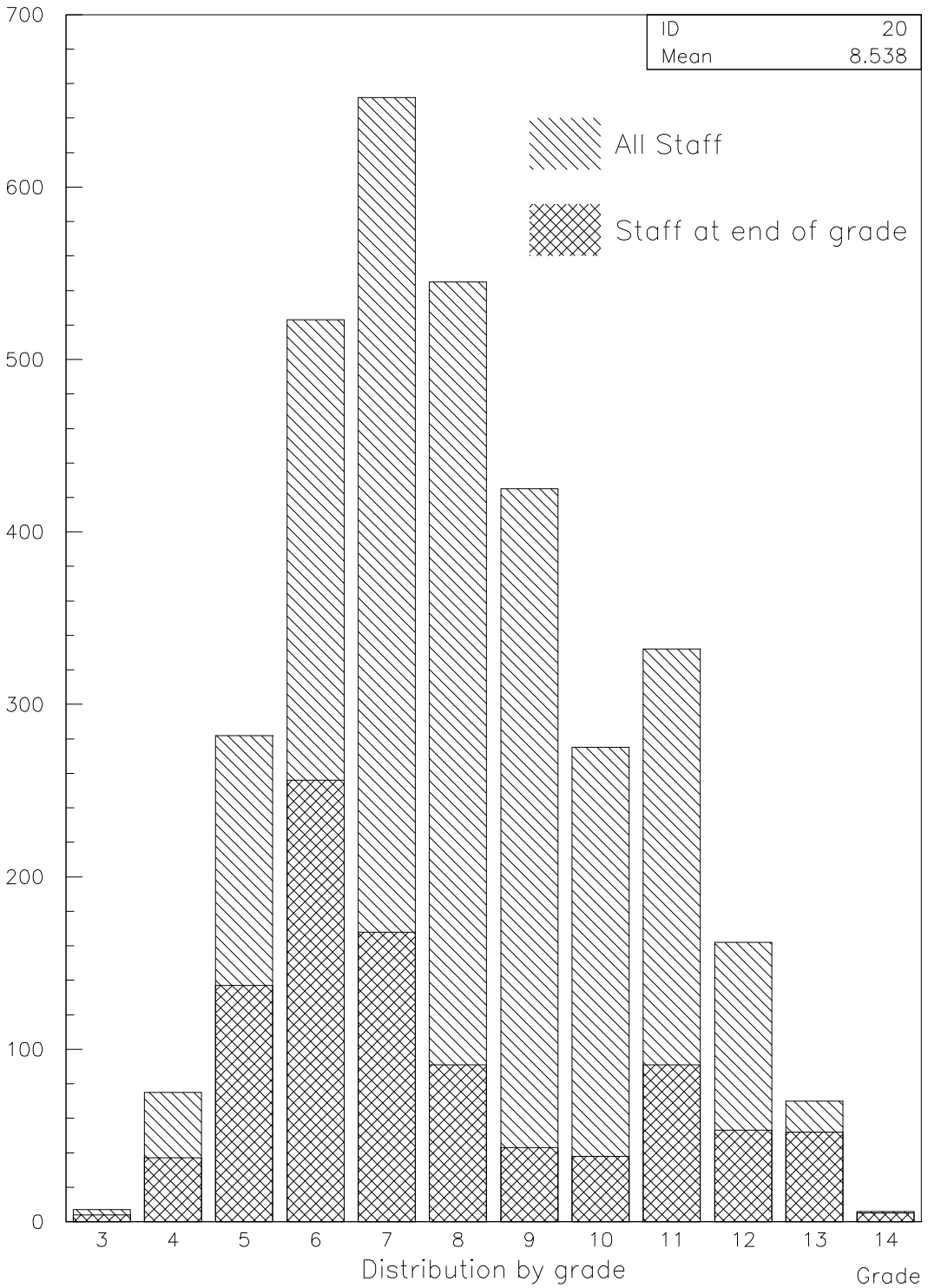
```
 =====> Current active selections in mask STMASK

 Bit  Nevents    Selection
   1      41     STEP=15

   2     877     GRADE>4.AND.STEP=13

   3      57     (GRADE=13.AND.STEP=10).OR.(GRADE=14.AND.STEP=7)

   4     975     STMASK(1).OR.STMASK(2).OR.STMASK(3)
```

❹ The option "C" in NT/MASK close the mask.

❺ Try NT/PLOT 10.GRADE STMASK(4): It produce the same result as the last NT/PLOT of the macro.

❻ Compare the execution time (with TIMING) of the two following commands:

```
NTUPLE/PLOT 10.GRADE (GRADE=13.AND.STEP=10).OR.(GRADE=14.AND.STEP=7)
NTUPLE/PLOT 10.GRADE STMASK(3)
```


ID 20
Mean 8.538
All Staff
Staff at end of grade
700
600
500
400
300
200
100
0
3
4
5
6
7
8
9
10
11
12
13
14
Distribution by grade
Grade

3.9.5 The use of Ntuple Cuts

The use of Ntuple Cuts

```
      hi/file 2 'rwn_aptuple.hbook'
❶     CUT $1 MOD(FLAG,2).EQ.0
❶     CUT $2 MOD(FLAG,4)>1
      1d 20 'Male/female and resident/non-resident Staff' 13 1 14
❷     OPT BAR
❷     SET BARW 0.4
❷     SET BARO 0.1
      max 20 600
❸     LABELS 1 13 AG DD DG EF EP FI LEP PE PS SPS ST TH TIS
      set NDVX 13.15
      set ndvy -506
      ntuple/plot 10.division ! -20
      set htyp 244
      ntuple/plot 10.division $2 -20 ! ! s
      set baro 0.5
      set htyp 145
      ntuple/plot 10.division $1 -20 ! ! s
      set htyp 154
      ntuple/plot 10.division $1.and.$2 -20 ! ! s
      ATITLE 'Division' 'Number of staff'
```

❶ `NTUPLE/CUTS` defines a cut identifier with the format $nn. It is possible to store the cuts in a file with the option "`W`" and read them afterwards with the option "`R`". When a cut is defined it can be used in commands like `NT/PLOT`, `NT/PROJ` etc ...

It also possible to define "graphical cuts". They are specified interactively with the mouse.

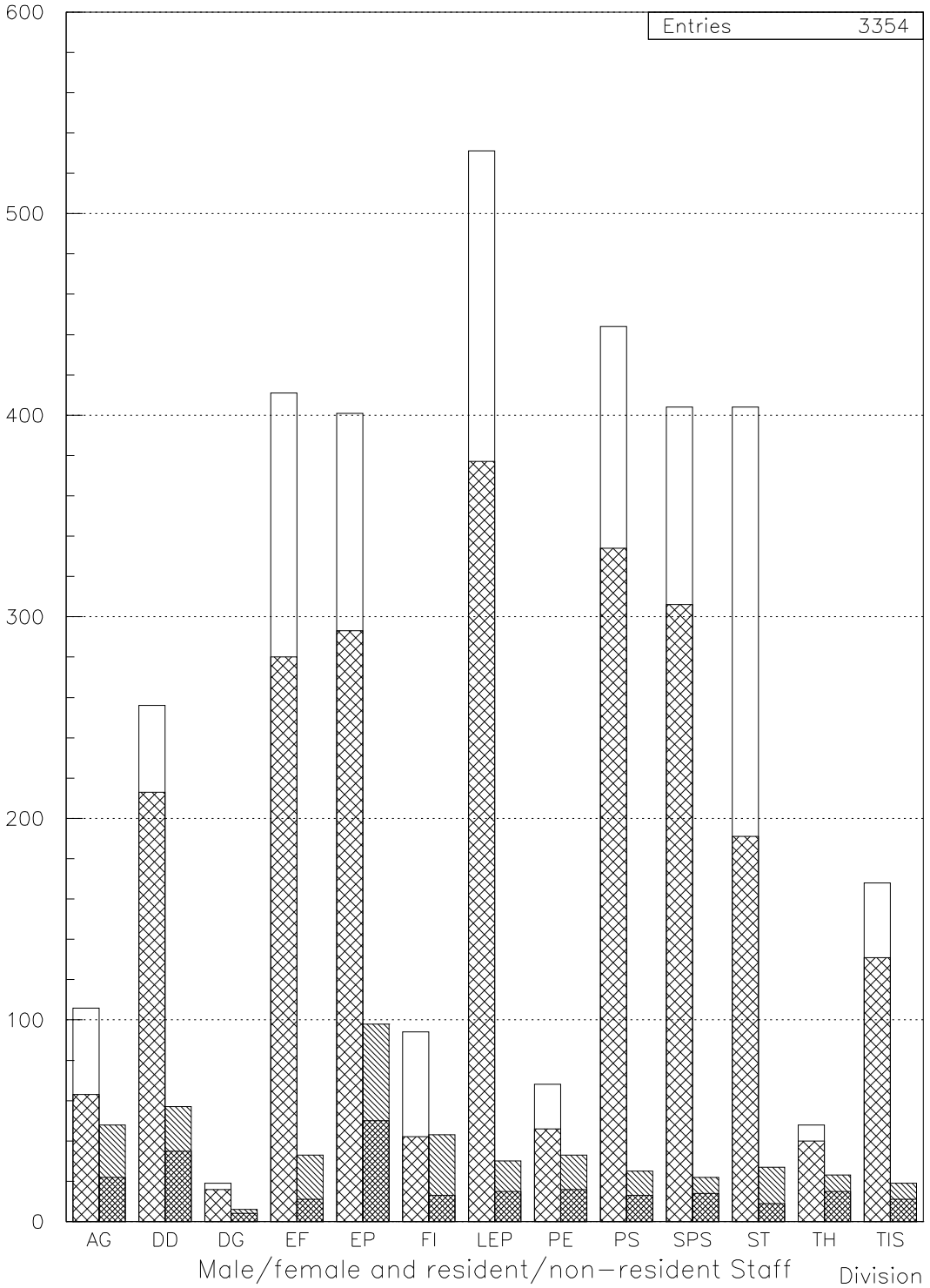
When option G is selected, graphical cuts are only operational for plots of the original Ntuple variables, not for expressions of these variables.

Note also:

❷ The "`BAR`" option and the attributes "`BARW`" and "`BARO`" allow to draw bar charts. `OPTION BAR` is also active on `LEGO` plots.

❸ `LABELS` used with `SET NDVX` or `SET NDVY` allows to produce alphanumeric labeling.

❹ Histograms with alphanumeric binning are now available in `hbook`. A set of routines is available to manage such histograms. In `paw`, the command `SORT` allows to reorder the labels.


Entries 3354
600
500
400
300
200
100
0
AG
DD
DG
EF
EP
FI
LEP
PE
PS
SPS
ST
TH
TIS
Male/female and resident/non−resident Staff
Division

3.9.6 Ntuple and 2D histograms

2D Ntuple distributions and 2D histograms projections

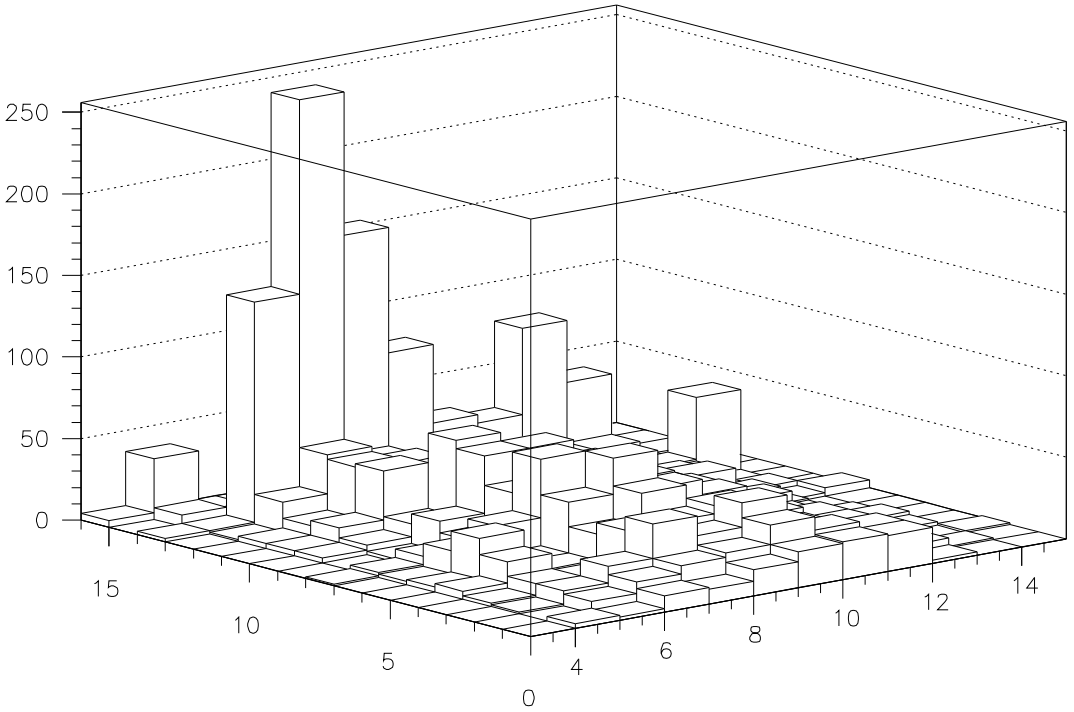
```
     hi/file 2 'rwn_aptuple.hbook'
     clr
     2d 20 ' ' 12 3 15 16 0 16 0.
❶    NT/PROJECT 20 //lun2/10.STEP%GRADE
     lego 20 20 40
❷    PROX 20
❸    H/PRO 20
❹    H/PLOT 20.prox
```

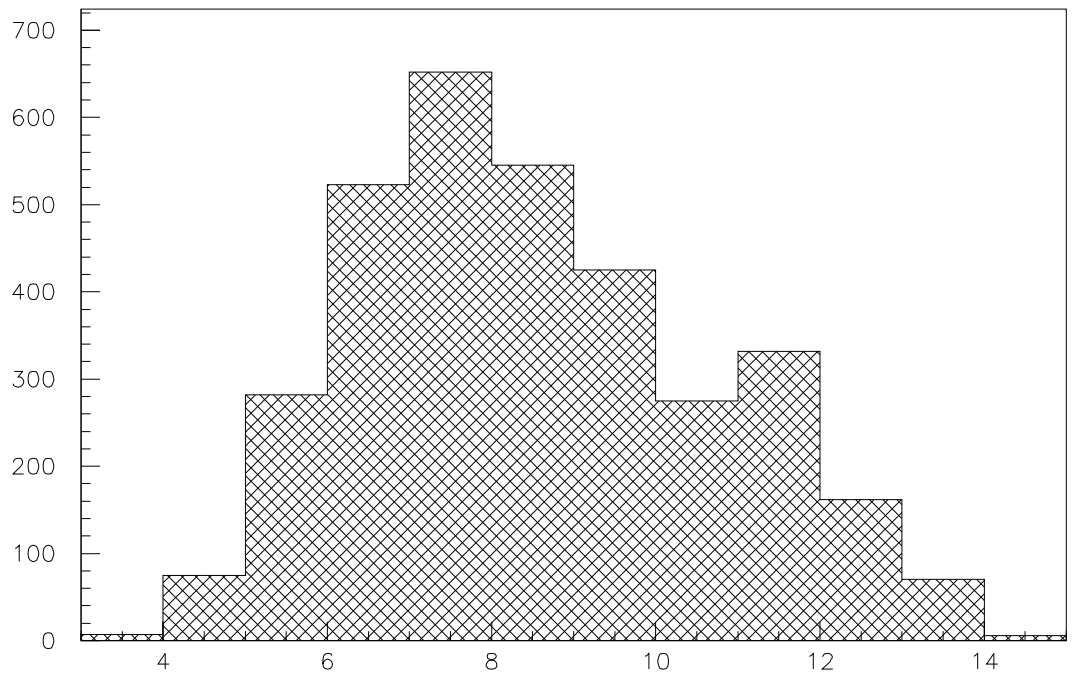
❶ The symbol "%" is used to produce multiple dimensional distributions with ntuples. The maximum number of dimension is 10. `NT/PROJ` allows to fill an histogram with data read in a Ntuple without plotting the result.

❷ Create the projection onto the x axis. The commands `PROX, SLIX, SLIY, BANX` and `BANY` allows to define other type of projections.

❸ Fill the projection.

❹ Plot the projection.


250
200
150
100
50
0
15
10
5
0
4
6
8
10
12
14


700
600
500
400
300
200
100
0
4
6
8
10
12
14

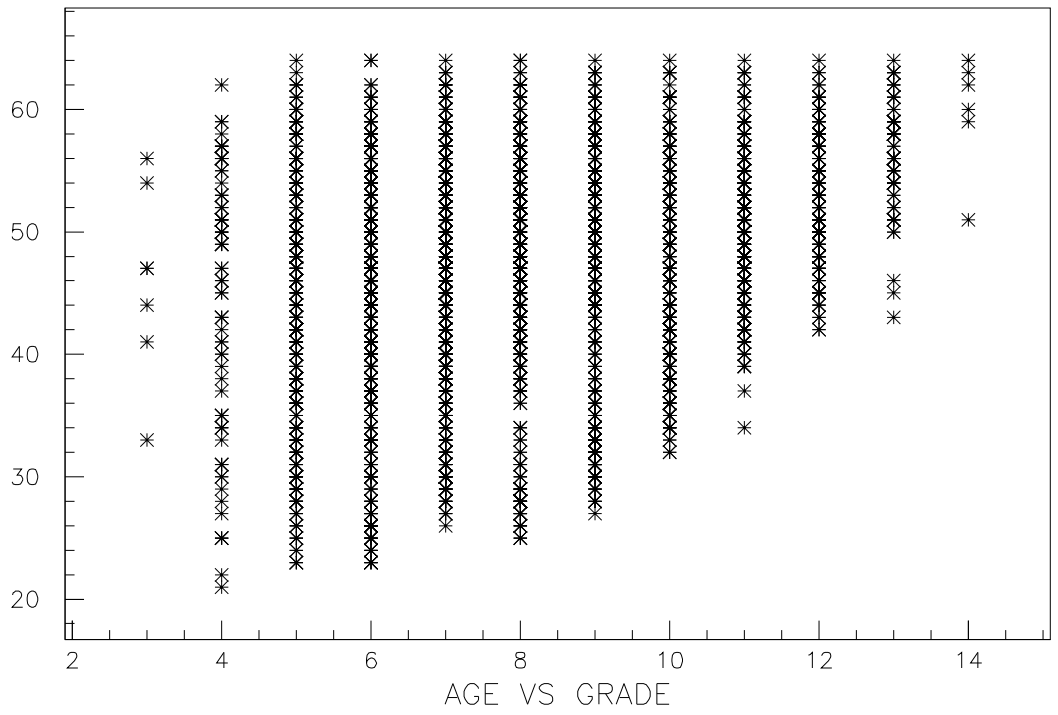
3.9.7 Profile histograms and Ntuples

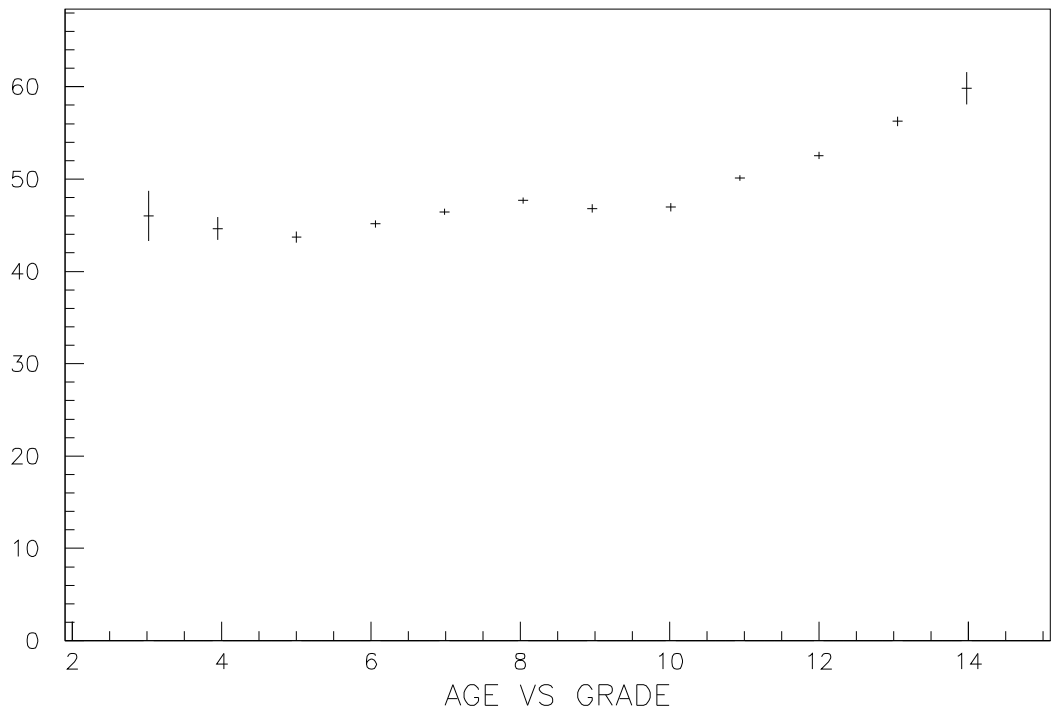
How to create a profile histogram from a Ntuple

```
    hi/file 2 'rwn_aptuple.hbook'
    zone 1 2
    set MTYP 3
❶   NT/PLOT //LUN2/10.age%grade
❷   NT/PLOT //LUN2/10.age%grade option=prof
```

❶ The command `NT/PLOT` produce a bi-dimensional distribution represented as a scatter plot with the current marker type.

❷ When the option `PROF` is used, a profile histogram is produce. A profile histogram, is a 1D histogram which gives for each value of `X` the mean value of `Y` and its RMS (for more details see the `hbook` manual: routine `HBPROF`).


60
50
40
30
20
2
4
6
8
10
12
14
AGE VS GRADE


60
50
40
30
20
10
0
2
4
6
8
10
12
14
AGE VS GRADE

3.9.8 Copy a Ntuple variable into a Vector

Copy a Ntuple variable into a Vector

```
      hi/file 2 'aptuple.hbook'
❶     UWFUNC 10 copy.f E
❷     NT/LOOP 10.age copy.f
      zone 1 2
      vect/draw x
      vect/plot x
```

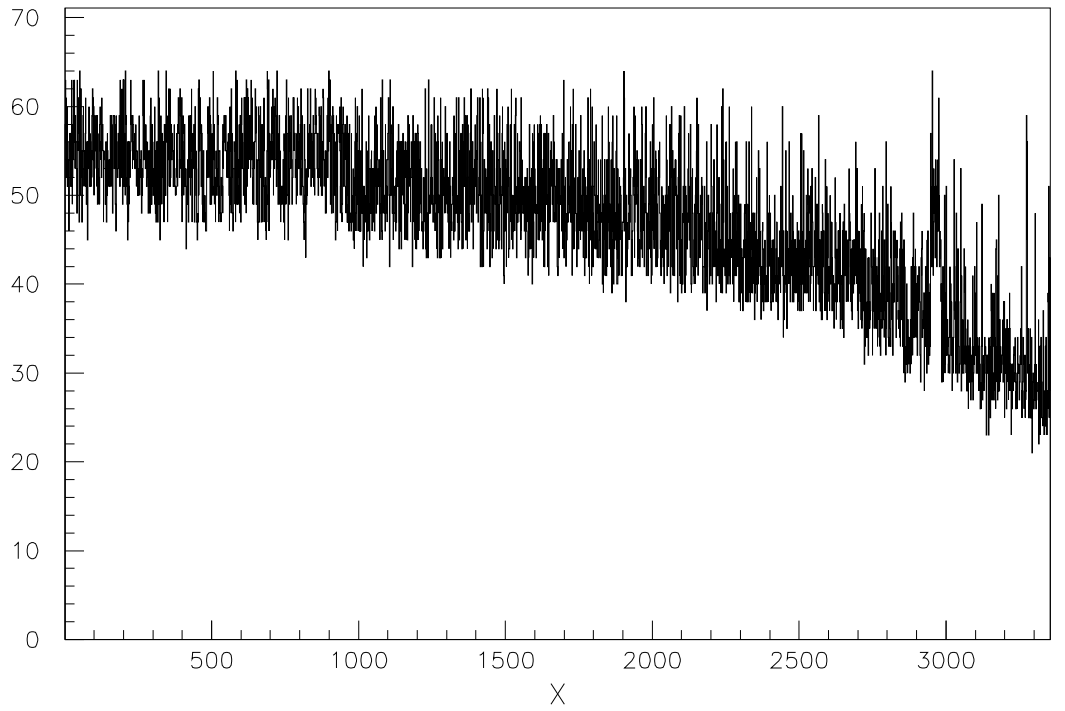
The routine copy.f

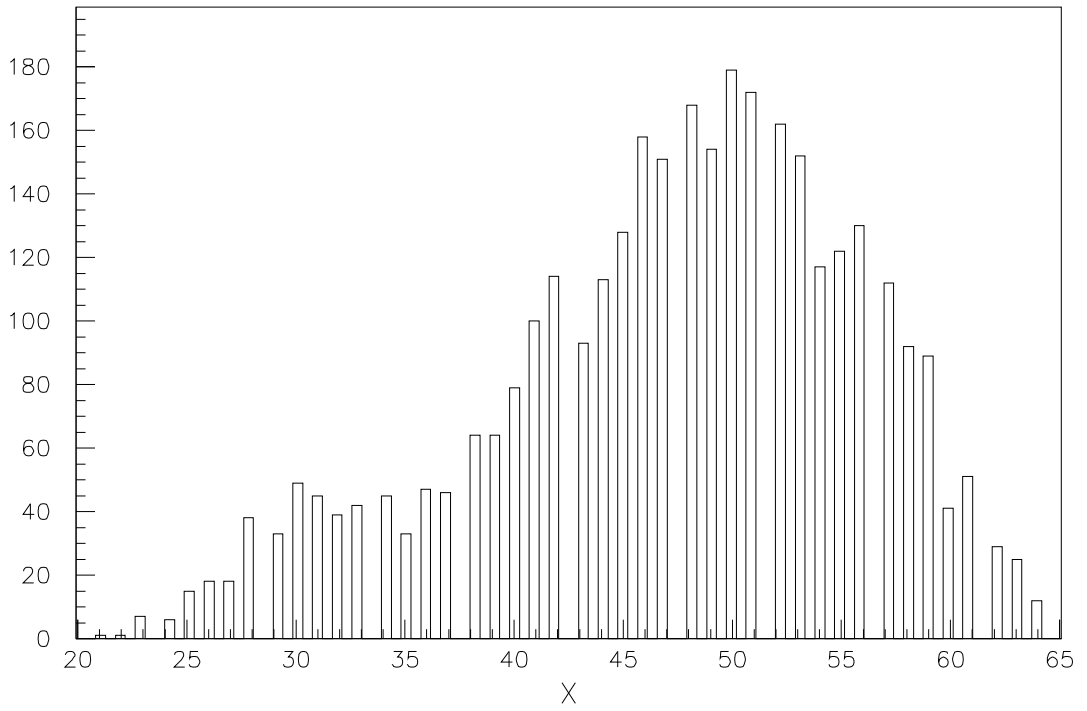
```
      REAL FUNCTION COPY(XDUMMY)
      REAL
     +CATEGORY,DIVISION,FLAG    ,AGE     ,SERVICE ,CHILDREN,
     +GRADE   ,STEP    ,NATION  ,HRWEEK  ,COST
      COMMON/PAWIDN/IDNEVT,VIDN1,VIDN2,VIDN3,VIDN(10),
     +CATEGORY,DIVISION,FLAG    ,AGE     ,SERVICE ,CHILDREN,
     +GRADE   ,STEP    ,NATION  ,HRWEEK  ,COST
*
❸     VECTOR X(3354)
      X(IDNEVT)=VIDN1
      END
```

❶ This command allows to define the skeleton of the FORTRAN routine used by NTUPLE/LOOP.

❷ For each event, NTUPLE/LOOP calls copy.f.

❸ The declaration VECTOR may be used inside a COMIS routine to address a KUIP vector. If the vector does not exist, it is created with the specifications provided by the declared dimension.


70
60
50
40
30
20
10
0
500
1000
1500
2000
2500
3000
X


180
160
140
120
100
80
60
40
20
0
20
25
30
35
40
45
50
55
60
65
X

3.9.9 Chain of Ntuples

This example simulate a CERN population of 335400 people.

A 10MB ntuple chain

```
      opt stat
❶     CHAIN MB05   newaptuple.hbook newaptuple.hbook newaptuple.hbook _
                   newaptuple.hbook newaptuple.hbook
❶     CHAIN MB1    MB05 MB05
❶     CHAIN MB10   MB1 MB1 MB1 MB1 MB1 MB1 MB1 MB1 MB1 MB1
❷     CHAIN
❸     CHAIN MB1>
❹     CD MB10
      Nt/plot 11.age
❺     CHAIN -MB10
```

❶ Create the chain.

❷ List all the chains.

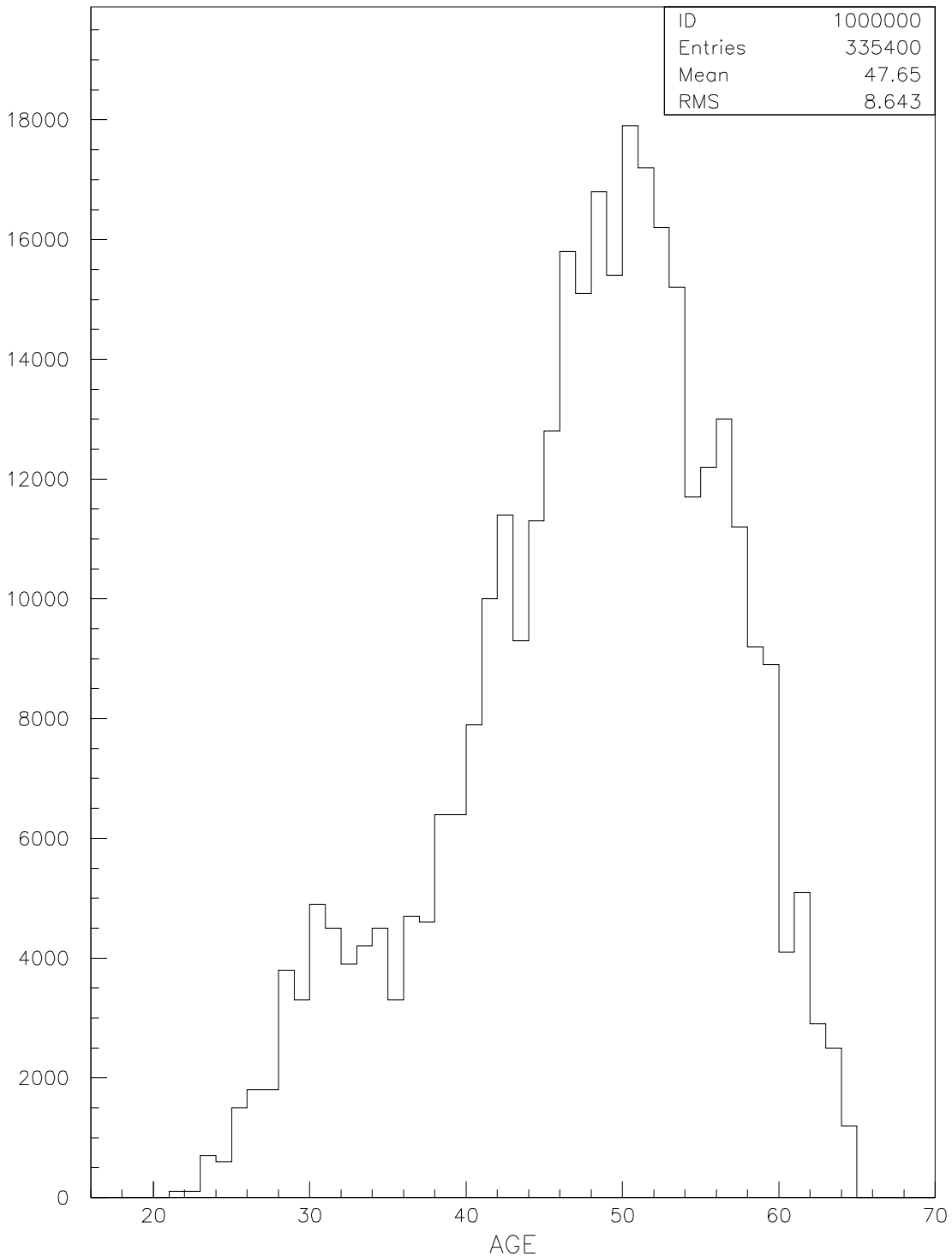
❸ Give the tree of the chain `MB1`.

❹ Set the current chain (`MB10`).

❺ Delete the chain `MB10`.

List of the chains and tree of `MB1`.

```
MB05      MB1       MB10

MB1
   MB05
      newaptuple.hbook   newaptuple.hbook   newaptuple.hbook
      newaptuple.hbook   newaptuple.hbook
   MB05
      newaptuple.hbook   newaptuple.hbook   newaptuple.hbook
      newaptuple.hbook   newaptuple.hbook
```


ID 1000000
Entries 335400
Mean 47.65
RMS 8.643
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
20
30
40
50
60
70
AGE

3.10 SIGMA—Examples

3.10.1 Examples of the SIGMA processor (1)

Examples of the SIGMA processor (1)

```
      zone 2 2
❷     APPLICATION SIGMA
         X=ARRAY(200,0#2*PI)
         sinus=sin(x)
         sinx=sin(x)/x
❷     EXIT
      gra 200 x sinus
      set dmod 2
      gra 200 x sinx l
      set dmod 0
❶     SIGMA x=array(300,0#8)
      sigma g=cosh(x)+sin(1/(.1+x*x))
      gra 300 x g
      sigma x=array(300,0#3)
❸     GRAPH 300 x $SIGMA(cosh(x)+sin(1/(.1+X*X)))
      sigma x=array(300,0#1)
❹     GRAPH 300 x $RSIGMA(cosh(x)+sin(1/(.1+X*X)))
```

This example (and the next one) shows how to use the array manipulation package `sigma`. There are four ways to give directives to `sigma`.

❶ Precede the statement by the prefix `SIGMA`.

❷ The `paw` command: `APPLication SIGMA`. All commands typed in after this command will be directly processed by `sigma`. The command `EXIT` will return control to `paw`.

❸ The `paw` system function `$SIGMA`. The expression to be evaluated must be enclosed in parentheses. The function will return the numerical value of the expression (if the result is a scalar) or the name of a temporary vector (if the result is a vector).

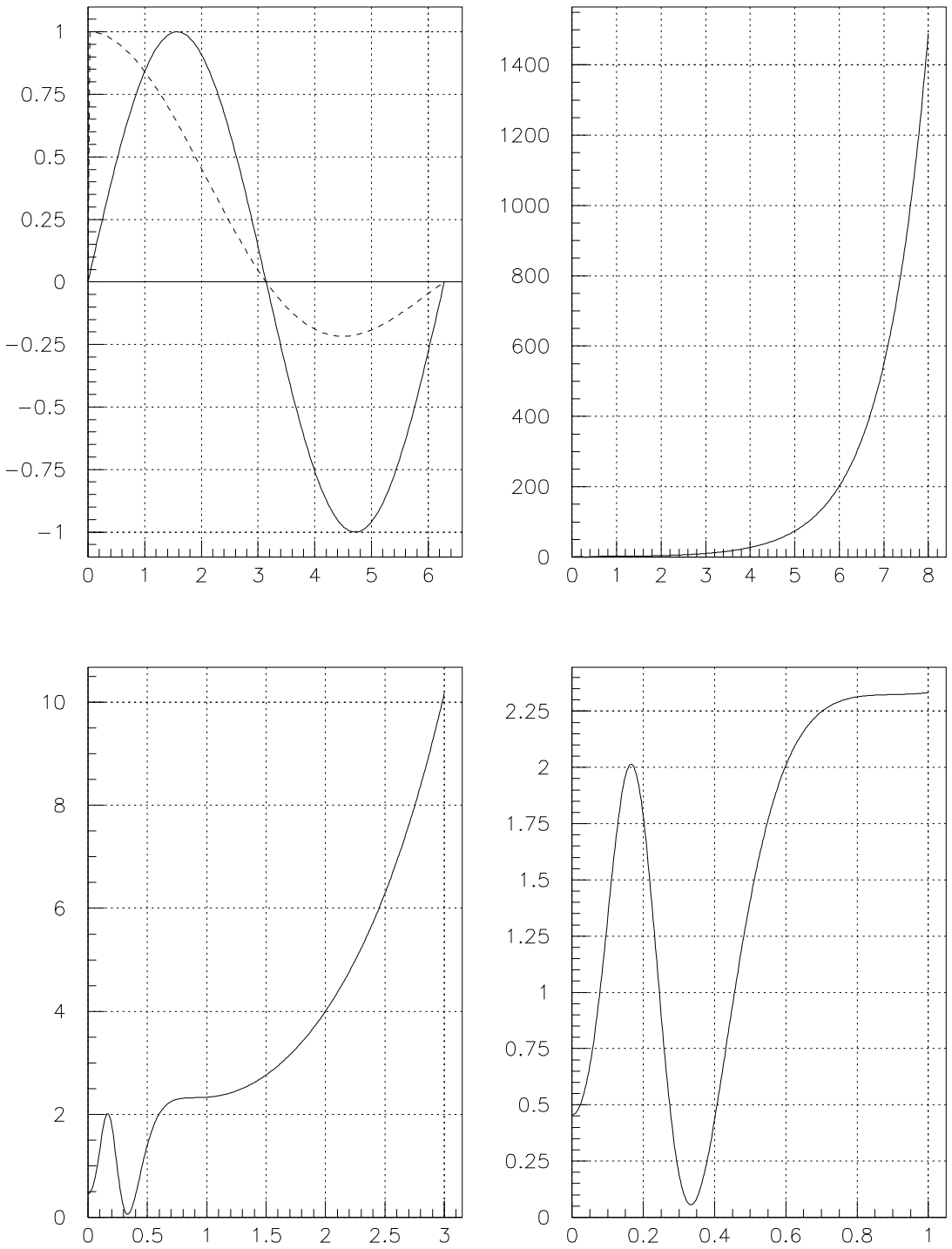
❹ The `paw` system function `$RSIGMA`. This function has be to used in `comis` calls expecting a `REAL` argument, e.g.

```
CALL file.f($RSIGMA(sqrt(x(1)))
```

Otherwise the value may be passed as an `INTEGER` if the `sigma` result turns out to be a whole number.

Note also:

The system function `$FORMAT(number,format)` to format a number according to a Fortran-like `FORMAT` string, e.g. `$FORMAT([x],F9.3)`. Supports F,E,G,I, and Z (hexadecimal). The complete list of the system functions available is given on next page.


1
0.75
0.5
0.25
0
−0.25
−0.5
−0.75
−1
0 1 2 3 4 5 6
1400
1200
1000
800
600
400
200
0
0 1 2 3 4 5 6 7 8
10
8
6
4
2
0
0 0.5 1 1.5 2 2.5 3
2.25
2
1.75
1.5
1.25
1
0.75
0.5
0.25
0
0 0.2 0.4 0.6 0.8 1

The function name (and arguments) is literally replaced, at run-time, by its current value. At present, the following functions are available:

The `kuip` System Functions

```
$DATE  ....................  Current date in format DD/MM/YY
$TIME  ....................  Current time in format HH.MM.SS
$CPTIME  ..................  CP time elapsed since last call (in sec)
$RTIME  ...................  Real time elapsed since last call (in sec)
$VDIM(VNAME,IDIM)  ........  Physical length of vector VNAME
                             on dimension IDIM (1..3)
$VLEN(VNAME,IDIM)  ........  As above, but for the logical length
                             (i.e. stripping trailing zeroes)
$NUMVEC  ..................  Current number of vectors
$VEXIST(VNAME)  ...........  Index of vector VNAME
                             (1..$NUMVEC or 0 if VNAME does not exist)
$SUBSTRING(STRING,IX,NCH)  ...  STRING(IX:IX+NCH-1)
$UPPER(STRING)  ...........  STRING changed to upper case
$LOWER(STRING)  ...........  STRING changed to lower case
$LEN(STRING)  .............  Length of STRING, stripping
                             leading/trailing blanks and single quotes
$SIGMA(Expression)  .......  Result of the Expression computed by SIGMA
$RSIGMA(Expression) .......  As above but a decimal point is added to
                             integer results
$FORMAT(number,format)  ....  Format a number according to a Fortran
                             format string,
                             e.g. $FORMAT(1.5,F5.2) ==> ' 1.50'
$ARGS  ....................  Command line at program invocation
$KEYNUM  ..................  Address of latest clicked key in style GP
$KEYVAL  ..................  Value of latest clicked key in style GP
$LAST  ....................  Latest command line executed
$ANUM  ....................  Number of aliases
$ANAM(I)  .................  Name of I-th alias
$AVAL(I)  .................  Value of I-th alias
$STYLE  ...................  Current style as defined by SET/STYLE
```

3.10.2 Examples of the SIGMA processor (2)

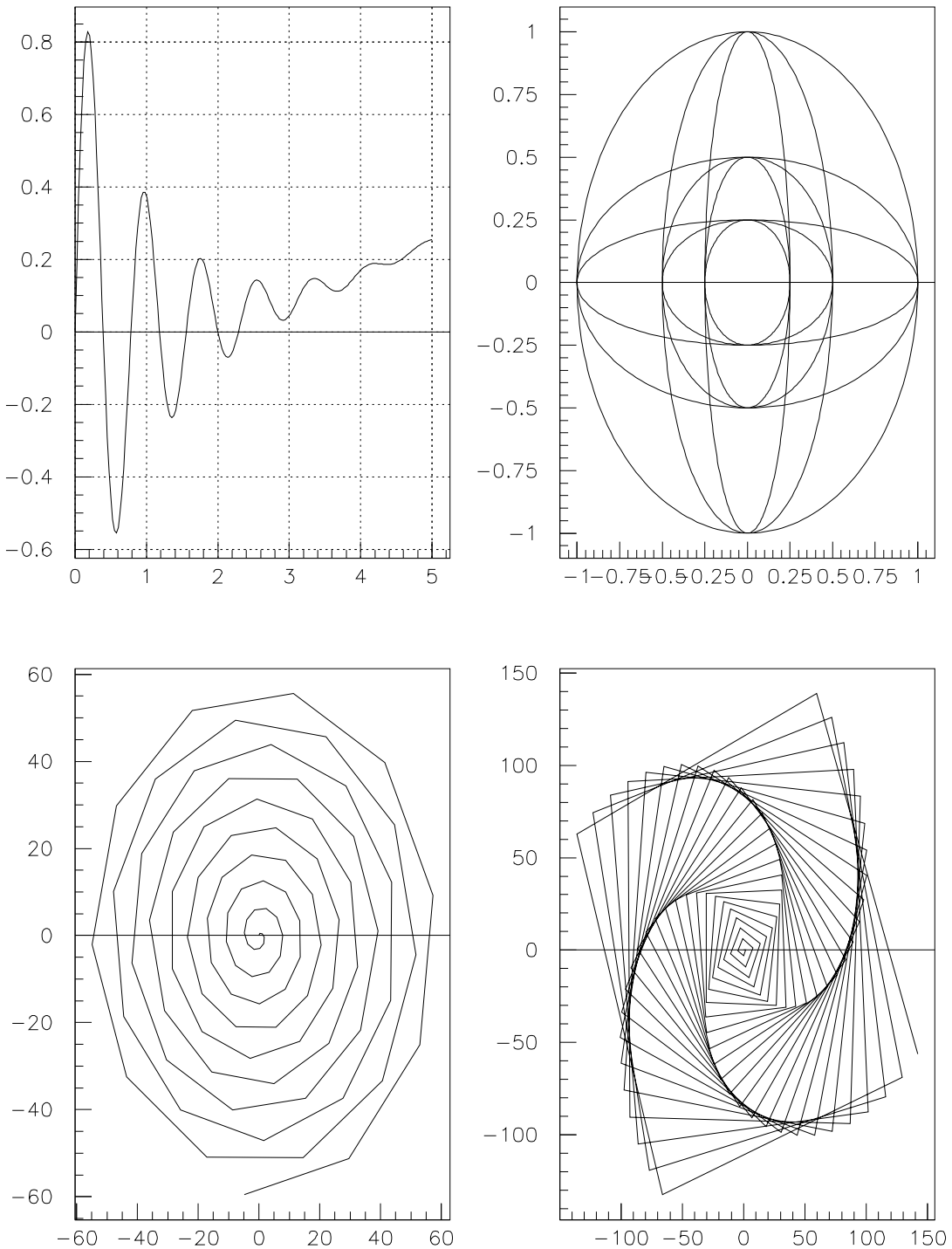
Examples of the SIGMA processor (2)

```
      zone 2 2
❶ ❸   SIGMA X=ARRAY(200,0#5)
❷     SIGMA A=8
      sigma b=.01
❸     SIGMA Y=EXP(-X)*SIN(A*X)+B*X*X
      gra 200 x y
      sigma x=array(200,0#2*pi)
      sigma s=sin(x)
      sigma s2=s/2
      sigma c=cos(x)
      sigma c2=c/2
      sigma s4=s/4
      sigma c4=c/4
      gra 200 s c
      gra 200 s2 c l
      gra 200 s4 c l
      gra 200 s c2 l
      gra 200 s2 c2 l
      gra 200 s4 c2 l
      gra 200 s c4 l
      gra 200 s2 c4 l
      gra 200 s4 c4 l
      sigma a=array(100,0#59.77)
      sigma nc=nco(a)
      sigma y=cos(a)*a
      sigma x=sin(a)*a
      gra nc x y
      sigma a=a*2.55555
      sigma y=cos(a)*a
      sigma x=sin(a)*a
      gra nc x y
```

❶ The command `V=ARRAY(L,x1#x2)` allows to create a vector `V` with the length `L` and initialize it in the range `x1,x2`.

❷ All the objects managed by `sigma` are vectors . In this example `A` is vector of length 1.

❸ The resulting vectors (if they don't exist) are created automatically by `sigma` (here `Y`).


0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
0 1 2 3 4 5
1
0.75
0.5
0.25
0
−0.25
−0.5
−0.75
−1
−1 −0.75 −0.5 −0.25 0 0.25 0.5 0.75 1
60
40
20
0
−20
−40
−60
−60 −40 −20 0 20 40 60
150
100
50
0
−50
−100
−100 −50 0 50 100 150

3.11 Pictures and PostScript

3.11.1 Merge pictures onto one plot

Merge pictures onto one plot

```
      histogram/file 1 pawhists.hbook
❺     SWITCH Z
❶     PIC/CR MERGE2
      set htyp 354
      hi/pl 110
      set htyp 345
      hi/pl 110(31:40) s
❶     PIC/CR MERGE1
      set htyp 354
      hi/pl 110(31:40)
❷     IZPICT MERGE2 C
      switch g
❸     PI/MERGE MERGE1 .5 .5 .3 D
❹     PI/DEL *
```

This example shows some application of the `higz` pictures.

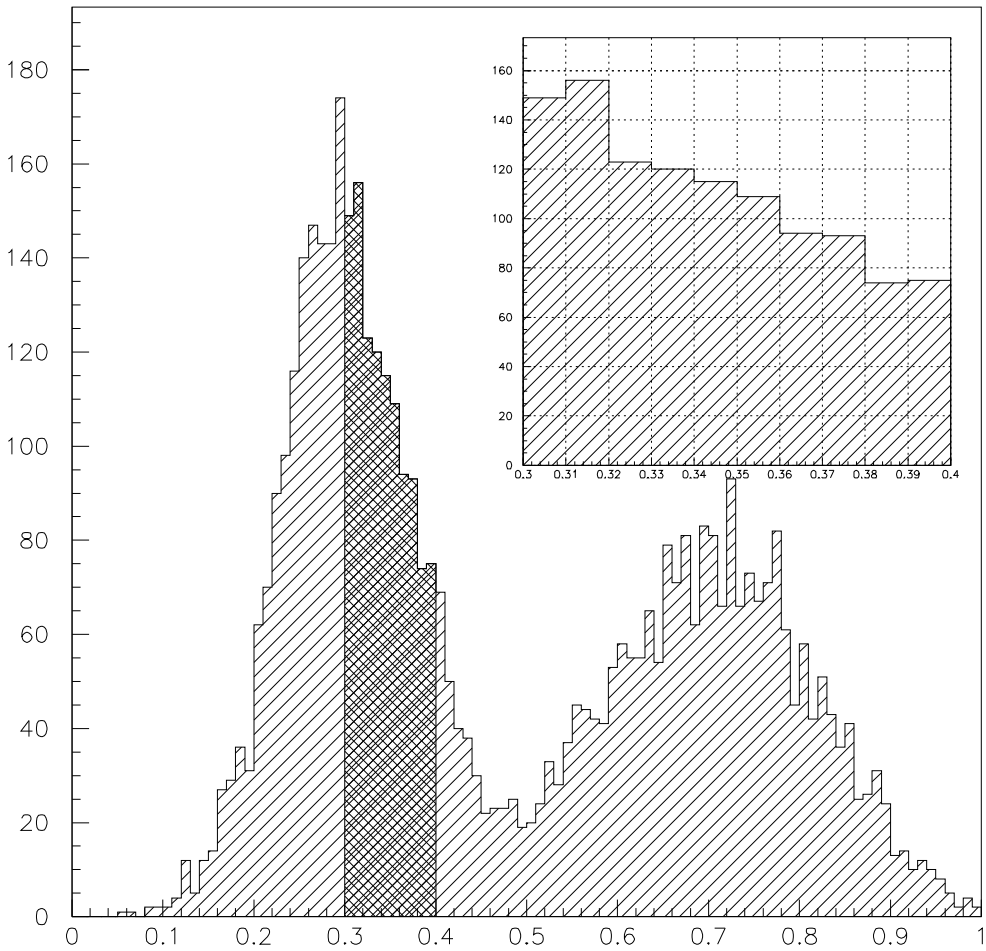
❶ `PI/CREATE` allows to create a new graphic picture in memory. After this call, all the graphic generated

❷ `IZPICT` is the generic function to perform all kind of actions on the `higz` pictures. Here the picture `MERGE2` is set as the current picture.

❸ `PI/MERGE` allows to merge a picture into the current picture.

❹ `PI/DEL` allows to delete a picture from memory. To delete a picture from a file the command `SCRATCH` should be used.

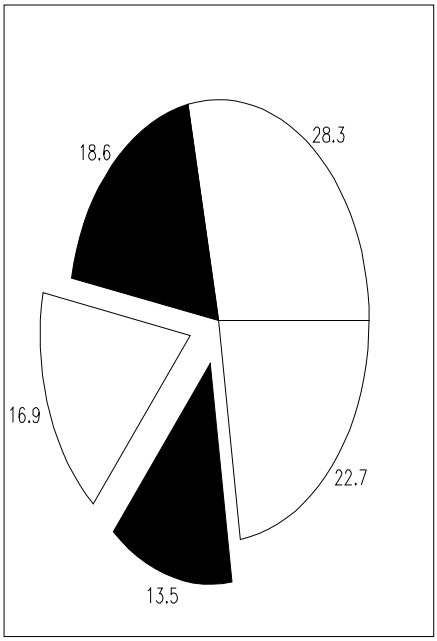
❺ The command `SWITCH` set the graphics switch to control plotting output to terminal (G) and/or picture in memory (Z).

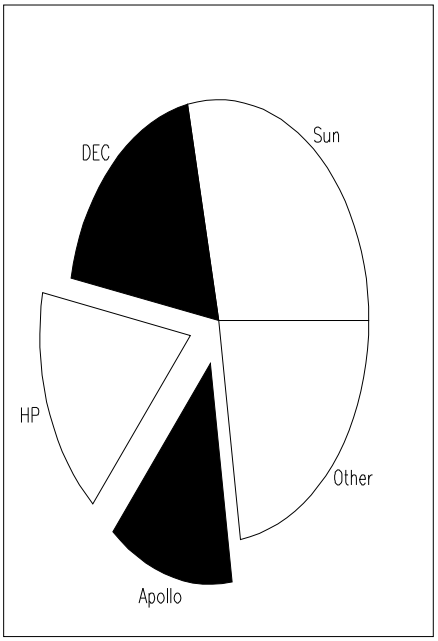

180
160
140
120
100
80
60
40
20
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
160
140
120
100
80
60
40
20
0
0.3
0.31
0.32
0.33
0.34
0.35
0.36
0.37
0.38
0.39
0.4
Test 1−dim Histo

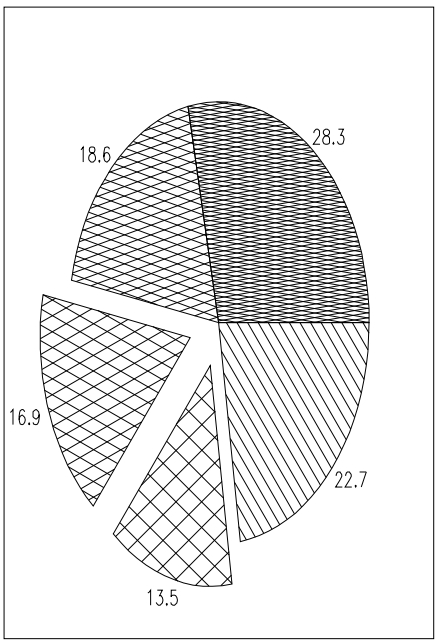
3.11.2 Pie charts

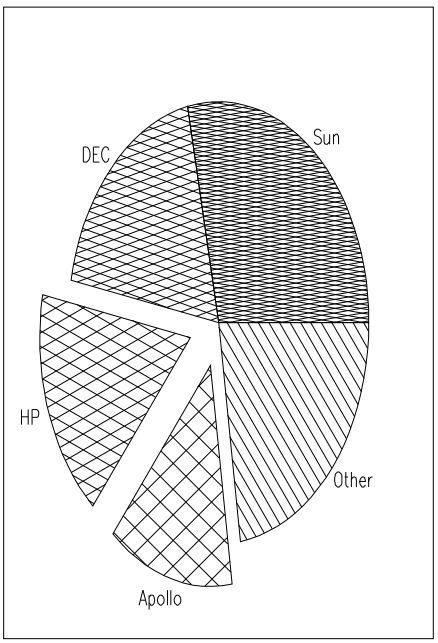
Pie chart and Bar chart

```
alias/cre colbackg  0
alias/cre colcompl  1
alias/cre colred    2
alias/cre colgreen  3
alias/cre colblue   4
alias/cre colyellow 5
alias/cre colpurple 6
alias/cre colcyan   7

v/cre vws(5) R 28.3 18.6 16.9 13.5 22.7
label 1 5 'Sun' 'DEC' 'HP' 'Apollo' 'Other'

v/cre offset(5)  R 2*0. 2*20. 0.
v/cre colour(5)  R colred colgreen colblue colyellow colpurple
v/cre style(5)   R 111 222 333 444 265

igset fais 1 ; igset bord 1
zon 2 2
null 0 20 0 20 a ; pie 10. 10. 7.  5 vws p offset ! colour
null 0 20 0 20 a ; pie 10. 10. 7.  5 vws l offset ! colour
null 0 20 0 20 a ; pie 10. 10. 7.  5 vws n offset style
null 0 20 0 20 a ; pie 10. 10. 7.  5 vws l offset style
```


18.6
28.3
16.9
22.7
13.5


DEC
Sun
HP
Other
Apollo


18.6
28.3
16.9
22.7
13.5


DEC
Sun
HP
Other
Apollo

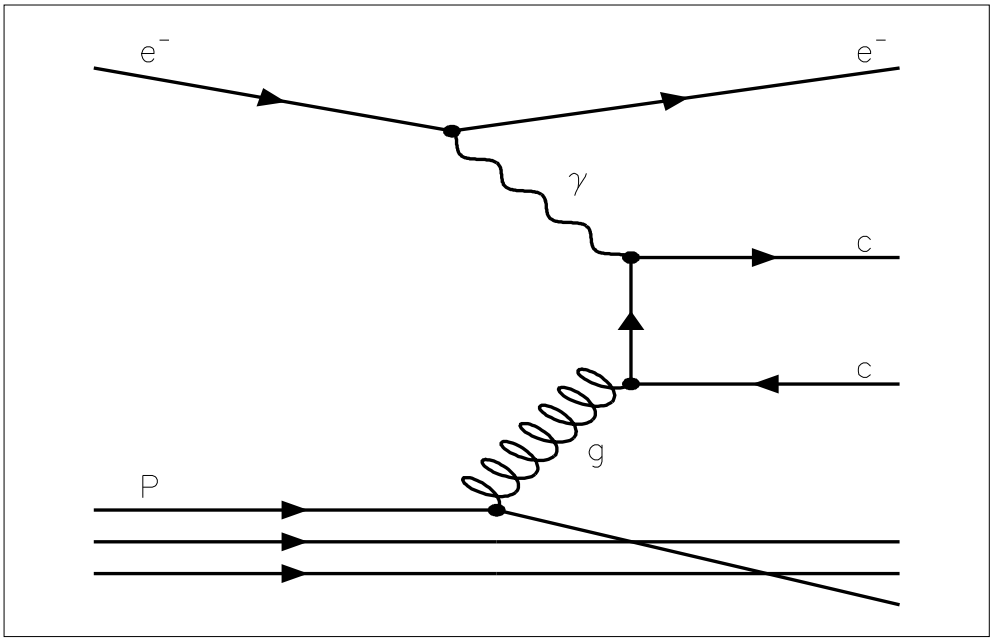
3.11.3 Feynman diagrams

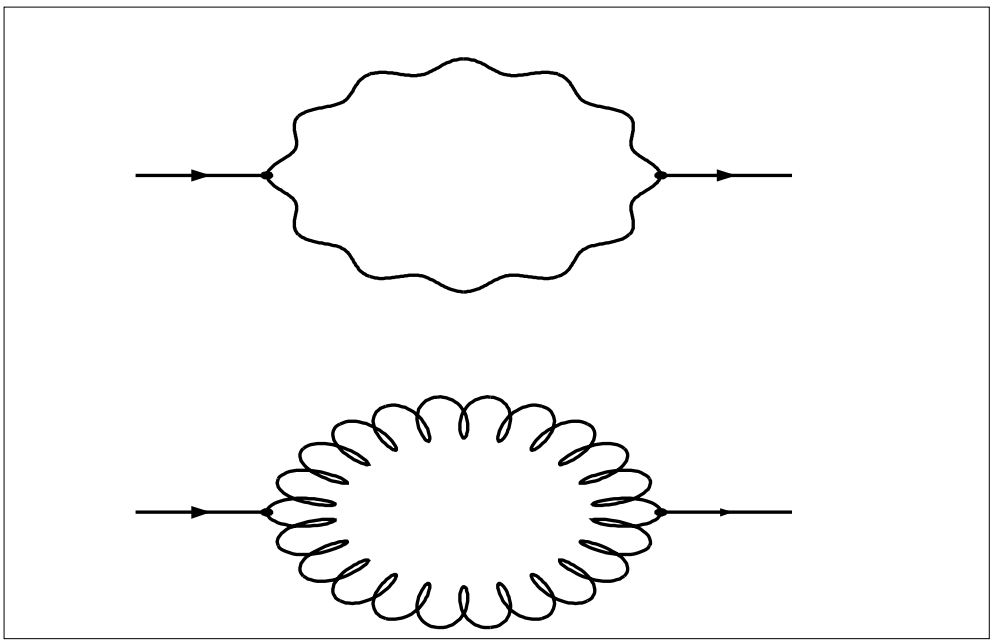
Feynman diagrams drawing

```
      Zone 1 2
❷     Nul 3 14.0 4 14.0 A
      Igset LWID 6 ; Igset FAIS 1
      * c-c system
❶     Arline  13.0  8.0 10.0  8.0 0.3
      Arline  10.0 10.0 13.0 10.0 0.3
      Arline  10.0  8.0 10.0 10.0 0.3
      * Proton
      Arline 4.0 5.0  8.5 5.0 0.3
      Arline 4.0 5.5  8.5 5.5 0.3
      Arline 4.0 6.0  8.5 6.0 0.3
      Line   8.5 6.0 13.0 4.5
      Line   8.5 5.0 13.0 5.0
      Line   8.5 5.5 13.0 5.5
      * Gluon
❶     Helix 10.0 8.0 8.5 6.0 0.3 7 30
      * Lepton
      Arline 4 13  8 12 0.3
      Arline 8 12 13 13 0.3
      * Photon
      Helix 8 12 10 10 0.1 4 0
      * Vertex
❶     Fpoint  8.0 12.0 0.1
      Fpoint 10.0 10.0 0.1
      Fpoint 10.0  8.0 0.1
      Fpoint  8.5  6.0 0.1
      Igset CHHE 0.35
      Itx 12.5 10.1 'c'
      Itx 12.5  8.1 'c'
      Itx 12.5 13.1 'e^-'
      Itx  4.5 13.1 'e^-'
      Itx  4.5  6.2 'P'
      Itx  9.3 11.1 '[g]'
      Itx  9.5  6.8 'g'
❷     Nul 0 15 0 15 A
      Arline    2.0  3.0  4.0  3.0 0.30
❶     Archelix  4.0  3.0 10.0  3.0 0.50 11 30 3.01
      Fpoint    4.0  3.0  0.1
      Fpoint   10.0  3.0  0.1
      Archelix 10.0  3.0  4.0  3.0 0.50 11 30 3.01
      Arline   10.0  3.0 12.0  3.0 0.20
      Arline    2.0 11.0  4.0 11.0 0.30
      Archelix  4.0 11.0 10.0 11.0 0.15  6  0 3.01
      Fpoint    4.0 11.0  0.1
      Fpoint   10.0 11.0  0.1
      Archelix 10.0 11.0  4.0 11.0 0.15  6  0 3.01
      Arline   10.0 11.0 12.0 11.0 0.30
```

❶ `paw` provides a set of commands to draw Feynman driagrams.

❷ `NULL` used with the option `'A'`, allows to define world coordinates without the axis. If in addition the option `'B'` is given, the box around the plot is not drawn.


e⁻
e⁻
γ
c
c
g
P

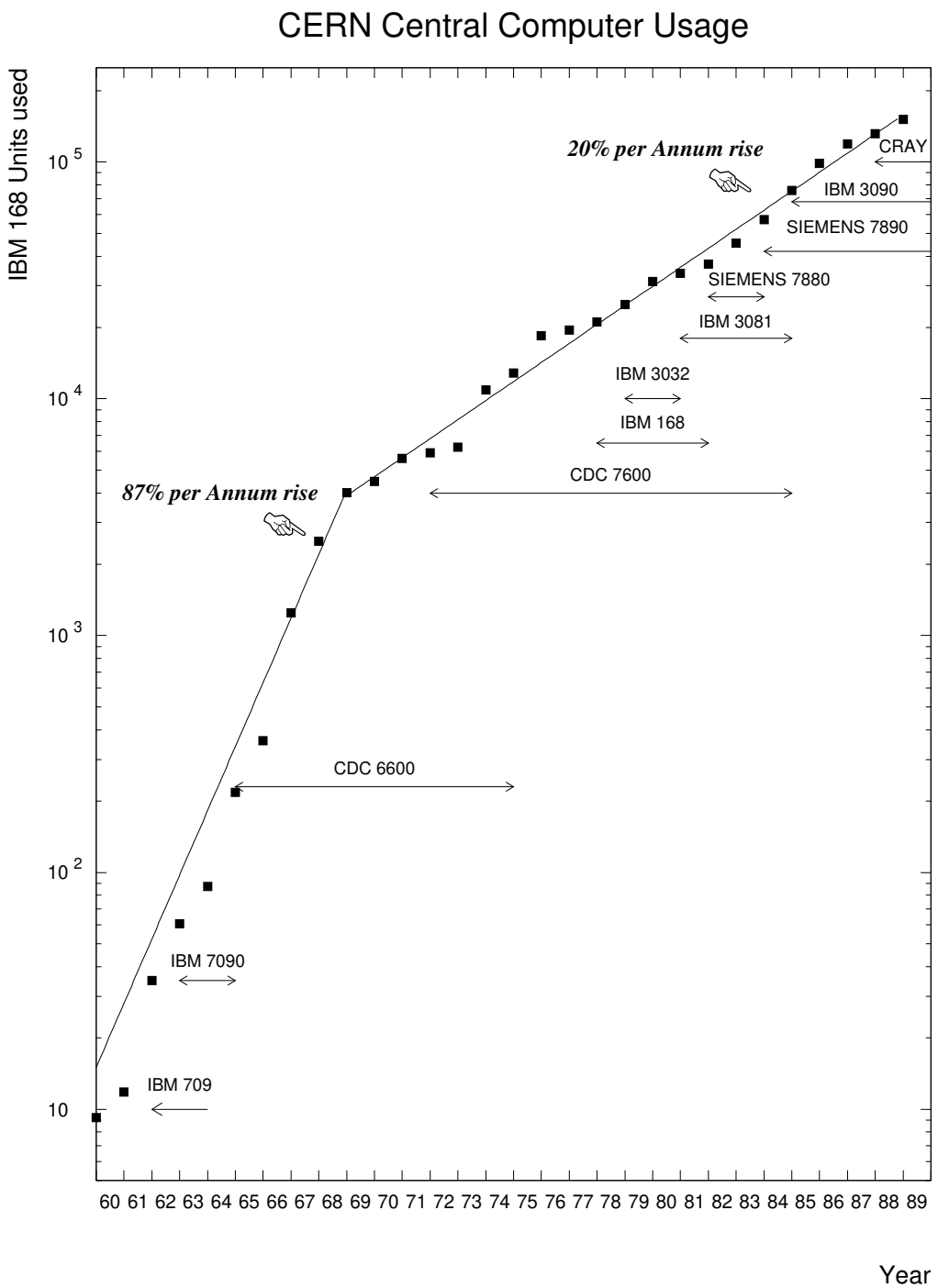


3.11.4 Making a complex graph with PAW

Pie chart and Bar chart

```
OPT NBOX
OPT LOGY
OPT TIC
OPT UTIT
opt ZFL1
size 16 20
set VSIZ 0.20
set YGTI 1.2
set XVAL 0.4
set YVAL 0.2
set XLAB 1.0
set YLAB 1.2
set XTIC 0.15
set YTIC 0.15
set ASIZ 0.26
set GSIZ 0.35
title_gl 'CERN Central Computer Usage'
vector/create vy(30) R 9.2 11.8 34.9 60.7 87.1 217.8 360 1250 2500 4006 _
  4478 5590 5910 6246 10879 12849 18429 19481 21171 25005 _
  31219 33928 37057 45520 57000 75957 98806 118993 131800 151138
sigma vx=array(30,60#89)
ve/cre f1(2) r 2*0.0
ve/cre f2(2) r 2*0.0
SET NDVX -30.05
NULL 60 90 5 250000
igset MSCF 0.75
igset mtyp 21
graph 30 vx vy p
sigma we=sqrt(vy)
ve/fi vx(:10) vy(:10) we e es ! f1
ve/fi vx(10:) vy(10:) we e es ! f2
arrow  64. 62.    10.    10.  0.15
igset txal 20
igset chhe 0.18
itx 63.    12. 'IBM 709'
arrow  65. 63.    35.    35. -0.11
itx 64.    40  'IBM 7090'
arrow 75. 65.    230.   230. -0.11
itx 70.   260. 'CDC 6600'
arrow 85. 72.   4000.  4000. -0.11
itx 78.5  4500. 'CDC 7600'
arrow 82. 78.   6500.  6500. -0.11
itx 80.   7500. 'IBM 168 '
arrow 81. 79.  10000. 10000. -0.11
itx 80.  12000. 'IBM 3032'
arrow 85. 81.  18000. 18000. -0.11
itx 83.  20000. 'IBM 3081'
igset txal 10
arrow 84. 82.  27000. 27000. -0.11
itx 82.  30000. 'SIEMENS 7880'
igset txal 20
arrow 90. 84.  42000. 42000.  0.11
itx 87.  50000. 'SIEMENS 7890'
arrow 90. 85.  68000. 68000.  0.11
```

```
itx 87.5 72000. 'IBM 3090'
arrow 90. 88.  100000. 100000.  0.11
itx 89. 110000. 'CRAY'

arise=$sigma(int((exp(f1(2))-1)*100+0.5))//'% per Annum rise'
xhand=68.
yhand=$sigma(exp(f1(1)+f1(2)*[xhand]))
EXEC DRAW X=[xhand] Y=[yhand] TEXT=[arise]
arise=$sigma(int((exp(f2(2))-1)*100+0.5))//'% per Annum rise'
xhand=84.
yhand=$sigma(exp(f2(1)+f2(2)*[xhand]))
EXEC DRAW X=[xhand] Y=[yhand] TEXT=[arise]
atitle 'Year ' 'IBM 168 Units used '
Return

MACRO DRAW
igset TXAL 30
igset TANG -35.
igset TXFP -140
igset CHHE  0.50
itx $SIGMA([X]-0.9) [Y] +
igset TXAL 30
igset TANG  0.
igset TXFP -30
igset CHHE  0.22
y = [y] * 1.70
itx [X] [Y] [TEXT]
RETURN
```


CERN Central Computer Usage
IBM 168 Units used
10 5
10 4
10 3
10 2
10
20% per Annum rise
87% per Annum rise
CRAY
IBM 3090
SIEMENS 7890
SIEMENS 7880
IBM 3081
IBM 3032
IBM 168
CDC 7600
CDC 6600
IBM 7090
IBM 709
60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89
Year

3.11.5 Making slides

Making slides

```
alias/create mainfont -60
opt zfl1
exec discomp
RETURN

MACRO SLIDE name='Author/CERN  CONF99' sn=' ' title=' '
xsize=18
ysize=22
width=0.4
xmax = [xsize]-[width]
ymax = [ysize]-[width]
size [xsize] [ysize]
next
igset *
igset lwid 2
pave 0 [xmax] 0 [ymax] [width] 0 1005 tr
igset lwid 1
xtitle = $sigma(([xsize]-0.2)/2.)
ytitle = [ysize]-1.5
igset txfp 2
igset txal 20
igset chhe 0.6
itx [xtitle] [ytitle] [title]
igset chhe 0.3
igset txal 10
xtext = [xmax]-0.2
ytext = 0.1
igset chhe 0.2
igset txal 30
itx [xtext] [ytext] [name]
igset txal 10
itx 0.1 0.1 [sn]
igset chhe 0.3
igset lwid 2
return

MACRO DISCOMP
exec slide sn='DisComp' title='Distributed Computing'
igset txfp mainfont
igset chhe 0.5
itx 2 17 'With a distributed operating system (not yet !)'
itx 2 15 'With tools on top (RPCs, NCS,.. ?)'
igset chhe 0.4
itx 3 14 Tmess
itx 3 13 Tfork
itx 3 12 Tdata
itx 3 11 Tcomp
igset txfp -70
itx 5 14 'Time to send message to remote process'
itx 5 13 'Time to fork a process'
itx 5 12 'Time to pass data (in and out)'
itx 5 11 'Time used for computation on remote process'
igset txfp mainfont
pave 2 16 2 9 0.3 0 1001 trs
```

```
igset txal 33
itx 6 7 'Efficiency ='
igset txal 20
line 6.1 7 14.1 7
itx 10 7.2 Tcomp
itx 10 6.3 'Tcomp + Tmess + Tfork + Tdata'
igset txfp -240
igset chhe 0.6
igset txal 30
itx 1.5 17 P
itx 1.5 15 P
igset chhe 0.3
igset txal 20
igset txfp 2
itx 9 4 'Many time consuming applications today have'
itx 9 3 'Efficiency > 0.9'
RETURN
```

Distributed Computing

- **With a distributed operating system (not yet)**
- **With tools on top (RPCs, NCS,..)**

Tmess	***Time to send message to remote process***
Tfork	***Time to fork a process***
Tdata	***Time to pass data (in and out)***
Tcomp	***Time used for computation on remote process***

$$\textbf{Efficiency} = \frac{\textbf{Tcomp}}{\textbf{Tcomp + Tmess + Tfork + Tdata}}$$

Many time consuming applications today have

Efficiency 0.9

DisComp Author/CERN CONF99

3.11.6 How to use PostScript files

This macro can be used to print the tutorial examples

```
MACRO PRINTEX 1=1
FOR/FILE 44 pawex[1].ps
METAFILE 44 -111
EXEC PAWEX[1]
CLOSE 44
SHELL local print command pawex[1].ps
```

The PostScript workstation types have the following format:

`-[Format][Nx][Ny][Type]`

Where:

Format is an integer between 0 and 99 which defines the format of the paper. For example if `Format=3` the paper is in the standard A3 format. `Format=4` and `Format=0` are the same and define an A4 page. The A0 format is selected by `Format=99`.

Nx, Ny specify respectively the number of zones on the x and y axis. `Nx` and `Ny` are integers between 1 and 9.

Type can be equal to:

1 Portrait mode with a small margin at the bottom of the page.

2 Landscape mode with a small margin at the bottom of the page.

4 Portrait mode with a large margin at the bottom of the page.

5 Landscape mode with a large margin at the bottom of the page. The large margin is useful for some PostScript printers (very often for the colour printers) they need more space to grip the paper for mechanical reasons. Note that some PostScript colour printers can also use the so called "special A4" format permitting the full usage of the A4 area; in this case larger margins are not necessary and `Type=1` or 2 can be used.

3 Encapsulated PostScript. This `Type` permits the generation of files which can be included in other documents, for example in LaTeX files. Note that with this `Type`, `Nx` and `Ny` must always be equal to 1, and `Format` has no meaning. The size of the picture must be specified by the user via the command `SIZE`. Therefore the workstation type for Encapsulated PostScript is -113. For example if the name of an Encapsulated PostScript file is `example.eps`, the inclusion of this file into a LaTeX file will be possible via (in the LaTeX file):

```
\begin{figure}
\epsffile{example.eps}
\caption{Example of Encapsulated PostScript in LaTeX}
\label{EXAMPLE}
\end{figure}
```

How to print all the tutorial examples on one page

```
MACRO PRINTALL
FOR/FILE 44 all.ps
METAFILE 44 -471
DO I=1,26
   EXEC PAWEX[I]
ENDDO
CLOSE 44
SHELL  local print command all.ps
```

Note also: The command `PICTURE/PRINT` allows to print the current picture in memory onto a PostScript file, and if required send it to the default PostScript printer.

Part II

PAW - Commands and Concepts

Chapter 4: User interface - KUIP

4.1 Command line syntax

The general syntax of a *command line* is a *command path* optionally followed by an *argument list*. The command path and the arguments have to be separated from each other by one or more space characters. Therefore arguments containing spaces or other special characters have to be quoted.

In the following we want to use an appropriate formalism to describe the syntax rules. The notation will be introduced step by step as needed. The verbal explanation given above can be written as:

command-line ::= *command-path* { *argument* }

The *slanted* symbols are non-terminal, i.e. they are composed of other terminal or non-terminal symbols. The definition of a non-terminal symbol is denoted by "**::=**". Symbols enclosed in braces ("{...}") are optional and they can appear zero or more times.

4.1.1 Command structure

The set of commands is structured as an (inverted) tree (see figure 4.1), comparable to a Unix file system. The command set can be dynamically extended by linking new commands or menus into the tree.

Compared to a flat list structure the tree allows a cleaner representation through menus, especially when the command set is large. `paw` has more than 200 commands. It would be hard to visualize such a number of command in a single graphics menu.

Abbreviations

A command path consists of a menu path and a command name. The menu path itself consists of a list of menu names up to an arbitrarily deep level of sub-menus.

command-path ::= [*menu-path*/]*command-name*

menu-path ::= [/]*menu-name*{/*menu-name*}

Here we introduced two more notations. Symbols in teletype mode ("/") are literals, i.e. the menu and command names have to be separated by a slash character. Symbols enclosed in brackets ("[...]") are optional which can appear zero or one times.

These syntax rules already show that a command path may be abbreviated by omitting part of the leading menu path. For example, if the complete command path is

```
/MENU/SUBMENU/COMMAND
```

valid abbreviations are

```
MENU/SUBMENU/COMMAND
SUBMENU/COMMAND
COMMAND
```

but **not** "`MENU/COMMAND`" or "`/SUBMENU/COMMAND`". Note that the command name matching is case-insensitive, i.e. the following are all valid possibilities:

```
COMMAND
command
Command
```

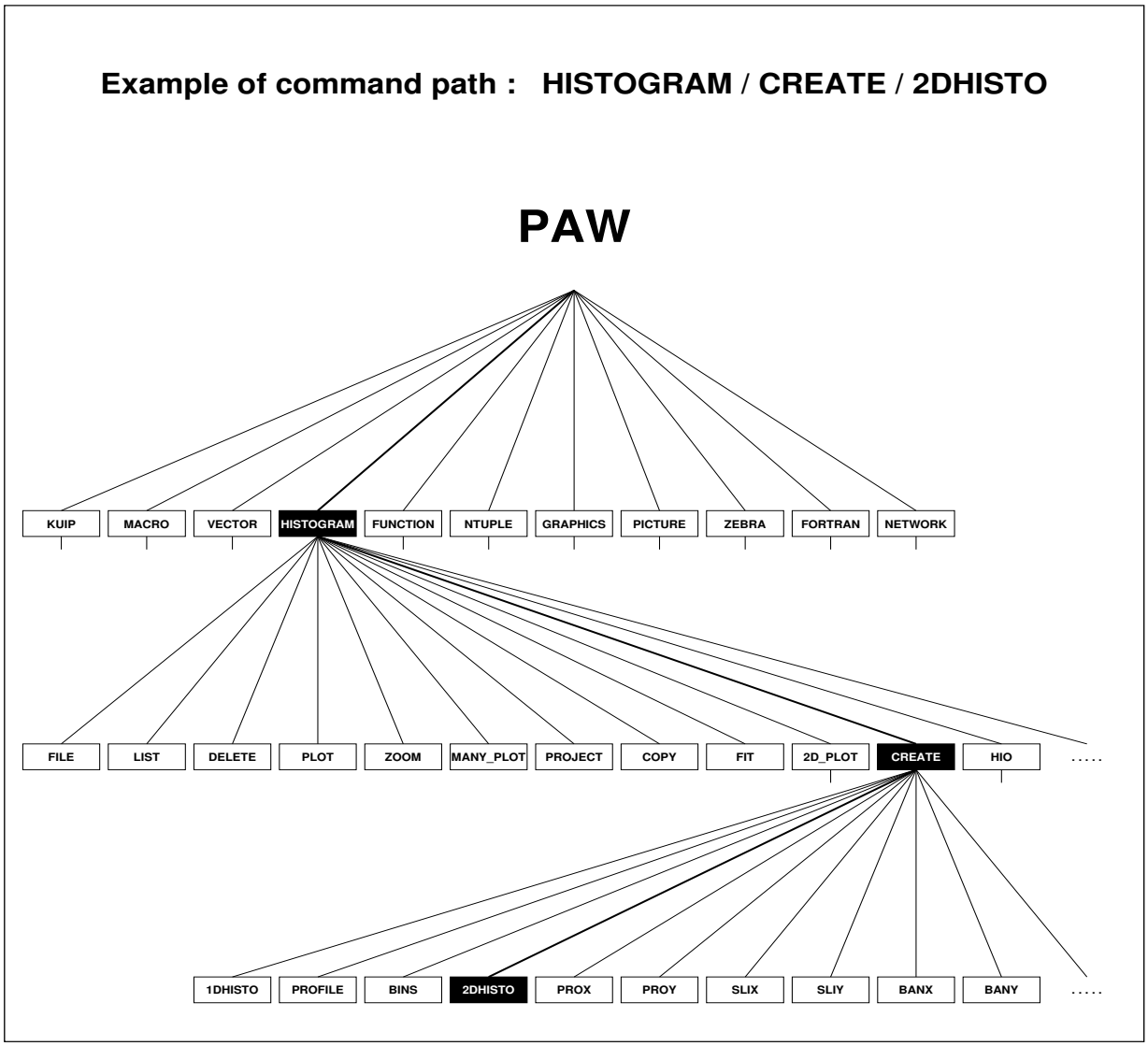


Figure 4.1: Example of the PAW command tree structure

Furthermore, menu and command names may be abbreviated by omitting trailing parts, i.e.

```
SUB/COMMAND
COMMA
/M/S/C
```

are also valid abbreviations.

The shortest unambiguous abbreviation for any command is not fixed but depends on the whole command set. `kuip` lists all possible ambiguities if a given abbreviation has no unique match:

```
PAW > LIST
 *** Ambiguous command list. Possible commands are :

 /KUIP/ALIAS/LIST
```

```
/MACRO/LIST
/VECTOR/LIST
/HISTOGRAM/LIST
/NTUPLE/LIST
/PICTURE/LIST
```

Ambiguity resolution

Abbreviations can lead to ambiguities if the abbreviation matches more than one command path. For example, in an application with the commands

```
/MENU/COMPUTE
/MENU/SUBMENU/COMMAND
/MENU/OTHERMENU/COMMA
```

typing "COM" matches all three commands and "COMM" still matches the last two.

The list of all executable commands can be obtained by just typing "/". The single slash matches every command element and therefore all available commands will be listed as possible ambiguities.

Since users tend to use abbreviations heavily also in command scripts adding a new command always risks to break these scripts by introducing a sudden ambiguity. In order to alleviate this problem a set of resolution rules apply before an abbreviation is finally considered ambiguous.

The first rule is that an exact match for the command name takes preference, i.e. "COMMA" resolves to the third command only. The second rule prefers the lowest number of menu levels. For example, "COM" resolves to the first command because the other two matches are one more menu level down.

More on command name resolution

kuip provides additonal commands which can affect the way the command name, i.e. the first token in a command line, is interpreted.

Changing the root menu The command SET/ROOT defines the menu from which the search for command name starts. It is not quite comparable to the Unix cd or VMS SET DEFAULT command. If no matching command is found going downwards from the SET/ROOT menu a second attempt is made starting off at the top menu "/".

Disabling commands The command SET/VISIBILITY allows to disable/enable individual commands. Disabled commands cannot be executed and they do not contribute to name ambiguities. However, the HELP information is still available. In STYLE G disabled commands are shown with a grey or hatched background.

Note that the VISIBILITY command can disable itself which makes it impossible to re-enable any command.

Automatic macro execution The command MACRO/DEFAULT implements two facilities. First it allows to define a directory search path used by the EXEC command for locating .kumac macro files. Second it controls the implicit interpretation of the command name token as a possible macro filename:

-Command This is the default setting which does not try to interpreted cmd as macro name.

`-Auto` If the search path contains a file `cmd.kumac` it is executed, i.e. the actual command becomes "`EXEC cmd`", otherwise the search for a command named `cmd` starts.

`-AutoReverse` If `cmd` is either not a command name or ambiguous and a file `cmd.kumac` exists the command is transformed into "`EXEC cmd`".

Command template The command `SET/COMMAND` allows to define a template which is used whenever the command token does not match any command name. The template can contain "`$1`", ..., "`$9`" which are substituted with the *n*'th token from the original command line, or "`$*`" which is replaced by the complete line. For example, a `kuip` application can be turned into a calculator by

```
PAW > SET/COMMAND 'mess $sigma($*)'
PAW > 17+2*5
 27
```

"`SET/COMMAND 'EXEC $*'`" has almost the same effect as "`DEFAULT -AutoReverse`" but these are two distinct facilities which can be active simultaneously. The difference is that for `SET/COMMAND` the token in the command name position must not match any command. If does not apply if the token is an ambiguous command name.

Both `Auto/AutoReverse` and `SET/COMMAND` logic are ignored during the execution of macro scripts.

4.1.2 Arguments

Most commands have *parameters* for which the user is expected to supply *argument values*. Parameters are either *mandatory* or *optional*. Mandatory arguments which are not specified on the command line are prompted for. If optional arguments are omitted a default value is used instead.

Mandatory parameters always precede the optional parameters. The command `USAGE` allows to see the number of parameters for a command:

```
PAW > usage manual

 * KUIP/MANUAL ITEM [ OUTPUT OPTION ]
```

The optional parameters are enclosed in square brackets. The default values can be seen from the help text for a command. The `STYLE` command shown in figure 4.2 has only optional arguments. The corresponding default values are indicated in the help information as "`D=`*value*". There is also the case of optional parameters without fixed default values. For these commands the application writer has to provide an appropriate default at execution time.

Mandatory parameters may also have a default value which is used if the prompt is acknowledged by simple hitting the `RETURN`-key. Otherwise the proposed default is the value used in the previous command execution.

The `STYLE` command also shows that there are three different kind of parameters: character values indicated by "`C`" after the parameter name, real values ("`R`") and integer values ("`I`").

Whether character values are case-sensitive is up to the application. The application writer has three choices to retrieve a character argument:

`KUGETC` returns the string converted to uppercase.

`KUGETS` returns the string as it was typed in.

```
PAW > HELP STYLE

 * KUIP/SET_SHOW/STYLE [ OPTION SGYLEN SGSIZE SGYSPA SGBORD WKTYPE ]

   OPTION     C 'Option' D='?'
   SGYLEN     R 'max Y LENgth of each menu item box' D=0.025 R=0.005:0.25
   SGSIZE     R 'space available for the application' D=0.8 R=0:0.90
   SGYSPA     R 'max Y length of space between menus' D=0.02 R=-0.5:0.50
   SGBORD     R 'X or Y border for menus' D=0.015 R=0:0.25
   WKTYPE     I 'Graphics workstation type' D=0

   Possible OPTION values are:

    ?   show current style
    C   Command line : select Command line input
    AN  Menu with Numbers : select general Alpha menu (with Numbers)
    AL  Menu with Letters : select general Alpha menu (with Letters)
```

Figure 4.2: Parameter types, default values, and range limits

KUGETF returns on operating systems with case-sensitive filenames (Unix) the string depending on the current setting of the FILECASE command. The string is either left as it is, or it is converted to lowercase. If filenames are not case-sensitive the argument value is converted to whatever case is required by the operating system.

Numeric (real or integer) parameters may be restricted in the range of acceptable values. In the help text this is indicated as "R=*lower*:*upper*. If the argument value is outside the range kuip prompts the user to enter an acceptable value before the command can be executed. The lower or upper range value may be missing to indicate an unlimited range in one direction. Instead of a simple numeric value the argument may also be an expression.

For both numeric and character parameters the range may also be given as a comma-separated list of values. kuip will accept an argument only if it matches one of the values in the list.

In general the arguments given on the command line are assigned to the command parameters from left to right but there are also ways to change the order. In our syntax notation, using "|" to indicate possible alternatives, we can write:

argument ::= *value* | ! | !! | *name*=*value* | -*value*

An argument given as a simple value is assigned to the next parameter expected. The special values "!" and "!!" are templates for the default value and the value from the previous command execution, respectively.

Named arguments

The form "*name*=*value*" allows to invert the argument order or to skip a list of optional parameters for which the default values should be used. For example,

```
STYLE G SGBORD=0.1
```

is equivalent to

```
STYLE G ! ! ! 0.1
```

kuip strips off the "*name=*" part before passing the argument values to the application. In fact the application program cannot distinguish which of these possible forms the user actually typed. A simple argument following a named argument is assigned to the parameter following the named parameter, i.e.

```
STYLE G SGBORD=0.1 1
```

is equivalent to

```
STYLE G ! ! ! SGBORD=0.1 WKTYPE=1
```

Parameter names are case-insensitive but in general they may not be abbreviated. However, the application write can allow abbreviations up to a certain minimum length. In the help text this is indicated by a "*" inside the parameter name. For example, if the parameter name is shown as

```
LIB*RARY
```

the acceptable abbreviations are "LIB=", "LIBR=", "LIBRA=", "LIBRAR=", and "LIBRARY=".

kuip does not insist that an argument of the form "*name=value*" matches one of the parameter names. The argument including the "*name=*" part is simply assigned to the next parameter expected.

Option arguments

The last alternative "*-value*" to specify an argument applies only to *option* parameters. (Note the distinction between *option* and *optional*. Option parameters are usually but not necessarily optional.) In the help text option parameters are tagged by the list of possible values (figure 4.3). Frequently these parameters are named "OPTION" or "CHOPT".

```
PAW > HELP MANUAL

 * KUIP/MANUAL ITEM [ OUTPUT OPTION ]

   ITEM       C 'Command or menu path'
   OUTPUT     C 'Output file name' D=' '
   OPTION     C 'Text formatting system' D=' '

   Possible OPTION values are:

   ' '     plain text : plain text format
    LATEX  LaTeX format (encapsulated)
    TEX    LaTeX format (without header)
```

Figure 4.3: Example for option parameters

The "*-value*" form allows to specify option arguments out of order, emulating the Unix style of options preceeded other command arguments. For example,

```
MANUAL -LATEX /KUIP
```

is equivalent to

```
MANUAL /KUIP OPTION=LATEX
```

Note that this is **not** equivalent to "`MANUAL OPTION=LATEX /KUIP`". Unlike to the "`-` *value*" form subsequent simple arguments are still assigned to the next parameter expected, not to the one following the option parameter itself.

Since a leading "`-`" can be part of a valid (non-option) argument the value is checked against a set of rules before it is actually interpreted as an option assignment.

The option argument can be a concatenation of several of the allowed option values. `kuip` checks that the argument string is exclusivly constructed from valid option values. This check is done by removing matches of option values from the argument string, starting with the longest option values first. For example, with the definition

```
Possible OPTION values are:    AB
                               ABC
                               CD
```

the argument "`-ABCD`" is not interpreted as option assignment because after removing the longest match "`ABC`" the remainder "D" is not anymore a valid option value. (This case would have to be written as "`-CDAB`". `kuip` does not check whether the combination of values is valid. It is left to the application to refuse execution, e.g. if some of the given option values are mutually exclusive.)

Even with this consistency check there is still a problem arising for commands using digits as option values. One example is the command `SMOOTH` (figure 4.4). The command line

```
SMOOTH -1 2
```

could be interpreted as

```
SMOOTH ID=2 OPTION=-1
```

Since histogram identifiers can have the form of a negative number the desired interpretation is the natural order

```
SMOOTH ID=-1 OPTION=2
```

The application writer has to inform `kuip` about this by giving the `ID` parameter the "`Minus`" attribute. For numeric parameters the "`Minus`" attribute is implicit. However, the argument is taken as an option assignment if the parameter has a limited range which does not include the corresponding negative value. For example,

```
SMOOTH 10 SENSIT=2 -1
```

is interpreted as

```
SMOOTH ID=-1 OPTION=1 SENSIT=2
```

since "`-1`" is outside the range for the `SMOOTH` parameter.

The "`-`" in an option assignment is usually stripped off before the value is passed to the application program. The exception is if the minus sign itself is one of the valid option values and the next argument expected belongs to the option parameter itself. Consider the command `HISTO/PLOT` (figure 4.5). The command line

```
 * HISTOGRAM/OPERATIONS/SMOOTH ID [ OPTION SENSIT SMOOTH ]

   ID         C 'Histogram or Ntuple Identifier' Minus
   OPTION     C 'Options' D='2M'
   SENSIT     R 'Sensitivity parameter' D=1. R=0.3:3.
   SMOOTH     R 'Smoothness parameter' D=1. R=0.3:3.

   Possible OPTION values are:

    0  Replace original histogram by smoothed.
    1  Replace original histogram by smoothed.
    2  Store values of smoothed function and its parameters without replacing
       the original histogram (but see note below) - the smoothed function can
       be displayed at editing time - see HISTOGRAM/PLOT.
    M  Invoke multiquadric smoothing.
```

Figure 4.4: HELP SMOOTH

```
 * HISTOGRAM/PLOT [ ID CHOPT ]

   ID         C 'Histogram Identifier' Loop Minus
   CHOPT      C 'Options' D=' ' Minus

   Possible CHOPT values are:

   ' '     Draw the histogram.
    C      Draw a smooth curve.
    S      Superimpose plot on top of existing picture.
    +      Add contents of ID to last plotted histogram.
    -      Substract contents of ID to last plotted histogram.
```

Figure 4.5: HELP HISTO/PLOT

```
H/PLOT -S 1
```

is interpreted as

```
HISTO/PLOT ID=1 CHOPT=S
```

while

```
H/PLOT 1 -S
```

is equivalent to

```
HISTO/PLOT ID=1 CHOPT=-S
```

Argument values

Since in command line blanks are used to separate the command name and the individual arguments string values containing blanks have to be quoted. The rules are the same as used by Fortran: the quote character is the apostroph "'", and apostroph inside a quoted string have to be duplicated:

```
MESS 'Hello world'
MESS 'Do or don''t'
```

The enclosing quote characters are stripped off before the argument value is passed to the application, even if they are redundant, i.e. the two forms

```
MESS 'Hello'
MESS Hello
```

are equivalent. Note that the `MESSAGE` command has only a single parameter:

```
 * KUIP/MESSAGE [ STRING ]

   STRING     C 'Message string' D=' '
...
```

Nevertheless, in most cases quoting the message string is not necessary. If the command line contains more arguments than there are parameters the additional values are concatenated to the argument for the last parameter. In the concatenation each value is separated by a (single) blank character, i.e. the commands

```
MESS 'Hello World'
MESS  Hello World
MESS  Hello       World
```

yield all the same output. Therefore the message text only needs quoting if the words should be separated by more than one space character.

Quoting inhibits the interpretation of the enclosed string as special argument values. Printing an exclamation mark as message text has to written as

```
MESS '!'
```

because "`MESS !`" would mean to take the default value for the parameter `STRING` and yield an empty line only.

Another instance is if an argument of the form "*name*`=`*value*" should be taken literally. For example, the command line

```
EXEC mac foo=bar
```

initializes the macro variable "`foo`" to the value "`bar`". However, if the intention is to pass the string "`foo=bar`" as argument to the macro quotes must be used:

```
EXEC mac 'foo=bar'
```

In addition, some commands, e.g.

```
 * NTUPLE/PLOT IDN [ UWFUNC NEVENT IFIRST NUPD OPTION IDH ]
```

use the form "*name*=*value*" for equality tests in the cut expression `UWFUNC`. For example, the command

```
NT/PLOT 10.energy year=1993
```

selects all event for which the Ntuple column `YEAR` has the value `1993`. Any name clash between the Ntuple column and one of the command parameters requires quoting. If the column was called `NUPD` instead of `YEAR` the command would have to be written as

```
NT/PLOT 10.energy 'nupd=1993'
```

or alternatively as "`NT/PLOT 10.energy UWFUNC=nupd=1993`".

Finally, quoted strings are also exempted from any substitutions of aliases, `kuip` system functions, and macro variables. For example,

```
MESS 'foo'
```

always prints "`foo`" while

```
MESS foo
```

can result in "`bar`" if preceded by the command "`ALIAS/CREATE foo bar`". Since square brackets denote macro variable substitution and system functions names start with a dollar-sign it is especially recommended to quote VMS file specifications.

The operator "`//`" allows to concatenate several parts to a single argument value. Unquoted strings on either side of the concatenation operator are implicitly treated as literals unless they are subject to a substitution, i.e. the command lines

```
MESS 'abc'//'def'
MESS 'abc'//def
MESS abc//'def'
MESS abc//def
MESS abcdef
MESS 'a'//'b'//'c'//'d'//'e'//'f'
```

are all equivalent (provided that `abc` and `def` are not defined as aliases). The character sequence "`//`" at the beginning or end of an argument is taken literally, e.g. in

```
CD //LUN2//1
```

the command receives the value "`//LUN21`".

4.1.3 More on command lines

The command line syntax allows to write several commands in one line and also to extend commands with long argument lists over several lines.

Multiple commands on a single line

An input line presented to the `kuip` command processor may contain several commands separated by "`;`". The commands are executed sequentially as if they were on separate lines:

```
MESS Hello world!; MESS How are you?
```

is equivalent to

```
MESS Hello world!
MESS How are you?
```

Note that the text following the semicolon will not be used to satisfy any prompts emitted by the preceeding command, e.g. "`usage; manual`" will not behave as "`usage manual`".
The semicolon is **not** interpreted as line separator if it is immediately followed by a digit or one of the characters

```
+ - * ? [
```

For example, issuing a VMS command with a file version number such as

```
SHELL delete *.tmp;*
```

does not require quoting. Note that this exception rule applies independently of the operating system. In order to avoid surprises we recommend to put always at least one blank after a semicolon intended to be a line separator.
Each command execution returns a status code which is zero for success and non-zero for failure. The sequences "`;&`" and "`;!`" allow to execute the remaining part of an input line depending on the status code of the preceeding command. With

```
cmd1 ;& cmd2 ; cmd3
```

the commands `cmd2` and `cmd3` are only executed if `cmd1` succeeded while with

```
cmd1 ;! cmd2 ; cmd3
```

the remaining commands are only executed if the first one failed. Note that the two characters must follow each other immediately without intervening blank.
In some commands, for example `HISTO/PLOT`, one of the parameters is marked in the help text with the attribute "`Loop`". If the corresponding argument is a comma-separated list of values `kuip` implicitly repeats the command for each value in the list individually:

```
HISTO/PLOT 10,20,30
```

is equivalent to

```
HISTO/PLOT 10
HISTO/PLOT 20
HISTO/PLOT 30
```

Note that "`,`" inside parentheses is not taken as value separator, i.e.

```
HISTO/PLOT 10(1:25,1:25)
```

executes a single command.

Single commands on multiple lines

For commands with very long argument lists it can become necessary to continue it on the next line. An input line ending with an "_" character is joined with the following line.

In the concatenation the underscore itself and all but one of the leading blanks from the next line are removed. Blanks preceding the underscore are left intact. For example,

```
ME_
SS _
'Hello_
       world'
```

is an extravagant way of writing

```
MESS 'Hello world'
```

Note that the interpretation of "_" as line continuation cannot be escaped. If the command line should really end with an underscore the last argument must be quoted.

Recalling previous commands

The command lines types during a session are written into a history file. By default the file is called `last.kumac` and is updated every 25 commands. The commands `LAST` and `RECORDING` allow one to change the file name and the frequency. At the start of a new session before creating a new `last.kumac` the existing file is renamed into `last.kumacold` (except on VMS). Comment lines indicate the date and time at which the sessions were started and stopped.

In this way the user can keep track of all commands entered in the previous and in the current session. The command "`LAST -99`" flushes the buffered lines into `last.kumac` and envokes the editor on the file. The user can then extract the interactively typed commands and copy them into another `.kumac` file from which they can be re-executed.

The command "`LAST -`*n*" prints the last *n* commands entered. On a workstation this allows to re-execute command sequences by doing cut-and-paste operations with the mouse.

`kuip` provides a mechanism similiar to the one found in the Unix `csh` shell for re-executing commands:

`!-`*n*	executes the *n*'th last command once more.
`!!`	is an short-cut for "`!-1`" re-executing the last command.
`!`*n*	re-executes the *n*'th command entered since the beginning of the session.
`!`	prints the commands together with their numbers. The number of lines printed depend on the recording frequency.
`!`*foo*	re-executed the latest command line starting with the string "*foo*".

The command line numbering can also be seen if the prompt string contains "`[]`":

```
PAW > PROMPT 'Paw[] '
Paw[2]
```

^A/^E	Move cursor to beginning/end of the line.
^F/^B	Move cursor forward/backward one character.
^D	Delete the character under the cursor.
^H, DEL	Delete the character to the left of the cursor.
^K	Kill from the cursor to the end of line.
^L	Redraw current line.
^O	Toggle overwrite/insert mode. Text added in overwrite mode (including yanks) overwrites existing text, while insert mode does not overwrite.
^P/^N	Move to previous/next item on history list.
^R/^S	Perform incremental reverse/forward search for string on the history list. Typing normal characters adds to the current search string and searches for a match. Typing ^R/^S marks the start of a new search, and moves on to the next match. Typing ^H or DEL deletes the last character from the search string, and searches from the starting location of the last search. Therefore, repeated DEL's appear to unwind to the match nearest the point at which the last ^R or ^S was typed. If DEL is repeated until the search string is empty the search location begins from the start of the history list. Typing ESC or any other editing character accepts the current match and loads it into the buffer, terminating the search.
^T	Toggle the characters under and to the left of the cursor.
^U	Kill from the prompt to the end of line.
^Y	Yank previously killed text back at current location. Note that this will overwrite or insert, depending on the current mode.
TAB	By default adds spaces to buffer to get to next TAB stop (just after every 8th column).
LF, CR	Returns current buffer to the program.

Table 4.1: Key-binding for recall style KSH

BS/^E	Move cursor to beginning/end of the line.
^F/^D	Move cursor forward/backward one character.
DEL	Delete the character to the left of the cursor.
^A	Toggle overwrite/insert mode.
^B	Move to previous item on history list.
^U	Delete from the beginning of the line to the cursor.
TAB	Move to next TAB stop.
LF, CR	Returns current buffer to the program.

Table 4.2: Key-binding for recall style DCL

On Unix and VMS kuip also provides recalling and editing of command lines for re-executing. The command RECALL allows to choose between different key-bindings:

- Recall style KSH has an Emacs-like binding (table 4.1) similar to the one used by the ksh and bash shells. If the terminal returns ANSI escape sequences the arrow keys can be used instead of ^B/^F/^N/^P.
- Recall style DCL implements the key-binding of VMS line editing (table 4.2).
- The style names KSHO and DCLO allow to switch to overstrike mode instead of the default insert mode.
- Recall style NONE directs kuip to do plain reading from the terminal input.

Although the default setting depends on the operating system both styles can be used on Unix and VMS. Style NONE is recommendable on systems which do swapping instead of paging. For example, on a Cray-X/MP kuip line-editing requires that the application program itself has to react to each individual keystroke.

On Apollo/DomainOS kuip starts up in style NONE, if the program runs in a Display Manager pad, and in style KSH otherwise. However, if crp is used from within a DM pad to run the program on a remote node the automatic identification fails and style NONE must be selected manually.

4.2 Aliases

kuip aliases allow the user to define abbreviations for parts of a command line. There are two types of aliases, *command aliases* and *argument aliases*, which differ in the way they are recognized in a command line. Both alias types can be defined by the ALIAS/CREATE command:

```
* KUIP/ALIAS/CREATE NAME VALUE [ CHOPT ]

  NAME       C 'Alias name'
  VALUE      C 'Alias value'
  CHOPT      C 'Option' D='A'

  Possible CHOPT values are:

   A  create an Argument alias
   C  create a Command alias
   N  No alias expansion of value
```

The alias value may be any string but the alias name can only consist letters, digits, "_", "-", "@", and "$" characters. Command and argument aliases share the same name space. If a command alias with the same name as an existing argument alias is created, the argument alias is deleted first, and vice versa.

4.2.1 Argument aliases

If an argument alias name is recognized anywhere in the command line it is substituted by its value. The name matching is case-insensitive and the substitution is literally, i.e. without case folding or insertion of additional blanks. The replacement is scanned for further occurrences of alias names which in turn will be replaced as well.

The alias name must be separated from the rest of the command line either by a blank or by one of the special characters

```
/  ,  =  :  ;  .  %  '  (  )
```

(not necessarily the same character on both sides). For example, if `foo` and `bar` are alias names, `foot` and `Bar-B-Q` are not affected. If two alias replacements need to be concatenated the "`//`" operator can be used, i.e.

```
ALIAS/CREATE DIR disk$user:[paw]
ALIAS/CREATE FIL file.dat
HISTO/FILE 1 DIR//FIL
```

translates into "`HISTO/FILE 1 disk$user:[paw]file.dat`". Since argument aliases are also recognized in the command position with the definition abbreviations like `HISTO/FIL` cannot be used anymore.

Alias substitution does not take place inside quoted strings. The `ALIAS` commands themselves are treated as a special case. In the command line parsing they are specifically exempted from alias translation in order to allow aliases can be deleted and redefined without quoting. For example,

```
PAW > ALIAS/DELETE *
PAW > ALIAS/CREATE foo bar
PAW > ALIAS/CREATE bar BQ
PAW > ALIAS/CREATE foo tball
PAW > ALIAS/LIST
 Argument aliases:
 BAR        => BQ
 FOO        => tball
 No Command aliases defined.
```

redefines `FOO` rather than creating a new alias name `BQ`. The value part, however, is subject to alias translations. If the aliases are created in reverse order

```
PAW > ALIAS/DELETE *
PAW > ALIAS/CREATE bar BQ
PAW > ALIAS/CREATE foo bar
PAW > ALIAS/LIST
 Argument aliases:
 BAR        => BQ
 FOO        => BQ
 No Command aliases defined.
```

the second alias is created as "`ALIAS/CREATE foo BQ`". In this case quoting the alias value does not avoid the translation. Writing instead

```
ALIAS/CREATE foo 'bar'
```

will yield the same result. Since the `ALIAS` commands bypass part of the command line parsing the translation of the value part has to be applied by the `ALIAS/CREATE` command itself. At that stage the information about quoting is no longer available.

The option "`N`" allows to inhibit the alias expansion in the value. Using this option can lead to an infinite recursion of alias translations which will be detected only when one the alias names involved is actually used.

```
PAW > ALIAS/DELETE *
PAW > ALIAS/CREATE foo bar
PAW > ALIAS/CREATE -N bar foo
PAW > ALIAS/LIST
 Argument aliases:
```

```
 BAR        => foo
 FOO        => bar
 No Command aliases defined.
PAW > foo
 *** Recursive command alias in foo
 *** Recursive argument alias in foo
 *** Unknown command: foo
PAW > bar
 *** Recursive command alias in bar
 *** Recursive argument alias in bar
 *** Unknown command: bar
```

Alias substitution happens before the command line is split-up into command name and arguments. Hence, aliases can represent several arguments at once. For example,

```
ALIAS/CREATE limits '100 -1.57 1.57'
FUN1 10 sin(x) limits
```

is equivalent to

```
FUN1 10 sin(x) 100 -1.57 1.57
```

The quotes in the ALIAS/CREATE command are necessary but they are not part of the alias value. If an alias value containing blanks is supposed to be treated as a single argument four extra quotes are needed in order that

```
ALIAS/CREATE htitle '''X vs. Y'''
1D 10 htitle 100 0 1
```

is equivalent to

```
1D 10 'X vs. Y' 100 0 1
```

Argument aliases can lead to unexpected interpretations of command lines. For example, a user defining

```
ALIAS/CREATE e EDIT
```

wants "E" to be short-hand for the command EDIT. However, the following consequence is probably not intended:

```
PAW > nt/plot 30.e
 ***** Unknown name ---> EDIT
```

For historic reasons the default option for the ALIAS/CREATE command is to define an argument alias. However, the use of argument aliases can lead to subtle side-effects and should therefore be restricted as much as possible.

4.2.2 Command aliases

This problem described above does not arise if a command alias is created instead:

```
ALIAS/CREATE -C e EDIT
```

Command aliases are only recognized if they appear at the beginning of a command line (ignoring leading blanks). Hence, there is no need to protect command arguments from inadvertent substitutions. Furthermore the match must be exact (ignoring case differences), i.e. the command

```
/GRAPHICS/HPLOT/ERRORS
```

can still be abbreviated as `HPLOT/E`.

Alias values can also represent several commands by using one of the line separators described in section 4.1.3, e.g.

```
ALIAS/CREATE -C ciao 'MESS Hello world! ; MESS How are you?'
```

4.3 System functions

`kuip` provides a set of built-in, so-called system functions which allow, for example, to inquire the current dialogue style or to manipulate strings. An application may provide additional functions. The complete list of available functions can be obtained from "`HELP FUNCTIONS`".

The function name is preceded by a `$`-sign. Arguments are given as a comma separated list of values delimited by "`(`" and "`)`". The arguments may be expressions containing other system functions.

Functions without arguments must be followed by a character which is different from a letter, a digit, an underscore, or a colon[1]. "`$OSMOSIS`" will not be recognized as the function "`$OS`" followed by "`MOSIS`". If that is the desired effect the concatenation operator has to be used: "`$OS//MOSIS`". Note however that two functions can follow each other, e.g. "`$OS$MACHINE`" because the `$`-sign does not belong to the function name.

Depending on the setting of the `SET/DOLLAR` command the name following the `$`-sign may also be an environment variable[2]. The replacement value for "$*xxx*" is obtained in the following order:

1. If *xxx* is a system function followed by the correct number and types of arguments, replace it by its value.
2. Otherwise if *xxx* is an argument-less system functions, replace it by its value.
3. Otherwise if *xxx* is a defined environment variable, replace it by its value.
4. Otherwise no replacement takes place.

[1] Excluding the colon as separator avoids the substitution of VMS logical name containing a dollar-sign such as in "`DISK$OS:[dir]file.dat`"

[2] On VMS there is a distinction between lowercase and uppercase names. Uppercase names (without the `$`-sign) are searched for first in the logical name tables and then in the symbol table while lowercase names are searched for only in the symbol table. The names `HOME`, `PATH`, `TERM`, and `USER` have a predefined meaning. In order to avoid conflicts with DCL symbols which are merely defined as abbreviations for running executables and DCL procedures all values starting with a "`$`" or "`@`" character are excluded from substitution.

4.3.1 Inquiry functions

Style inquiries

- `$STYLE` returns the name of the currently active dialogue style ("`C`", "`G`", "`GP`", etc.). This allows, for example, to a common logon macro containing different default setups depending whether the application is started in command line mode or in Motif mode:

```
IF $STYLE='XM' THEN
   ...
ELSE
   ...
ENDIF
```

- `$LAST` returns the previously executed command sequence:

```
PAW > MESS Hello world! ; MESS How are you?
 Hello world!
 How are you?
PAW > MESS $LAST
 MESS Hello world! ; MESS How are you?
```

- `$KEYVAL` returns the content of the last selected panel box in style `GP` and
- `$KEYNUM` returns row/column address in the form "*row*.*col*". The column address is always given as a two-digit number.

Alias inquiries

- `$ANUM` returns the number of *argument* aliases currently defined.
- `$ANAM(n)` returns the name and
- `$AVAL(n)` returns the value of the *n*'th argument alias. No substitution takes place if *n* is not a number between 1 and `$ANUM`. There is no guarantee that "`$ANAM($ANUM)`" refers to the most recently created alias.

Vector inquiries

- `$NUMVEC` returns the number of vectors currently defined.
- `$VEXIST(name)` returns a positive number if a vector *name* is currently defined. The actual value returned is undefined and may even change between tests on the same *name*. If the vector is undefined the value "`0`" is returned.
- `$VDIM(name,dim)` returns the vector size along index dimension *dim*; $dim = 1$ is used if the second argument is omitted. If the vector is undefined the value "`0`" is returned.
- `$VLEN(name)` returns for a 1-dimensional vector the index of the last non-zero element. For 2- and 3-dimensional vectors the result is the same as for `$VDIM`. If the vector is undefined the value "`0`" is returned.

```
PAW > V/CREATE v1(10) R 1 2 3 4 0 6
PAW > MESS $VDIM(v1) $VLEN(v1)
 10 6
PAW > V/CREATE v2($VLEN(v1))
PAW > MESS $VDIM(v2) $VLEN(v2)
 6 0
```

Environment inquiries

- `$ARGS` returns the program arguments with which the application was invoked.
- `$DATE` returns the current date in the format "*dd*/*mm*/*yy*".
- `$TIME` returns the current time in the format "*hh*/*mm*/*ss*".
- `$RTIME` returns the number of seconds elapsed since the previous usage of `$RTIME`.
- `$CPTIME` returns the seconds of CPU time spent since the previous usage of `$CPTIME`.
- `$OS` returns an identification for the operating system the application is running on, e.g. "`UNIX`", "`VM`", or "`VMS`".
- `$MACHINE` returns an identification for the particular hardware platform or Unix brand, e.g. "`HPUX`", "`IBM`", or "`VAX`". Table 4.3 shows the `$OS` and `$MACHINE` values for the different platforms. On Unix platforms the operating system version can be obtained by `$SHELL('uname -r')`.
- `$PID` returns the process number or "`1`" if the operating system does not support the notion of process IDs.
- `$IQUEST(i)` returns the *i*'th component of the status vector

  ```
  COMMON /QUEST/ IQUEST(100)
  ```

  `IQUEST(1)` always contains the return code of the most recently executed command.
- `$DEFINED(name)` returns *name* if a variable of that name is defined, or the empty string if the variable is not defined. The argument can contain "`*`" as wildcard matching any sequence of characters. All matching variable names are returned as a blank separated list.
- `$ENV(name)` returns the value of the environment variable *name*, or the empty string if the variable is not defined.
- `$FEXIST(filename)` returns "`1`" if the file exists, or "`0` otherwise.
- `$SHELL(command,n)` returns the *n*'th line of output from the shell command.
- `$SHELL`(*command,sep*) returns the output from the shell command, where newlines are replaced by the separator string. The *sep* argument can be omitted and defaults to a single blank character. The `$SHELL` function is operational only on Unix systems. The *command* string is passed to the shell set by the `HOST_SHELL` command. Alias definitions etc. in the shell specific startup script (e.g. `.cshrc`) are taken into account.

4.3.2 String manipulations

- `$LEN(string)` returns the number of characters in *string*.
- `$INDEX(string,substring)` returns the position of the first occurence of *substring* inside *string* or zero if there is none.
- `$LOWER(string)` and
- `$UPPER(string)` return the argument *string* converted to lower or upper case, respectively.
- `$SUBSTRING(string,k,n)` returns the substring
  - *string*(*k*:*k*+*n*-1) if *k*¿0, or
  - *string*(*l*+*k*+1:*l*+*k*+*n*) if $k \leq 0$, where *l*=`$LEN`(*string*).

  In any case the upper bound is clamped to `$LEN`(*string*). The argument *n* may be omitted and the result will extend to the end of *string*. Character counting starts with 1; by definition the replacement is empty if *k*=0 or *n*=0. If *n*¡0 an error message is emitted.

$OS	$MACHINE	Platform
UNIX	ALPHA	DEC Alpha OSF
UNIX	APOLLO	HP/Apollo DomainOS
UNIX	CONVEX	Convex
UNIX	CRAY	Cray Unicos
UNIX	DECS	DECstation Ultrix
UNIX	HPUX	HP/UX
UNIX	IBMAIX	AIX for IBM/370
UNIX	IBMRT	AIX for RS/6000
UNIX	LINUX	Linux for PCs
UNIX	NEXT	NeXT
UNIX	SGI	Silicon Graphics Irix
UNIX	SOLARIS	Sun Solaris
UNIX	SUN	SunOS
VM	IBM	VM/CMS for IBM/370
MVS	IBMMVS	MVS for IBM/370
VMS	ALPHA	VMS for Alpha
VMS	VAX	VMS for Vax
MSDOS	IBMPC	MSDOS for PCs
WINNT	ALPHA	Windows/NT for DEC Alpha
WINNT	IBMPC	Windows/NT for PCs

Table 4.3: Platform identification with $OS and $MACHINE

```
PAW > MESS $SUBSTRING(abcde,2)/$SUBSTRING(abcde,2,3)
 bcde/bcd
PAW > MESS $SUBSTRING(abcde,-2)/$SUBSTRING(abcde,-4,3)
 de/bcd
```

– $WORDS(*string*,*sep*) returns the number of words in *string* separated by the *sep* character. Leading and trailing separators are ignored and strings of consecutive separators count as one only. The second argument may be omitted and defaults to blank as the separator character.

```
PAW > MESS $WORDS(',abc,def,,ghi',',')
 3
```

– $WORD(*string*,*k*,*n*,*sep*) returns *n* words starting from word *k*. The last two arguments may be omitted default to blank as separator character and the replacement value extending to the last word in *string*.

```
PAW > MESS $WORD('abc def ghi',2)
 def ghi
PAW > MESS $WORD('abc def ghi',2,1)
 def
```

– $QUOTE(*string*) returns a quoted version of *string*, i.e. the string is enclosed by quote characters

and quote characters inside *string* are duplicated. The main use of this function is if an alias value containing blanks should be treated as a single lexical token in a command line:

```
ALIAS/CREATE htitle 'Histogram title'
1d 10 $QUOTE(htitle) 100 0 1
```

Another useful application of $QUOTE is to pass the value of an alias or macro variable as a character constant to a comis function, for example

```
foo = 'bar'
CALL fun.f($QUOTE([foo]))
```

is equivalent to "CALL fun.f('bar')". Since the quotes around "'bar'" are not part of the variable value the construct "CALL fun.f([foo])" would given the desired result only if the value contains blanks forcing the implicit quoting in the variable substitution.

– $UNQUOTE(*string*) returns a *string* with enclosing quote characters removed. The main use of this function is if a macro variable should be treated as several blank-separated lexical tokens:

```
limits = '100 0 1'
1d 10 'Histogram title' $UNQUOTE([limits])
```

4.3.3 Expression evaluations

– $EXEC(*cmd*) executes a macro command and returns the macro's EXITM value. Thus

```
mess $EXEC('mname 5')
```

is equivalent to

```
EXEC mname 5
mess [@]
```

– $EVAL(*expr*) returns the value of a numeric expression. The expression can contain numeric constants and references to vector elements joined by "+", -", "*", "/". Parentheses may be used to override the usual operator precedence. In addition, the functions ABS(*x*) (absolute value), INT(*x*) (truncation towards zero), and MOD(*x*,*y*) (modulus) are available. Note that all operations, including division of two integer numbers, use floating point arithmetic.

```
PAW > V/CREATE vec(3) R 1.2 3.4 4.5
PAW > MESS $EVAL((2+3)/4) $EVAL(vec(1)+vec(2)+vec(3))
 1.25 9.1
```

Even if *expr* is merely a constant, the result is always in a canonical format with a maximum of 6 non-zero digits. Non-significant zeroes and the decimal point are omitted after rounding the last digit towards $+\infty$ or $-\infty$. A mantissa/exponent notation is used if the absolute value is $\geq 10^6$ or $< 10^-4$.

```
PAW > MESS $EVAL(1.500) $EVAL(14.99999) $EVAL(0.000015)
 1.5 15 1.5E-05
```

The explicit use of $EVAL is only necessary if the result should be inserted in a place where a string is expected, for example in the MESSAGE command. In the instances where a command expects an integer or real argument expressions are implicitly evaluated even without the $EVAL function.

– $SIGMA(*expr*) passes the expression to sigma for evaluation. sigma is an array manipulation package which supports a multitude of mathematical functions (SQRT, EXP, etc.) operating on scalars

and kuip vectors:

```
PAW > V/CREATE v10(10) R 1 2 3 4 5 6 7 8 9 10
PAW > MESS $SIGMA(2*pi) $SIGMA(vsum(v10))
 6.28319 55
```

For a description of the complete sigma expression syntax refer to chapter 6.
sigma expressions do not follow the syntax rules for kuip expressions. Therefore they cannot contain kuip system functions with arguments. They may, however, contain argument-less system functions, alias names, and macro variables.

– $RSIGMA is a slight variation of $SIGMA. Both functions return a scalar result in the same canonical format used by $EVAL. The only difference is that $SIGMA removes the decimal point from integral values while $RSIGMA leaves it in. For example, $RSIGMA should be used to calculate argument values to be passed to a comis routine

```
      SUBROUTINE FUN(X)
      PRINT *,X
      END
```

as floating point constants:

```
PAW > CALL fun.f($SIGMA(sqrt(8)))
  2.828430
PAW > CALL fun.f($SIGMA(sqrt(9)))
  .4203895E-44
PAW > CALL fun.f($RSIGMA(sqrt(9)))
  3.000000
```

If the expression evaluates to a vector result $SIGMA (and $RSIGMA) return the name of a temporary vector containing the result. $SIGMA with a vector result can be used in all places where a vector name is expected, e.g.

```
PAW > V/PRINT $SIGMA(sqrt(array(3,1#3)))
 ?SIG1(1) = 1
 ?SIG1(2) = 1.41421
 ?SIG1(3) = 1.73205
```

The lifetime of these vectors is limited to the current command. Hence, their names should not be assigned to macro variables and not be used in alias definitions:

```
PAW > A/CREATE square_roots $SIGMA(sqrt(array(3,1#3)))
PAW > V/PRINT square_roots
 *** VECTOR/PRINT: unknown vector ?SIG1
```

– $FORMAT(*expr,format*) returns the expression value formatted according to the Fortran *format* specifier. The possible formats are “F”, “E”, “G”, “I”, and “Z” (hexadecimal).

```
PAW > MESS 'x = '//$FORMAT(1.5,F5.2)
 x =  1.50
PAW > MESS 'i = '//$FORMAT(15,I5)
 i =    15
PAW > MESS 'j = '//$FORMAT(15,I5.4)
 j =  0015
```

– `$INLINE(name)` allows to insert the value of an alias or macro variable into an expression which is then treated as being part of the expression. For example,

```
convert = '$UPPER'
foo = $INLINE([convert])('bar')
```

is equivalent to "`foo = $UPPER('bar')`", i.e. "`foo = 'BAR'`". Without `$INLINE` the content of `[convert]` would be treated as a text string with the result that "`foo = '$UPPER(''bar'')'`".

4.3.4 Histograms inquiry functions

– `$HEXIST(id)` returns 1 if histogram `id` exists or 0 otherwise
– `$HINFO(id,'ENTRIES')` returns the number of entries.
– `$HINFO(id,'MEAN')` returns the mean value.
– `$HINFO(id,'RMS')` returns the standard deviation.
– `$HINFO(id,'EVENTS')` returns the number of equivalent event.
– `$HINFO(id,'OVERFLOW')` returns the content of overflow channel.
– `$HINFO(id,'UNDERFLOW')` returns the content of underflow channel.
– `$HINFO(id,'MIN')` returns the minimum bin content.
– `$HINFO(id,'MAX')` returns the maximum bin content.
– `$HINFO(id,'SUM')` returns the total histogram content.
– `$HINFO(id,'XBINS')` returns the number of bins in X direction.
– `$HINFO(id,'XMIN')` returns the lower histogram limit in X direction.
– `$HINFO(id,'XMAX')` returns the upper histogram limit in X direction.
– `$HINFO(id,'YBINS')` returns the number of bins in Y direction.
– `$HINFO(id,'YMIN')` returns the lower histogram limit in Y direction.
– `$HINFO(id,'YMAX')` returns the upper histogram limit in Y direction.
– `$HTITLE(id)` returns the histogram title.

4.3.5 Graphics inquiry functions

– `$GRAFINFO('XZONES')` returns the number of zones in X direction.
– `$GRAFINFO('YZONES')` returns the number of zones in Y direction.
– `$GRAFINFO('NT')` returns the current normalization transformation number.
– `$GRAFINFO('WNXMIN')` returns the lower X limit of window in current NT.
– `$GRAFINFO('WNXMAX')` returns the upper X limit of window in current NT.
– `$GRAFINFO('WNYMIN')` returns the lower Y limit of window in current NT.
– `$GRAFINFO('WNYMAX')` returns the upper Y limit of window in current NT.
– `$GRAFINFO('VPXMIN')` returns the lower X limit of viewport in current NT.
– `$GRAFINFO('VPXMAX')` returns the upper X limit of viewport in current NT.
– `$GRAFINFO('VPYMIN')` returns the lower Y limit of viewport in current NT.
– `$GRAFINFO('VPYMAX')` returns the upper Y limit of viewport in current NT.
– `$GRAFINFO('?attr')` returns the current value of the `hplot/higz` attribute `attr`. See the HELP of the command `SET` to have the list of the valid values of `attr`.

4.3.6 Cuts manipulations

- `$CUT(n)` returns the cut expression `$n`
- `$CUTEXPAND(string)` replace `$n` in the (quoted) string by `$CUT(n)`

4.4 Vectors

`kuip` provides optionally (`VECDEF`) the facilities to store vectors of integer or real data. These vectors, or rather arrays with up to 3 index dimensions, can be manipulated by `kuip` built-in commands (see "`HELP VECTOR`"). They are also accessible to application routines (`KUGETV` and `KUVECT`). Furthermore they are interfaced to the array manipulation package `sigma` and to the Fortran interpreter `comis` (see chapter 6).

Vectors are memory resident only, i.e. they are not preserved across program executions. The commands `VECTOR/READ` and `VECTOR/WRITE` allow to save and restore vector data from an external file in text format.

Vector names may be composed of letters, digits, underscores and question marks up to a maximum length of 32 characters[3]. Names starting with "`?`" are reserved for internal use by `kuip`.

The only exception is the predefined vector simply called "`?`" which has a fixed size of 100 real elements. Unlike the others the "`?`" vector is mapped to a fixed memory location (the common block `/KCWORK/`). It does not show up in `VECTOR/LIST` output and it is not counted in the result of `$NUMVEC`.

4.4.1 Creating vectors

Vectors can be created with the `VECTOR/CREATE` command. The size of the index dimensions is given in Fortran notation, e.g.

```
VECTOR/CREATE v1(100)
VECTOR/CREATE v2(10,10)
```

The lower index bound always starts off at 1, i.e. "`V/CREATE v(-10:10)`" is not allowed. Omitting the index dimension as in

```
VECTOR/CREATE v
```

is equivalent to

```
VECTOR/CREATE v(1)
```

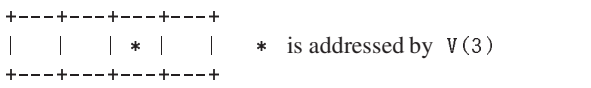
`kuip` does not keep track of the actual number of index dimension given in the `VECTOR/CREATE` command, i.e.

```
VECTOR/CREATE v1(10)
VECTOR/CREATE v3(10,1,1)
```

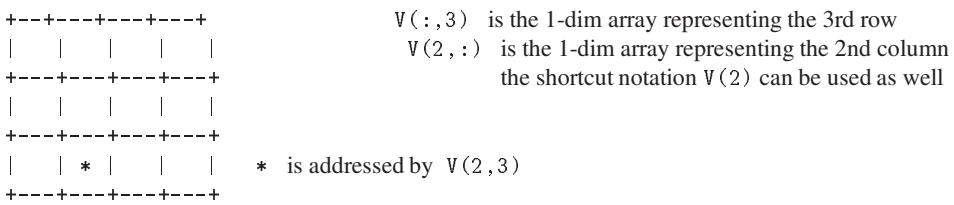
are equivalent.

[3] Vector names which should be processed by `sigma` are currently limited to 7 characters.

Definition: `VECTOR/CREATE V(NCOL)`



Definition: `VECTOR/CREATE V(NCOL,NROW)`



Definition: `VECTOR/CREATE V(NCOL,NROW,NPLANE)`

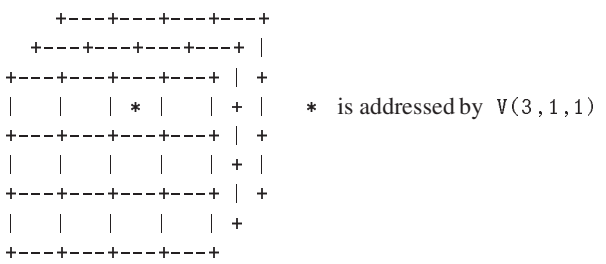


Figure 4.6: Addressing scheme for `kuip` vectors

4.4.2 Accessing vectors

Single vector elements can be used in `kuip` expressions where they are treated as numeric constants. Vectors with a single element only we will refer to as "*scalar vectors*". They have the special property that in expressions it is sufficient to give the name without the "`(1)`" subscript.

Complete vectors and vector subranges can be used in the `$SIGMA` function and as argument to commands expecting a vector name. The subrange notation is the same as in Fortran, e.g. `v(3:5)`. The elements of arrays are stored in column-major order, i.e. the elements `v(1,2)` and `v(2,2)` are adjacent in memory (see figure 4.6).

The vector processing commands are expected to deal only with contiguous vectors. Therefore a subrange referring to a non-contiguous set of elements is copied into a temporary vector and cannot be used as output parameter.

4.5 Expressions

`kuip` has a built-in parser for different kinds of expressions: arithmetic expressions, boolean expressions, string expressions, and "garbage expressions".

4.5.1 Arithmetic expressions

The syntactic elements for building arithmetic expressions are shown in table 4.4. They can be used
- in the macro statements DO, FOR, and EXITM
- in macro variable assignments
- as system function arguments where a numeric value is expected
- as command arguments retrieved with KUGETI or KUGETR
- as argument to the $EVAL function

Note that all arithmetic operations are done in floating point, i.e. "5/2" becomes "2.5". If a floating point result appears in a place where an integer is expected, for example as an index, the value is truncated.

4.5.2 Boolean expressions

Boolean expressions can only be used in the macro statements IF, WHILE, and REPEAT. The possible syntactic elements are shown in table 4.5.

In addition, a single arithmetic expression is also accepted as boolean expression, interpreting any non-zero value as *true*. This allows, for example, the short-cuts

```
IF $VEXIST(v1) THEN
...
WHILE 1 DO
...
```

instead of the explicit forms

```
IF $VEXIST(v1)<>0 THEN
...
WHILE 1=1 DO
...
```

Note, however, that an arithmetic expression is **not** equivalent to a boolean value. This implies that

```
IF $VEXIST(v1) .and. $VEXIST(v2) THEN   | error
```

is not accepted and has to be written as

```
IF $VEXIST(v1)<>0 .and. $VEXIST(v2)<>0 THEN
```

4.5.3 String expressions

String expressions can be used
- in the macro statements CASE, FOR, and EXITM
- in macro variable assignments
- as system function arguments where a string value is expected
- as argument to the $EVAL function

They may be constructed from the syntactic elements shown in table 4.6.

expr	::=	*number*	
	|	*vector-name*	for scalar vectors
	|	*vector-name* (*expr*)	
	|	*vector-name* (*expr* , *expr*)	
	|	*vector-name* (*expr* , *expr* , *expr*)	
	|	[*variable-name*]	if variable value has form of a numeric constant or is the name of a scalar vector
	|	[*variable-name*] (*expr* ...)	if variable value is a vector name
	|	*alias-name*	if alias value has form of a numeric constant
	|	$*system-function* (...)	if function returns a numeric value
	|	- *expr*	
	|	*expr* + *expr*	
	|	*expr* - *expr*	
	|	*expr* * *expr*	
	|	*expr* / *expr*	
	|	(*expr*)	
	|	ABS (*expr*)	
	|	INT (*expr*)	
	|	MOD (*expr* , *expr*)	

Table 4.4: Syntax for arithmetic expressions

bool	::=	*expr rel-op expr*	*rel-op*	::=	.LT.	|	.LE.
	|	*string eq-op string*		|	<	|	<=
	|	*expr eq-op string*		|	.GT.	|	.GE.
	|	.NOT. *bool*		|	>	|	>=
	|	*bool* .AND. *bool*		|	*eq-op*		
	|	*bool* .OR. *bool*	*eq-op*	::=	.EQ.	|	.NE.
	|	(*bool*)		|	=	|	<>

Table 4.5: Syntax for boolean expressions

string	::=	*quoted-string*	
	|	*unquoted-string*	
	|	*string* // *string*	concatenation
	|	*expr* // *string*	value of expression converted to string representation
	|	[*variable-name*]	
	|	*alias-name*	
	|	$*system-function* (...)	

Table 4.6: Syntax for string expressions

4.5.4 Garbage expressions

Expressions which do not satisfy any of the above syntax rules we want to call "garbage" expressions. For example,

```
s = $OS$MACHINE
```

is not a proper string expression. Unless they appear in a macro statement where specifically only an arithmetic or a boolean expression is allowed, `kuip` does not complain about these syntax errors. Instead the following transformations are applied:

1. alias substitution
2. macro variable replacement; values containing a blank character are implicitly quoted
3. system function calls are replaced one by one by their value provided that the argument is a syntactically correct expression
4. string concatenation

The same transformations are also applied to command arguments. Therefore the concatenation operator "`//`" can be omitted in many cases. For example,

```
MESS $OS$MACHINE
MESS $OS//$MACHINE
MESS $EVAL($OS$MACHINE)
MESS $EVAL($OS//$MACHINE)
```

give all the same result.

4.5.5 The small-print on `kuip` expressions

`kuip` expressions are evaluated by a `yacc`-generated parser. `Yacc` ("*Yet Another Compiler-Compiler*") is a standard Unix tool. It produces a C routine to parse an token stream which follows the syntax rules fixed by the grammar definition.
The parser needs as front-end a lexical analyzer which reads the input stream, separates it into tokens, and returns the token type and its value to the parser. There is another Unix tool `lex` which can produce an appropriate lexical analyzer from a set of rules. In the case of `kuip` the lexical analyzer had to be hand-crafted because the interpretation of a symbol depends very much on the global context. For example, if the input stream consists is simply "`foo`" the lexical analyzer has to check consecutively:

- If `foo` is defined as an alias:
  - If the alias value looks like a number, classify it as a *number*.
  - Otherwise classify the alias value as a *string*.
- Otherwise classify it as the *string* "`'foo'`".

A similar reasoning has to be applied for "`[foo]`":

- If `foo` is a defined macro variable:
  - If the variable value looks like a number, classify it as a *number*.
  - If the variable value is the name of a scalar vector, classify it as a *number*.
  - Otherwise classify the variable value as a *string*.
- Otherwise classify it as the *string* "`'[foo]'`".

`kuip` macro variables do not have to (and cannot) be declared. The value is always stored as a string and it depends on the context whether the value should be interpreted as a number. Also there is no way to tell in the beginning whether the right-hand side of an assignment is an arithmetic or a string expression.
The lexical analyzer starts off interpreting tokens as a numbers if it can. For example,

```
a = '1'
b = '2'
c = [a]+[b]
```

is tokenized as "*number* + *number*" and gives "`c = 3`" even though the values assigned to `a` and `b` are originally quoted. If we have a string expression

```
[foo]//[bar]
```

this could result in the possible token sequences

```
string // string
number // string
string // number
number // number
```

depending whether the values of `foo` and `bar` look like a number. Accordingly we would have to define four grammar rules to cover these different cases. The same problem occurs in system functions expecting a string argument, e.g.

```
$SUBSTRING([foo],2,3)
```

would need two rules for `foo` being a number or a genuine string.
`Yacc` allows to avoid this inflation of necessary rules by using so-called lexical tie-ins. After having seen "`//`" or "`$SUBSTRING(`" the parser can instruct the lexical analyzer that it should not attempt to classify the next token as a number. Therefore a single rule for each system function is sufficient.
However, a lexical tie-in can only be used after the parser found a unique match between the token sequence and all grammar rules In the case of string concatenation we still have to provide two separate rules for

```
string // string
number // string
```

The grammar rule (see above) actually says that the left-hand side of the "`//`" operator can be either an arithmetic or a string expression. An arithmetic expression is evaluated and then transformed into the result's string representation. For example,

```
2*3//4
```

gives "`'64'`". On the other hand,

```
4//2*3
```

gives "`'42*3'`". It does not become "`'46'`" because the right-hand side is not consider to be an arithmetic expression. It does also not become "`126`" because a result of a string operation is never again treated as a number even if it looks like one.
The lexical analyzer forwards numbers in arithmetic expressions as floating point values to the parser. The result is converted back to the string representation when it has to be stored in the macro variable. Since a single numeric value already counts as an arithmetic expression the original string representation can be lost. For example,

```
a = '0123456789'
b = [a]
MESS $LEN([a]) $LEN([b])
```

results in "`10 11`" because the assignment "`b = 0123456789`" is taken as an arithmetic expression which is reformatted into `1.23457E+08`. The reformatting can be inhibited by using

```
b = $UNQUOTE([a])
```

The `$UNQUOTE` function removes quotes around a string. If the string is already unquoted it does nothing except that in this case the parser will treat the value of `[a]` as a string.
Macros should not depend on this reformatting behavior. We consider it as an obscure side-effect of the present implementation rather than a feature. If it causes inconvenience and we have a good idea how to avoid it the behavior may change in a future `kuip` version.

4.6 Macros

A macro is a set of command lines stored in a file, which can be created and modified with any text editor. The command EXEC invokes the macro and allows for two ways of specifying the macro name:

```
EXEC file
EXEC file#macro
```

The first form executes the first macro contained in *file* while the second form selects the macro named *macro*. The default extension for *file* is ".kumac".

Example of macro calls

```
PAW > EXEC abc    | Execute first (or unnamed) macro of file abc.kumac
PAW > EXEC abc#m  | Execute macro M of file abc.kumac
```

In addition to all available kuip commands the special "macro statements" in table 4.7 are valid only inside macros (except for EXEC and APPLICATION, which are valid both inside and outside).

Note that the statement keywords are fixed. Aliasing such as "ALIAS/CREATE jump GOTO" is not allowed.

4.6.1 Macro definitions and variables

A .kumac file can contain several macros. An individual macro has the form

```
MACRO macro-name  [ parameter-list ]
   statements
RETURN [ expression ]
```

Each statement is either a command line or one of the macro constructs described below. For the first macro in the file the MACRO header can be omitted. For the last macro in the file the RETURN trailer may be omitted. Therefore a .kumac file containing only commands (like the LAST.KUMAC) already constitutes a valid macro.

Input lines starting with an asterisk ("*") are comments. The vertical bar ("|") acts as in-line comment character unless it appears inside a quoted string. An underscore ("_") at the end of a line concatenates it to the next line.

Invoking a macro triggers the compilation of the whole .kumac file—not just the single macro called for. The

```
ENDKUMAC
```

statement fakes an end-of-file condition during the compilation. This allows to keep unfinished material, which would cause compilation errors, simply by moving it after the ENDKUMAC statement rather than having to comment the offending lines.

The APPLICATION statement has the same form and similar functionality as the SET/APPLICATION command:

Macro Statements	
STATEMENT	DESCRIPTION
`MACRO mname [ var1=val1 ... ]`	define macro `mname`
`RETURN [ value ]`	end of macro definition
`ENDKUMAC`	end of macro file
`EXEC mname [ val1 ... ]`	execute macro `mname`
`EXITM [ value ]`	return to calling macro
`STOPM`	return to command line prompt
`APPLICATION command marker`	In-line text passed to application command
`name = expression`	assign variable value
`READ var [ prompt ]`	prompt for variable value
`SHIFT`	shift numbered macro variables
`GOTO label`	continue execution at `label`
`label:`	`GOTO` target label (must terminate with a colon)
`IF expr GOTO label`	continue at `label` if `expr` is true
`IF-THEN, ELSEIF, ELSE, ENDIF`	conditional block statement
`CASE, ENDCASE`	Macro flow control
`WHILE-DO, ENDWHILE`	Macro flow control
`REPEAT, UNTIL`	Macro flow control
`DO, ENDDO`	Macro flow control
`FOR, ENDFOR`	Macro flow control
`BREAKL`	Macro flow control
`NEXTL`	Macro flow control
`ON ERROR CONTINUE`	ignore error conditions
`ON ERROR GOTO label`	continue at `label` on error condition
`ON ERROR EXITM value`	return to calling macro on error condition
`ON ERROR STOPM`	return to command input on error condition
`OFF ERROR`	deactivate the `ON ERROR GOTO` handling
`ON ERROR`	reactivate the previous `ON ERROR GOTO` setting

Table 4.7: `kuip` macro statements

```
APPLICATION  command  marker
   text
marker
```

The text up to the next line containing only the end marker starting in the first column is written to a temporary file and then passed to the application command. The text is not interpreted in any way, i.e. variable substitution etc. does not take place.
Instead of the full spelling APPLICATION any valid abbreviation of /KUIP/SET_SHOW/APPLICATION be used, e.g. "APPL". A call to SET/APPLICATION as a result of an alias expansion, however, is not allowed.

Macro execution

Inside a macro the EXEC statement can call other macros. A macro may also call itself recursively. The EXEC command allows two different forms for specifying the macro to be executed:

```
EXEC  fname#mname  [ argument-list ]
```

and

```
EXEC  name  [ argument-list ]
```

Between the EXEC *statement* and the EXEC *command* there is a slight difference. The command "EXEC name" executes the first macro in name.kumac while the EXEC statement will try first whether a macro name is defined within the current .kumac file.
Macro execution terminates when one of the statements

```
    EXITM  [ expression ]
```

or

```
    RETURN  [ expression ]
```

or

```
    STOPM
```

is encountered. The EXITM and RETURN statements return to the calling macro. They allow to pass a return value which is stored into the special variable [@] of the calling macro. If no value is given it defaults to "0". Note that the RETURN statement also flags the end of the macro definition, i.e. the construct

```
IF ... THEN
  RETURN         | error!
ENDIF
```

is illegal. The STOPM statement unwinds nested macro calls and returns to the command line prompt immediately.

Macro variables

Macro variables do not have to be declared. They become defined by an assignment statement:

```
name = expression
```

The right-hand side of the assignment can be an arithmetic expression, a string expression, or a garbage expression (see section 4.5). The expression is evaluated and the result is stored as a string (even for arithmetic expressions).

The variable value can be used in other expressions or in command lines by enclosing the name in square brackets:

```
[name]
```

For example,

```
greet = Hello
msg = [greet]//' World'
MESS [msg]
```

If the name enclosed in brackets is not a macro variable then no substitution takes place.

Variable values can also be queried from the user during macro execution. The statement

```
READ  name  [ prompt ]
```

prompts for the variable value. If the prompt string is omitted it is constructed from the macro and variable names. The variable value prior to the execution of the READ statement is proposed as default value and will be left unchanged if the user answers simply be hitting the RETURN-key.

Macro using the READ statement

```
MACRO m
READ foo
bar = abc
READ bar
MESS [foo] [bar]
msg = ''
READ msg 'Enter message:'
MESS You said [msg].
```

Output when executing

```
PAW > EXEC m
 Macro m: foo ? (<CR>=[foo]) 123
 Macro m: bar ? (<CR>=abc)
 123 abc
 Enter message: (<CR>=) Hello
 You said Hello.
```

Macro arguments

The EXEC command can pass arguments to a macro. The arguments are assigned to the numbered variables [1], [2], etc. For example, with the macro definition

```
MACRO m
MESS p1=[1] p2=[2]
```

we get the result

```
PAW > EXEC m foo bar
 p1=foo p2=bar
```

Unlike named variables undefined numbered variables are always replaced by the blank string ' ', i.e.

```
PAW > EXEC m foo
 p1=foo p2=' '
```

The MACRO statement can define default values for missing arguments. With the macro definition

```
MACRO m 1=abc 2=def
MESS p1=[1] p2=[2]
```

we get the result

```
PAW > EXEC m foo
 p1=foo p2=def
```

The macro parameters can also be named, for example:

```
MACRO m arg1=abc arg2=def
MESS p1=[arg1] p2=[arg2]
```

Even if the parameters are named the corresponding numbered variables are created nevertheless. The named variables are a copy of their numbered counterparts rather that aliases, i.e. the above macro definition is equivalent to

```
MACRO m 1=abc 2=def
arg1 = [1]
arg2 = [2]
```

The named parameters can be redefined by a variable assignment which leaves the value of the numbered variable untouched. For example,

```
MACRO m arg=old
MESS [1] [arg]
arg = new
MESS [1] [arg]
```

yields

```
PAW > EXEC m
 old old
 old new
```

The EXEC command allows to give values for named parameters in non-positional order. For example,

```
MACRO m arg1=abc arg2=def
MESS [arg1] [arg2]
```

can be used as

```
PAW > EXEC m arg2=foo
 abc foo
```

Unnamed EXEC arguments following a named argument are assigned to numbered variables beyond the parameters listed in the MACRO definition. For example,

```
PAW > EXEC m arg1=foo bar
 foo def
```

i.e. the second argument "bar" is not assigned to [arg2] or [2] but to [3]. Note that this differs from the behavior for command arguments (see section 4.1.2).

The construct *name*=*value* may also be used in the EXEC command for names not defined in the macro's parameter list. The variable *name* is implicitly defined inside the macro. For example,

```
MACRO m
MESS [foo]
```

yields

```
PAW > EXEC m
 [foo]
PAW > EXEC m foo=bar
 bar
```

Therefore a string containing a "=" must be quoted[4] if it should be passed to the macro literally:

```
PAW > EXEC m 'foo=bar'
 foo=bar
```

Since a undefined variable name can be thought of as having the value '[name]', the construct

```
IF [var]<>'[var]' THEN
```

allows to test whether such an external variable definition was provided.

Passing a value as argument to a macro is not quite the same as assigning the value to a variable inside the macro. The macro argument is not tried to be evaluated as an arithmetic expression. String operations, however, such as concatenation and alias substitutions, are applied. For example, "EXEC m1 2*3 4//5" with

```
MACRO m1 a=0 b=0
mess [a] [b]
```

yields "2*3 45", while "EXEC m2" with

[4] Up to the 94a release of kuip the treatment of quoted strings as macro arguments was very primitive. The value assigned to the macro variable was obtained by simply stripping off the quote character on both sides. For example, a cut expression "nation='F*'" had to be written as EXEC m 'nation='F*''

In the 94b release the quoting of macro arguments was made consistent with the general rules and the correct quoting is now EXEC m 'nation=''F*''' The old spelling is still accepted but emits a warning message Old style use of quotes in macro argument fixed to 'nation=''F*'''

```
MACRO m1
a = 2*3
a = 4//5
mess [a] [b]
```

yields "6 45". Macro arguments are not tried as arithmetic expressions in order to allow passing of vector names without the use of quotes. Otherwise "EXEC m v1", where v1 is a scalar vector, would pass the value of v1(1) rather than the string 'v1'.

Note that the result "6 45" can also be obtained from the first of the above examples by means of the $INLINE function:

```
MACRO m1 1=0 b=0
a = $INLINE([1])
mess [a] [b]
```

Special variables

A numbered variable cannot be redefined, i.e. an assignment such as "1 = foo" is illegal. The only possibly manipulation of numbered variables is provided by the

```
SHIFT
```

statement which copies [2] into [1], [3] into [2], etc. and discards the value of the last defined numbered variable. For example, the construct

```
WHILE [1] <> ' ' DO
  arg = [1]
  ...
  SHIFT
ENDDO
```

allows to traverse the list of macro arguments.

For each macro the following special variables are always defined:

- [0] contains the fully qualified macro file name, e.g. "./fname.kumac#mname"
- [#] contains the number of macro arguments
- [*] is the concatenation of all macro arguments separated by blanks
- [@] contains the return value of the most recent EXEC call

Like for numbered variables these names cannot be used on the left-hand side of an assignment. The values or [#] and [*] are updated by the SHIFT statement.

Variable indirection and arrays

Macro variables can be referenced indirectly by constructing the name using other variables, for example

Example of Input Macros

```
MACRO EXITMAC
  MESSAGE At first, '[@]' = [@]
  EXEC EXIT2
  IF [@] = 0 THEN
     MESSAGE Macro EXIT2 successful
  ELSE
     MESSAGE Error in EXIT2 - code [@]
  ENDIF
RETURN
MACRO EXIT2
  READ NUM
  IF [NUM] > 20 THEN
     MESSAGE Number too large
     EXITM [NUM]-20
  ELSE
     V/CREATE V([NUM])
  ENDIF
RETURN
```

Output when executing

```
PAW > EXEC EXITMAC
At first, [@] = 0
Macro EXIT2: NUM ? 25
Number too large
Error in macro EXIT2 - code 5
PAW > EXEC EXITMAC
At first, [@] = 0
Macro EXIT2: NUM ? 16
Macro EXIT2 successful
```

```
DO i = 1, 10
  a_[i] = [i] * [i]
ENDDO
s = 0
DO i = 1, 10
  s = [s] + [a_[i]]
ENDDO
```

While for `kuip` we simply created ten variables `a_1`, ..., `a_10`, we can also look at it as an array `a_`*i*. We don't even need to remember the dimension of the array. The system function `$DEFINED` returns all defined variables matching a wildcard, for example

```
s = 0
DO i = 1, $WORDS($DEFINED('a_*'))
  s = [s] + [a_[i]]
ENDDO
```

Instead of `a_`*i* we can also use the more conventional array notation `a(`*i*`)`

```
DO i = 1, 10
  a([i]) = [i] * [i]
ENDDO
s = 0
DO i = 1, $WORDS($DEFINED('a(*)'))
  s = [s] + [a([i])]
ENDDO
```

as long as we have the possibility to match all array elements with a single wildcard expression.

Since for `kuip` all array elements are just simple variables the indices do not even need to be numeric. We can also construct associative arrays where the indices are names, for example

```
events(mu) = 1000
events(el) = 100
events(tau) = 10
total = 0
names = $DEFINED('events(*)')
DO i = 1, $WORDS([names])
  name = $WORD([names],[i],1)
  total = [total] + [[name]]
ENDDO
```

By the same token we can also create multi-dimensional arrays, for example

```
DO i = 1, 3
  DO j = 1, 3
     a([i],[j]) = [i]*2+[j]
  ENDDO
ENDDO
```

The `$DEFINED` function returns the matching variable names sorted in alphabetical order, i.e.

```
$DEFINED('events(*)') is 'events(el) events(mu) events(tau)'
$DEFINED('a(*)') is 'a(1) a(10) a(2) ... a(9)'
```

and not necessarily in the order in which they were created.

The indirection only allows for variable substitution when constructing the actual variable name. Expression evaluation etc. does not take place and constructs such as

```
  total = [total] + [$WORD([names],[i],1)]   | invalid!
```

are not allowed.

The construct `[[name]]` can also be written as

```
[%name]
```

For example, this is another way to traverse the list of macro arguments:

```
DO i=1,[#]
  arg = [%i]
  ...
ENDDO
```

Except for the `[%name]` construct variable indirection is available only since the 95a release.

Global variables

Global variables can be made visible inside a macro by executing the commands `GLOBAL/CREATE` or `GLOBAL/IMPORT`. Technically these commands create a local variable with the same name initialized to the value of the global variable. When assigning a value to the local variable the change is also propagated to the global variable. Therefore, once they are made visible inside a macro, global variables are assigned to and used in the same way as local variables.

The `GLOBAL/CREATE` command creates a global variable allowing to specify an initial value and a comment text, e.g.

```
GLOBAL/CREATE m_e  0.0005 'Electron mass (GeV)'
GLOBAL/CREATE m_mu 0.106  'Muon mass (GeV)'
```

If executed inside a macro the global variable becomes visible there.

The `GLOBAL/IMPORT` command has an effect only when executed inside a macro. It allows to make global variables visible which have been created elsewhere. The import list may contain "`*`" as a wildcard for any character sequence, for example

```
GLOBAL/IMPORT m_*
```

Only those global variables existing at the time the `GLOBAL/IMPORT` is executed become visible. Therefore, global variables created in an inferior macro do not become visible even if they match the wildcard. For example, in

```
MACRO a
GLOBAL/IMPORT m_*
EXEC b
...
RETURN

MACRO b
GLOBAL/CREATE m_tau 1.784 'Tau mass (GeV)'
RETURN
```

`m_tau` is not visible in macro `a` unless it is imported after executing `b`.

Deleting a global variable in an inferior macro, on the other hand, also deletes the associated local variables in the macro call stack. For example, in

```
MACRO a
GLOBAL/IMPORT m_*
EXEC b
...
RETURN

MACRO b
GLOBAL/DELETE m_mu
RETURN
```

when returning from macro `b` the imported variable `m_mu` will become undefined.

Global variables can also be set and used from the command line, for example,

```
PAW > g/cre x 2
PAW > x=[x]*2
PAW > mess [x]
 4
```

However, the implicit creation when assigning a value to an undefined variables does not apply:

```
PAW > y=0
 *** Unknown command: y=0
```

Global variables are available only since the 95a release.

4.6.2 Flow control constructs

There are a variety of constructs available for controlling the flow of macro execution. Most for the constructs extend over several lines up to an end clause. The complete block counts as a single statement and inside each block may be nested other block statements.
The simplest form of flow control is provided by the

```
GOTO label
```

statement which continues execution at the statement following the target label:

```
label:
```

If the jump leads into the scope of a block statement, for example a DO-loop, the result is undefined. The target may be given as an expression evaluating to the actual label name, e.g.

```
name = label
  ...
GOTO [name]
  ...
label:
```

In the label definition the colon must follow the label name immediately without any intervening blanks. The label may be followed by a command on the same line, e.g.

```
label: MESS Hello
```

Conditional execution

```
IF expression THEN
   statements
ELSEIF expression THEN
   statements
...
ELSEIF expression THEN
   statements
ELSE
   statements
ENDIF
```

The general IF construct executes the statements following the first IF/ELSEIF clause for with the boolean expression is true and then continues at the statement following the ENDIF.
The ELSEIF clause can be repeated any number of times or can be omitted altogether. If none of the expressions is true, the statements following the optional ELSE clause are executed.

```
IF expression GOTO label
```

This old-fashioned construct is equivalent to

```
IF expression THEN
   GOTO label
ENDIF
```

```
CASE expression IN
(label)  [ statements ]
...
(label)  [ statements ]
ENDCASE
```

The CASE switch evaluates the string expression and compares it one by one against the label lists until the first match is found. If a match is found the statements up to the next label are executed before skipping to the statement following the ENDCASE. None of the statements are executed if there is no match with any label.

Each label is a string constant and the comparison with the selection expression is case-sensitive. If a label is followed by another label without intervening statements then a match of the first label will skip to the ENDCASE immediately. In order to execute the same statement sequence for distinct labels a comma-separated list of values can be used. The "*" character in a label item acts as wild-card matching any string of zero or more characters, i.e. "(*)" constitutes the default label.

Example for CASE labels with wild-cards

```
MACRO CASE
   READ FILENAME
   CASE [FILENAME] IN
   (*.ftn, *.for)  TYPE = FORTRAN
   (*.c)           TYPE = C
   (*.p)           TYPE = PASCAL
   (*)             TYPE = UNKNOWN
   ENDCASE
   MESSAGE [FILENAME] is a [TYPE] file.
RETURN
```

Loop constructs

The loop constructs allow the repeated execution of command sequences. For DO-loops and FOR-loops the number of iterations is fixed before entering the loop body. For WHILE and REPEAT the loop count depends on the boolean expression evaluated for each iteration.

```
DO loop = start_expr, finish_expr  [, step_expr ]
   statements
ENDDO
```

The step size defaults to "1". The arithmetic expressions involved can be floating point values but care must be taken of rounding errors. A DO-loop is equivalent to the construct

```
count = ( finish_expr - start_expr ) / step_expr
loop = start_expr
step = step_expr
label:
IF [count] >= 0 THEN
   statements
   loop = [loop] + [step]
   count = [count] - 1
   GOTO label
ENDIF
```

where all variables except for `loop` are temporary.

Note that "`DO i=1,0`" results in zero iterations and that the expressions are evaluated only once. i.e. the loop

```
n = 10
DO i=1,[n]
   MESS [i] [n]
   n = [n] - 1
ENDDO
```

is iterated 10 times and leaves "`i = 11`" afterwards.

```
FOR name IN expr_1 [ expr_2 ... expr_n ]
   statements
ENDFOR
```

In a `FOR`-loop the number of iterations is determined by the number of items in the blank-separated expression list. The expression list must not be empty. One by one each expression evaluated and assigned to the variable `name` before the statements are executed. The equivalent construct is the loop-unrolling

```
name = expr_1
statements
name = expr_2
statements
...
name = expr_n
statements
```

The expressions can be of any type: arithmetic, string, or garbage expressions, and they do not need to be all of the same type. In general each expression is a single list item even if the result contains blanks. For example,

```
foobar = 'foo bar'
FOR item IN [foobar]
   MESS [item]
ENDFOR
```

results in a single iteration. The variable `[*]` is treated as a special case being equivalent to the expression list "`[1] [2] ... [n]`" which allows yet another construct to traverse the macro arguments:

```
FOR arg IN [*]
   ...
ENDFOR
```

```
WHILE expression DO
   statements
ENDWHILE
```

The WHILE-loop is iterated while the boolean expression evaluates to true. The loop body is not executed at all if the boolean expression is false already in the beginning. The equivalent construct is:

```
label:
IF expression THEN
   statements
   GOTO label
ENDIF
```

```
REPEAT
   statements
UNTIL expression
```

The body of a REPEAT-loop is executed at least once and iterated until the boolean expression evaluates to true. The equivalent construct is:

```
label:
   statements
IF .NOT. expression GOTO label
```

```
BREAKL  [ levels ]
```

allows to terminate a loop prematurely. The BREAKL statement continues executing after the end clause of the enclosing DO, FOR, WHILE, or REPEAT block.

```
NEXTL  [ levels ]
```

allows to terminate one loop iteration and to continue with the next one. The NEXTL statement continues executing just before the end clause of the enclosing DO, FOR, WHILE, or REPEAT block.

Both BREAKL and NEXTL allow to specify the number of nesting levels to skip as an integer constant.

Error handling

Each command returns a status code which should be zero if the operation was successful or non-zero if any kind of error condition occurred. The status code is stored in the IQUEST(1) status vector and can be tested as, for example

Example of using BREAKL and NEXTL

```
WHILE 1=1 DO
   ...
   IF expr THEN
      BREAKL
   ENDIF
   ...
   DO i=1,[#]
      ...
      IF [%i]='-' THEN
         NEXTL
      ENDIF
      IF [%i]='--' THEN
         NEXTL 2
      ENDIF
      ...
   ENDDO
   ...
ENDWHILE
```

Equivalent code using GOTOs

```
WHILE 1=1 DO
   ...
   IF expr GOTO break_while
   ...
   DO i=1,[#]
      ...
      IF [%i]='-' GOTO next_do
      IF [%i]='--' GOTO next_while
     ...
   next_do:
   ENDDO
   ...
next_while:
ENDWHILE
break_while:
```

```
HISTO/FILE 1 foo.hbook
IF $IQUEST(1)<>0 THEN
*-- cannot open file
   ... do some cleanup
   EXITM 1
ENDIF
```

```
ON ERROR GOTO label
```

installs an error handler which tests the status code after each command and branches to the given label when a non-zero value is found. The error handler is local to each macro.

```
ON ERROR EXITM  [ expression ]
```

and

```
ON ERROR STOPM
```

are short-hand notations for an `ON ERROR GOTO` statement with a `EXITM` or `STOPM` statement, respectively, at the target label.

```
ON ERROR CONTINUE
```

nullifies the error handling. Execution continues with the next command independent of the status code. This is the initial setting when entering a macro.

```
OFF ERROR
```

and

```
ON ERROR
```

allow to temporarily suspend and afterwards reinstate the previously installed error handling. Note that the OFF/ON settings do not nest, for example

```
ON  ERROR EXITM
OFF ERROR        | behave like ON ERROR CONTINUE
ON  ERROR STOPM
OFF ERROR
ON  ERROR        | restore ON ERROR STOPM
ON  ERROR        | unchanged, i.e. not ON ERROR EXITM !
```

Another way of testing the status code of a command is to use the line separators "; &" and "; !" (see section 4.1.3). These operators take precedence over the ON ERROR setting.

```
cmd1 ;& cmd2 ; cmd3
```

is roughly equivalent to

```
OFF ERROR
cmd1
IF $IQUEST(1)=0 THEN
   cmd2
   ON ERROR
   cmd3
ENDIF
ON ERROR
```

except that the ON/OFF ERROR statements are virtual and do not overwrite the setting saved by a real OFF ERROR statement.

4.7 Motif mode

4.7.1 The KUIP/Motif Browser Interface

The KUIP/Motif Browser interface is a general tool to display and manipulate a tree structure of objects which are defined either by `kuip` itself (commands, files, macros, etc.) or by the application (e.g. in `paw++`: Zebra and Hbook files, Chains, etc.). The objects contained in the currently selected directory can be displayed in various forms: big icons, small icons, text only, etc. It is possible to perform actions on these objects or the directories it-selves by accessing pop-up menus directly attached to them: this behavior of the browser gives access to a "direct object manipulation" user interface by opposition to the usual "command mode interface". Adding your application specific objects into the browser is mainly done through the `kuip` "Command Definition File" (CDF): you will not get involved in any kind of Motif programming.

Description of the "Main Browser" Window

For any application based on KUIP/Motif one browser will be automatically created and displayed: it is called the **"Main Browser"**. Later on it is possible to "clone" this browser (by pressing the corresponding button at the bottom/right) when it is in a certain state. This will give to the user the possibility to have several instances of the browser window, and look at the same time to different kind of objects.
A "browser window" is composed of (Fig. 4.7):

- A menu bar with the menu entries "File" ①, "View" ②, "Options" ③, "Commands" ④ and "Help" ⑤.
- A two lines text/label area (❶ and ❷).
- The middle part of the browser is divided into two scroll-able windows: the "FileList" or **"Browsable window"** ❸ at the left and the "DirList" or **"Object window"** ❹ at the right.
- Two lines of information at the bottom (❺ et ❻), plus a "Clone" ❽ and a "Close" ❾ buttons.

Below follows a description of the middle (and main) part of the browser which is divided into two scroll-able windows on the left and right sides (Fig. 4.7):

- The left hand "FileList" or **"Browsable window"** ❸ shows the list of all the currently connected browsables. A "browsable" is simply a container of objects and is defined with the "`>Browse`" directive in the CDF. The browsables "Commands", "Files" and "Macros" are built-in inside `kuip` itself and are always displayed. Each application can add to this list its own definitions for any kind of browsables (e.g. in `paw++`: "Zebra", "Hbook", "Chains" and "PAWC") Some browsables can also be attached at run time by selecting the corresponding "Open" entry in the menu "File" (e.g. in `paw++`: ZEBRA/RZ files for access to histograms and Ntuples).
  Pressing the right mouse button in this window shows a pop-up menu with all the possible actions which have been defined for this browsable.
  Selecting one item (or browsable) in this window with the left mouse button executes by default the "List" action (first entry of the pop-up menu): it displays the content of the browsable in the right hand window ("DirList" or **"Object window"**)
  Note that the first entry of the pop-up menu of actions for one browsable is always "List" and that the last entry is always "Help" : it should give information concerning the selected browsable.
- The right hand "DirList" or **"Object window"** ❹ shows the content of the currently selected browsable for the selected path. E.g. when you select the browsable "Macro" (built-in inside `kuip`), you will get all the `kuip` macro files and sub-directories which are contained in the selected directory.

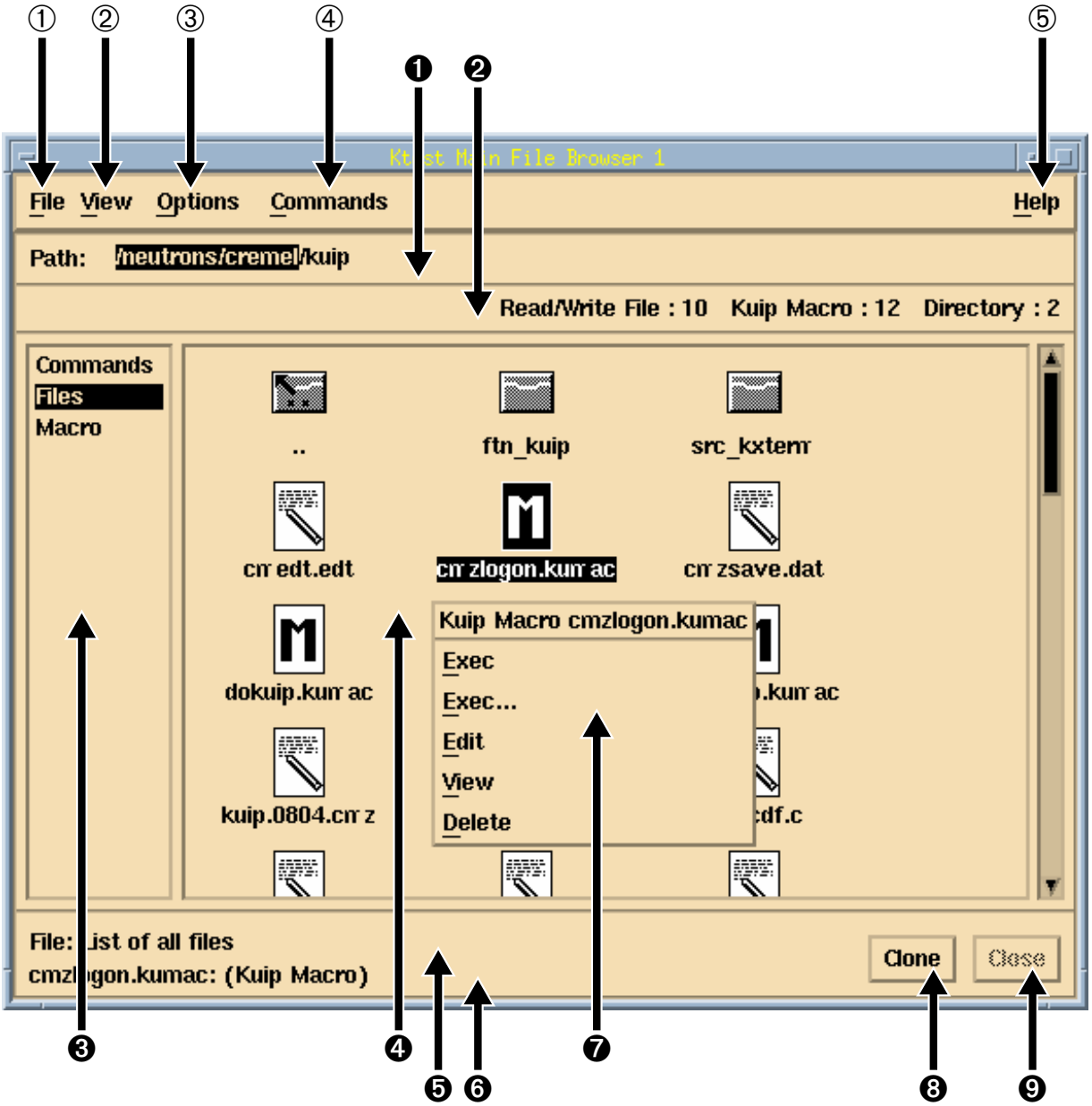


Figure 4.7: KUIP/Motif **"Main Browser"** Window

Objects are selected by clicking on them with the left mouse button. Pressing the right mouse button pops up a menu of possible operations depending on the object type ❼.
An item in a pop-up menu is selected by pointing at the corresponding line and releasing the right mouse button. Double clicking with the left mouse button is equivalent to selecting the first menu item.
Each menu item executes a command sequence where the name of the selected object is filled into the appropriate place. By default the command is executed immediately whenever possible. (The commands executed can be seen by selecting "Echo Commands" in the "Options" menu of the **"Executive Window"**.) In case some mandatory parameters are missing the corresponding "Command

Argument Panel" is displayed, and he remaining arguments have to be filled in. The command is executed then by pressing the "OK" or "Execute" button. (Note that if it is not the last one in the sequence of commands bound to the menu item, the application is blocked until the "OK" or "Cancel" button is pressed.)

All the application specific definitions for the entities accessible through the browser (objects, browsables and action menus) have to be made in the "Command Definition File" (CDF) with a very simple and easy-readable syntax.
The two lines text/label area at the top displays information about (Fig. 4.7):

- the current path (or directory) for the selected browsable ❶ (entry "Path:"). The directory can be changed by pointing at the tail of the wanted sub-path and clicking the left mouse button. Clicking a second time on the same path segment performs the directory change and updates the "DirList" window with the list of objects.
- the number of objects of all the different classes defined for the selected browsable in the current directory ❷.

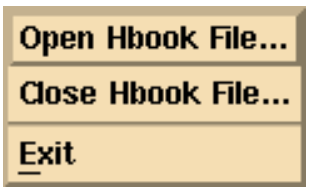
The two lines of information at the bottom are filled with (Fig. 4.7):

- a short description of the browsable which is currently selected ❺ (entry "File:"),
- a short description of the object which is selected in the "object window" for a given browsable ❻.

Below follows a description of the different Browser menus:

File

The **File** menu can be filled by the application with menu entries (buttons) which give access to the commands that can be used to connect or de-connect a new browsable at run time (e.g. in `paw++` the commands to open or close ZEBRA/RZ files).
These buttons/menu entries are automatically generated from the definition of the action menus for the browsables made in the CDF. For example, the **File** menu in the `paw++` **"Main Browser"** is shown below. The last entry of this menu is always "Exit", to exit from the application.



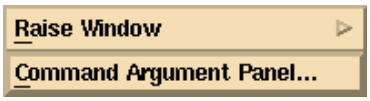
Open Hbook File...	Open one ZEBRA/RZ file.
Close Hbook File...	Close one ZEBRA/RZ file.
Exit	Exit from the application.

View

The **View** menu allows to change the way objects are displayed or selected.

Icons	display objects with normal size icons and names (default).
Small Icons	display objects with small icons and names.
No Icons	display objects without icons, but names and small titles.
Titles	display objects without icons, but long titles.
Select All	select all the objects.
Filter...	ask for a filter to be put on object names.

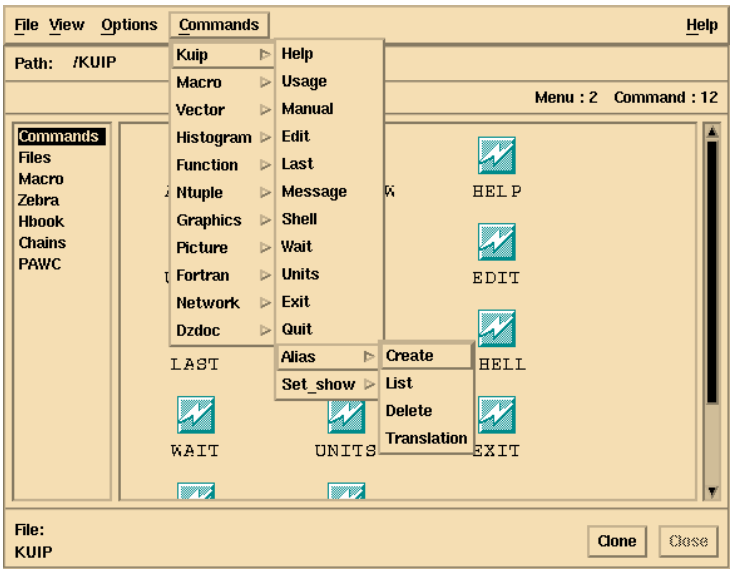
Options



Raise Window "cascade button" with the list of all opened windows. Selecting one of this window will pop-up the window on top of the others.

Command Argument Panel selecting this entry will prompt the user for a command name. If the command is valid then the corresponding "Command Argument Panel" with the list and description of all parameters will be displayed. If the command is ambiguous (e.g. command "list") the user will be proposed a list of all the possible commands. He can then select one and the corresponding "Command Argument Panel" will be displayed. If the command does not exist an error message is displayed.

Commands



This menu gives access to the complete tree of commands defined by `kuip` and the application in the form of a pull-down menu. When a terminal item (command) in this menu is selected then the corresponding "Command Argument Panel" is displayed. The functionality of this menu is quite similar to the browsable "Commands" (this is just a matter of taste whether the user prefer to access commands through this pull-down menu or through the "Commands" browser).

Help

On {**Appl.**}	Help specific to the application (has to be written in the CDF (Command Definition File)).
On {**Appl.**} **Resources**	Help specific to the application resources (has to be written in the CDF (Command Definition File)). Resources control the appearance and behavior of an application.
On Kuip Resources	List the X resources available to any KUIP/Motif based application.
On Browser	Help on the KUIP/Motif Browser interface (**"Main Browser"**).
On Panel	Help on the KUIP/Motif "**PANEL** interface".
On System Functions	List `kuip` all internal system functions currently available.

Browser Setting or Initialization

The following KUIP/Motif command can be used to set up the browser in a given state, without having to click with the mouse:

```
/MOTIF/BROWSER browsable [path]
```

- *browsable* is the name of the file (browsable) you want to open (corresponding item is selected in the list of browsables).
- *path* (optional) is the pathname to be used for this browsable.

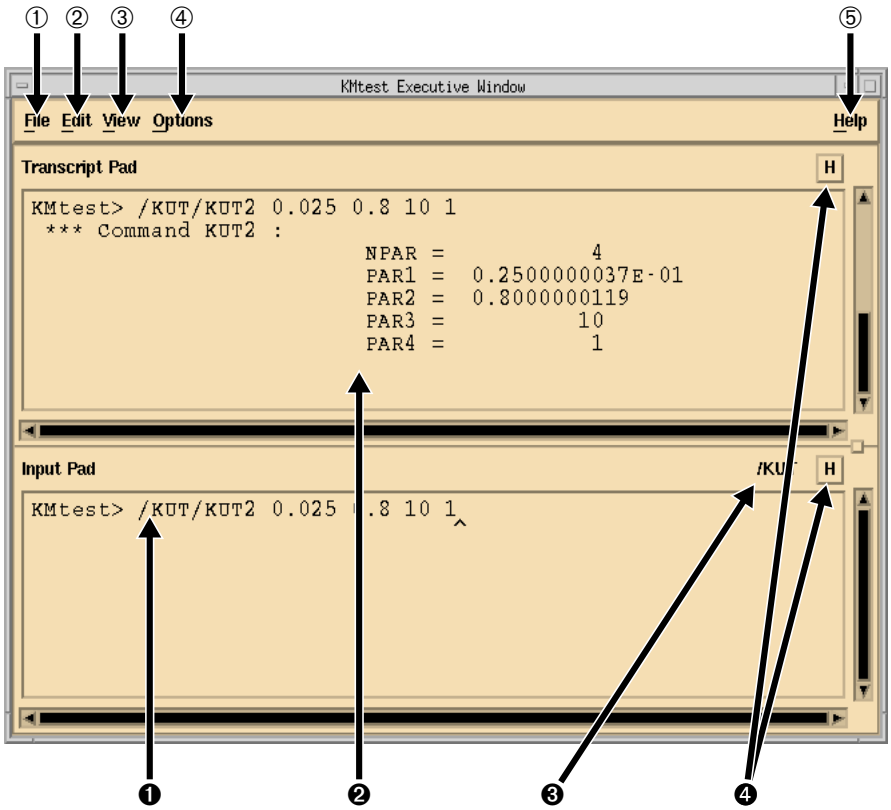
E.g. If you want to open the browser in the state displayed in Fig. 4.7, without having to click with the mouse, you can execute the KUIP command:

```
/MOTIF/BROWSER Files /neutrons/cremel/kuip
```

It is also possible, for the application programmer, to initialize the browser in a certain state when the application is starting. For that we provide the Motif user callable C routine *km_browser_set* which can be called just before entering the Motif main loop.

4.7.2 KXTERM: the `kuip` Terminal Emulator (or "Executive Window")

This terminal emulator combines features from Apollo DM pads (**Input Pad** and **Transcript Pad**, automatic file backup of **Transcript Pad**, string search in pads, etc.) and the Korn shell emacs-style command line editing and command line recall mechanism.



① Menu bar entry "File".
② Menu bar entry "Edit".
③ Menu bar entry "View".
④ Menu bar entry "Options".
⑤ Menu bar entry "Help".

❶ **Input Pad**
❷ **Transcript Pad**
❸ Current working directory indicator.
❹ Hold buttons.

Figure 4.8: KXTERM (KUIP/Motif **"Executive Window"**)

Description and Behavior

KXTERM (or what we call the **"Executive Window"** in a `kuip` based application) is composed of three main parts (Fig. 4.8):

- A "menu bar" with the menu entries "File" ①, "Edit" ②, "View"③, "Options"④, and "Help"⑤,.
- A **Transcript Pad** ❷ which contains any kind of output coming from `kuip` or from the application.
- An **Input Pad** ❶ which is an edit-able "scrolled window" where the user can type commands.

Commands are typed in the input pad behind the application prompt. Via the toggle buttons ❹ labeled "H" the **Input Pad** and/or **Transcript Pad** can be placed in hold mode. In hold mode one can paste or type a number of commands into the **Input Pad** and edit them without sending the commands to the application. Releasing the hold button will causes Kxterm to submit all lines, up to the line containing the cursor, to the application. To submit the lines below the cursor, just move the cursor down. In this way one can still edit the lines just before they are being submitted to the application.

Commands can be edited in the **Input Pad** using emacs-like key sequences (see section 4.7.2). The **Transcript Pad** shows the executed commands and command output. When in hold mode the **Transcript Pad** does not scroll to make the new text visible.

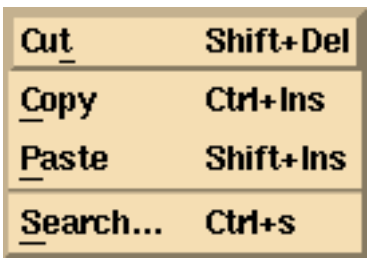
Every time the current directory is changed, the **Current working directory indicator** ❸ is updated. The current working directory is the one which is currently selected in the **"Main Browser"**.

Below follows a description of the different Kxterm menus. All Kxterm menus can be dynamically extended by the application.

File

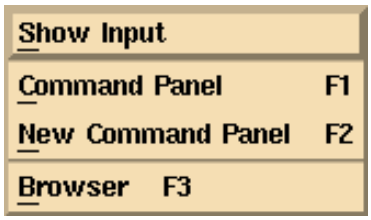
About Kxterm...	Displays version information about Kxterm.
About {**Appl.**} **...**	Displays version information about the application Kxterm is servicing.
Save Transcript	Write the contents of the **Transcript Pad** to the current file. If there is no current file a file selection box will appear.
Save Transcript As...	Write the contents of the **Transcript Pad** to a user-specified file.
Print...	Print the contents of the **Transcript Pad** (not yet implemented).
Kill {**Appl.**}	Send a SIGINT signal to the application to cause it to core dump. This is useful when the application is hanging or blocked. Use only in emergency situations.
Exit	Exit Kxterm and the application.

Edit



Cut	Remove the selected text. The selected text is written to the Cut & Paste buffer. Using the "Paste" function, it can be written to any X11 program. In the **Transcript Pad** "Cut" defaults to the "Copy" function.
Copy	Copy the selected text. The selected text is written to the Cut & Paste buffer. Using the "Paste" function, it can be written to any X11 program.
Paste	Insert text from the Cut & Paste buffer at the cursor location into the **Input Pad**.
Search...	Search for a text string in the **Transcript Pad**.

View



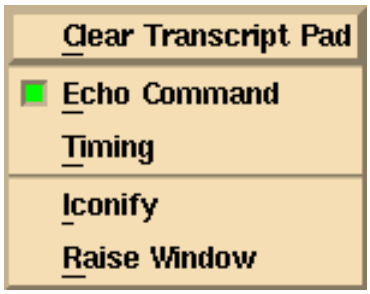
Show Input Show in a window all commands entered via the **Input Pad**.

Command Panel Gives access to the KUIP/Motif "**PANEL** interface" for a panel which has been predefined in a `kuip` macro file (see section 4.7.3).

New Command Panel Gives access to the KUIP/Motif "**PANEL** interface" for setting a new and empty panel to be filled interactively (see section 4.7.3).

Browser Display another instance of the browser.

Options



Clear Transcript Pad Clear all text off of the top of the **Transcript Pad**.

Echo Command Echo executed commands in **Transcript Pad**.

Timing Report command execution time (real and CPU time).

Iconify Iconify Kxterm and all windows of the application.

Raise Window Display a list of all windows connected to the application. The user can select the window he wants to pop-up.

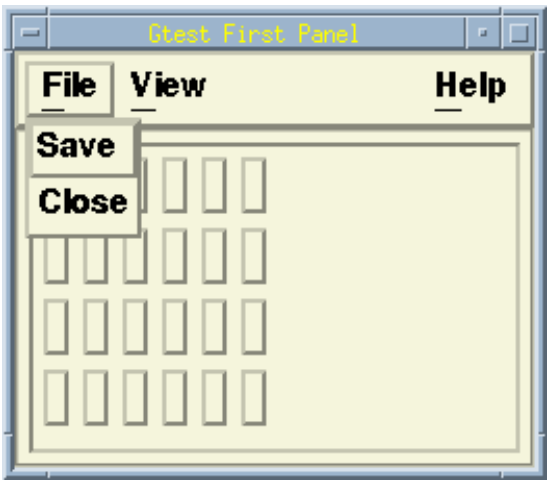
Help

On Kxterm	The help you are currently reading.
On Edit Keys	Help on the emacs-style edit key sequences.
On {Appl.}	Help specific to the application (has to be written in the CDF).
On {Appl.} Resources	Help specific to the application resources (has to be written in the CDF). Resources control the appearance and behavior of an application.
On Kuip Resources	List the X resources available to any KUIP/Motif based application.
On Browser	Help on the KUIP/Motif Browser interface (**"Main Browser"**).
On Panel	Help on the KUIP/Motif "**PANEL** interface".
On System Functions	List all `kuip` internal system functions currently available.

Edit Key Sequences

Please note that "C-b" means holding down the Control key and pressing the "b"-key. "M-" stands for the Meta or Alt key.

```
C-b:              backward character
M-b:              backward word
Shift M-b:        backward word, extend selection
M-[:              backward paragraph
Shift M-[:        backward paragraph, extend selection
M-<:              beginning of file
C-a:              beginning of line
Shift C-a:        beginning of line, extend selection
C-osfInsert:      copy to clipboard
Shift osfDelete:  cut to clipboard
Shift osfInsert:  paste from clipboard
Alt->:            end of file
M->:              end of file
C-e:              end of line
Shift C-e:        end of line, extend selection
C-f:              forward character
M-]:              forward paragraph
Shift M-]:        forward paragraph, extend selection
C-M-f:            forward word
C-d:              kill next character
M-BS:             kill previous word
C-w:              kill region
C-y:              yank back last thing killed
C-k:              kill to end of line
C-u:              kill line
```




```
NEWPANEL 4 6 'First panel' -
         250 200 500 600
```

This KUIP/Motif command creates an empty panel with 4 rows and 6 columns of buttons. The title of this panel will be set to "Gtest First panel" ("Gtest" is the application class-name). The panel size in pixels is 250 (width) x 200 (height), and the panel position (in pixels) is 500 (along X axis), 600 (along Y axis).

Figure 4.9: New Panel of Commands

```
M-DEL:          kill to start of line
C-o:            newline and backup
C-j:            newline and indent
C-n:            get next command, in hold mode: next line
C-osfLeft:      page left
C-osfRight:     page right
C-p:            get previous command, in hold mode: previous line
C-g:            process cancel
C-l:            redraw display
C-osfDown:      next page
C-osfUp:        previous page
C-SPC:          set mark here
C-c:            send kill signal to application
C-h:            toggle hold button of pad containing input focus
F8:             re-execute last executed command
Shift F8:       put last executed command in input pad
Shift-TAB:      change input focus
```

4.7.3 User Definable Panels of Commands ("PANEL interface")

KUIP/Motif includes a built-in "**PANEL** interface" that allows to define command sequences which are executed when the corresponding button in the panel is pressed.

As you will see, this "**PANEL** interface" is quite powerful compared to the "STYLE GP" which was available in the basic `kuip` for graphical screens. In particular it is possible to set-up panels with graphical keys (icons) representation.

New Panel

It is possible to fill a new and empty panel interactively (see section 4.7.3) giving a label to each button.

In the top menu bar 3 pull-down menus ('File", "View" and "Help") are available. The pull-down menu "File", whose contents is displayed, contains the 2 items "Save" (to save the actual panel configuration after editing) and "Close" (to close the panel and erase it from the screen). The "View" menu contains

various options for displaying the same panel in different ways (see section 4.7.3), and the "Help" menu contains various items to help the user concerning this panel interface.

This new panel definition can also be done with the command `PANEL` using the sequence

```
PANEL 0
PANEL 4.06 ' '
PANEL 0 D 'This is my first panel' 250x200+500+600
```

You can get automatically access to the command "NEWPANEL" (and its corresponding "Command Argument Panel") by selecting the menu item "New Command Panel" in the "View" menu of the **"Executive Window"** (KXTERM, Fig. 4.7.2).

Predefined Panel of Commands

The command "PANEL" for a key (or button) definition has to be used if you want to describe your panel in a `kuip` macro file in order to keep trace of the panel definition, and be able to retrieve it later on. You can predefine as many panels as you want, and you can easily access them by selecting the menu item "Command Panel" in the "View" menu of the **"Executive Window"** (section 4.7.2).

You have to describe in the `kuip` macro file(s) each button individually. You can also request the macro(s) execution in your "`kuip` logon" file so that the panel(s) will be automatically displayed at the beginning of the session.

The general syntax of the KUIP/Motif command "PANEL" for a key definition is:

```
panel x.y command [label] [pixmap]
```

- *x.y* is the key position (column and row number),
- *command* is the complete command (or list of commands) to be executed when the corresponding button is pressed,
- *label* (optional) is an alias-name for this command. If specified, this alias-name is used for the button label (when the appropriate "View" option is selected) instead of the complete command (which is generally too long for a "user-friendly" button label).
- *pixmap* (optional) has to be specified for graphical keys (fully described in the next section 4.7.3).

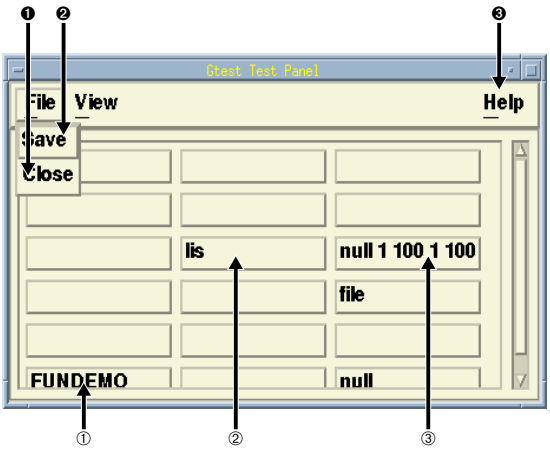
An example of a panel definition is given in figure 4.10.

Panel with Graphical keys (Icons) and "View" Selection

As seen in the previous section, the general syntax of the KUIP/Motif command "PANEL" for a key definition allows the user to define graphical keys (or buttons) where pixmaps are used instead of alphanumerical labels:

```
panel x.y command [label] [pixmap]
```

The last parameter *pixmap* (optional) is the pixmap to be used for representing the key (button) graphically. If it is specified the graphical representation is displayed by default. It is anyway always possible



kuip Macro for panel definition

```
*
* MOTIF_PANEL panel_test.kumac
*
panel 0
panel 3.02 'list'
panel 3.03 'null 1 100 1 100'
panel 4.03 'file'
panel 6.01 'FUNDEMO'
panel 6.03 'null'
panel 0 d 'Test Panel' 450x250+600+600
```

❶ Close button (to close panel)
❷ Save button (to save panel into a `kuip` macro file)
❸ Access to various "helps" on the "**PANEL** interface"

① ② ③ User defined buttons

Figure 4.10: Predefined Panel of Commands

at run time to ask for an alpha-numerical representation by selecting the appropriate entry in the "View" menu of the panel.

The application can define its own icons (pixmaps). This can be done by the application programmer in the CDF (following the KUIP/Motif directive "`>Icon_bitmaps`") or by the user himself (at run-time and for his own user-defined panels of commands) using the KUIP/Motif command:

```
/MOTIF/ICON pixmap filename
```

- *pixmap* is the name given to the icon bitmap and used in the "panel" command for a graphical key definition.
- *filename* is the name of the file where the icon bitmap data is stored.

N.B. All application-defined pixmaps (in the CDF) are available to the user in the "panel" command, without having to use this "/MOTIF/ICON" command. This command is only useful when you want to make new icons known by the application (and the command "panel").

To create a new icon bitmap (or pixmap) one can use the X11 standard bitmap editor "bitmap". E.g., to get a 20×20 pixel icon called "`m1`", one can type: `bitmap m1.bm 20x20`. The output file `m1.bm` containing "`#define m1_width 20` ..." has to be referred in the command "/MOTIF/ICON" (with the correct path for the filename), e.g. `/MOTIF/ICON m1 /user/.../.../m1.bm`

The following `kuip` macro is a general example for a panel definition with graphical keys.

****

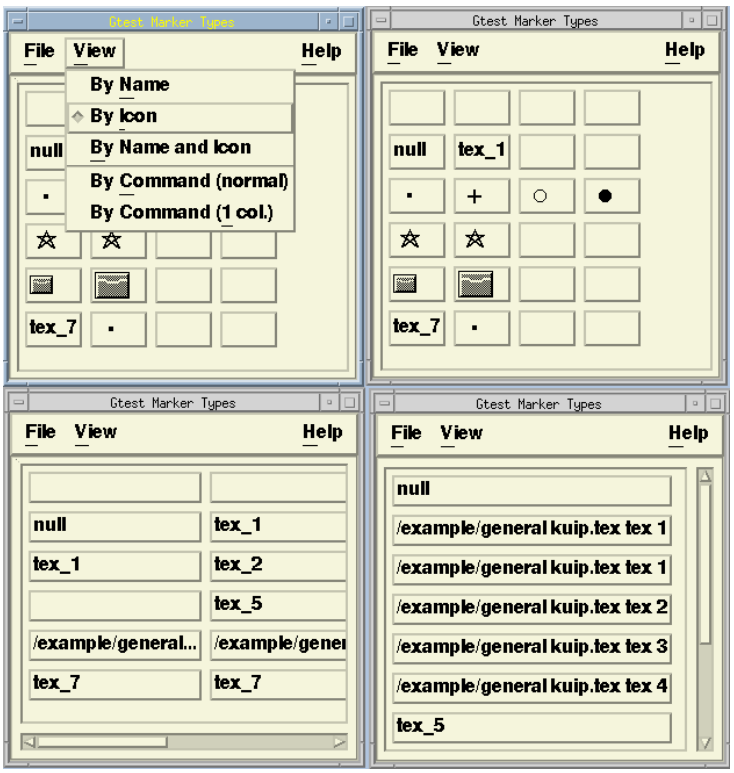
```
*                                                    *
*             *** panel.kumac ***                    *
*                                                    *
* General example for a panel with icons definition *
*                                                    *
*                                                    *
******************************************************
*
* Icon bitmaps
*
/motif/icon m1 mk1.bm
/motif/icon m2 mk2.bm
/motif/icon m3 mk3.bm
/motif/icon m4 mk4.bm
/motif/icon m5 mk5.bm
*
* Panel keys definition
* N.B. General syntax:
*      panel r.c command [label] [pixmap]
*      label --> command alias
*                (written in the panel and executed for <Button press>).
*                if <label> (optional) is defined then:
*                   /KUIP/ALIAS/CREATE <label> <command>
*                is automatically generated.
*                if <label> is not defined then "command" is used
*                for button label.
*
panel 0
panel 2.01 null
panel 2.02 tex_1
panel 3.01 '/example/general kuip.tex tex 1' 'tex_1' m1
panel 3.02 '/example/general kuip.tex tex 2' 'tex_2' m2
panel 3.03 '/example/general kuip.tex tex 3' . m3
panel 3.04 '/example/general kuip.tex tex 4' . m4
panel 4.01 ' ' . m5
panel 4.02 'tex_5' . m5
panel 5.01 '/example/general kuip.tex tex 6' . sm_menu
panel 5.02 '/example/general kuip.tex tex 6' . big_menu
panel 6.01 '/example/general kuip.tex tex 7' 'tex_7'
panel 6.02 '/example/general kuip.tex tex 7' 'tex_7' m1
panel 0 d 'Marker Types' 300x300+500+500
```

Figure 4.11 shows the panel defined in the macro listed above with different “View(ing)” options. In the first window (top/right) the “View” menu is displayed, with the different possibilities which are offered to the user to see the same panel in different ways.

Panel Edition and Saving

All the panels (new or predefined) can be edited interactively. Clicking with the left mouse button on a panel button removes its definition. Clicking with the right mouse button on an empty panel button the user will be asked to give a definition to this button (figure 4.12).

The PANEL commands needed to recreate a panel can be automatically saved into a macro file by pressing the ”Save” button ❷ (Fig. 4.10). The panel configuration with its current size and position (which can be modified interactively) is kept into the macro. Panels can be reloaded either by executing the com-



‘‘By Name’’ (bottom left): The panel is displayed with alphanumeric labels. If the alias-name “label” is specified in the “panel” command it is used for the button label, otherwise the complete command is displayed.

‘‘By Icon’’ (top right): The panel is displayed with graphical labels (icons), if “pixmap” is specified in the “panel” command. Otherwise “label” or the complete command are used instead (no graphical representation). This “view” setting is the default one (the setting can be changed interactively at run time, and the default setting can be changed with the appropriate resource in the “.Xdefaults” for each user individually).

‘‘By Name and Icon’’: The panel is displayed with both alphanumeric and graphical (if any) labels. (Not yet implemented ...).

‘‘By Command (normal)’’ The panel is displayed with the complete command names. The arrangement of the buttons stay the same (which might not be very convenient ... See below).

‘‘By Command (1 col.) (bottom right): The panel is displayed with the complete command names BUT the arrangement of the buttons is modified: all buttons are displayed on one column, and “blank” buttons are suppressed (this can save a lot of space, and is more user-friendly, for this kind of viewing option).

Figure 4.11: Panel “View” Selection

mand ’PANEL 0 D’ or by pressing the ”Command Panel” button in the ”View” menu of the **“Executive Window”** and entering the corresponding macro file name.

Some characters in the panel keys/buttons have a special meaning:

– The dollar sign inside a key is replaced by additional keyboard input. For example:

```
'V/PRINT V($)'    | entering 11:20 will execute V/PRINT V(11:20)
```

– Keys ending with a double minus sign make an additional request of keyboard input. For example:

```
V/PRINT V--'      | entering AB will execute V/PRINT VAB
```

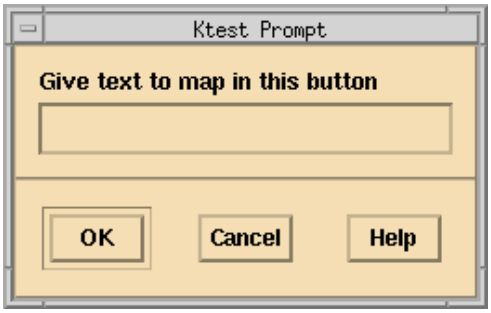
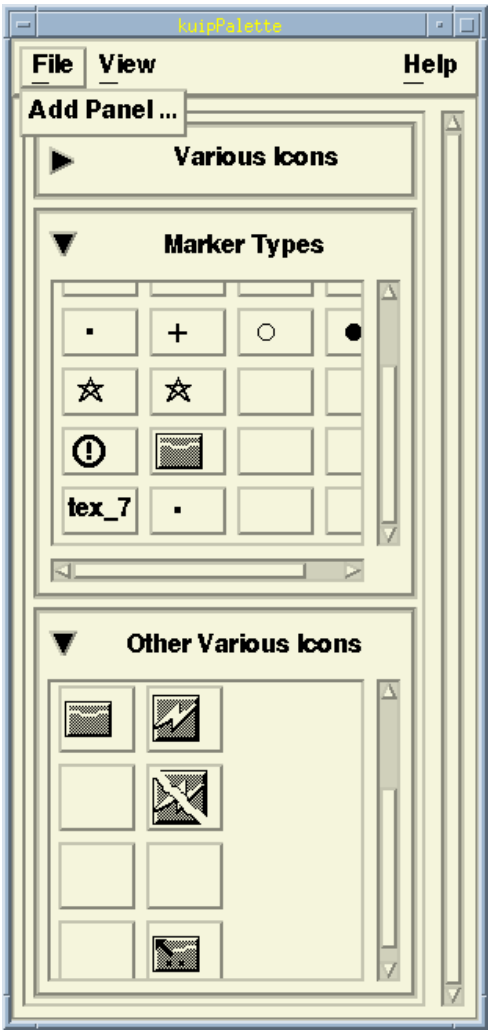


Figure 4.12: Interactive panel button definition



User-defined KUIP/Motif “palette” with 3 panels :

“Various Icons” : this panel is not displayed (arrow turned left to right) at the moment. One would just have to press the arrow button to make it visible ...

“Marker Types” : this user-defined panel is visible (arrow turned top to down). One can turned it off by pressing the arrow button.

”Other Various Icons” : this user-defined panel is also visible.

Figure 4.13: Multi˙panel (or Palette)

“Multi˙panel” or Palette of panels Definition

It may be nice or more user-friendly to group a certain number of panels (related to similar actions or objects to be manipulated) in a so-called “palette” of panels. This is possible with the KUIP/Motif command “MULTI˙PANEL” which opens such a widget. [5]

```
/MOTIF/MULTI_PANEL [title] [geometry]
```

E.g. `MULTI_PANEL 'My Palette' '200x100+0+0'` will display a ”multi˙panel” widget with title “My Palette” and geometry “200x100+0+0” (Position=0,0 in X and Y, width=200, height=100). When this command is executed all panel definitions and executions will go into this ”multi˙panel” (or palette) widget. This can be done simply by executing KUIP macro(s) containing your panel definition(s), or by selecting the ”Add button” entry in the menu “File” available in the ”multi˙panel” widget. To terminate a ”multi-panel” setting one just have to type: `MULTI_PANEL end`. This means that the following panel definitions and executions will be displayed as individual panels and will not go into this ”palette” anymore, unless another palette is opened (by executing again the command “MULTI˙PANEL”). Then the panels will go into that new palette.

The following sequence of commands (which can be put inside a macro) can be used to set up a palette:

```
MULTI_PANEL
EXEC PANEL1.KUMAC
EXEC PANEL2.KUMAC
EXEC PANEL3.KUMAC
MULTI_PANEL end
```

N.B. `panel1.kumac`, `panel2.kumac`, and `panel3.kumac` are KUIP macro files with “usual” panel setting and definition.

Figure 4.13 shows an example of a user-defined palette (with some predefined panels). The “arrow buttons” can be pressed either to reduce the panel to a label containing the panel title (arrow button is then turned left to right) or to display it (arrow button turned up to down). One can see that the KUIP/Motif “palette” is a good way to have many panels defined and save space on the screen.

4.7.4 KUIP/Motif X-Windows Resources

X-Windows resources control the appearance and behavior of an application. Users who are not pleased with the default values proposed for any resources that can affect their KUIP/Motif based application, can override them by specifying their own values in the standard X11 way : i.e. by editing their private “.Xdefaults” file or the system wide “/usr/lib/X11/app-defaults/¡appl˙class¿”.

Each new resource has to be specified on a separate line. The syntax for editing one specific resource is always the following:

¡appl.˙class¿*¡resource˙name¿: ¡resource˙value¿

where:

– “appl.˙class” has to be replaced by the real application class name (e.g. “Paw++” for `paw++`) which is the input parameter of the routine `KUWHAM`.

[5] For those who are familiar with the “UIMX” User Interface Management System, this is an emulation of the “Palette” widget which is built-in inside this program.

– "resource˙value" is the value to be given to the corresponding "resource˙name". It can be an integer, a boolean value, a color, a font, or any kind of predefined syntax (e.g. for geometry).

The following is a (non exhaustive) list of the most important or frequently used X-Windows resources for a KUIP/Motif based application. The default values provided by KUIP/Motif (if any) are put inside "[]".

– Background and foreground color for all windows (except KXTERM):

  ...*background: ...

  ...*foreground: ...

– Geometry ([width]x[height]+[xpos]+[ypos]) of the **"Executive Window"** (KXTERM):

  ...*kxtermGeometry: ... [650x450+0+0]

– Geometry of the Browser(s):

  ...*kuipBrowser˙shell.geometry: ... [-0+0] (1) or [+0+485] (2)

  (1) without any graphics window - (2) with graphics window(s) managed by HIGZ.

– Geometry of the Graphics Window(s) (if any):

  ...*kuipGraphics˙shell.geometry: ... [600x600-0+0]

– Character font for menus, buttons and dialog area:

  ...*fontList: ... [-adobe-helvetica-bold-r-normal–12-120-75-75-p-70-iso8859-1]

– Character font for the **Input Pad** and **Transcript Pad** (KXTERM):

  ...*kxtermFont: ... [*-courier-medium-r-normal*-120-*]

– Character font for the "HELP" windows:

  ...*helpFont: ... [*-courier-bold-r-normal*-120-*]

– Character font for all "Text" widgets:

  ...*XmText*fontList: ...

  ...*XmTextField*fontList: ...

– Character font for the icon labels in the browser(s) **"Object window"**:

  ...*dirlist*fontList: ...

– Background and foreground colors for the **"Object window"** in browser(s):

  ...*dirlist*background: ...

  ...*dirlist*foreground: ...

– Background and foreground colors for the icons associated to the object class "objclass":

  ...*dirlist*¡objclass¿*iconBackground: ... [white]

  ...*dirlist*¡objclass¿*iconForeground: ... [black]

– Background and foreground colors for the icon-labels associated to the object class "objclass":

  ...*dirlist*¡objclass¿*iconLabelBackground: ... [white]

  ...*dirlist*¡objclass¿*iconLabelForeground: ... [black]

– Possibility to turn on/off the zooming effect when traversing directories structures inside the browser(s):

  ...*zoomEffect: ... [on]

- Speed of the zooming effect in the browser(s) when turned on:
  ...*zoomSpeed: ... [10]
- Double click interval in milliseconds (time span within which 2 button clicks must occur to be considered as a double click rather than two single clicks):
  ...*doubleClickInterval: ... [250]
- Background and foreground colors for the **"Browsable window"** in browser(s):
  ...*fileList*background: ...
  ...*fileList*foreground: ...
- Focus policy:
  ...*keyboardFocusPolicy: ...
  If "explicit" focus is set by the mouse or a keyboard command. If "pointer" focus is determined by the mouse pointer position.

The appearance and behavior of the **"Executive Window"** are managed by "KXTERM" whose classname is "KXterm". It means that, for instance, to change the background and foreground color of the **"Executive Window"**, one has to override the following resources:

KXterm*background: ...

KXterm*foreground: ...

To this list of resources one can add all the resources which can affect any Motif widgets which are used by KUIP/Motif.

Concerning the appareance of the icons built-in inside KUIP/Motif (browsers for "Commands", "Files" and "Macro"), the classes of objects which are currently predefined are:

```
Cmd        -- Command
InvCmd     -- Deactivated command
Menu       -- Menu tree
MacFile    -- Macro File
RwFile     -- Read-write file
RoFile     -- Readonly file
NoFile     -- No access file
ExFile     -- Executable file
DirFile    -- Directory
DirUpFile  -- Up directory (..)
```

4.8 Nitty-Gritty

4.8.1 System dependencies

`kuip` tries to provide as far as possible a homogenous environment across different operating systems and hardware platforms. Here we want to summarize the remaining system-dependencies. To a large extend the comments made on Unix apply also to the MS-DOS and Windows/NT implementations.

SHELL command

The `SHELL` command allows to pass a command line to the underlying operating system for execution. If used without arguments the `SHELL` command suspends the application program and allows to enter OS commands interactively. When leaving the subprocess, either with the command `return` or `exit` depending on the system, the application resumes execution.

Unix The command HOST_SHELL defines the shell to be invoked. The start-up value is taken from the environment variable SHELL or set to an appropriate default such as /bin/sh. On some Unix implementations the SHELL command can fail if there is not enough free swap space to duplicate the current process.

VMS The SHELL command spawns a subprocess with a DCL command processor. This is notoriously slow and there is no way to combine several DCL commands into one SHELL command.

VM/CMS "SHELL cmd" first tries to find the file "CMD EXEC *" and execute it as a REXX script. Otherwise the command is passed as-is which will either run "CMD MODULE *" or execute the genuine CMS command CMD. There are some restrictions on the kind of modules that can be executed in CMS subset mode. CP commands have to be prefixed, e.g. "SHELL CP Q TIME".

EDIT command

The EDIT command allows to edit a file without leaving the application program. The command HOST_EDITOR defines the editor to be invoked. The start-up value is taken from the environment variables KUIPEDITOR, EDITOR, or set to a system dependent default.

HOST_EDITOR sets the shell command (sans filename) for starting the editor. Some values have a system dependent special meaning.

Unix The default editor is vi. The shell command containing a "&" does not necessarily mean that the editor will run as a background process (see section 4.8.2).
On Apollo/DomainOS "DM" uses the Display Manager editor. This is the default if the application program is started from a DM pad.

VMS The special names EDT and TPU use the callable interface to these two editors. The startup time is much less than, for example EDIT/TPU which spawns a subprocess. However, there is a problem with the callable EDT. If any error condition occurs (invalid filename etc.) the callable EDT will be unusable for the rest of the session.

VM/CMS There is only one possible HOST_EDITOR: XEDIT. For editing large files the virtual machine's size must be dimensioned that the application program and XEDIT fit into the available memory at the same time.

Exception handling

kuip installs a signal handler in order to catch exceptions and return to the command input prompt. The command "BREAK OFF" disables the signal handler, i.e. the program aborts in case of an exceptions. For some systems "BREAK ON" allows to request a traceback of where the exception has happened.

There are two major types of exceptions caught by the signal handler. Program exceptions indicate either a bug in the application program (or kuip) or insufficient protection against invalid input:

Floating point exceptions are caused by divide by zero, floating point overflow, square root of negative numbers etc. Floating point underflows are usually silently ignored and the result is treated as being zero.

Segmentation violation indicates an attempt to read or write a memory location outside the address space reserved by the process, e.g. if an array index is out of bounds. In C code it is most often caused by dereferencing a `NULL` pointer which is prohibited on many systems.

Bus error is usually caused by an unaligned access. Most RISC processors have strict requirements for properly aligned data.

Illegal instruction can mean that the program tries to executed data as code, for example if the return address on the stack has been overwritten.

Don't be surprised if the program shows irregular behaviour after an exception!

The second type of exceptions handled by the `kuip` signal handler are user breaks. Hitting the break key (usually `Ctrl`-C) aborts a running command and returns to the input prompt. Using `Ctrl`-C is potentially unsafe unless the application is properly coded to block keyboard interrupts in critical sections. Otherwise the interrupt can happen at an inconvinient moment which leaves the program's data structures in an inconsistent state. The signal handler prompts the user after three consecutive keyboard interrupts to allow exiting from a run-away process.

Unix The actual break key can be changed with the Unix command `stty`. The default setup usually is "`stty intr ^C`". Unix provides a second kind of keyboard interrupt which is intentionally **not** caught by the `kuip` signal handler to allow killing run-away processes. A convenient setting is "`stty quit '\\'`"
User break interception does not work for Windows/NT. Tell Microsoft that signal handlers are pretty useless if they are not allowed to use `printf` and `longjmp`.

VMS The user break key is `Ctrl`-C. `Ctrl`-Y is treated like `Ctrl`-C, i.e. it does not bring up the DCL prompt.

VM/CMS There is no user break for VM/CMS. To abort a run-away session use

```
#CP EXT
HX
```

4.8.2 The edit server

By default editing from within a `kuip` application is synchronous, i.e. the application is suspended until the editor terminates. On a workstation this is an inconvenient restriction because the editor can run in a separate window while the application continues to accept commands.

Although not an issue for the `KUIP/EDIT` command itself there are applications (notably `cmz`) which have to process the file content after it has been edited. Therefore the editor cannot be simply started as a background process.

To take care of this problem `kuip` provides a facility called the "edit server". Instead of calling the editor directly, `kuip` starts the editor server as a background process which leaves the application program ready to accept more commands. The server invokes the editor and waits for it. When the editor terminates the server informs the application program about the file which is ready. `kuip` can then call the application routine for processing the edited file.

The processing routine cannot be called at the very instant the file is ready. `kuip` waits until the user hits the `RETURN`-key to execute the next command. The file is then checked in *before* the command just entered is executed.

As a protection especially for users working alternately on a terminal or on a workstation `kuip` does not try asynchronous editing if one of the following conditions is missing:

- The edit server module `kuesvr` must be found in the search path.
- The editor command set by `HOST_EDITOR` must end with an ampersand ("`&`").
- The environment variable `DISPLAY` must be set.

Note that the editor command must create its own window, possibly by wrapping the editor into a terminal window. For convenience "`HOST_EDITOR 'vi &'`" is interpreted automatically as "`xterm -e cmd &`".

The edit server cannot be used for the Apollo DM editor. Some Unix windowing editors tend to fork themselves as a detached process by default. For example the `jot` editor found on Silicon Graphics systems requires a special option "`-noFork`". Otherwise the edit server and the application think that the editor has already terminated leaving the file unchanged.

In the KUIP/Motif interface it is essential to use the edit server mechanism. Otherwise invoking the editor from a pop-up menu freezes the screen when the right-hand mouse button is pressed before the subprocess terminates[6]. The screen can only be unlocked by logging in remotely and killing the application program.

For asynchronous editing on VMS either the Motif version of TPU must be used or the hosteditor command must create its own terminal window, e.g.

```
HOST_EDITOR TPU/DISPLAY=MOTIF
HOST_EDITOR 'CREATE/TERM/WAIT EDT'
```

4.8.3 Implementation details

Command search order

With the various possibilities of changing the interpretation of a command line it is sometimes important to know the exact order in which the different mechanisms are applied:

1. If the input line contains a semicolon line separator (section ??), split off the front part and deal with the rest later. In case the line separator is "`;&`" or "`;!`" the execution of the remaining line depends on the status code of the first command.
2. If executing a macro script, substitute all variables by their values.
3. If the first token is a command alias (section ??), substitute it by its value. If the replacement contains a semicolon line separator, start again at step 1. In order to protect against recursive aliases stop if a reasonable upper limit on the number of iterations is exceeded.
4. Unless the command name belongs to the `KUIP/ALIAS` menu, substitute argument aliases. Argument aliases can occur in the command name position but they may not contain semicolon line separators.
5. Substitute system functions (section ??).
6. If executing a macro with "`TRACE ON`", show the present command line. If "`TRACE ON WAIT`" prompt for further actions:
   - execute command
   - skip execution of this command
   - quit execution of macro script
   - continue macro execution without further prompting

[6] Can somebody elucidate this problem or knows a workaround? It seems that the application does not receive the button-release event and therefore the Motif pop-up menu never releases the pointer grab???

7 Separator first token (command name `cmd`) from the rest of the line (argument list).

8 Unless executing a macro, if "`DEFAULT -AutoReverse`" (section ??) is active and `cmd.kumac` is found in the macro search path, transform the command name into `EXEC`. The command token itself has to be put back in the front of the other argument. If the command token contains a "`#`" character we had to separate the front part before searching for the `.kumac` file.

9 Match `cmd` as abbreviation against the command tree:

- If `cmd` begins with a slash, start at the top menu.
- Otherwise start at the `SET/ROOT` menu; if there is no match and the current root is not the top menu itself, start again at the top menu.

10 Unless executing a macro, if "`DEFAULT -Auto`" is active and `cmd` is either not a command or ambiguous, try again procedure of step 8.

11 If a `SET/COMMAND` template is defined and `cmd` is unknown as command name, i.e. not just ambiguous, apply the template replacement and go back to step 1. `SET/COMMAND` must be disabled temporarily to avoid an infinite recursion in case the template itself is an invalid command.

12 If `cmd` is ambiguous, show the list of possible solutions and exit.

13 If `cmd` is not a valid command name, print error message and exit.

14 Otherwise tokenize the argument list and call the action routine for the command.

Name spaces

There is an admittedly confusing difference in the characters allowed to form the various `kuip` identifiers which we summarize here:

Alias names allow letters, digits, "`_`", "`@`", "`-`", "`$`".

Macro variable names allow letters, digits, "`_`". The first character may not be a digit.

System function names allow letters, digits, "`_`". The first character may not be a digit. Uppercase and lowercase letters are distinct when the name is looked up as environment variable.

Vector names allow letters, digits, "`_`", "`?`". The first character may not be a digit. Names starting with "`?`" are reserved.

Although not in the hands of the application user but only the application writer:

Command and menu names allow letters, digits, and "`_`".

Parameter names allow letters, digits, and "`_`". The first character may not be a digit.

Chapter 5: Vectors

Vectors are named arrays of numerical data, memory resident, which can be created during a session, loaded from HBOOK objects, typed in by hand, read from disk files, operated upon using the full functionality of SIGMA or COMIS. Vectors can be used to produce graphics output, and, if necessary, stored away on disk files for further usage. Vectors provide a very convenient mechanism to transport numerical information between different PAW objects, and to manipulate mathematically their content. At the end of an interactive session, they are lost, unless previously saved onto disk files.

Vectors can have up to 3 dimensions (in fact they are "arrays", called "vectors" for historical reasons). They can be handled in PAW either interactively, by using `VECTOR/...` commands, or by means of KUIP routines which return the addresses of a given vector.

Simple arithmetic operations can be applied to vectors. In addition, as SIGMA is part of PAW, powerful array manipulation operations are available, through the SIGMA, `$SIGMA` and `APPLICATION SIGMA` commands (see section 6.1 on page 239).

An "invisible" vector named `?`, mono-dimensional and of length 100, is always present. It is used for communication between arrays in the user code (for instance in a COMIS[1] routine) and KUIP vectors, being equivalenced with the real array `VECTOR(100)` in the labelled common block `/KCWORK/`.

5.1 Vector creation and filling

A vector is **created** either by the **PAW command** `VECTOR/CREATE`, by the **SIGMA function** `ARRAY`. or by the **COMIS statement** `VECTOR`.

Example of vector creation

```
VECTOR/CREATE  X(100)      will create a 100-components vector, values = 0.
SIGMA X=ARRAY(100,1#100)   will create a 100-components vector and assign
                            to each element the values 1,2,...100
 VECTOR X(100)             in a COMIS routine creates a 100-components vector
                           and initialises each element to zero
```

Once the vector is created, it can be manipulated using the following PAW commands:

`VECTOR/INPUT vlist`	Input from the terminal values into the vector elements specified by the list `vlist`.
`VECTOR/READ vlist`	Values can be **read in** from a file into the vector elements specified by the list vlist.
`VECTOR/COPY v1 v2`	Values in `v1` are copied into `v2`.
`VECTOR/WRITE vlist`	Values in the vector elements specified by the list vlist can be **saved** on a file.
`VECTOR/PRINT vlist`	Values of the vector elements specified in `vlist` will be **printed** on the terminal.
`VECTOR/LIST`	A **list** of existing vectors and their **characteristics** is printed on the terminal.
`VECTOR/DELETE`	Allows global or selective deletion of vectors.

5.2 Vector addressing

Indexing of vectors is possible[1].

Example of vector indices

Vec	for all elements
Vec(13)	for element 13
Vec(12:)	for elements 12 up to the last
Vec(:10)	for elements 1 to 10
Vec(5:8)	for elements 5 to 8

Sub-elements of the two-dimensional vector `Vec(3,100)` (3 columns by 100 rows) may be addressed by:

Using two-dimensional vectors

Vec(2,5:8)	for elements 5 to 8 in column 2
Vec(2:3,5:8)	for elements 5 to 8 columns 2 to 3
Vec(2,5)	for element 5 in column 2
Vec(:,3)	for all elements in row 3
Vec(2)	for all elements in the 2-nd column (SPECIAL CASE)

5.3 Vector arithmetic operations

A number of basic vector arithmetic operations is available:

```
VBIAS v1 bias v2        v2(I) = bias + v1(I)
VSCALE v1 scale v2      v2(I) = scale * v1(I)
VADD v1 v2 v3           v3(I) = v1(I) + v2(I)
VMULTI v1 v2 v3         v3(I) = v1(I) * v2(I)
VSUBTR v1 v2 v3         v3(I) = v1(I) - v2(I)
VDIVID v1 v2 v3         v3(I) = v1(I) / v2(I), if v2(I)<>0
```

In all operations only the minimum vector length is considered, i.e. an operation between a vector `A` of dimension 10 and a vector `B` of dimension 5 will involve the first 5 elements for both vectors. If the destination vector does not exist, it is created with the same length as specified in the source vector.

5.4 Vector arithmetic operations using SIGMA

A more complete and convenient mechanism for the mathematical manipulation of entire vectors is provided by SIGMA. SIGMA-generated arrays are stored as PAW vectors and therefore are accessible to PAW commands, and PAW vectors are accessible to SIGMA. The facilities available via SIGMA are described in the next chapter.

[1] Note that the indexing permitted in PAW can be considered as a superset of that permitted by FORTRAN. This feature cannot be used from within SIGMA.

5.5 Using KUIP vectors in a COMIS routine

The declaration `VECTOR vector_name` may be used inside a COMIS routine to address a KUIP vector. If the vector does not exist, it is created with the specifications provided by the declared dimension.

5.6 Usage of vectors with other PAW objects

Vectors can be used to transport numerical information between different PAW objects, and to manipulate mathematically their content.

`VECTOR/HFILL VNAME ID`	Each vector element of vector `VNAME` is used to fill an existing histogram with identifier `ID`.
`HISTOGRAM/GET_VECTOR/CONTEN`	Provides an interface between vectors and histograms.
`HISTOGRAM/PUT_VECTOR/CONTEN`	Provides an interface between histograms and vectors.

5.7 Graphical output of vectors

`VECTOR/DRAW VNAME`	Interprets the content of the vector `VNAME` as a histogram contents and draw a **graph**.
`VECTOR/PLOT VNAME`	Vector elements are considered as individual values to be entered into a histogram and a graph is produced. If `VNAME` is the name of a vector, then each vector element of `VNAME` is used to fill a histogram which is automatically booked with 100 channels and plotted. If `VNAME` has the form `VNAME1%VNAME2` then a scatter-plot of vector `VNAME1` versus `VNAME2` is plotted.

See section 3.4.4 in the tutorial section for an explanation of the difference between `VECTOR/DRAW` and `VECTOR/PLOT`.

A number of HIGZ [10] macro-primitives are available in PAW. Those directly related to the graphical output of vectors are:

`GRAPH N X Y`	Draw a curve through a set of points defined by arrays `X` and `Y`.
`HIST N X Y`	Draw an histogram defined by arrays `X` and `Y`.
`PIE XO YO RAD N VAL`	Draw a pie chart, of `N` slices, with size of slices given in `VAL`, of a radius `RAD`, centered at `XO`, `YO`.

5.8 Fitting the contents of a vector

A user defined (and parameter dependent) function can be fitted to the points defined by the two vectors `X` and `Y` and the vector of associated errors `EY`. The general syntax of the command to fit vectors is:

```
VECTOR/FIT x y ey func [ chopt np par step pmin pmax errpar ]
```

For more information the reader is referred to the reference part of the present manual.

Chapter 6: SIGMA

6.1 Access to SIGMA

The SIGMA array manipulation package can be accessed in three different ways in PAW:

Precede the statement by the prefix SIGMA

Example

```
PAW > SIGMA xvec=array(100,-pi#pi*2)
PAW > SIGMA y=sin(xvec)*xvec
```

Note the use of the predefined constant `PI` in SIGMA with the obvious value.

The PAW command: APPLication SIGMA

All commands typed in after this command will be directly processed by SIGMA. The command `EXIT` will return control to PAW, e.g.

```
PAW > APPLication SIGMA
SIGMA > xvec=array(100,-pi#pi*2)
SIGMA > sinus=sin(xvec)*xvec
SIGMA > cosinus=cos(xvec)*xvec
SIGMA > exit
PAW > vector/list
 Vector Name                    Type    Length    Dim-1    Dim-2    Dim-3

 XVEC                              R       100      100
 SINUS                             R       100      100
 COSINUS                           R       100      100

 Total of  3 Vector(s)
```

The PAW system function $SIGMA

The expression to be evaluated must be enclosed in parentheses. The function will return the numerical value of the expression (if the result is a scalar) or the name of a temporary vector (if the result is a vector).

Assuming that the computation of the function `sin(x)*x` in the above example would be only for the purpose of producing a graph, (i.e. the result is not needed for further calculations), then one could just have typed the following commands:

```
PAW > SIGMA xvec=array(100,-pi#pi*2)
PAW > GRAph 100 xvec $SIGMA(SIN(XVEC)*XVEC)
```

6.2 Vector arithmetic operations using SIGMA

A complete and convenient mechanism for the mathematical manipulation of vectors is provided by SIGMA. In the following, we use the words "array" and "vector" as synonyms. In both cases, we refer to PAW vectors, in the sense that SIGMA offers an alternative way to generate and to manipulate PAW vectors (see section 5 on page 236). The notation of SIGMA is similar to that of FORTRAN, in the sense that is based upon formulae and assignment statements.

The special operator `ARRAY` is used to generate vectors:

```
vname = ARRAY (arg1,arg2)
```

`vname` Name of the vector (array) being created.

`arg1` Defines the array **structure**, i.e. the Number of COmponents (`NCO`) of the array.

`arg2` Provides the **numerical values** filling the array row-wise.
If `arg2` is absent (or does not provide enough values) the array is filled with 1.

6.2.1 Basic operators

`+`	Add	`*`	Multiply	`**`	Exponentiation
`-`	Subtract	`/`	Divide	`&`	Concatenation

Note that ill defined operations will give `0.` as result. For instance: a division by zero gives zero as result.

6.2.2 Logical operators

Logical operators act on entities that have **Boolean** values `1` (true) or `0` (false). The result is Boolean.

`AND`	Logical operation AND	`EQ`	EQual to	`LE`	Less or Equal to
`NOT`	Logical operation NOT	`GE`	Greater or Equal to	`LT`	Less Than
`OR`	Logical operation OR	`GT`	Greater Than	`NE`	Not Equal

6.2.3 Control operators

`!PRINT` Provides the automatic printing of every newly created array or scalar.

`!NOPRINT` Suppresses the automatic printing of every newly created array or scalar.

Examples

```
A=ARRAY (6,1#6)              1  2  3  4  5  6
A=ARRAY (4)                  1  1  1  1
A=ARRAY (5,2&3&-1&2&1.2)     2  3  -1  2  1.2
A=ARRAY (3)*PI               3.1415927  3.1415927  3.1415927
A=ARRAY (1,123E4)            1230000.0
```

6.3 SIGMA functions

SIGMA provides some functions which perform a task on a whole array. These functions have no analogues in FORTRAN because all FORTRAN functions operate on one or more single numbers. Presently available SIGMA functions are listed in table 6.1 below.

Name	Result	Explanation
ANY	Scalar	The result is a Boolean scalar of value 1 (true) if at least one component of the argument is true and 0 (false) otherwise.
DEL	Vector	Analog to the **Dirac-DELta Function**. V1=DEL(V) sets each element of V1 to 0.0 (if corresponding element in V is **non-zero**) or to 1.0 (if corresponding element is **zero**).
DIFF	Vector	V2=DIFF(V) **forward difference** of V. The rightmost value in V1 is obtained by quadratic extrapolation over the last three elements of V.
LS	Vector	V1=LS(V,N) **shifts** index of V to the left by N steps (cyclic).
LVMAX	Scalar	S1=LVMAX(V1) sets S1 equal to the **index** (location) of the **maximum** value in vector V1.
LVMIM	Scalar	S1=LVMIN(V1) sets S1 equal to the **index** (location) of the **minimum** value in vector V1.
MAX	Vector	V3=MAX(V1,V2) sets each element of V3 equal to the **maximum** of the corresponding elements in V1 and V2.
MAXV	Vector	V1=MAXV(V) sets each element of V1 equal to the **maximum** value in V.
MIN	Vector	V3=MIN(V1,V2) sets each element of V3 equal to the **minumum** of the corresponding elements in V1 and V2.
MINV	Vector	V1=MINV(V) sets each element of V1 equal to the **minimum** value in V.
NCO	Scalar	V1=NCO(V) **Number of COmponents** of vector of V.
ORDER	Vector	V1=ORDER(V,V2) finds a permutation that brings V2 in a non-descending order and applies it to V to generate V1.
PROD	Vector	V1=PROD(V) V1 is the **running product** of V.
QUAD	Vector	V2=QUAD(V1,H) The quadrature function QUAD numerically integrates each row of V1 with respect to the scalar step size H.
SUMV	Vector	V2=SUMV(V1) **running sum** of V.
VMAX	Scalar	S1=VMAX(V1) sets S1 equal to the **maximum** value in vector V1.
VMIN	Scalar	S1=VMIN(V1) sets S1 equal to the **minimum** value in vector V1.
VSUM	Scalar	S1=VSUM(V) **sum** of all components of V.

Table 6.1: SIGMA functions

6.3.1 SIGMA functions - A detailed description

In the following description of the SIGMA functions, the letter R always denotes the **result** and arg denotes one or more **arguments**. Any argument may itself be an expression. In that case arg means the

result of this expression. Let OP denote any of the above array functions, then the statement:

```
R = OP  (arg1,arg2,...)
```

produces R without doing anything to the contents stored under the names appearing in arg1,arg2,.... Thus, although in the description we may say "...OP does such and such to arg ...", in reality it leaves arg intact and works on the argument to produce R.

```
R  = ANY  (arg)
```

The function ANY considers the result of the argument expression as a Boolean array. SIGMA represents "true" by 1 and "false" by 0. Thus the components of arg must be either 0 or 1, otherwise an error is generated.

If at least one component of the result of the argument expression is 1, than ANY returns the scalar 1. If all components of the result of the argument expression are 0 then ANY returns the scalar 0. If arg is a Boolean scalar, R = arg.

Example of the ANY command

```
PAW > APPL SIGMA
SIGMA > !PRINT                          | Print newly created vectors and scalars
SIGMA > W=(-2)**ARRAY(10,1#10)
NCO(W       )=   10
W       =
 -2.000      4.000     -8.000      16.00     -32.00      64.00
 -128.0      256.0     -512.0      1024.
SIGMA > X=W GT 0
NCO(X       )=   10
X       =
 0.0000      1.000     0.0000      1.000     0.0000      1.000
 0.0000      1.000     0.0000      1.000
SIGMA > R=ANY(X)
NCO(R       )=    1
R         1.000
```

```
R  = DEL  (arg)
```

DEL is a discrete analogue of a **Dirac delta function**. DEL works independently on each row of the argument array. If the elements of any row of the argument are denoted by X_1, X_2, ..., X_i, ..., X_n then the corresponding row of the result of the delta function operation will be Z_1, Z_2, ..., Z_i, ..., Z_n where all $Z_i = 0$ except in three cases, in which $Z_i = 1$, namely:

1. When the component X_i is itself zero.
2. When X_{i-1}, X_i are of opposite sign and $|X_i| < |X_{i-1}|$ If $i = 1$ then linear extrapolation to the left is used.
3. When X_i, X_{i+1} are of opposite sign and $|X_i| \leq |X_{i+1}|$ If $i = 1$ then linear extrapolation to the right is used.

If arg is a scalar, the value of DEL(arg) will be 1 if arg is zero, and 0 otherwise.

Example of the `del` command

```
SIGMA > W=array(11,-1#1)
NCO(W       )=   11
W       =
 -1.000    -0.8000    -0.6000    -0.4000    -0.2000    -0.2980E-07
 0.2000     0.4000     0.6000     0.8000      1.000

SIGMA > X= (W+1.01)*W*(W-.35)*(W-.42)
NCO(X       )=   11
X       =
-0.1917E-01 -0.2357    -0.2384    -0.1501    -0.5524E-01-0.4425E-08
 0.7986E-02 -0.5640E-03 0.4347E-01 0.2476     0.7578

SIGMA > R=del(x)
NCO(R       )=   11
R       =
  1.000     0.0000     0.0000     0.0000     0.0000      1.000
 0.0000      1.000     0.0000     0.0000     0.0000
```

`R = DIFF (arg)`

The `DIFF` function generates the **forward difference** of each row of the argument array, say $X_1, X_2, \ldots, X_i, \ldots, X_n$ and creates an array with components equal to the forward difference of X: $X_2 - X_1$, $X_3 - X_2, \ldots, X_n - X_{n-1}, X_0$ where the rightmost value X_0 is obtained by quadratic extrapolation over the last three elements of the result of `arg`. Applied to a scalar `DIFF` gives a zero result.

Example of the `DIFF` command

```
SIGMA > x=array(6,5#0)
NCO(X       )=    6
X       =
  5.000      4.000      3.000      2.000      1.000     0.0000
SIGMA > Y=x*x
NCO(Y       )=    6
Y       =
  25.00      16.00      9.000      4.000      1.000     0.0000
SIGMA > Z=Diff(Y)
NCO(Z       )=    6
Z       =
 -9.000     -7.000     -5.000     -3.000     -1.000      1.000
```

`R = LS (arg1,arg2)`

The `LS` rearrangement function performs a **left shift**. `arg1` is the array to be shifted; `arg2` must be a scalar value (rounded if necessary by the system), interpreted as the number of places the array has to be shifted to the left. The scalar `arg2` can be negative, in which case `LS` shifts to the right a number of places equal to the absolute value of `arg2`.

It should be noted the the shift is performed circularly `modulo N`, where `N` is the number of components in the rows of the array to be shifted. Hence, `LS(X,N+1)` shifts the `N` component rows of `X` by `1` to the left, and `LS(X,-1)` shifts the rows by `N-1` to the left (or by `1` to the right). If `arg1` is a scalar, `R = arg1`.

Example of the left shift command

```
SIGMA > X=array(4&5,array(20,1#20))
NCO(X       )=    4    5
X       =
  1.000       2.000       3.000       4.000
  5.000       6.000       7.000       8.000
  9.000       10.00       11.00       12.00
  13.00       14.00       15.00       16.00
  17.00       18.00       19.00       20.00
SIGMA > y=ls(x,1)
NCO(Y       )=    4    5
Y       =
  2.000       3.000       4.000       1.000
  6.000       7.000       8.000       5.000
  10.00       11.00       12.00       9.000
  14.00       15.00       16.00       13.00
  18.00       19.00       20.00       17.00
SIGMA > y=ls(x,-3)
NCO(Y       )=    4    5
Y       =
  2.000       3.000       4.000       1.000
  6.000       7.000       8.000       5.000
  10.00       11.00       12.00       9.000
  14.00       15.00       16.00       13.00
  18.00       19.00       20.00       17.00
SIGMA > X=array(5,1#5)
NCO(X       )=    5
X         1.000       2.000       3.000       4.000       5.000
SIGMA > z=ls(x,3)
NCO(Z       )=    5
Z         4.000       5.000       1.000       2.000       3.000
SIGMA > z1=ls(x,-4)
NCO(Z1      )=    5
Z1        2.000       3.000       4.000       5.000       1.000
```

R = LVMAX (arg1) and **R = LVMIN (arg1)**

The functions LVMAX and LVMIN returns as a scalar result the index (position) of the largest or smallest element, respectively, in the argument array.

Example of using the LVMAX and LVMIN commands

```
SIGMA > x=sin(array(10,1#10))
NCO(X       )=   10
X       =
 0.841     0.909     0.141    -0.757    -0.959    -0.279     0.657
 0.989     0.412    -0.544

SIGMA > r=lvmax(x)
NCO(R       )=    1
R         8.00
```

`R = MAX (arg1,arg2)` and `R = MIN (arg1,arg2)`

The functions `MAX` and `MIN` work independently on each element of their arguments. `arg2` can be a scalar. The result has the same dimension as the argument array `arg1` and each element of the result is set equal to the largest or smallest element, respectively, of the corresponding element of the argument arrays.

Example of using the `MAX` and `MIN` commands

```
SIGMA > x=sin(array(10,1#10))
NCO(X       )=   10
X       =
 0.841     0.909     0.141    -0.757    -0.959    -0.279     0.657
 0.989     0.412    -0.544
SIGMA > y=cos(array(10,1#10))
NCO(Y       )=   10
Y       =
 0.540    -0.416    -0.990    -0.654     0.284     0.960     0.754
-0.146    -0.911    -0.839
SIGMA > z=min(x,y)
NCO(Z       )=   10
Z       =
 0.540    -0.416    -0.990    -0.757    -0.959    -0.279     0.657
-0.146    -0.911    -0.839
```

`R = MAXV (arg)` and `R = MINV (arg)`

The extrema functions `MAXV` and `MINV` work on each element of their argument and the result has the same dimension as the argument array `arg1`. Each element of of the result is set equal to the largest or smallest element, respectively, of the corresponding row of the argument array.

All these functions, if applied to a scalar argument, yield `R=arg`.

Example of using the `MAX` and `MIN` commands

```
SIGMA > x=array(10,0#10)
NCO(X       )=   10
X       =
 0.0000     1.111     2.222     3.333     4.444     5.556
 6.667     7.778     8.889     10.00
SIGMA > s=sin(x)*x
NCO(S       )=   10
S       =
 0.0000    0.9958     1.767    -0.6352    -4.286    -3.695
 2.494     7.755     4.539    -5.440
SIGMA > x=minv(s)
NCO(X       )=   10
X       =
-5.440    -5.440    -5.440    -5.440    -5.440    -5.440
-5.440    -5.440    -5.440    -5.440
```

R = NCO (arg)

The "Number of COmponents" (NCO) control function obtains the NCO vector of the arg. The NCO vector of a scalar is the scalar 1. For any argument the NCO(NCO(arg)) gives the number of dimensions of the arg.

Using the NCO command

```
SIGMA > x=array(4&3&2,array(24,2#48))
NCO(X       )=    4    3    2
X       =
  2.000      4.000      6.000      8.000
  10.00      12.00      14.00      16.00
  18.00      20.00      22.00      24.00

  26.00      28.00      30.00      32.00
  34.00      36.00      38.00      40.00
  42.00      44.00      46.00      48.00

SIGMA > r=nco(x)
NCO(R       )=    3
R         4.000      3.000      2.000
SIGMA > ndim=nco(nco(x))
NCO(NDIM    )=    1
NDIM      3.000
```

R = ORDER (arg1,arg2)

The ordering function ORDER acts independently on each row of arg1. arg2 must have the same row length as arg1.

ORDER finds the permutation that brings arg2 into a non-descending sequence (row-wise) and constructs the result by applying this permutation to arg1. It may in some cases be expanded to that structure by using the techniques of the topological arithmetic. This is particularly useful if arg2 is a single vector with the length of the rows of arg1.

Using the ORDER command

```
SIGMA > X = 1&1&2&4&-3&1&3
NCO(X       )=    7
X       =
  1.00       1.00       2.00       4.00      -3.00       1.00       3.00
SIGMA > P = ORDER(X,X)
NCO(P       )=    7
P       =
 -3.00       1.00       1.00       1.00       2.00       3.00       4.00
SIGMA > P = ORDER(X,-X)
NCO(P       )=    7
P       =
  4.00       3.00       2.00       1.00       1.00       1.00      -3.00
SIGMA > Y = ARRAY(7,1# 7)
NCO(Y       )=    7
Y       =
  1.00       2.00       3.00       4.00       5.00       6.00       7.00
SIGMA > P = ORDER(Y,X)
NCO(P       )=    7
P       =
  5.00       1.00       2.00       6.00       3.00       7.00       4.00
```

R = **PROD** (arg)

The PROD function generates the **running product** of each row of the argument array, say $X_1, X_2, \ldots, X_n$ and creates an array with components equal to the running product of the component of the argument: $X_1,\ X_2, \ldots,\ X_n\ X_1, X_1 \times X_2,\ \ldots,\ X_1 \times X_2 \times \ldots X_n$

Using the TIMES command

```
SIGMA > x=array(6&4,array(24,1#24))
NCO(X      )=    6    4
X      =
  1.000      2.000      3.000      4.000      5.000      6.000
  7.000      8.000      9.000      10.00      11.00      12.00
  13.00      14.00      15.00      16.00      17.00      18.00
  19.00      20.00      21.00      22.00      23.00      24.00

SIGMA > y=prod(x)
NCO(Y      )=    6    4
Y      =
  1.000      2.000      6.000      24.00      120.0      720.0
  7.000      56.00      504.0      5040.     0.5544E+05 0.6653E+06
  13.00      182.0      2730.     0.4368E+05 0.7426E+06 0.1337E+08
  19.00      380.0      7980.     0.1756E+06 0.4038E+07 0.9691E+08
```

R = **QUAD** (arg1,arg2)

The **quadrature function** QUAD numerically integrates each row of arg1 with respect to the scalar step size h defined by arg2.

The result R has the same dimension as arg1 and the integration constant is fixed by choosing the first point of the result to be zero.

The method uses a four-point forward and backward one-strip-formula based on Lagrange interpolation. We have for the first point of the result:

$$R_1 = \int_{x_1}^{x_1} (arg1)\mathrm{d}x = 0$$

for the second and third points

$$R_{i+1} = R_i + \frac{h}{24}(9f_i + 19f_{i+1} - 5f_{i+2} + f_{i+3})$$

and for all subsequent points

$$R_i = R_{i-1} + \frac{h}{24}(f_{i-3} - 5f_{i-2} + 19f_{i-1} + 9f_i)$$

where the f_i are elements of `arg1` and are assumed to be values of some functions evaluated at equidistant intervals with interval width equal to `h` (`h` being equal to the value of `arg2`).

```
SIGMA > *********************
SIGMA > * SIGMA application *
SIGMA > *  showing use of   *
SIGMA > *   QUAD numeric    *
SIGMA > *   integration     *
SIGMA > *********************
SIGMA > x=array(101,0#2*pi)
SIGMA > * Function value array
SIGMA > y=sin(x)
SIGMA > * Step size
SIGMA > dx=0.6283186E-01
SIGMA > print dx
 NCO(DX      )=    1
 DX       0.6283186E-01
SIGMA > * Integration of SIN(X)
SIGMA > * in interval 0<=X<+2*PI
SIGMA > f=quad(y,dx)
SIGMA > * Analytical result
SIGMA > * is   1-COS(X)
SIGMA > g=1-cos(x)
SIGMA > * Compute the difference
SIGMA > erro=(g-f)*10**6
SIGMA > * Plot the difference
SIGMA > *  in units of 10**-6
SIGMA > exit
PAW > opt GRID
PAW > gra 101 x erro
```

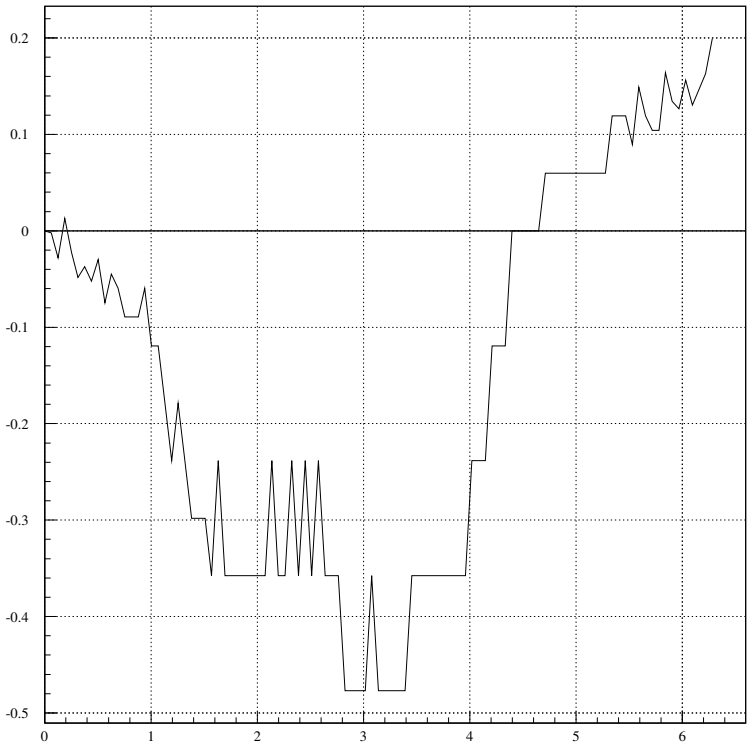


Figure 6.1: Using numerical integration with SIGMA

R = SUMV (arg)

The `SUMV` function generates the **running summation** of each row of the argument array, say $X_1, X_2, \ldots, X_i, \ldots, X_n$ and creates an array with components equal to the running sum of the X_i namely: X_1, $X_1 + X_2, \ldots, X_1 + X_2 + \ldots X_i, \ldots, X_1 + X_2 + \ldots X_n$.

Using the `SUM` function

```
SIGMA >  x=array(6&4,array(24,1#24))
NCO(X       )=    6    4
X       =
  1.000      2.000      3.000      4.000      5.000      6.000
  7.000      8.000      9.000      10.00      11.00      12.00
  13.00      14.00      15.00      16.00      17.00      18.00
  19.00      20.00      21.00      22.00      23.00      24.00

SIGMA > y=sumv(x)
NCO(Y       )=    6    4
Y       =
  1.000      3.000      6.000      10.00      15.00      21.00
```

```
   7.000      15.00      24.00      34.00      45.00      57.00
   13.00      27.00      42.00      58.00      75.00      93.00
   19.00      39.00      60.00      82.00      105.0      129.0
```

`R = VMAX (arg)` and `R = VMIN (arg)`

The functions VMAX and VMIN return a scalar equal to the largest or smallest element of the array arg.

`R = VSUM (arg1)`

The VSUM function generates the **sum** of each element of the argument array, say $X_1, X_2, \ldots, X_i, \ldots, X_n$ and creates a scalar whose value is equal to the sum of all the components of X namely: $X_1 + X_2 + X_3, \ldots, X_n$

Using the VSUM function

```
SIGMA > x=array(10)
NCO(X       )=    10
X       =
  1.00       1.00       1.00       1.00       1.00       1.00       1.00
  1.00       1.00       1.00

SIGMA > r=vsum(x)
NCO(R       )=     1
R          10.0
```

6.4 Available library functions

The library functions available under SIGMA are listed below. All these functions have a single argument, unless otherwise indicated. The number indicated between parentheses corresponds to the number of the same function in the CERN program library.

ABS	ABSolute value
ACOS	ArCOSine
ALOGAM	LOGarithm of the GAMma Function (C341)
ASIN	ArcSINe
ATAN	ArcTANgent
ATAN2	ArcTANgent2 (2 arguments)
BESI0	Mod. Bessel Function I0 (C313)
BESI1	Mod. Bessel Function I1 (C313)
BESJ0	Bessel Function J0 (C312)
BESJ1	Bessel Function J1 (C312)
BESK0	Mod. Bessel Function K0 (C313)
BESK1	Mod. Bessel Function K1 (C313)

`BESY0`	Bessel Function Y0 (C312)
`BESY1`	Bessel Function Y1 (C312)
`COS`	COSine
`COSH`	Hyperbolic COSine
`COSINT`	COSine INTegral (C336)
`DILOG`	DILOGarithm Function (C304)
`EBESI0`	$\exp(-\|x\|)I_0(x)$ (C313)
`EBESI1`	$\exp(-\|x\|)I_1(x)$ (C313)
`EBESK0`	$\exp(x)K_0(x)$ (C313)
`EBESK1`	$\exp(x)K_1(x)$ (C313)
`ELLICK`	Complete Elliptic Integral K (C308)
`ELLICE`	Complete Elliptic Integral E (C308)
`ERF`	Error Function ERF (C300)
`ERFC`	Error Function ERFC (C300)
`EXP`	EXPonential
`EXPINT`	EXPonential INTegral (C337)
`FREQ`	Normal Frequency Function FREQ (C300)
`GAMMA`	GAMMA Function (C305)
`INT`	Takes INTegral part of decimal number
`LOG`	Natural LOGarithm
`LOG10`	Common LOGarithm
`MOD`	Remaindering
`RNDM`	Random Number Generator: `V1=RNDM(V)`, with `NCO(V1)=NCO(V)` generates random numbers between `0` and `1`.
`SIGN`	Transfer of SIGN: `V2=SIGN(V,V1)`, `V2=|V|*V1/|V1|`
`SIN`	SINe Function
`SINH`	Hyperbolic SINe
`SININT`	SINe INTegral (C336)
`SQRT`	SQuare RooT
`TAN`	TANgent
`TANH`	Hyperbolic Tangent

Ill defined functions will return `0.` as result. (e.g. `SQRT` of a negative number is taken as `0`).

Chapter 7: HBOOK

7.1 Introduction

Many of the ideas and functionality in the area of data presentation, manipulation and management in PAW find their origin in the HBOOK subroutine package [2], which handles statistical distributions (histograms and Ntuples). HBOOK is normally run in a batch environment, and it produces generally graphics output on the line printer or, optionally, via the HPLOT [11] package on a high resolution graphic output device.

The HBOOK system consists of a few hundred FORTRAN subroutines which enable the user to symbolically define, fill and output one- and two-dimensional density estimators, under the form of **histograms, scatter-plots** and **tables**.

Furthermore the analysis of large data samples is eased by the use of **Ntuples**, which are two-dimensional arrays, characterised by a **fixed** number N, specifying the number of entries per element, and by a **length**, giving the total number of elements. An element of a Ntuple can be thought of as a physics "event" on e.g. a Data Summary Tape (micro-DST). **Selection criteria** can be applied to each "event" or element and a complete Ntuple can be statistically analysed in a fast, efficient and interactive way.

7.1.1 The functionality of HBOOK

The various user routines of HBOOK can be subdivided by functionality as follows:

Booking	Declare a one- or two-dimensional histogram or a Ntuple
Projections	Project two-dimensional distributions onto both axes
Ntuples	Way of writing micro data-summary-files for further processing. This allows to make later projections of individual variables or correlation plots. Selection mechanisms may be defined
Function representation	Associates a real function of 1 or 2 variables to a histogram
Filling	Enter a data value into a given histogram, table or Ntuple
Access to information	Transfer of numerical values from HBOOK-managed memory to Fortran variables and back
Arithmetic operations	On histograms and Ntuples
Fitting	Least squares and maximum likelihood fits of parametric functions to histogramed data
Smoothing	Splines or other algorithms
Random number generation	Based on experimental distributions
Archiving	Information is stored on mass storage for further reference in subsequent programs
Editing	Choice of the form of presentation of the histogramed data

7.2 Basic ideas

The basic data elements of HBOOK are the **histogram** (one- and two-dimensional) and the **Ntuple**. The user identifies his data elements using a **single integer**. Each of the elements has a number of **attributes** associated with it.
The HBOOK system uses the ZEBRA [7] data manager to store its data elements in a COMMON block `/PAWC/`, shared with the KUIP [4] and HIGZ [10] packages, when the latter are also used (as is the case in PAW). In fact the first task of a HBOOK user is to declare the length of this common to ZEBRA by a call to `HLIMIT`, as is seen in figures 7.3 and 7.5[1].
In the `/PAWC/` data store, the HBOOK, HIGZ and KUIP packages have all their own **division** (see [7] for more details on the notion of divisions) as follows (figure 7.1):

`LINKS` Some locations at the beginning of `/PAWC/` for `ZEBRA` pointers.

`WORKS` Working space (or division `1`) used by the various packages storing information in `/PAWC/`

`HBOOK` Division `2` of the store. Reserved to `HBOOK`

`HIGZ` A division reserved for the `HIGZ` graphics package.

`KUIP` A division reserved for the `KUIP` user interface package.

`SYSTEM` The `ZEBRA` system division. It contains some tables, as well as the Input/Output buffers for `HRIN` and `HROUT`.

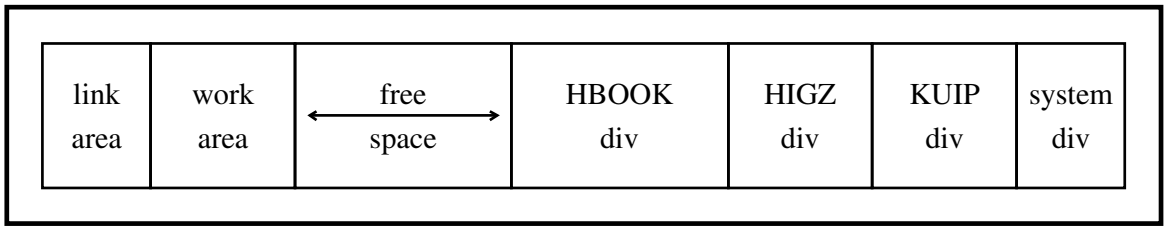


Figure 7.1: The layout of the `/PAWC/` dynamic store

7.2.1 RZ directories and HBOOK files

An advantage of using ZEBRA in HBOOK is that ZEBRA's direct access **RZ package** is available. The latter allows data structures to be uniquely addressed via **pathnames**, carrying a mnemonic meaning and showing the relations between data structures. Related data structures are addressed from a **directory**. Each time a RZ file is opened via a call to `HRFILE` a supplementary top directory is created with a name specified in the calling sequence. This means that the user can more easily keep track of his data and also the **same** histogram identifiers can be used in various files, what makes life easier if one wants to study various data samples with the same program, since they can be addressed by changing to the relevant file by a call to `HCDIR` first.

[1] This is of course not necessary in PAW, which is already precompiled when it is run. However when treating very large data samples or in other special applications, it might be necessary to specify a different value for the length of the dynamic store, which is defined by a call to `PAWINT` from the main initialisation routine `PAMAIN`. The "default" value for the length of `/PAWC/` is 500000 (Apollo), 200000 (IBM) or 300000 (other systems), with respectively 10000 and 68000 words initially reserved for HIGZ and KUIP.

Example of using directories

```
CALL HRFILE(1,'HISTO1',' ')    ! Open first  HBOOK RZ file (read only)
CALL HRFILE(2,'HISTO2','U')    ! Open second HBOOK RZ file (update)
CALL HCDIR('//HISTO1',' ')     ! Make HISTO1 current directory
CALL HRIN(20,9999,0)           ! Read ID 20 on file 1
  ....
CALL HCDIR('//HISTO2',' ')     ! Make HISTO2 current directory
CALL HRIN(10,9999,0)           ! Read ID 10 on file 2
  ....
CALL HROUT(20,ICYCLE,' ')      ! Write ID 20 to file 2
CALL HREND('HISTO1')           ! Close file 1
CALL HREND('HISTO2')           ! Close file 2
```

In the previous example (and also in figures 7.3 and 7.5) it is shown how an external file is available via a directory name inside HBOOK (and PAW), and that one can change from one to the other file by merely **changing directory**, via the PAW command `CDIR`, which calls the HBOOK routine `HCDIR`.

7.2.2 Changing directories

One must pay attention to the fact that **newly** created histograms go to **memory** in the `//PAWC` directory (i.e. the `/PAWC/` common). As an example suppose that the current directory is `//LUN1`, and an operation is performed on two histograms. These histograms are first copied to memory `//PAWC`, the operation is performed and the result is **only** available in `//PAWC`,

```
PAW > CDIR //LUN1            | Set current directory to //LUN1
PAW > ADD 10 20 30           | Add histograms 10 and 20 into 30
                             | Histogram 30 is created in //PAWC
PAW > Histo/Plot //PAWC/30   | Show the result of the sum
PAW > CD //PAWC              | Set the current directory to memory
PAW > Histo/plot 30          | Show the result once more
```

Similarly when histograms or Ntuples are plotted (e.g. by the `HISTO/PLOT` command), they are copied to memory possibly replacing an old copy of the same ID. As long as the copy in memory is not changed, each time the ID is read from the **external** file. This is because in a **real time** environment, e.g. using **global sections** on VMS or **modules** with OS9, the data base on the external medium can be changed by concurrent processes. However if the HBOOK data structure, associated with the histogram or Ntuple in memory is **altered** (e.g. by a `MAX, IDOPT, FIT` command), then it becomes the **default** for subsequent operations. If one wants the **original copy** one first must delete the copy from memory or **explicitly** use the pathname for the external file.

```
PAW > Histo/file 1 his.dat   | The file contains ID=10
PAW > Histo/Plot 10          | ID=10 read from file and plotted
PAW > H/plot 10              | ID=10 read again from file and plotted
PAW > H/fit 10 ! G           | Read from file, make a Gaussian fit on //PAWC/10
PAW > H/plot 10              | ID=10 read from memory since it changed
PAW > H/del 10               | Delete histogram 10 from memory
PAW > H/plot 10              | ID=10 read again from file and plotted
```

7.3 HBOOK batch as the first step of the analysis

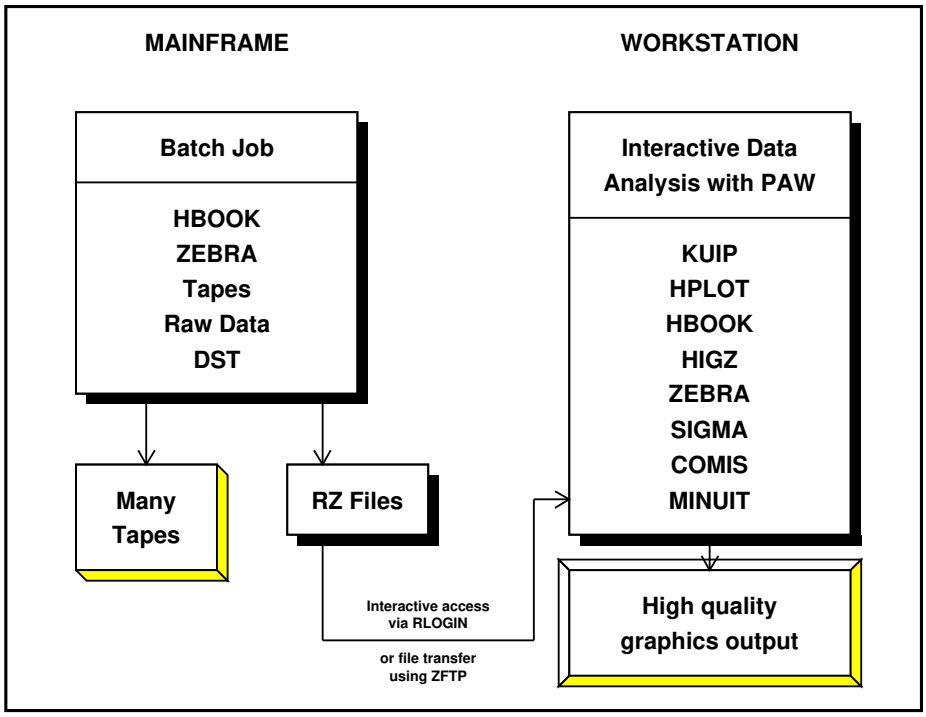


Figure 7.2: Schematic presentation of the various steps in the data analysis chain

Although it is possible to define histograms interactively in a PAW session, and then read the (many thousands of) events, in general for large data samples the relevant variables are extracted from the **Data Summary Files** or **DST**s and stored in **histograms** or an **Ntuple**. The histogram needs already that a certain choice has to be made as to the range of values for the plotted parameter, because the **binning**, or the coarseness, of the distribution has to be specified when the histogram is defined (**booked**). Also only one- and two-dimensional histograms are possible, hence the correlations between various parameters can be difficult to study. Hence it seems in many cases more appropriate to store the value of the important parameters for each event in an **Ntuple**. This approach preserves the correlation between the parameters and allows selection criteria to be applied on the (reduced) data sample at a later stage.

In general, the time consuming job of analysing all events available on tape is run on a mainframe or CPU server, and the important event parameters are stored in a Ntuple to allow further detailed study. For convenience the Ntuple can be output to disk for each run, and then at a later stage the Ntuples can be **merged** in order to allow a global interactive analysis of the complete data sample.

A typical batch job in which data are analysed offline and some characteristics are stored in HBOOK is given in 7.3. After opening the RZ HBOOK file, HBOOK is initialised by a call to `HLIMIT`, which declares a length of 20000 words for the length of the `/PAWC/` dynamic store. Then the one- and two- dimensional histograms 110 and 210 are filled respectively according to the functions `HTFUN1` and `HTFUN2`. The output generated by the program is shown in Figure 7.4.

```
      PROGRAM HTEST
      PARAMETER (NWPAWC=20000)
      COMMON/PAWC/H(NWPAWC)
      EXTERNAL HTFUN1,HTFUN2
*.___________________________________________________________
      CALL HLIMIT(NWPAWC)
*             Book histograms and declare functions
      CALL HBFUN1(100,'Test of HRNDM1',100,0.,1.,HTFUN1)
      CALL HBOOK1(110,'Filled according to HTFUN1',100,0.,1.,1000.)
      CALL HBFUN2(200,'Test of HRNDM2',100,0.,1.,40,0.,1.,HTFUN2)
      CALL HSCALE(200,0.)
      CALL HBOOK2(210,'Fill according to HTFUN2',100,0.,1.,40,0.,1.,30.)
*             Fill histograms
      DO 10 I=1,10000
         X=HRNDM1(100)
         CALL HFILL(110,X,0.,1.)
         CALL HRNDM2(200,X,Y)
         CALL HFILL(210,X,Y,1.)
  10  CONTINUE
*             Save all histograms on file HTEST.DAT
      CALL HRPUT(0,'HTEST.DAT','N')
      CALL HDELET(100)
      CALL HDELET(200)
      CALL HPRINT(0)
      END
      FUNCTION HTFUN2(X,Y)
*             Two-dimensional guassian
      HTFUN2=HTFUN1(X)*HTFUN1(Y)
      RETURN
      END
      FUNCTION HTFUN1(X)
*             Constants for gaussians
      DATA C1,C2/1.,0.5/
      DATA XM1,XM2/0.3,0.7/
      DATA XS1,XS2/0.07,0.12/
*             Calculate the gaussians
      A1=-0.5*((X-XM1)/XS1)**2
      A2=-0.5*((X-XM2)/XS2)**2
      X1=C1
      X2=C2
      IF(ABS(A1).GT.0.0001)X1=C1*EXP(A1)
      IF(ABS(A2).GT.0.0001)X2=C2*EXP(A2)
*             Return function value
      HTFUN1=X1+X2
      RETURN
      END
```

Figure 7.3: Writing data to HBOOK with the creation of a HBOOK RZ file

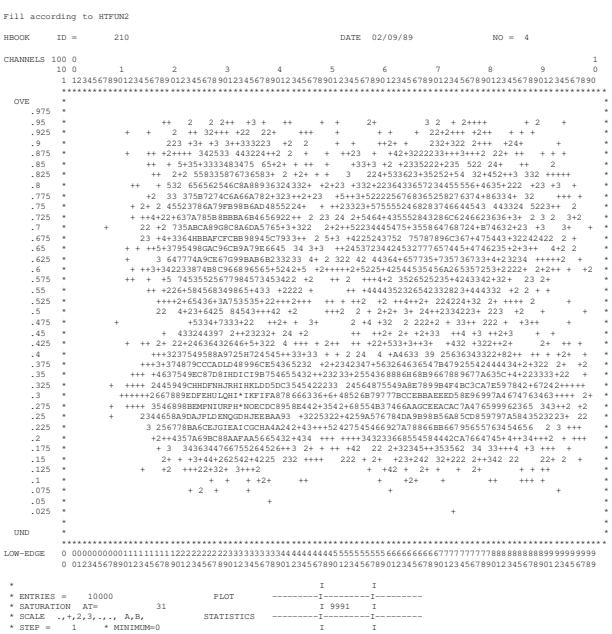


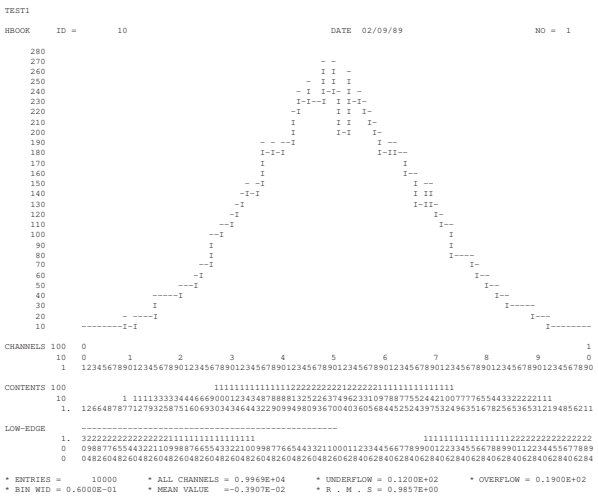
Figure 7.4: Output generated by job HTEST

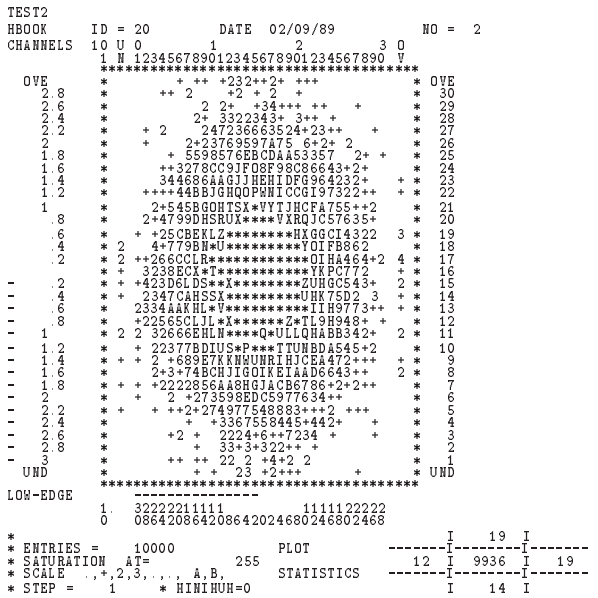
7.3.1 Adding some data to the RZ file

The second run using program HTEST1 shows how to add some data to the HBOOK RZ file created in the job HTEST. After opening the file in question in update mode ('U' option) with the name EXAM2, a new directory NTUPLE is created, known as //EXAM2/NTUPLE as seen in the output of HLDIR command at the end of the output. A one- and a two-dimensional histogram and a Ntuple with identifiers of respectively 10, 20 and 30 are booked. Each Ntuple element or "event" is characterised by three **variables** (labelled 'X', 'Y' and 'Z'). The Ntuple data, when the initial size of 1000 words is exhausted, will be written to the directory specified in the call to HBOOKN, i.e. //EXAM2/NTUPLE, and the data in memory are replaced with those newly read. A one- and a two-dimensional projection of X and X Y are then made onto histograms 10 and 20 respectively, before they are printed and written on the HBOOK RZ file. At the end the **current** and **parent** directories are listed. The contents of the latter shows that the data written in the first job (HTEST) are indeed still present in the file under the top directory //EXAM2. The call to RZSTAT shows usage statistics about the RZ file.

Example of adding data to a HBOOK RZ file

```
      PROGRAM HTEST1
      PARAMETER (NWPAWC=20000)
      COMMON/PAWC/H(NWPAWC)
      DIMENSION X(3)
      CHARACTER*8 CHTAGS(3)
      DATA CHTAGS/'   X   ','   Y   ','   Z   '/
*.------------------------------------------------
      CALL HLIMIT(NWPAWC)
*             Reopen data base
      CALL HROPEN(1,'EXAM2','HTEST.DAT',0,'U')
      CALL HMDIR('NTUPLE','S')
      CALL HBOOK1(10,'TEST1',100,-3.,3.,0.)
      CALL HBOOK2(20,'TEST2',30,-3.,3.,30,-3.,3.,250.)
      CALL HBOOKN(30,'N-TUPLE',3,'//EXAM2/NTUPLE',
     +            1000,CHTAGS)
*
      DO 10 I=1,10000
         CALL RANNOR(A,B)
         X(1)=A
         X(2)=B
         X(3)=A*A+B*B
         CALL HFN(30,X)
  10  CONTINUE
*
      CALL HPROJ1(10,30,0,0,1,999999,1)
      CALL HPROJ2(20,30,0,0,1,999999,1,2)
      CALL HPRINT(0)
      CALL HROUT(0,ICYCLE,' ')
      CALL HLDIR(' ',' ')
      CALL HCDIR('-',' ')
      CALL HLDIR(' ',' ')
      CALL RZSTAT(' ',999,' ')
      CALL HREND('EXAM2')
      END
```






```
**********************************************************
* NTUPLE ID=   30  ENTRIES=  10000   N-TUPLE             *
**********************************************************
*  Var numb  *    Name     *     Lower     *     Upper     *
**********************************************************
*       1     *     X       * -.422027E+01 * 0.386411E+01 *
*       2     *     Y       * -.411076E+01 * 0.378366E+01 *
*       3     *     Z       * 0.485187E-04 * 0.179518E+02 *
**********************************************************


===> Directory : //EXAM2/NTUPLE
        30 (N)   N-TUPLE
        10 (1)   TEST1
        20 (2)   TEST2

===> Directory : //EXAM2
       100 (1)   Test of HRNDM1
       110 (1)   Filled according to HTFUN1
       200 (2)   Test of HRNDM2
       210 (2)   Fill according to HTFUN2


     NREC    NWORDS    QUOTA(%)  FILE(%)   DIR. NAME
      34       34064       0.85      0.85   //EXAM2/NTUPLE
      41       40431       1.02      1.02   //EXAM2
```

Figure 7.5: Adding data to a HBOOK RZ file

7.4 Using PAW to analyse data

After transferring the HBOOK RZ file, which was created in the batch job as explained in the previous section, we start a PAW session to analyse the data which were generated[2]. The PAW session below shows that the file `HTEST.DAT` is first opened via a call to `HISTO/FILE`. The data on the file are now accessible as the top directory `//LUN1`. When listing with the `LDIR` command the contents of the top directory `//LUN1` and its `NTUPLE` subdirectory, the same information (histograms and Ntuples) is found as in the batch job (figure 7.5)

Reading a HBOOK direct access file

```
PAW > histo/file 1 htest.dat     | open the HBOOK RZ file
PAW > ldir                       | list current directory

************** Directory ===> //LUN1 <===

                Created 890902/1955  Modified 890902/1958

===> List of subdirectories
NTUPLE          Created 890902/1958 at record     9

===> List of objects
    HBOOK-ID  CYCLE   DATE/TIME   NDATA   OFFSET    REC1    REC2
       100       1   890902/1955    153       1       3
       110       1   890902/1955     88     154       3
       200       1   890902/1955   4335     242       3       4 ==>     7
       210       1   890902/1955    767     481       7       8

 NUMBER OF RECORDS =    7  NUMBER OF MEGAWORDS =  0 +  6367 WORDS
 PER CENT OF DIRECTORY QUOTA USED =   0.175
 PER CENT OF FILE USED            =   0.175
 BLOCKING FACTOR                  =  74.540
PAW > ldir ntuple                | list directory in NTUPLE

************** Directory ===> //LUN1/NTUPLE <===

                Created 890902/1958  Modified 890902/1958

===> List of objects
    HBOOK-ID  CYCLE   DATE/TIME   NDATA   OFFSET    REC1    REC2
        30       2   890902/1958   1082     215      41      42
                 1   890902/1958   1082     725      39      40
        10       1   890902/1958    151     783      40
        20       1   890902/1958    305     934      40      41

 NUMBER OF RECORDS =   34  NUMBER OF MEGAWORDS =  0 + 34064 WORDS
 PER CENT OF DIRECTORY QUOTA USED =   0.851
 PER CENT OF FILE USED            =   0.850
 BLOCKING FACTOR                  =  94.899
```

Figure 7.6: Reading a HBOOK direct access file

[2] In fact it is possible to leave the data on the disk of the machine where they were written in the batch job, and connect with `NETWORK/RLOGIN host` to the machine in question, getting access to the file via TCP/IP. See page 321 for more details.

7.4.1 Plot histogram data

The analysis of the data can now start and we begin by looking at the histograms in the top directory. Figure 7.7 shows the commands entered and the corresponding output plot. They should be compared with the lineprinter output in figure 7.4.

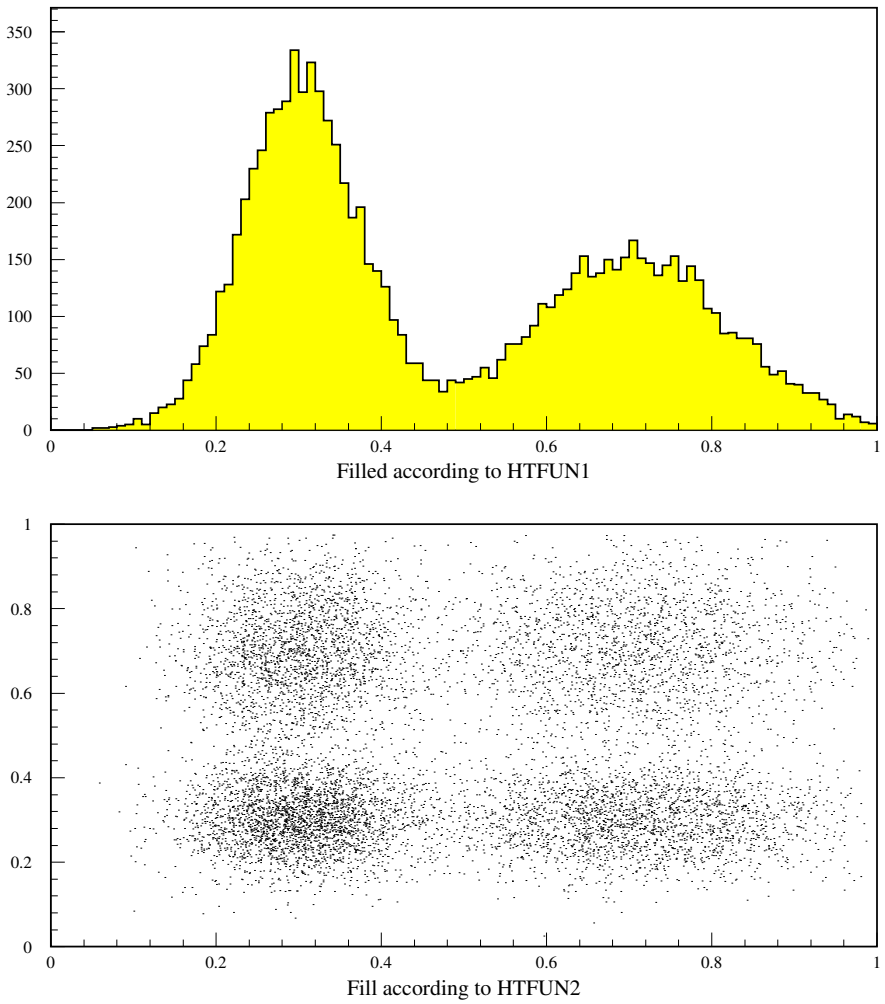


Figure 7.7: Plot of one- and two-dimensional histograms

Plotting histogram data

```
PAW > zon 1 2          | Divide picture into 2 vertically
PAW > set htyp -3      | Set hatch style for histogram
PAW > hi/pl 110        | Plot 1-dimensional histogram 110
PAW > hi/pl 210        | Plot 2-dimensional histogram 210
```

7.5 Ntuples: A closer look

We now turn our attention to the NTUPLE directory to show the functionality and use of Ntuples. After making NTUPLE the **current** directory the available HBOOK objects are listed. The structure of the Ntuple with identifier 30 is PRINTed. The contents of the various Ntuple elements ("events") can be viewed by the NTUPLE/SCAN command. As with most Ntuple commands a **selection criterion** can be given to treat only given "selected" subsamples of the Ntuple (two examples are seen with the further NTUPLE/SCAN commands.

Looking at Ntuple elements

```
 PAW > cd ntuple                                  | move to NTUPLE directory
 PAW > hi/li                                      | list HBOOK objects
 ===> Directory : //LUN1/NTUPLE
        30 (N)   N-TUPLE
        10 (1)   TEST1
        20 (2)   TEST2

 PAW > nt/print 30                                | print summary for Ntuple 30
 *********************************************************
 * NTUPLE ID=   30  ENTRIES=  10000   N-TUPLE           *
 *********************************************************
 *  Var numb  *   Name    *    Lower     *    Upper     *
 *********************************************************
 *      1     *    X      * -.422027E+01 * 0.386411E+01 *
 *      2     *    Y      * -.411077E+01 * 0.378365E+01 *
 *      3     *    Z      * 0.485188E-04 * 0.179518E+02 *
 *********************************************************
 PAW > nt/scan 30                                 | scan the first elements
 ************************************************
 * ENTRY *      X      *      Y      *      Z      *
 ************************************************
 !     1 !  -1.0765    !   1.4405    !   3.2337    !
 !     2 !  -1.2429    !  -1.6043    !   4.1185    !
 !     3 !  0.54489    !   1.7043    !   3.2017    !
 !     4 ! -0.81803    !  0.66588    !   1.1126    !
 !     5 !  -1.8752    !  0.38176    !   3.6621    !
 !     6 !  0.37968    !  -1.0601    !   1.2680    !
 !     7 ! -0.52406    ! -0.68243E-01!  0.27930    !
 !     8 !   1.2175    !  0.91701    !   2.3231    !
 !     9 ! -0.21487    ! -0.26670    !  0.11730    !
 !    10 !  0.70368    !  0.82514    !   1.1760    !
 !    11 !  0.93648E-01!  -2.0311    !   4.1343    !
 !    12 ! -0.48216    !  -2.5980    !   6.9820    !
 !    13 ! -0.45801    !  0.71523    !  0.72132    !
 !    14 ! -0.60272    !  0.98909E-01!  0.37306    !
 !    15 !  0.70454    ! -0.25562    !  0.56172    !
More...? ( <CR>/N ): N
 ==>     15 events have been scanned

 PAW > nt/sc 30 z>16                              | example of a condition on the Z variable
 ************************************************
 * ENTRY *      X      *      Y      *      Z      *
 ************************************************
 !    43 !   3.8641    !  -1.5822    !   17.435    !
 !  1964 !  -4.2203    ! -0.37562    !   17.952    !
```

```
!  7480 !  0.94503    !  -4.1108    !   17.791    !
!  9213 !  0.71434    !  -4.0068    !   16.565    !
==>      4 events have been scanned
PAW > nt/sc 30 abs(x)>4.or.abs(y)>4          | example of a more complex selection criterium
************************************************
* ENTRY *      X      *      Y      *      Z      *
************************************************
!  1964 !  -4.2203    ! -0.37562    !   17.952    !
!  7480 !  0.94503    !  -4.1108    !   17.791    !
!  9213 !  0.71434    !  -4.0068    !   16.565    !
==>      3 events have been scanned
```

7.5.1 Ntuple plotting

The general format of the command NTUPLE/PLOT to project and plot a Ntuple as a (1-Dim or 2-Dim) histogram with automatic binning, possibly using a selection algorithm is:

NTUPLE/PLOT idn [uwfunc nevent ifirst nupd chopt]

IDN	Ntuple Identifier and variable(s) (see table 7.1)
UWFUNC	Selection function (see table 7.2) - Default no function
NEVENT	Number of events to be processed (default is 999999)
IFIRST	First event to be procesed (default is 1)
NUPD	Frequency with which to update histogram (default is 1000000)
CHOPT	HPLOT options (C,S,+,B,L,P,*,U,E,A)

7.5.2 Ntuple variable and selection function specification

Format	Explanation	Example
IDN.CHNAME	The variable **named** "CHNAME"	30.x variable x
IDN.n	The Ntuple variable at **position** n	30.3 variable 3
IDN.expression	Expression is any numerical expression of Ntuple variables. It may include a call to a COMIS function.	30.X**2+Y**2 30.X*COMIS.FOR
IDN.B%A	Scatter-plot of variable B versus A for each event	30.Y%X Y versus X
IDN.2%1	Scatter-plot of variable nb. 2 versus variable nb. 1	30.1%3 1 versus 3
IDN.expr1%expr2	expr1 and expr2 can be any numerical expression of the Ntuple variables. They can be COMIS functions.	30.SQRT(X**2+Y**2)%SIN(Z) 30.COMIS1.FTN%COS(Z)
	Any combination of the above	30.3%COMIS2.FTN*SIN(X)

Table 7.1: Syntax for specifying Ntuple variables

Format	Explanation	Example
0 or missing	No selection is applied (weight is 1).	NT/PLOT 30.X
Combination of cuts	A CUT or combination of CUTs, each created by the command NTUPLE/CUTS	NT/PLOT 30.X 1 (use cut 1) NT/PLOT 30.X 1.AND.2 NT/PLOT 30.X .NOT.(1.AND.3).OR.2
Combination of masks	A MASK or a combination of MASKs, each created by a command NTUPLE/MASK	Assuming there exists a mask vector MSK: NT/PLOT 30.X MSK(4) (bit 4) NT/PLOT 30.X MSK(1).OR.MSK(6)
Logical expression	Any **logical** combination of conditions between Ntuple variables, cuts and masks.	NT/PLOT 30.X X>3.14.AND.(Y<Z+5.) NT/PLOT 30.X 1.AND.MASK(3).OR.Z<10
Numerical expression	Any **numerical** combination of constants and Ntuple variables. In this case the value of the expression will be applied as a **weight** to the element being plotted.	NT/PLOT 30.X Y weight X by Y NT/PLOT 30.X X**2+Y**2 weight X by X^2+Y^2
Selection function	Name of a selection function in a text file of the form fun.ftn (Unix), FUN FORTRAN (IBM) and FUN.FOR (VAX). The function value is applied as a **weight**	NTUPLE/PLOT 30.X SELECT.FOR For each event the plotted value of X will be multiplied by the value of the selection function SELECT calculated for that event.
	Any combination of the above	NT/PL 30.3%F1.FTN*SIN(X) 1.OR.F2.FTN

Table 7.2: Syntax of a selection function used with a Ntuple

7.5.3 Ntuple selection mechanisms

With most Ntuple operations a selection "function" `UWFUNC` of a form described in table 7.2 can be used, i.e. it can take the form of a simple or composed **expression** or an **external FORTRAN function**, executed by COMIS [1], a **cut** or a **mask**. When used together with the `NTUPLE/PLOT` command the selection function also acts as a **weighting** factor.

7.5.4 Masks

The mask facility allows the user to specify up to **32** selection criteria associated with a Ntuple. These criteria are defined like cuts, but their value for each event are written to an external direct access file, from which the information can be readily retrieved at a later stage, without recalculating the condition value in question. In the example session below first a **new** mask file `MNAME.MASK` is defined, which can contain information for up to 10000 Ntuple elements. Next we define event election criteria and store their result at various bit positions in the mask vector `MNAME`.

Defining cuts and masks

```
PAW > NT/CUT 4 Z>X**2                    | Define cut 4
PAW > NT/MASK MNAME N 10000
PAW > NT/PLOT 30.X X**2+Y**2>2>>MNAME(1)
PAW > NT/PLOT 30.X 4.AND.Y>1>>MNAME(2)
PAW > NT/PLOT 30.Y SIN(Z).GT.SIN(Y)>>MNAME(3)
PAW > NT/MASK MNAME P                    | Print mask definitions

  =====> Current active selections in mask MNAME

  Bit  Nevents    Selection
    1    3723     X**2+Y**2>2
    2    1558     4.AND.Y>1
    3    7051     SIN(Z)>SIN(Y)
PAW > NT/MASK MNAME C                    | close MNAME.MASK file
```

Of course doing this kind of gymnastics makes sense only if a **time consuming** selection mechanism is used and only a few events are selected. In a subsequent run the mask file can then be read to display the information much more quickly.

Using a mask file of a previous run

```
PAW > NT/MASK MNAME                      | open the mask file for read
PAW > NT/PLOT 30.X MNAME(1)              | plot using bit 1
PAW > NT/PLOT 30.X MNAME(2)              | plot using bit 2
PAW > NT/PLOT 30.Y MNAME(3)              | plot using bit 3
PAW > NT/MASK MNAME C                    | close MNAME.MASK file
```

Cuts

A **cut** is identified by an integer (between `0` and `100`) and is a **logical** expression of Ntuple elements, other cuts, masks or functions.

Example of cuts

```
PAW > NT/CUT 1 4<X                        | variable
PAW > NT/CUT 2 0.4<X<0.8.AND.Y<SQRT(Z)    | ditto
PAW > NT/CUT 3 FUN.FOR                    | external function
PAW > NT/CUT 4 FUN.FOR.AND.Z>X**2         | ditto plus variable
PAW > NT/CUT 5 (1.AND.2).OR.4             | combination of cuts
PAW > NT/CUT 6 1.AND.Z<0                  | cut and variable
PAW > NT/CUT 7 X                          | event weight
PAW > NT/CUT 8 SQRT(Y)                    | ditto
PAW > NT/CUT 9 MASK(23).AND.8             | mask and cut
```

Cut definitions can be written to a file and later re-read.

```
PAW > NT/CUT 0 W cuts.dat                 | write all cuts to file
PAW > NT/CUT 4 R cuts.dat                 | read cut 4 from file
PAW > NT/CUT 4 P                          | print cut 4

   CUT number=  4 Points= 1  Variable=  1
            FUN.FOR.AND.Z>X**2
```

Graphical cut

One can also define a cut on the screen in a **graphical** way, by pointing out the upper and lower limits (1-dimensional case) or an area defined by up to 20 points (2-dimensional case) by using the mouse or arrow keys (see figure 7.8).

Note that graphical cuts are only valid for the **original** Ntuple variables and not for combinations of the latter.

Using graphical cuts

```
PAW > nt/pl 30.x%y                                  | plot y versus x
PAW > CUT 1 G                                       | graphical cut 1 for current plot
PAW > zon 1 2                                       | define picture layout
PAW > title 'Graphical cuts'                        | title for picture
PAW > 2d 211 'X versus Y' 50 -2.5 2.5 50 -2.5 2.5 0. | user binning
PAW > 1d 212 'X - Before and after cut' 60 -3. 3. 0. |    ditto
PAW > 1d 213 'Y - Before and after cut' 60 -3. 3. 0. |    ditto
PAW > nt/pl 30.x%y ! -211                           | plot y versus x in histogram 211
PAW > cut 1 d                                       | draw graphical cut 1
PAW > zon 2 2 3 s                                   | redefine the picture layout
PAW > nt/pl 30.x ! -212                             | plot x BEFORE cut in histogram 212
PAW > set htyp -3                                   | use hatch for plot after cut
PAW > nt/pl 30.x 1 -212 ! !  S                      | plot x AFTER cut on same plot
PAW > set htyp 0                                    | no hatch for plot without cut
PAW > nt/pl 30.y ! -213                             | plot y BEFORE cut in histogram 213
PAW > set htyp -3                                   | use hatch for plot after cut
PAW > nt/pl 30.y 1 -213 ! !  S                      | plot y AFTER cut on same plot
```

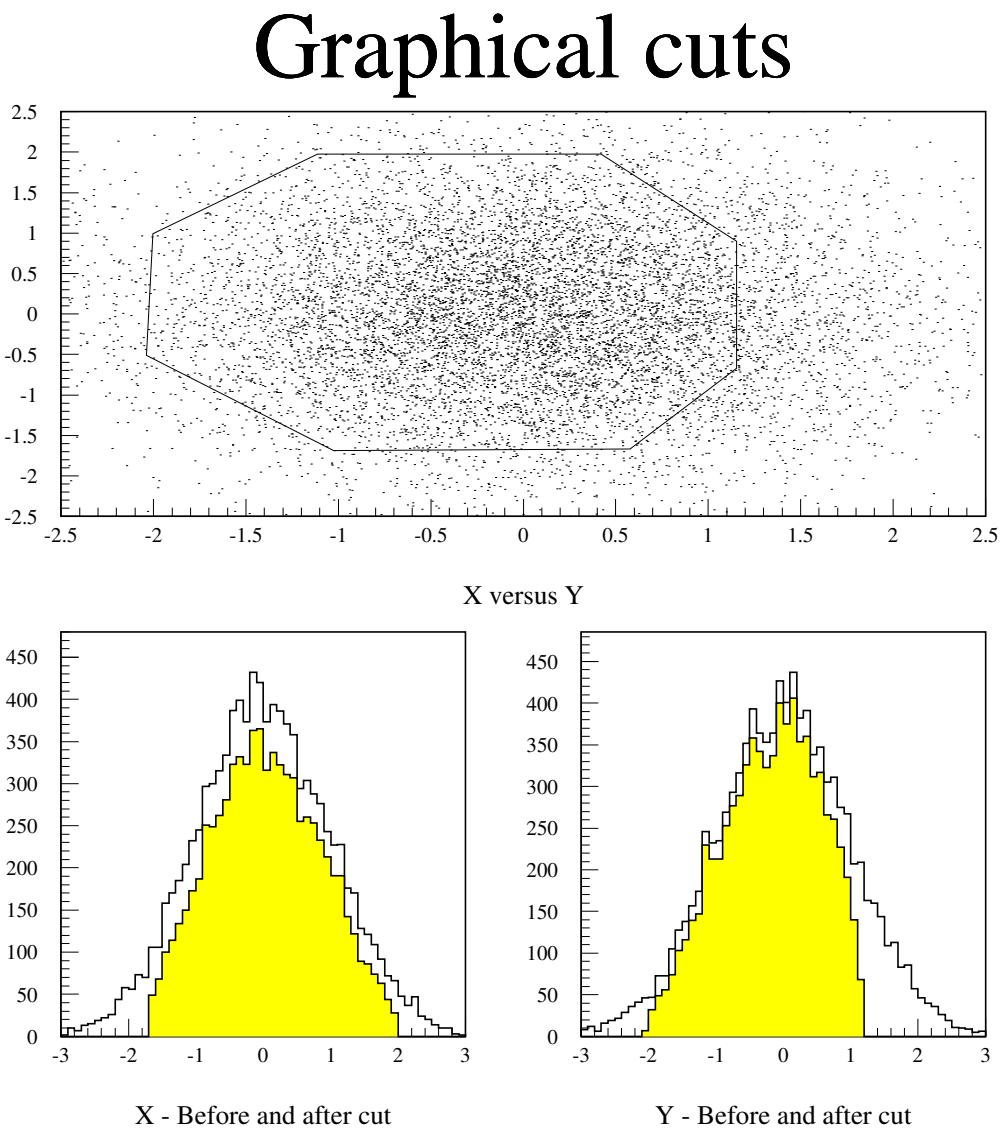


Figure 7.8: Graphical definition of cuts

COMIS selection function

In the definition of a selection criterion an external function (in the sense that it has not been compiled and linked together with PAW) can be used. This function is interpreted by the COMIS [1] package. The functions which are callable from within such a function are given below.

Type of function	List of callable routines
FORTRAN library	SQRT LOG LOG10 EXP SIN COS TAN ASIN ACOS ATAN2 ABS MOD MIN MAX INT REAL DBLE LEN INDEX
HBOOK package	HBOOK1 HBOOK2 HBOOKN HFILL HF1 HPRINT HDELET HRESET HFITGA HFITPO HFITEX HPROJ1 HPROJ2 HFN HGNPAR HROPEN PAOPEN PACLOS PAREAD PAWRIT HPAK HPAKE HUNPAK HGIVE HGN HGNF HF2 HFF1 HFF2 HBFUN1 HBFUN2 HRIN HROUT HI HIE HIX HIJ HIDALL HNOENT HX HXY HCOPY HSTATI HBPROF HOPERA HIDOPT HDERIV HRNDM1 HRNDM2 HBARX HBARY
ZEBRA package	FZIN FZOUT FZENDI FZENDO FZFILE RZCDIR RZLDIR RZFILE RZEND RZIN RZOUT RZVIN RZVOUT
HPLOT package	HPLOT HPLSYM HPLERR HPLEGO HPLNT HPLSUR HPLSOF HPLSET HPLGIV HPLOC HPLSET HPLGIV HPLOC
KUIP package	KUGETV KUDPAR KUVECT KILEXP KUTIME KUEXEL
HIGZ package	IPL IPM IFA IGTEXT IGBOX IGAXIS IGPIE IGRAPH IGHIST IGARC IGLBL IGRNG IGMETA IGSA IGSET IRQLC IRQST ISELNT ISFAIS ISFASI ISLN ISMK ISVP ISWN ITX ICLRWK ISCR
KERNLIB library	JBIT JBYT LENNOC RANNOR RNDM SBITO SBIT1 SBYT UCOPY UCTOH UHTOC VZERO
COMMON blocks	/PAWC/, /QUEST/, /KCWORK/, /PAWPAR/, /PAWIDN/

Table 7.3: Function callable and common blocks which can be referenced from an external function with PAW.

The command NTUPLE/UWFUNC allows a selection function for a Ntuple to be prepared more easily. It generates a function with a name specified by the user and with code making available the variables corresponding to the given Ntuple identifier via a COMMON block. As an example consider the Ntuple number 30 used previously.

Specifying a user selection function

```
 PAW > NTUPLE/UWFUNC 30 SELECT.FOR PT   | Generate SELECT.FOR
 PAW > EDIT SELECT.FOR                 | Look at file SELECT.FOR
      REAL FUNCTION SELECT(XDUMMY)
      REAL X    ,   Y    ,   Z
      COMMON/PAWIDN/IDNEVT,VIDN1,VIDN2,VIDN3,
     +    X    ,   Y    ,   Z
      DIMENSION XDUMMY(  3)
      CHARACTER*8 CHTAGS(  3)
      DATA CHTAGS/'   X    ','   Y    ','   Z    '/
*
      SELECT=1.
      PRINT 1000,IDNEVT
      DO 10 I=1,  3
         PRINT 2000,I,CHTAGS(I),XDUMMY(I)
  10  CONTINUE
*
 1000 FORMAT(8H IDNEVT=,I5)
 2000 FORMAT(5X,I3,5X,A,1H=,G14.7)
      END
```

The user can add further FORTRAN code with the command `EDIT`. Remember that the value of the function can be used for weighting each event.

7.5.5 Examples

To put into practice the syntax explained above let us consider figure 7.9. We first plot variable `Z` with the binning automatically calculated by HBOOK. Then we define a histogram with identifier `300` into which we want HBOOK to plot the squared sums of the elements `X` and `Y`. This corresponds to the definition of the `Z` variable as can be seen in the FORTRAN listing in figure 7.5. As the `MEAN` and `RMS` are only calculated on the events within the histogram boundaries, they differ slightly between the top and bottom plot in figure 7.9.

Plotting Ntuples

```
PAW > zon 1 2                          | 2 histograms one above the other
PAW > opt STAT                         | Write statistics on plot
PAW > NT/PL 30.Z                       | plot variable Z of Ntuple 30
PAW > 1d 300 'Z recalculated and user binning' 100 0. 10.
PAW > NT/PL 30.X**2+Y**2 ! -300        | Recalculate variable Z + plot with user binning
```

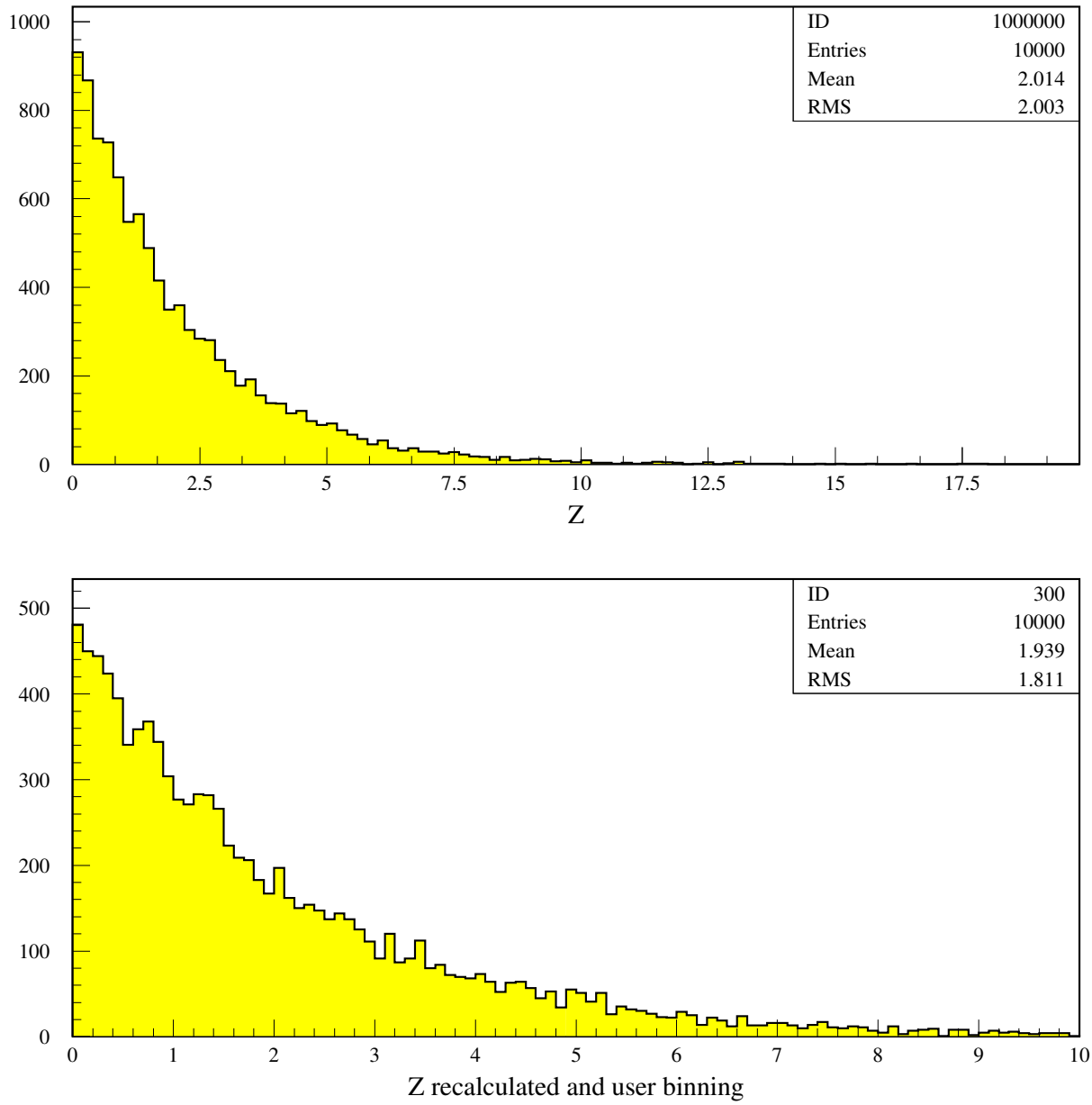


Figure 7.9: Read and plot Ntuple elements

More complex Ntuple presentations

```
PAW > zon 2 2                                  | Divide plot in 4 zones
PAW > opt STAT                                 | Select option to write statistics on plot
PAW > set HTYP -3                              | Define histogram hatch type
PAW > 1d 401 'NT/PL - X' 100. -2.5 2.5         | Book 1 dim histogram
PAW > nt/pl 30.1 ! -401                        | Plot variable 1 (x) using histogram 401
PAW > 1d 402 'NT/PL E option - Y' 100. -2.5 2.5 | 1 dim histogram (different title)
PAW > igset mtyp 21                            | Select market type for points on plot
PAW > nt/pl 30.y ! -402 ! ! E                  | Plot y variable with Error bar option
PAW > 1d 403 'NT/PL B option - X' 40. -2.5 2.5 | 1 dim histogram (different title + binning)
PAW > set barw 0.4                             | Define bar width for bar chart
PAW > set baro 0.3                             | Define bar origin for bar chart
PAW > csel NB 0.33                             | Print selection criterion on plot
PAW > set hcol 1001                            | Histogram colour black
PAW > nt/pl 30.x y>0 -403 ! ! b                | Plot x variable as bar chart
PAW > 1d 404 'NT/PL PL option - Y' 100. -2.5 2.5 | 1 dim histogram (different title)
PAW > max 404 160                              | Fix maximum for plotting hist 404
PAW > nt/pl 30.y sqrt(z)>1 -404 ! ! pl         | Plot y variable with PL option
```

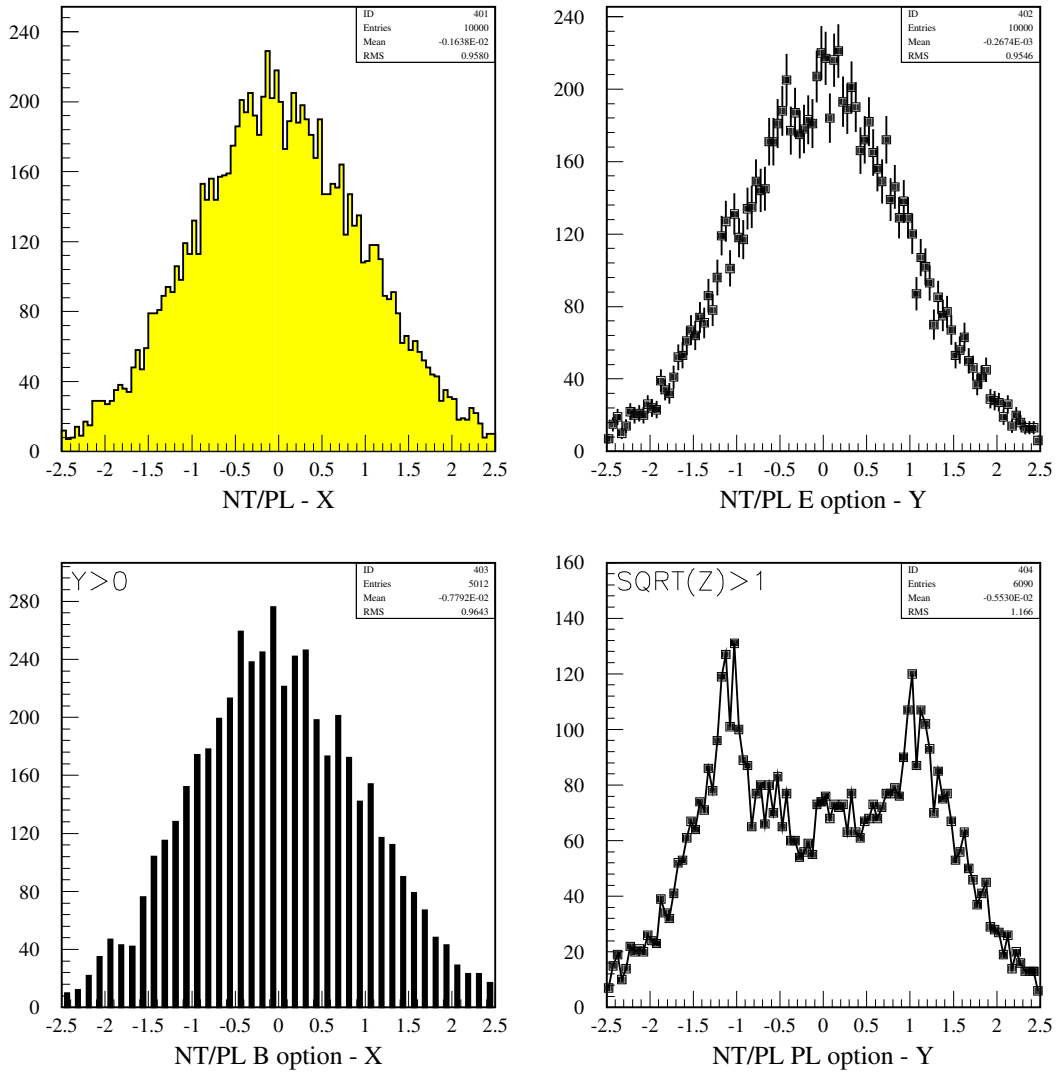


Figure 7.10: Selection functions and different data presentations

7.6 Fitting with PAW/HBOOK/MINUIT

Minuit[5][3] is conceived as a tool to find the minimum value of a multi-parameter function and analyze the shape of the function around the minimum. The principal application is foreseen for statistical analysis, working on chisquare or log-likelihood functions, to compute the best-fit parameter values and uncertainties, including correlations between the parameters. It is especially suited to handle difficult problems, including those which may require guidance in order to find the correct solution.

7.6.1 Basic concepts of MINUIT.

The MINUIT package acts on a multiparameter FORTRAN function to which one must give the generic name `FCN`. In the PAW/HBOOK implementation, the function `FCN` is called `HFCNH` when the command `Histo/Fit` (PAW) or the routine `HFITH` are invoked. It is called `HFCNV` when the command `Vector/Fit` or the routine `HFITV` are invoked. The value of `FCN` will in general depend on one or more variable parameters.

To take a simple example, suppose the problem is to fit a polynomial through a set of data points with the command Vector/Fit. Routine `HFCNV` called by `HFITV` calculates the chisquare between a polynomial and the data; the variable parameters of `HFCNV` would be the coefficients of the polynomials. Routine `HFITV` will request MINUIT to minimize `HFCNV` with respect to the parameters, that is, find those values of the coefficients which give the lowest value of chisquare.

7.6.2 Basic concepts - The transformation for parameters with limits.

For variable parameters with limits, MINUIT uses the following transformation:

$$P_{\text{int}} = \arcsin\left(2\frac{P_{\text{ext}} - a}{b - a} - 1\right) \qquad P_{\text{ext}} = a + \frac{b - a}{2}\left(\sin P_{\text{int}} + 1\right)$$

so that the internal value P_{int} can take on any value, while the external value P_{ext} can take on values only between the lower limit a and the upper limit b. Since the transformation is necessarily non-linear, it would transform a nice linear problem into a nasty non-linear one, which is the reason why limits should be avoided if not necessary. In addition, the transformation does require some computer time, so it slows down the computation a little bit, and more importantly, it introduces additional numerical inaccuracy into the problem in addition to what is introduced in the numerical calculation of the `FCN` value. The effects of non-linearity and numerical roundoff both become more important as the external value gets closer to one of the limits (expressed as the distance to nearest limit divided by distance between limits). The user must therefore be aware of the fact that, for example, if he puts limits of $(0, 10^{10})$ on a parameter, then the values 0.0 and 1.0 will be indistinguishable to the accuracy of most machines.

The transformation also affects the parameter error matrix, of course, so MINUIT does a transformation of the error matrix (and the "parabolic" parameter errors) when there are parameter limits. Users should however realize that the transformation is only a linear approximation, and that it cannot give a meaningful result if one or more parameters is very close to a limit, where $\partial P_{\text{ext}}/\partial P_{\text{int}} \approx 0$. Therefore, it is recommended that:

– Limits on variable parameters should be used only when needed in order to prevent the parameter from taking on unphysical values.

[3] The following information about Minuit has been extracted from the Minuit documentation.

– When a satisfactory minimum has been found using limits, the limits should then be removed if possible, in order to perform or re-perform the error analysis without limits.

7.6.3 How to get the right answer from MINUIT.

MINUIT offers the user a choice of several minimization algorithms. The `MIGRAD` (Other algorithms are available with Interactive MINUIT, as described on Page 280) algorithm is in general the best minimizer for nearly all functions. It is a variable-metric method with inexact line search, a stable metric updating scheme, and checks for positive-definiteness. Its main weakness is that it depends heavily on knowledge of the first derivatives, and fails miserably if they are very inaccurate. If first derivatives are a problem, they can be calculated analytically inside the user function and communicated to PAW via the routine `HDERIV`.

If parameter limits are needed, in spite of the side effects, then the user should be aware of the following techniques to alleviate problems caused by limits:

Getting the right minimum with limits.

If MIGRAD converges normally to a point where no parameter is near one of its limits, then the existence of limits has probably not prevented MINUIT from finding the right minimum. On the other hand, if one or more parameters is near its limit at the minimum, this may be because the true minimum is indeed at a limit, or it may be because the minimizer has become "blocked" at a limit. This may normally happen only if the parameter is so close to a limit (internal value at an odd multiple of $\pm\frac{\pi}{2}$ that MINUIT prints a warning to this effect when it prints the parameter values.

The minimizer can become blocked at a limit, because at a limit the derivative seen by the minimizer $\partial F/\partial P_{\rm int}$ is zero no matter what the real derivative $\partial F/\partial P_{\rm ext}$ is.

$$\frac{\partial F}{\partial P_{\rm int}} = \frac{\partial F}{\partial P_{\rm ext}}\frac{\partial P_{\rm ext}}{\partial P_{\rm int}} = \frac{\partial F}{\partial P_{\rm ext}} = 0$$

Getting the right parameter errors with limits.

In the best case, where the minimum is far from any limits, MINUIT will correctly transform the error matrix, and the parameter errors it reports should be accurate and very close to those you would have got without limits. In other cases (which should be more common, since otherwise you wouldn't need limits), the very meaning of parameter errors becomes problematic. Mathematically, since the limit is an absolute constraint on the parameter, a parameter at its limit has no error, at least in one direction. The error matrix, which can assign only symmetric errors, then becomes essentially meaningless.

7.6.4 Interpretation of Parameter Errors:

There are two kinds of problems that can arise: the **reliability** of MINUIT's error estimates, and their **statistical interpretation**, assuming they are accurate.

Statistical interpretation:

For discussuion of basic concepts, such as the meaning of the elements of the error matrix, or setting of exact confidence levels, see [12, 13, 14].

Reliability of MINUIT error estimates.

MINUIT always carries around its own current estimates of the parameter errors, which it will print out on request, no matter how accurate they are at any given point in the execution. For example, at initialization, these estimates are just the starting step sizes as specified by the user. After a `MIGRAD` or `HESSE` step, the errors are usually quite accurate, unless there has been a problem. MINUIT, when it prints out error values, also gives some indication of how reliable it thinks they are. For example, those marked `CURRENT GUESS ERROR` are only working values not to be believed, and `APPROXIMATE ERROR` means that they have been calculated but there is reason to believe that they may not be accurate.

If no mitigating adjective is given, then at least MINUIT believes the errors are accurate, although there is always a small chance that MINUIT has been fooled. Some visible signs that MINUIT may have been fooled are:

- Warning messages produced during the minimization or error analysis.
- Failure to find new minimum.
- Value of `EDM` too big (estimated Distance to Minimum).
- Correlation coefficients exactly equal to zero, unless some parameters are known to be uncorrelated with the others.
- Correlation coefficients very close to one (greater than 0.99). This indicates both an exceptionally difficult problem, and one which has been badly parameterized so that individual errors are not very meaningful because they are so highly correlated.
- Parameter at limit. This condition, signalled by a MINUIT warning message, may make both the function minimum and parameter errors unreliable. See the discussion above "*Getting the right parameter errors with limits*".

The best way to be absolutely sure of the errors, is to use "independent" calculations and compare them, or compare the calculated errors with a picture of the function. Theoretically, the covariance matrix for a "physical" function must be positive-definite at the minimum, although it may not be so for all points far away from the minimum, even for a well-determined physical problem. Therefore, if MIGRAD reports that it has found a non-positive-definite covariance matrix, this may be a sign of one or more of the following:

A non-physical region: On its way to the minimum, MIGRAD may have traversed a region which has unphysical behaviour, which is of course not a serious problem as long as it recovers and leaves such a region.

An underdetermined problem: If the matrix is not positive-definite even at the minimum, this may mean that the solution is not well-defined, for example that there are more unknowns than there are data points, or that the parameterization of the fit contains a linear dependence. If this is the case, then MINUIT (or any other program) cannot solve your problem uniquely, and the error matrix will necessarily be largely meaningless, so the user must remove the underdeterminedness by reformulating the parameterization. MINUIT cannot do this itself.

Numerical inaccuracies: It is possible that the apparent lack of positive-definiteness is in fact only due to excessive roundoff errors in numerical calculations in the user function or not enough precision. This is unlikely in general, but becomes more likely if the number of free parameters is very large, or if the

parameters are badly scaled (not all of the same order of magnitude), and correlations are also large. In any case, whether the non-positive-definiteness is real or only numerical is largely irrelevant, since in both cases the error matrix will be unreliable and the minimum suspicious.

An ill-posed problem: For questions of parameter dependence, see the discussion above on positive-definiteness.

Possible other mathematical problems are the following:

Excessive numerical roundoff: Be especially careful of exponential and factorial functions which get big very quickly and lose accuracy.

Starting too far from the solution: The function may have unphysical local minima, especially at infinity in some variables.

7.6.5 Fitting histograms

The general syntax of the command to fit histograms is:

```
HISTOGRAM  id func [ chopt np par step pmin pmax errpar ]
```

Only the parameters, which are of more general use, are described in detail. The full description can be found in part 3 of this manual.

`ID` A histogram identifier (1-dim or 2-dim)
A bin range may be specified, e.g. `Histo/Fit 10(25:56) ...`

`FUNC` Name of a function to be fitted to the histogram.
This function can be of various forms:

1. The name of a file which contains the user defined function to be minimized. Function name and file name must be the same. For example file `FUNC.FOR` is:

```
FUNCTION FUNC(X)   or FUNC(X,Y) for a 2-Dim histogram
COMMON/PAWPAR/PAR(2)
FUNC=PAR(1)*X +PAR(2)*EXP(-X)
END
```

2. One of the keywords below **(1-dim histograms only)**, which will use the parameterization described at the right for the fit.

   `G` `Func=par(1)*exp(-0.5*((x-par(2))/par(3))**2)`

   `E` `Func=exp(par(1)+par(2)*x)`

   `Pn` `Func=par(1)+par(2)*x+par(3)*x**2...+par(n+1)*x**n, 0<n<20`

3. A combination of the keywords above with the 2 operators `+` or `*`.

   Note that in this case, the order of parameters in PAR must correspond to the order of the basic functions. Blanks are not allowed in the expression.

`CHOPT` All options of the `HISTO/PLOT` command plus the following additional ones:

`O` Do not plot the result of the fit. By default the fitted function is drawn unless the option "N" below is specified.

- B Some or all parameters are bounded. In this case vectors STEP,PMIN,PMAX must be specified. Default is: All parameters vary freely.
- D The user is assumed to compute derivatives analytically using routine HDERIV. By default, derivatives are computed numerically.
- L Use Log Likelihood method. Default is χ^2 method.
- M Invokes interactive Minuit (See on Page 280)
- N Do not st ore the result of the fit bin by bin with the histogram. By default the function is calculated at the centre of each bin and the fit results stored with the histogram data structure.
- Q Quiet mode. No output printed about the fit.
- V Verbose mode. Results are printed after each iteration. By default only final results are printed.
- W Sets weights equal to 1.

NP Number of parameters in fit ($0 \leq$ NP ≤ 34)

PAR Vector containing the fit parameters.
Before the fit: Vector containing the initial values
After the fit: Vector containing the fitted values.

STEP Vector with step size for fit parameters

PMIN Vector with lower bounds for fit parameters

PMAX Vector with upper bounds for fit parameters

ERRPAR Vector with errors on the fitted parameters

When using predefined functions (case 2 for the FUNC parameter) initial values need not be specified when NP=0. In this case the parameter vector PAR, if specified, is only filled with the fitted parameters on **output**.

7.6.6 A simple fit with a gaussian

Example of simple fit with gaussian in PAW

```
PAW > opt stat     | Select option to show histogram statsitics on plot
PAW > opt fit      | Select option to show fitted parameters on plot
PAW > hi/fit 10 G  | Fit histogram 10 with a single gaussian
    ********************************************
    *                                          *
    * Function minimization by SUBROUTINE HFITGA *
    * Variable-metric method                   *
    * ID =         10  CHOPT = T               *
    *                                          *
    ********************************************
Convergence when estimated distance to minimum (EDM) .LT.  0.10E-03

FCN=   96.97320     FROM MIGRAD    STATUS=CONVERGED  CALLS=  549 EDM=  0.26E-03
                    STRATEGY= 1    ERROR DEF=    1.0000

INT EXT  PARAMETER                                    STEP         FIRST
NO. NO.    NAME        VALUE          ERROR          SIZE      DERIVATIVE
  1  1  Constant      239.83         2.8178       0.00000       0.57627E-02
  2  2  Mean        -0.53038E-02   0.77729E-04   0.00000        22.025
```

```
 3  3  Sigma         0.98766       0.70224E-02   0.00000      -0.88534

CHISQUARE = 0.1021E+01  NPFIT =   98
```

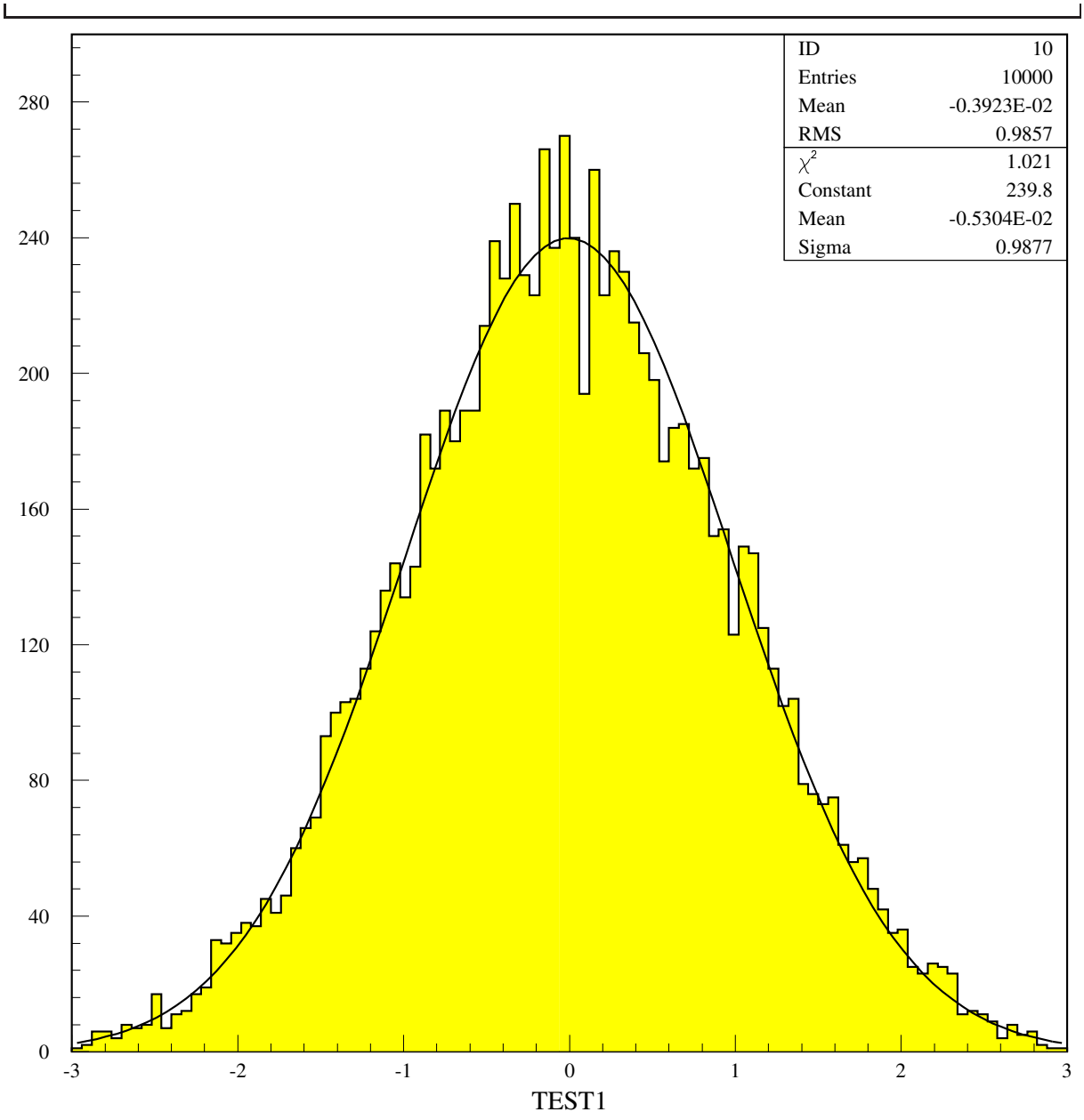


Figure 7.11: Example of a simple fit of a one-dimensional distribution

Fit parts of histogram separately

```
PAW > opt NSTA                        | Turn off option showing statistics on plot
PAW > ve/cr par(6)                    | Create a vector with 6 elements
PAW > set fit 111                     | Show fitted parameters + errors on plot
PAW > hi/fit 110(1:50) G ! 0 par      | Fit first half with a gaussian and plot

    ********************************************
    *                                          *
    * Function minimization by SUBROUTINE HFITGA *
    * Variable-metric method                   *
    * ID =        110  CHOPT = TR              *
    *                                          *
    ********************************************
Convergence when estimated distance to minimum (EDM) .LT.  0.10E-03

FCN=   90.66560     FROM MIGRAD    STATUS=CONVERGED  CALLS=  152 EDM=  0.68E-05
                    STRATEGY= 1    ERROR DEF=    1.0000

INT EXT  PARAMETER                                    STEP         FIRST
NO. NO.    NAME        VALUE          ERROR          SIZE      DERIVATIVE
  1  1  Constant      300.28         5.0681       0.13342       0.97075E-04
  2  2  Mean         0.30698       0.10511E-02  -0.13885E-04  -0.57797
  3  3  Sigma        0.73832E-01   0.67896E-03  -0.57602E-04   -4.6407

CHISQUARE = 0.2159E+01  NPFIT =   45

PAW > hi/fit 110(50:99) G 0 0 par(4)  | Fit second half with gaussian, do not plot

    ********************************************
    *                                          *
    * Function minimization by SUBROUTINE HFITGA *
    * Variable-metric method                   *
    * ID =        110  CHOPT = TR              *
    *                                          *
    ********************************************
Convergence when estimated distance to minimum (EDM) .LT.  0.10E-03

FCN=   30.16534     FROM MIGRAD    STATUS=CONVERGED  CALLS=  221 EDM=  0.87E-04
                    STRATEGY= 1    ERROR DEF=    1.0000

INT EXT  PARAMETER                                    STEP         FIRST
NO. NO.    NAME        VALUE          ERROR          SIZE      DERIVATIVE
  1  1  Constant      153.27         3.0227       0.65005E-01   0.36877E-02
  2  2  Mean         0.70186       0.19599E-02   0.40388E-03    4.8103
  3  3  Sigma        0.11965       0.18242E-02  -0.25292E-03    6.9011

CHISQUARE = 0.6418E+00  NPFIT =   50

PAW > hi/plot 110 SFUNC     | Plot result of fit on Same plot
PAW > ve/pr par(1:6)        | Print the fitted parameters in PAR
PAR (    1 ) =   300.2846
PAR (    2 ) =  0.3069752
PAR (    3 ) =  0.7383241E-01
PAR (    4 ) =   153.2716
PAR (    5 ) =  0.7018576
PAR (    6 ) =  0.1196475
```

Parameter	Input value	Result of Figure 7.12	Result of Figure 7.13
First Gaussian:			
Height	1. (normalised)	$300. \pm 5.$	$308. \pm 5.$
Mean value	0.3	0.307 ± 0.001	0.303 ± 0.001
Width (sigma)	0.07	0.074 ± 0.001	0.070 ± 0.001
Second Gaussian:			
Height	0.5 (normalised)	$153. \pm 3.$	$154. \pm 4.$
Mean value	0.7	0.702 ± 0.002	0.703 ± 0.002
Width (sigma)	0.12	0.120 ± 0.002	0.119 ± 0.002

Table 7.4: Results for the fitted parameters of the gaussian distributions as compared to the initial values which the gaussian distributions were generated in the "batch" job in figure 7.3. The table also includes the result of the double gaussian fit in section 7.13

.

Example of a more complex fit

```
PAW > * Create vector of 6 elements and give initial values for combined fit of two gaussians
PAW > ve/cr par2(6) r 200 0.3 0.1 100 0.7 0.1  | initial values for the 6 fit parameters
PAW > set fit 111                              | display fitted parameters plus errors
PAW > hi/fit  110(2:99) G+G  ! 6 par2          | perform the fit (sum of 2 gaussians)

*******************************************
*                                         *
* Function minimization by SUBROUTINE HFITH  *
* Variable-metric method                  *
* ID =        110  CHOPT = R              *
*                                         *
*******************************************
Convergence when estimated distance to minimum (EDM) .LT.  0.10E-03

FCN=   57.41251     FROM MIGRAD    STATUS=CONVERGED  CALLS=  597 EDM=  0.10E-03
                    STRATEGY= 1    ERROR DEF=    1.0000

INT EXT  PARAMETER                                      STEP         FIRST
NO. NO.    NAME        VALUE          ERROR          SIZE      DERIVATIVE
  1  1  P1            307.86         5.3896         1.3393      -0.51814E-03
  2  2  P2           0.30265       0.10750E-02   0.18577E-03    3.5622
  3  3  P3           0.70029E-01   0.86285E-03   0.19967E-03    11.689
  4  4  P4            153.62         3.0170       0.73111       0.30406E-02
  5  5  P5           0.70303       0.20652E-02   0.43051E-03   -1.2694
  6  6  P6           0.11865       0.18645E-02   0.39360E-03    3.2237

CHISQUARE = 0.6524E+00  NPFIT =   94
```

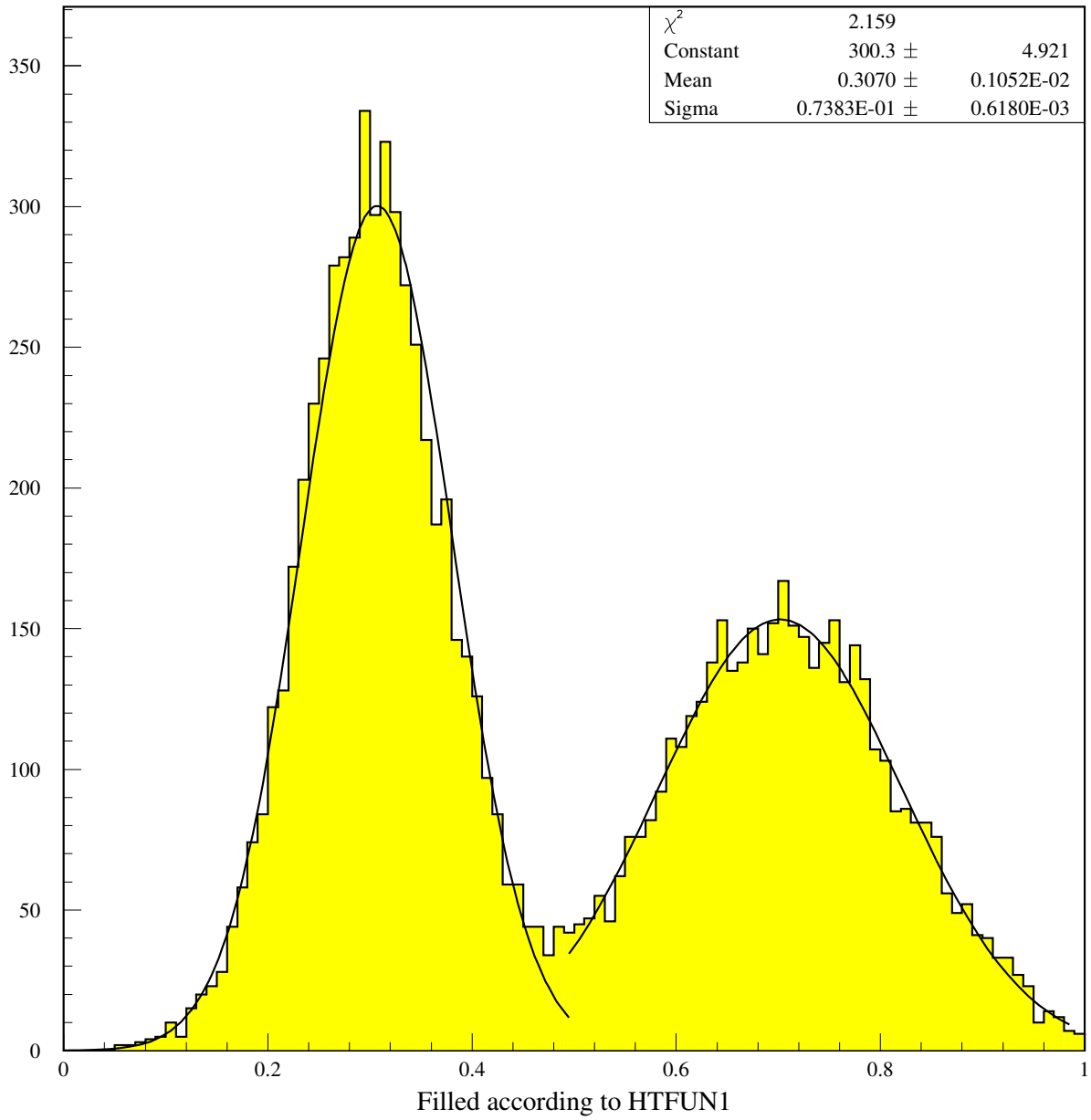


Figure 7.12: Example of a fit using sub-ranges bins

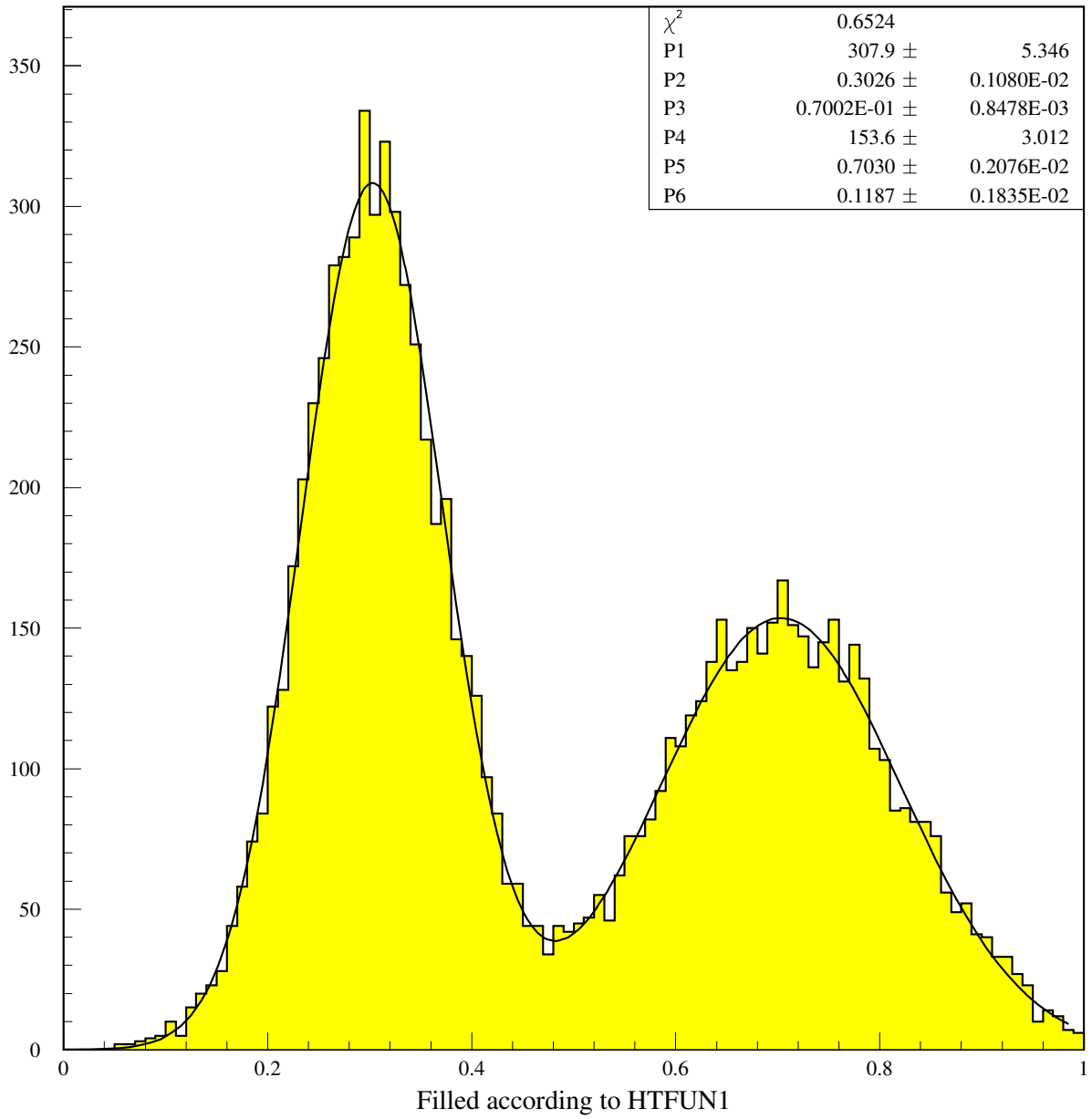


Figure 7.13: Example of a fit using a global double gaussian fit

7.7 Doing more with Minuit

When the `HISTO/FIT` or `VECTOR/FIT` command is invoked, PAW/HBOOK will set a default environment for Minuit. Control may be given to Minuit if the option "`M`" is specified in the command. In this case, the user may enter Minuit control statements.

Overview of available MINUIT commands

CLEar

Resets all parameter names and values to undefined. Must normally be followed by a PARAMETER command or equivalent, in order to define parameter values.

CONtour par1 par2 [devs] [ngrid]

Instructs MINUIT to trace contour lines of the user function with respect to the two parameters whose external numbers are **par1** and **par2**. Other variable parameters of the function, if any, will have their values fixed at the current values during the contour tracing. The optional parameter [**devs**] (default value 2.) gives the number of standard deviations in each parameter which should lie entirely within the plotting area. Optional parameter [**ngrid**] (default value 25 unless page size is too small) determines the resolution of the plot, i.e. the number of rows and columns of the grid at which the function will be evaluated.

EXIT

End of Interactive MINUIT. Control is returned to PAW.

FIX parno

Causes parameter **parno** to be removed from the list of variable parameters, and its value will remain constant (at the current value) during subsequent minimizations, etc., until another command changes its value or its status.

HELP [SET] [SHOw]

Causes MINUIT to list the available commands. The list of SET and SHOw commands must be requested separately.

HESse [maxcalls]

Instructs MINUIT to calculate, by finite differences, the Hessian or error matrix. That is, it calculates the full matrix of second derivatives of the function with respect to the currently variable parameters, and inverts it, printing out the resulting error matrix. The optional argument [**maxcalls**] specifies the (approximate) maximum number of function calls after which the calculation will be stopped.

IMProve [maxcalls]

If a previous minimization has converged, and the current values of the parameters therefore correspond to a local minimum of the function, this command requests a search for additional distinct local minima. The optional argument [**maxcalls**] specifies the (approximate) maximum number of function calls after which the calculation will be stopped.

MIGrad [maxcalls] [tolerance]

Causes minimization of the function by the method of Migrad, the most efficient and complete single method, recommended for general functions (see also MINImize). The minimization produces as a by-product the error matrix of the parameters, which is usually reliable unless warning messages are produced. The optional argument **[maxcalls]** specifies the (approximate) maximum number of function calls after which the calculation will be stopped even if it has not yet converged. The optional argument **[tolerance]** specifies required tolerance on the function value at the minimum. The default tolerance is `0.1`. Minimization will stop when the estimated vertical distance to the minimum (`EDM`) is less than `0.001*[tolerance]*UP` (see `SET ERR`).

MINImize [maxcalls] [tolerance]

Causes minimization of the function by the method of Migrad, as does the MIGrad command, but switches to the SIMplex method if Migrad fails to converge. Arguments are as for MIGrad.

MINOs [maxcalls] [parno] [parno]...

Causes a Minos error analysis to be performed on the parameters whose numbers **[parno]** are specified. If none are specified, Minos errors are calculated for all variable parameters. Minos errors may be expensive to calculate, but are very reliable since they take account of non-linearities in the problem as well as parameter correlations, and are in general asymmetric. The optional argument **[maxcalls]** specifies the (approximate) maximum number of function calls *per parameter requested,* after which the calculation will be stopped for that parameter.

RELease parno

If **parno** is the number of a previously variable parameter which has been fixed by a command: **FIX parno**, then that parameter will return to variable status. Otherwise a warning message is printed and the command is ignored. Note that this command operates only on parameters which were at one time variable and have been FIXed. It cannot make constant parameters variable; that must be done by redefining the parameter with a PARAMETER command.

REStore [code]

If no **[code]** is specified, this command restores all previously FIXed parameters to variable status. If **[code]=1**, then only the last parameter FIXed is restored to variable status.

SCAn [parno] [numpts] [from] [to]

Scans the value of the user function by varying parameter number **[parno]**, leaving all other parameters fixed at the current value. If **[parno]** is not specified, all variable parameters are scanned in sequence. The number of points **[numpts]** in the scan is 40 by default, and cannot exceed 100. The range of the scan is by default 2 standard deviations on each side of the current best value, but can be specified as from **[from]** to **[to]**. After each scan, if a new minimum is found, the best parameter values are retained as start values for future scans or minimizations. The curve resulting from each scan is plotted on the output unit in order to show the approximate behaviour of the function. This command is not intended for minimization, but is sometimes useful for debugging the user function or finding a reasonable starting point.

SEEk [maxcalls] [devs]

Causes a Monte Carlo minimization of the function, by choosing random values of the variable parameters, chosen uniformly over a hypercube centered at the current best value. The region size is by default 3 standard deviations on each side, but can be changed by specifying the value of [**devs**].

SET ERRordef up

Sets the value of **up** (default value= 1.), defining parameter errors. MINUIT defines parameter errors as the change in parameter value required to change the function value by **up**. Normally, for chisquared fits **up=1**, and for negative log likelihood, **up=0.5**.

SET LIMits [parno] [lolim] [uplim]

Allows the user to change the limits on one or all parameters. If no arguments are specified, all limits are removed from all parameters. If [**parno**] alone is specified, limits are removed from parameter [**parno**]. If all arguments are specified, then parameter [**parno**] will be bounded between [**lolim**] and [**uplim**]. Limits can be specified in either order, MINUIT will take the smaller as [**lolim**] and the larger as [**uplim**]. However, if [**lolim**] is equal to [**uplim**], an error condition results.

SET PARameter parno value

Sets the value of parameter **parno** to **value**. The parameter in question may be variable, fixed, or constant, but must be defined.

SET PRIntout level

Sets the print level, determining how much output MINUIT will produce. The allowed values and their meanings are displayed after a **SHOw PRInt** command. Possible values for **level** are:

`-1` No output except from SHOW commands

`0` Minimum output (no starting values or intermediate results)

`1` Default value, normal output

`2` Additional output giving intermediate results.

`3` Maximum output, showing progress of minimizations.

SET STRategy level

Sets the strategy to be used in calculating first and second derivatives and in certain minimization methods. In general, low values of **level** mean fewer function calls and high values mean more reliable minimization. Currently allowed values are 0, 1 (default), and 2.

SHOw XXXX

All **SET XXXX** commands have a corresponding **SHOw XXXX** command. In addition, the SHOw commands listed starting here have no corresponding SET command for obvious reasons. The full list of SHOw commands is printed in response to the command **HELP SHOw**.

SHOw CORrelations

Calculates and prints the parameter correlations from the error matrix.

SHOw COVariance

Prints the (external) covariance (error) matrix.

SIMplex [**maxcalls**] [**tolerance**]

Performs a function minimization using the simplex method of Nelder and Mead. Minimization terminates either when the function has been called (approximately) [**maxcalls**] times, or when the estimated vertical distance to minimum (EDM) is less than [**tolerance**]. The default value of [**tolerance**] is 0.1*UP (see SET ERR).

Chapter 8: Graphics (HIGZ and HPLOT)

8.1 HPLOT, HIGZ and local graphics package

Graphics input/output in PAW is handled by the two packages HPLOT (Histograms PLOTting) and HIGZ (High level Interface to Graphics and Zebra). HIGZ is the basic graphics system of PAW interfacing an basic graphics package while HPLOT, sitting on top of HIGZ, is used for plotting HBOOK objects (Histograms, Ntuples, etc.). The figure below shows the hierarchy between HPLOT, HIGZ and the basic graphics package (GKS, DI3000, X Windows, etc.).

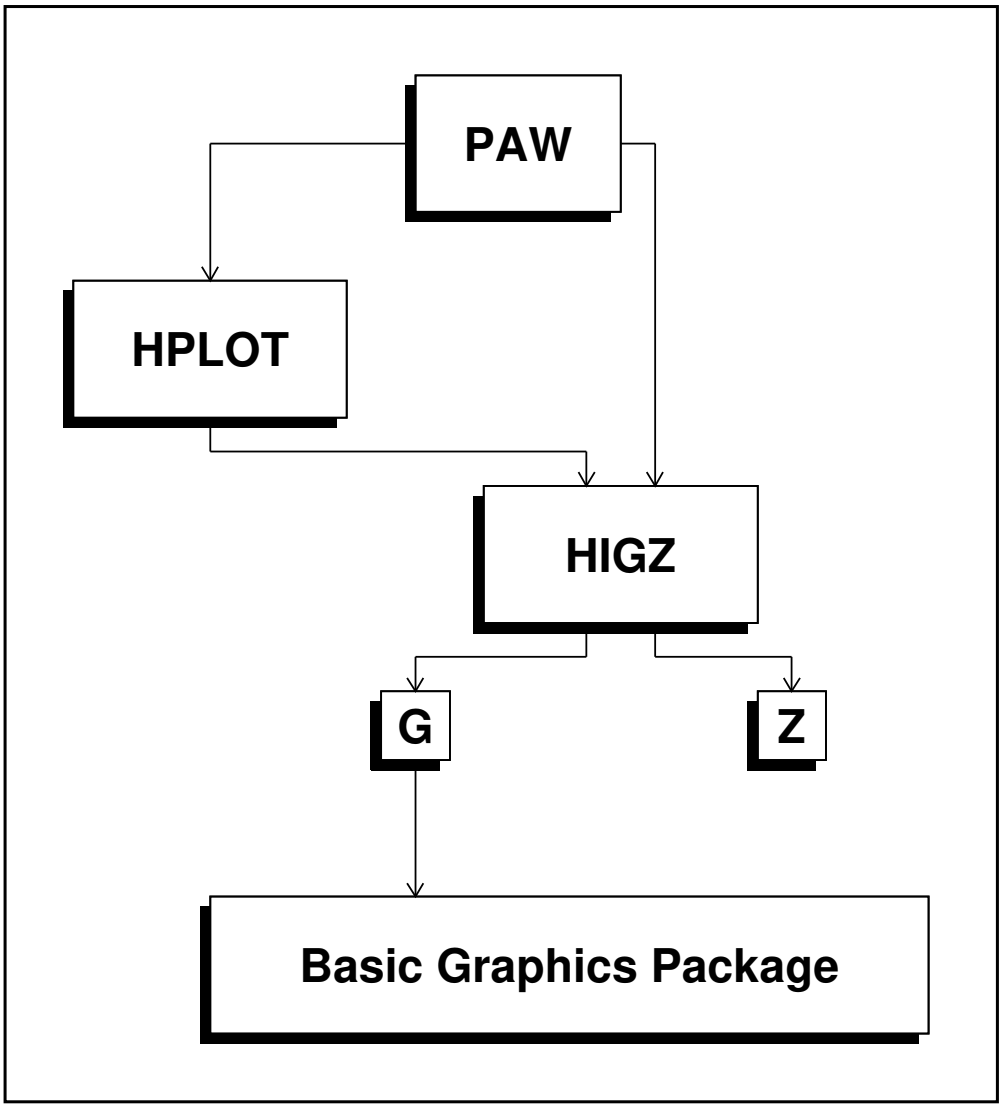


Figure 8.1: HPLOT and HIGZ in PAW

Graphics could be produced in PAW either directly by HIGZ commands or by HPLOT commands. In both cases, all the graphics is under the control of HIGZ. Two distinct modes are available in HIGZ: one is purely graphics (the `G` mode) interfacing the basic graphics package, and the second (the `Z` mode) allows the management of the HIGZ structures (pictures). As an example, the simple PAW command `HISTOGRAM/PLOT` is handled at the different levels as follows:

PAW Level	`HISTOGRAM/PLOT ID`
HPLOT Level	Takes care of `ZONE`, `SET`, `OPTION`, etc.
HIGZ Level	Windows and Viewport, Axis, Boxes, Histogram, Text and Attributes
Basic graphics	Line, Text, Attributes, etc.

8.2 The metafiles

Metafiles are text files used as device independent sources of graphics output for printers of different type. PAW is able to produce two types of metafiles.

The first one is the basic graphics package metafile (for example a GKS metafile). This file is produced by the basic graphics package and it usually needs a special interpreter to be sent to the printers. For example, at CERN, the GKS metafile (workstation type `4`) must be printed with `GRPLOT`.

The second type of metafile is directly produced by HIGZ and is independent from the basic graphics package used. This type of metafile is a PostScript metafile and could be sent directly to a PostScript printer The PostScript workstation types have the following format:

`-[Format][Nx][Ny][Type]`

Where:

`Format` Is an integer between 0 and 99 which defines the format of the paper. For example if `Format=3` the paper is in the standard A3 format. `Format=4` and `Format=0` are the same and define an A4 page. The A0 format is selected by `Format=99`. The US format Letter is selected by `Format=100`. The US format Legal is selected by `Format=200`. The US format Ledger is selected by `Format=300`.

`Nx, Ny` Specify respectively the number of zones on the x and y axis. `Nx` and `Ny` are integers between 1 and 9.

`Type` Can be equal to:

- `1` Portrait mode with a small margin at the bottom of the page.
- `2` Landscape mode with a small margin at the bottom of the page.
- `4` Portrait mode with a large margin at the bottom of the page.
- `5` Landscape mode with a large margin at the bottom of the page.
  The large margin is useful for some PostScript printers (very often for the colour printers) as they need more space to grip the paper for mechanical reasons.
  Note that some PostScript colour printers can also use the so called ”special A4” format permitting the full usage of the A4 area; in this case larger margins are not necessary and `Type=1` or 2 can be used.

3 Encapsulated PostScript. This `Type` permits the generation of files which can be included in other documents, for example in LaTeX files. Note that with this `Type`, `Nx` and `Ny` must always be equal to 1, and `Format` has no meaning. The size of the picture must be specified by the user via the `SIZE` command. Therefore the workstation type for Encapsulated PostScript is -113. For example if the name of an Encapsulated PostScriptfile is `example.eps`, the inclusion of this file into a LaTeX file will be possible via (in the LaTeX file):

```
\begin{figure}
\epsffile{example.eps}
\caption{Example of Encapsulated PostScript in LaTeX.}
\label{EXAMPLE}
\end{figure}
```

Note that all the figures in this manual are included in this way.

With `Type=1,2,4` and `5` the pictures are centered on the page, and the usable area on paper is proportional to the dimensions of A4 format.

Examples:

`-111` or `-4111` defines an A4 page not divided. `-6322` define an A6 landscape page divided in 3 columns and 2 rows.

1	2	3
4	5	6

The first picture will be drawn in the area 1. The next image will appear in the next area in the order defined above. If a page is filled, a new page is used with the same grid. Note that empty pages are not printed in order to save paper.

Ignoring formats smaller than A12, the total number of possible different PostScript workstation types is: $4 \times 9 \times 9 \times 13 + 1 = 4213$!

The command `GRAPHICS/METAFILE LUN METAFL` is designed to produce metafiles. `LUN` is the logical unit number of an open FORTRAN file and `METAFL` the metafile type. For example, the following four commands will produce a HIGZ/PostScript metafile with the name `"PAW.PS"` containing the graphics representation of histogram number `10`:

```
PAW > FORTRAN/FILE 66 PAW.PS
PAW > GRAPHICS/META 66 -111
PAW > HISTO/PLOT 10
PAW > FORTRAN/CLOSE 66
```

8.3 The HIGZ pictures

The HIGZ pictures have four main goals:

- HIGZ graphics primitives and attributes can be stored in a ZEBRA structure in memory in order to display them later.
- They can be stored on direct access files (in a very compact way), in order to build a picture data base.
- They can be modified with the graphics editor.
- They are structured i.e. they can contains so called “graphics objects” which are used to retrieve objects names and type in the “direct graphics mode” of PAW++.

8.3.1 Pictures in memory

The general command to manage pictures in memory is: PICTURE/IZPICT. This command has two parameters:

PNAME Picture name:

- CH Character string specifying picture name (must begin with a letter)
- N Picture number as displayed by PICT/LIST.
- * All pictures in memory.
- ' ' A blank indicates the current picture.

CHOPT Option value:

- AL Give a full listing of the pictures in memory.
- C Picture PNAME becomes the current picture.
- D Display the picture PNAME.
- F First picture in memory becomes the current picture.
- L List pictures in memory.
- M Make a new picture in memory with the name PNAME.
- N Next picture in memory becomes the current picture.
- P Print the contents of the picture PNAME.
- S Scratch picture PNAME from memory.

In addition, simpler and more mnemonic commands are available:

```
PAW > PICT/CREATE PNAME          | Create a picture in memory
PAW > PICT/LIST                  | List pictures in memory
 1: PNAME <-- Current Picture
```

The last created picture in memory is called the **current** picture. All graphics primitives (line, text, histogram, etc.) produced by PAW commands will be stored in this picture if it is **active**, i.e. if mode Z is on.

```
PAW > SWITCH Z                   | Switch Z mode on
PAW > PICT/LIST
 1: PNAME <-- Current Picture (Active)
```

Note that the command PICTURE/CREATE will switch automatically Z mode on.

```
PAW > PICT/PLOT PNAME
```

will display picture PNAME. If picture PNAME is not in memory and if the current working directory (as given by CDIR) is a picture file, PAW will try to take this picture from the file before displaying it.

HIGZ pictures can be created automatically by HPLOT via the command:

```
PAW > OPTION ZFL
```

If this command has been typed, each new plot produced by HPLOT will result in a HIGZ picture created in memory. The following example shows how for each HIST/PLOT ID command a new HIGZ picture is created with an automatic naming:

```
PAW > HIST/PLOT 10
PAW > HIST/PLOT 110
PAW > HIST/PLOT 20
PAW > PICT/LIST
 1: PICT1
 2: PICT2
 3: PICT3 <-- Current Picture (Active)
```

A similar command is given by:

```
PAW > OPTION ZFL1
```

which works exactly like OPTION ZFL except that only the last created picture is kept in memory. For example, if we had typed OPTION ZFL1 instead of OPTION ZFL in the example above, the result would be:

```
PAW > PICT/LIST
 1: PICT3 <-- Current Picture (Active)
```

The following example is a useful macro showing how to use the HIGZ pictures (via OPTION ZFL1) and the metafiles in order to produce a hard copy of the graphics screen:

Macro showing how to convert the current picture in PostScript

```
MACRO POST
FORTRAN/FILE 66 PAW.PS  | Open the FORTRAN file PAW.PS on unit 66
META -66 -111           | PAW.PS is an A4 PostScript file
PICT/PLOT ' '           | Convert the current picture in PostScript
CLOSE 66                | Close PAW.PS
SHELL PRINT PAW.PS      | Send PAW.PS to the local printer
RETURN
```

Typing EXEC POST, the current HPLOT picture on the screen will be sent to the printer using the SHELL command which issues a system-dependent "print" command to the local operating system (e.g. lp or lpr on Unix).
The command PICTURE/PRINT do the same thing:

```
PAW > PICT/PRINT PAW.PS
```

This command transform the current picture into a printable file. The file type is defined according to the extension of the file name i.e.

- **FILE = filename.ps** A PostScript file is generated (-111)
- **FILE = filename.eps** A Encapsulated PostScript file is generated (-113)
- **FILE = filename.tex** A LaTex file is generated (-778)

With this command the metafile type is predefined. It is not possible to change it like in the macro POST previously described. If FILE=HIGZPRINTER or FILE=' ' the PostScript file paw.ps (-111) is generated and the operating system command defined by the environment variable HIGZPRINTER is executed. The environment variable HIGZPRINTER should be defined as follow:
On UNIX sytems:

```
setenv HIGZPRINTER 'lp -dprinter_name paw.ps'
```

or

```
export HIGZPRINTER='lp -dprinter_name paw.ps'
```

On VAX/VMS sytems:

```
HIGZPRINTER == "XPRINT paw.ps /PRINTER=printer_name"
```

On CERNVM:

```
setenv HIGZPRINTER 'XPRINT PAW PS (PR printer_name'
```

Note that if the environment variable `HIGZPRINTER` is not defined the file `paw.ps` is created but not printed.

Other available commands working on pictures in memory are:

```
PAW > PICT/RENAME PNAME PNAME2
PAW > PICT/COPY PNAME PNAME2
PAW > PICT/DELETE PNAME
```

- `PNAME` can be the complete name, the picture number in memory or `' '`.
- `PNAME2` is the complete picture name.

8.3.2 Pictures on direct access files

HIGZ pictures are stored on direct-access files and hence access times to pictures are fast. Moreover, due to the fact that HIGZ uses high level primitives to describe the picture's structural tree, a storage compaction factor as compared to the equivalent GKS metafiles of between `10` and `100` is routinely obtained.

As HIGZ is interfaced to various basic graphics packages, a picture file can be created on one system (e.g. DECGKS, X11, GL etc.) and transported to another machine to be interpreted with a different graphics package (e.g GKSGRAL, GDDM, DI3000 etc.).

All available commands to handle pictures with ZEBRA files are shown below. Note that in the example the picture names could be "*" (all pictures in memory), " " (current picture) or a number (picture number in memory).

Handling pictures with ZEBRA

```
PAW > * Open an existing picture file PICT.DAT on LUN 4 in Update mode
PAW > PICT/FILE 4 PICT.DAT ! U  | Open the existing file PICT.DAT
PAW > LDIR                      | List the content of the file PICT.DAT

************** Directory ===> //LUN4 <===

                Created 890512/1110  Modified 890622/1732

===> List of objects
             PICTURE   NAME                          CYCLE
    UNIX                                               1
    ZEBRA                                              1
    CERN                                               1
    MARKER                                             1

PAW > IZIN CERN                 | Put picture "CERN" in memory
PAW > PICT/LIST                 | List pictures in memory
 1: CERN
PAW > IZOUT CERN                | Store picture "CERN" in PICT.DAT
PAW > LDIR                      | List the content PICT.DAT

************** Directory ===> //LUN4 <===

                Created 890512/1110  Modified 890622/1732

===> List of objects
             PICTURE   NAME                          CYCLE
    UNIX                                               1
    ZEBRA                                              1
    CERN                                               1
                                                       2
    MARKER                                             1

PAW > PURGE                     | Purge the file PICTURES
PAW > SCRATCH ZEBRA             | Delete the picture ZEBRA from PICT.DAT
PAW > LDIR                      | List the content of PICT.DAT

************** Directory ===> //LUN4 <===

                Created 890512/1110  Modified 890622/1732

===> List of objects
             PICTURE   NAME                          CYCLE
    UNIX                                               1
    CERN                                               2
    MARKER                                             1
```

8.3.3 Automatic storage pictures in memory

After typing the command:

```
PAW > IGSET AURZ 1
```

the AURZ mode is on and all the subsequent created pictures are stored automatically in the last picture file opened via the command PICTURE/FILE.

Example of the use of pictures in memory

```
PAW > PICT/FILE 4 PICT.DAT ! N    | Open a new picture file PICT.DAT
PAW > HIST/FILE 3 HEXAM.DAT       | Open an existing histogram RZ file
PAW > LDIR                        | List the contain of HEXAM.DAT

 ************** Directory ===> //LUN3 <===

                Created 880104/1414  Modified 880104/1414

 ===> List of objects
     HBOOK-ID  CYCLE   DATE/TIME   NDATA   OFFSET    REC1    REC2
         10       1   880104/1414     75     725      32
         20       1   880104/1414   1815     800      32      33
         30       1   880104/1414   1066     567      34      35

PAW > OPT ZFL       | Each new plot will result in a HIGZ picture
PAW > IGSET AURZ 1  | Each new HIGZ picture is stored in PICT.DAT
PAW > HIST/PLOT 0   | All histograms in HEXAM.DAT are plotted
PAW > CDIR //LUN4   | Set the current working directory on PICT.DAT
PAW > LDIR          | List the content of PICT.DAT

 ************** Directory ===> //LUN4 <===

                Created 890928/1024  Modified 890928/1024

 ===> List of objects
              PICTURE   NAME                          CYCLE
     PICT1                                              1
     PICT2                                              1
     PICT3                                              1
```

Note that if the command PICTURE/FILE is invoked with the option 'A', the AURZ mode is automatically enable.

8.3.4 HIGZ pictures generated in a HPLOT program

HIGZ pictures can be generated in a batch HPLOT program and later visualized in an interactive session with PAW. The HIGZ picture file, like any HBOOK file, can be exchanged between computers using the FTP in binary mode. As the size of the picture data base (see page 286), and hence the associated disk storage requirements, is much smaller than the size of the metafile generated by the basic graphics package, transfer times are drastically reduced. The example below show how to interactively visualize (with PAW) HIGZ pictures produced by HPLOT. In the same way we can visualize and edit pictures generated by any HIGZ based application (GEANT, event scanning programs, etc.)

Store HPLOT pictures with HIGZ

```
      PROGRAM HPICT
*.==========>
*.  HPLOT Program to demonstrate how to store HPLOT
*.  pictures onto direct access HIGZ picture file
*..=========>
      COMMON/PAWC/H(20000)
      DIMENSION SIG(2)
      CHARACTER*20 TITLE
*.------------------------------------------
*.
      CALL HLIMIT(20000)
* --        Create histograms
      DO 10 ID=1,10
         WRITE(TITLE,1000)ID
 1000    FORMAT('Test number',I3)
         CALL HBOOK1(ID,TITLE,100,-3.,3.,0.)
  10  CONTINUE
* --        Fill histograms
      DO 30 ID=1,10
         DO 20 I=1,1000
            CALL RANNOR(A,B)
            CALL HFILL(ID,A,0.,1.)
  20     CONTINUE
         CALL HFITGA(ID,COEFF,AV,SIGH,CHI2,2,SIG)
  30  CONTINUE
* --        Initialize HPLOT. Set various graphics options.
      CALL HPLINT(0)
      CALL HPLZON(1,2,1,' ')
      CALL HPLOPT('ZFL',1)
      CALL HPLOPT('FIT',1)
      CALL HPLOPT('STAT',1)
      CALL HPLSET('STAT',1.)
      CALL HPLSET('HTYP',244.)
      CALL HPLSET('FWID',5.)
      CALL HPLSET('VFON',-40.)
      CALL HPLSET('TFON',-60.)
      CALL HPLSET('PWID',4.)
      CALL HPLSET('BCOL',1.01)
      CALL HPLSET('CSIZ',0.25)
      CALL HPLSET('CFON',-10.)
*
*   Open a picture file called "hpict.dat".
*   Option 'A' means "Automatic saving of pictures"
*   Option 'N' means "New file"
*   (option 'U' instead of 'N' updates an existing file)
*
      CALL IZOPEN(1,'Pictures','hpict.dat','AN',1024,ISTAT)
*
*   Select HIGZ option to store graphics in ZEBRA memory only
*   No calls to the local graphics package.
*
      CALL IGZSET('Z')
* --     Plot all histograms
      CALL HPLOT(0,' ',' ',0)
      CALL HPLEND
*
      END
```

Using the picture in Paw

```
PAW > PICT/FILE 20 HPICT.DAT
PAW > LDIR
            Directory ===> //LUN20 <===

  Created 891006/1026  Modified 891006/1026

===> List of objects
    PICTURE  NAME                    CYCLE
       PICT1                          1
       PICT2                          1
       PICT3                          1
       PICT4                          1
       PICT5                          1
PAW > META 10 -111
PAW > PICT/PLOT PICT2
PAW > CLOSE 10
PAW > * Print metafile
PAW > *   (see pages 286 and following)
PAW > SHELL print PAW.METAFILE
PAW > EXIT
```

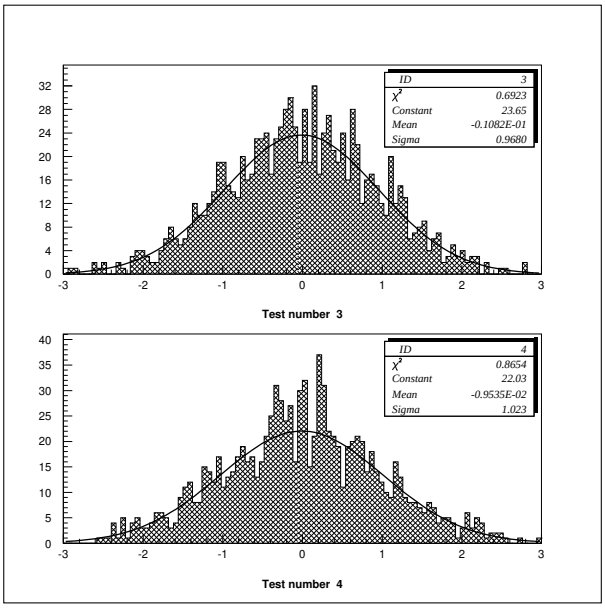


Figure 8.2: Visualising a HIGZ picture produced in a batch HPLOT program

8.4 Setting attributes

Attributes are parameters like: colour, character font, etc. which could be changed interactively in PAW via the commands PICTURE/IGSET, GRAPHICS/SET and GRAPHICS/OPTION. Each attribute is linked to one or more objects (lines, histogram, etc.). The aim of this section is to give a complete description of the attributes available in PAW and to clarify the differences between IGSET, which changes attributes at the HIGZ level, and SET and OPTION, which act at the HPLOT level.

IGSET [CHOPT VAL]

This command is used to set the value of attributes related to primitives and macroprimitives. The first parameter is the mnemonic name of the attribute, the second is the value to be assigned.

CHOPT Character variable specifying the name of the attribute to be set. This a character string of 4 characters.

VAL Value of the attribute. A value of 0 or no value specified, indicates that the attribute value must be reset to its default value.

Examples of IGSET commands

```
PAW > IGSET MTYP 20        | Change marker type to 20.
                           | This new marker is used by all subsequent
                           | commands using the current marker type.

PAW > IGSET LWID           | Set the line width to its default value.

PAW > IGSET                | Display actual and default values of all HIGZ attributes
PAW > IGSET *              | Set ALL HIGZ attributes to their default values
```

OPTION [CHOPT]

The OPTION command has one optional parameter:

CHOPT Option name (four characters). Special values are:

- '*' Set all HPLOT options to their default values
- ' ' Display actual and default values of all HPLOT options

SET [CHOPT VAL]

Sets an HPLOT parameter; see table 8.3 and figures 8.3, 8.4, 8.5 and 8.6 for details.

CHOPT Character variable of length 4 identifying the parameter to be redefined (must be given in uppercase). Special values are:

- '*' All parameters are set to their default values.
- 'SHOW' A list of all parameters and their values is printed.

VAR New value for the parameter specified. Special values are:

- 0. The corresponding parameters is set to its default value.

NAME	default	Explanation
'AURZ'	0.	If 1. the last current picture is automatically saved on disk when a new picture is created.
'AWLN'	0.0	Axis wire length. Default is length=0 (no grid)
'BARO'	0.25	Offset of the left edge of the bar with respect to the left margin of the bin for a bar chart (expressed as a fraction of the bin width).
'BARW'	0.50	Width of the bar in a bar chart (expressed as a fraction of the bin width).
'BASL'	0.01	Basic segment length in NDC space (0-1) by (0-1) for dashed lines
'BORD'	0.	Border flag. If = 1., a border is drawn in boxes, pie charts,....
'CHHE'	0.01	CHaracter HEight.
'CSHI'	0.02	Distance between each shifted drawing of a character (in percentage of character height) for characters drawn by TEXT
'FACI'	1.	Fill Area Colour Index.
'FAIS'	0.	Fill Area Interior Style (0.,1.,2.,3.).
'FASI'	1.	Fill Area Style Index.
'LAOF'	0.013	LAbels OFfset.
'LASI'	0.018	LAbels SIze (in World coordinates).
'LTYP'	1.	Line TYPe.
'LWID'	1.00	Line WIDth.
'MSCF'	1.00	Marker SCale Factor.
'MTYP'	1.	Marker TYPe.
'PASS'	1.	Text width (given by number of PASSes) of characters drawn by TEXT. The width is simulated by shifting the "pen" slightly at each pass.
'PICT'	1.	Starting number for automatic pictures naming.
'PLCI'	1.	PolyLine Colour Index.
'PMCI'	1.	PolyMarker Colour Index.
'TANG'	0.00	Text ANGle (for calculating Character up vector).
'TMSI'	0.019	Tick Marks SIze (in world coordinates)
'TXAL'	0.	10*(horizontal alignment)+(vertical alignment).
'TXCI'	1.	TeXt Colour Index.
'TXFP'	10.	10*(TeXt Font) + (TeXt Precision). (0: hard, 1: string, 2: soft)
'*'		All attributes are set to their default values.
'SHOW'		The current and default values of the parameters controlled by IGSET are displayed.

Table 8.1: Parameters and default values for IGSET

Table 8.2: Parameters and default values for OPTION

Default	Alternative	Effect
' '	'A0', 'A1',...	Picture size. Predefined options are: A0, A1, A2, A3, A4, A5, A6
'NOPG'	'*P','**P', '***P'	Suppresses ('NOPG') or adds a 1, 2 or 3 digit page numbers to a plot (Each '*' stands for a digit). The page numbers are incremented automatically
'NEAH'	'EAH'	Plots Errors bars And Histogram, if both are present
'VERT'	'HORI'	Vertical or horizontal orientation of paper
'NAST'	'AST'	Functions are drawn with ('AST ') or without ('NAST') asterisks in each channel.
'NCHA'	'CHA'	Scatter plot are plotted with dots randomised within each bin ('NCHA') or by printing a single character in the middle of the bin ('CHA ')
'SOFT'	'HARD'	Use SOFTware or HARDware characters
'TAB '	'NTAB'	tables (HTABLE) are plotted as tables ('TAB ') or as scatter plots ('NTAB')
'HTIT'	'UTIT'	Option for printing titles. 'HTIT' means use the hbook titles, while 'UTIT' signals the use of user titles
'LINX'	'LOGX'	The scale for the X axis is linear or logarithmic.
'LINY'	'LOGY'	The scale for the Y axis is linear or logarithmic.
		Note that if in hbook the HIDOPT option 'LOGY' or HLOGAR was selected for a particular ID and if neither options 'LINY' nor 'LOGY' are selected then the scale will be logarithmic. If HLOGAR or HIDOPT with option 'LOGY' was called and the option 'LINY' is selected then the scale will be linear
'LINZ'	'LOGZ'	The scale for the Z axis is linear or logarithmic (for lego plots or surfaces).
'BOX '	'NBOX'	By default a rectangular box is drawn around a picture. 'NBOX' suppresses this box
'NTIC'	'TIC'	Cross-wires are drawn ('TIC ') or not drawn ('NTIC') after each plot
'NSTA'	'STA'	Statistics information are printed ('STA ') or not printed ('NSTA') on the picture
'NFIT'	'FIT'	Fit parameters are printed ('FIT ') or not printed ('NFIT') on the picture
'NSQR'	'SQR'	The size of the histogram boxes is set to the largest square (SQR)
'NZFL'	'ZFL'	The picture is stored ('ZFL ') or not stored ('NZFL') in a ZEBRA data base The procedure to create a higz picture is given below.
'NZFL'	'ZFL1'	'ZFL1' has the same effect as 'ZFL ', but only the picture last created is kept in memory.
'NPTO'	'PTO'	"Please Turn Over". With 'PTO ' a carriage return is requested between each new plot.
'NBAR'	'BAR'	1-dimensional histograms are plotted as "Bar charts" ('BAR ') or as contours ('NBAR')
'DVXR'	'DVXI'	Real ('DVXR') or integer ('DVXI') labels are computed for the X axis
'DVYR'	'DVYI'	Real ('DVYR') or integer ('DVYI') labels are computed for the Y axis

Table 8.2: Overview of the HPLOPT options (continued)

Default	Alternative	Effect
'GRID'	'NGRI'	Grid on X and Y axis
'NDAT'	'NDAT'	The date is printed or not on each plot
'NFIL'	'NFIL'	The file name is printed or not on each plot

Table 8.3: Parameters and default values in SET

CHOPT	VAR (default)	Explanation
ASIZ	0.28 cm	axis label size
BARO	0.25	bar offset for "bar charts"
BARW	0.5	bar width for "bar charts"
BCOL	1	zone fill area colour index
BTYP	0	zone fill area style index
BWID	1	box line width
CFON	2	comment font (10*font+precision)
CSHI	0.03	character shift between two pass
CSIZ	0.28 cm	comment size
DASH	0.15	length of basic dashed segment for dashed lines
DATE	2	date position
DMOD	1	line style for histogram contour (see HPLOT)
ERRX	0.50	error on X (% of bin width)
FCOL	1	function fill area COLor
FILE	1	file name position
FIT	101	fit values to be plotted
FPGN	1	first PaGe Number
FTYP	0	function fill area TYPe
FWID	1	function line width
GFON	2	global title font (10*font+precision)
GRID	3	grid line type
GSIZ	0.28 cm	global title size
HCOL	1	histogram fill area colour index
HMAX	0.90	histogram maximum for scale (in percent)
HTYP	0	histogram fill area style index
HWID	1	histogram line width
KSIZ	0.28 cm	Hershey character size (cf. KEY)
LFON	2	axis labels font (10*font+precision)
NDVX	10510.00	number of divisions for X axis
NDVY	10510.00	number of divisions for Y axis

Table 8.3: Parameters and default values in SET (continued)

CHOPT	VAR **(default)**	**Explanation**
NDVZ	10510.00	number of divisions for Z axis
PASS	1.	number of pass for software characters
PCOL	1	picture fill area colour index
PSIZ	0.28 cm	page number size
PTYP	0	picture fill area style index
PWID	1	picture line width
SMGR	0.	stat margin right (in percent)
SMGU	0.	stat margin up (in percent)
SSIZ	0.28 cm	asterisk size (for functions)
STAT	1111	stat values to be plotted
TFON	2	general comments font (10*font+precision)
TSIZ	0.28 cm	histogram title size
VFON	2	axis values font (10*font+precision)
VSIZ	0.28 cm	axis values size
XCOL	1	X axis COLor
XLAB	1.40 cm	distance Y axis to labels
XMGL	2.00 cm	X margin left
XMGR	2.00 cm	X margin right
XSIZ	20.0 cm	length of picture along X
XTIC	0.30 cm	X axis tick mark length
XVAL	0.40 cm	distance between the Y axis and the axis values
XWID	1	X ticks width
XWIN	2.00 cm	X space between zones
YCOL	1	Y axis COLor
YGTI	1.50 cm	Y position of global title
YHTI	1.20 cm	Y position of histogram title
YLAB	0.80 cm	distance X axis to labels
YMGL	2.00 cm	Y margin low
YMGU	2.00 cm	Y margin up
YNPG	0.60 cm	Y position for the page number
YSIZ	20.0 cm	length of picture along Y
YTIC	0.30 cm	Y axis tick mark length
YVAL	0.20 cm	distance between the X axis and the axis values
YWID	1	Y ticks width
YWIN	2.00 cm	Y space between zones
2SIZ	0.28 cm	scatter plot and table character. size

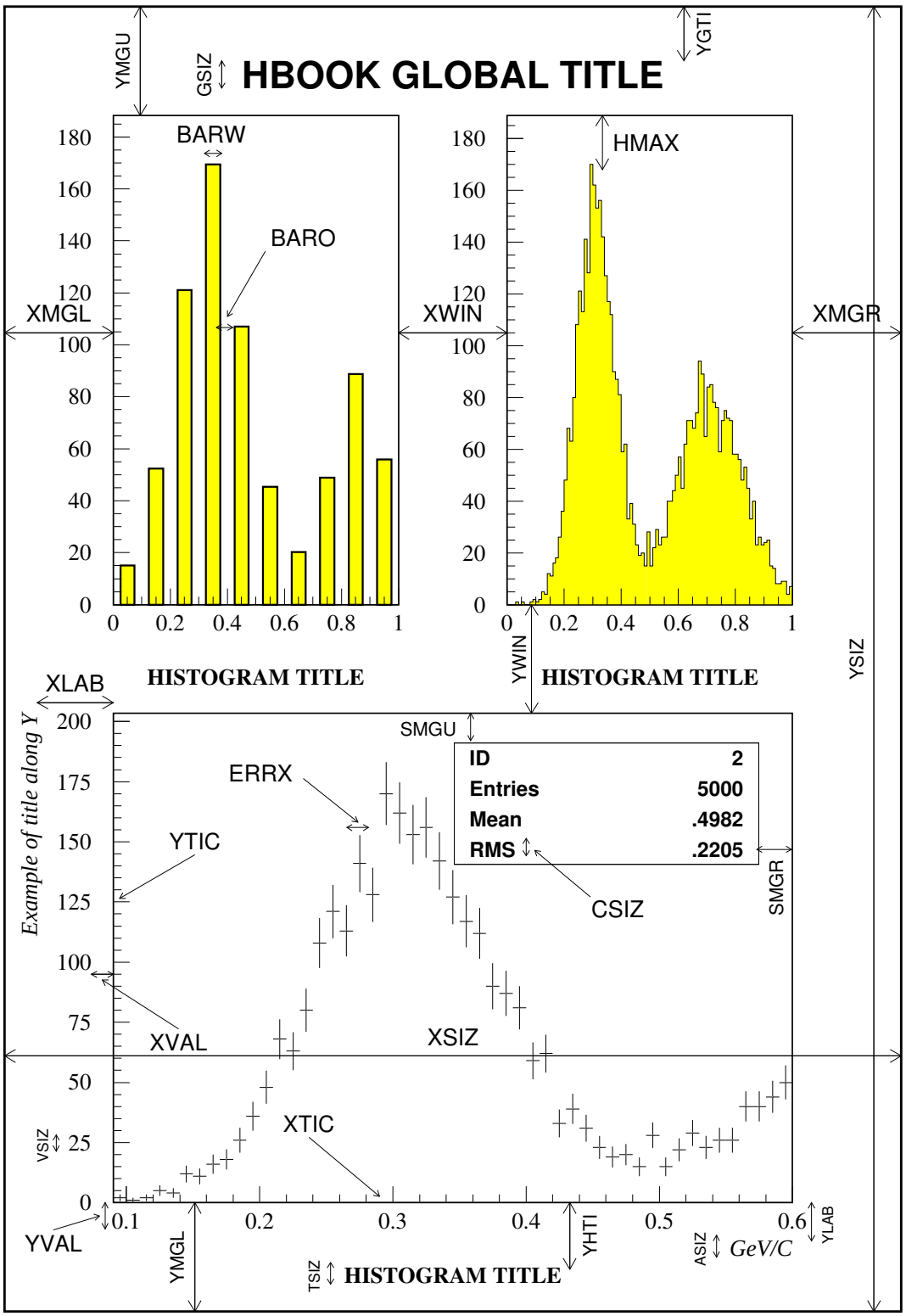


Figure 8.3: A graphical view of the SET parameters

8.5 More on labels

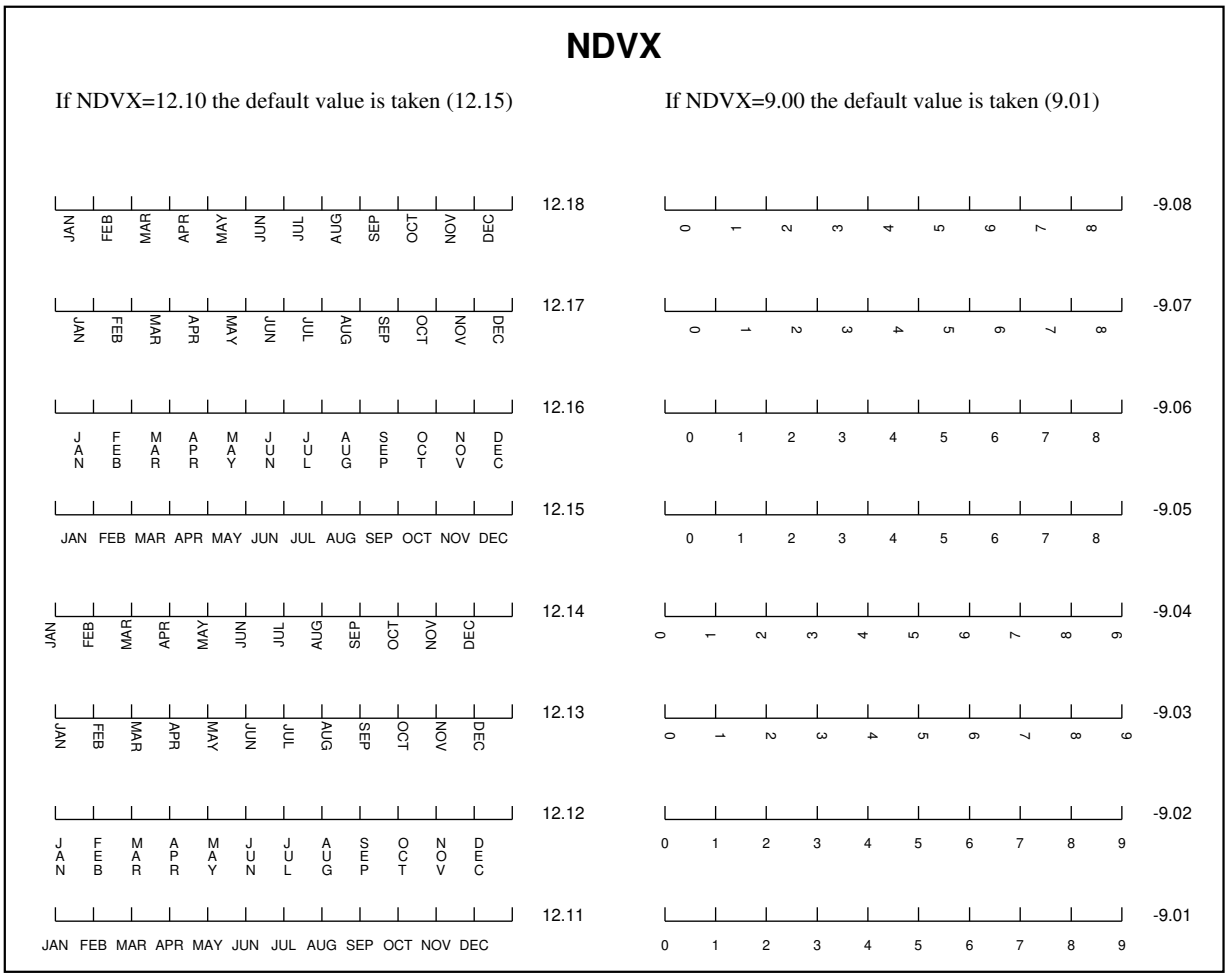


Figure 8.4: Example of labelling for horizontal axes

By default, labels used by `AXIS` and `PIE` are numeric labels. The command `GRAPHICS/PRIMITIVES/LABELS` (or `LABELS` for short), allows the user to define up to nine alphanumeric set of labels (numbered from `1` to `9`). These labels can then be used in subsequent commands using `PIE` or `AXIS` primitives of HIGZ.

The `LABELS` command has three parameters:

`LABNUM` An integer between `1` and `9`. It identifies the labels set.

`NLABS` The number of items to be placed on the labels (up to `50`).

`CHLABS` `NLABS` character strings specifying the label items.

The label sets thus defined can be used for axes on all plots produced by PAW (HPLOT histograms, graphs, vectors drawing, etc.) via the SET NDVX (NDVY) command. These commands have the following structure:

Example of NXDV specification

```
    SET NDVX i            e.g. SET NDVX 512
or
    SET NDVX i.jk         e.g. SET NDVX 10.25
```

In the first case the number i contains 100 times the number of secondary divisions plus the number of primary divisions. (e.g. 512 means 12 primary and 5 secondary division. By adding 10000 times N3 to i a third level of divisions is available.

In the second case the number in front of the dot (i) indicates the total number of divisions, the first digit following the dot (j) the label identifier (LABNUM) (if this number is equal to 0 numeric labels are drawn). The second digit after the (k) dot indicates the position where the labels have to be drawn (i.e. the *text justification* parameter, in this case 5, indicating horizontally written text centered on the interval). Study figures 8.4 and 8.5 for details. These two figures show that the labels can be centered on the tick marks (1 to 4) or on the divisions (5 to 8). If the labels are centered on the tick marks, note that the number of items in the command LABELS must be equal to the number of tick marks (which is equal to the number of divisions **plus one**), otherwise the last alphanumeric label on the axis will be undefined.

By default, the number of primary divisions given by SET NDVX n, SET NDVY n or SET NDVZ n is optimized to have a reasonable labelling. If the number of divisions has to be exactly equal to the number given by SET NDVX n, SET NDVY n or SET NDVZ n, a negative value must be used i.e.:

Forcing an exact number of divisions

```
    SET NDVX -i           e.g. SET NDVX -512
or
    SET NDVX -i.jk        e.g. SET NDVX -10.25
```

For example to label each subsequent X-axis with the names of the months of the year centered in the middle of each bin one can use:

Example of alphanumeric labels on an axis

```
PAW > LABEL 1 12 JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC
PAW > SET NDVX -12.15
```

8.6 Colour, line width, and fill area in HPLOT

The aspect of HPLOT pictures can be modified via the xWID, xTYP and xCOL attributes, where x can be H, B, P, or F, defined as follows:

B zone Box

F Function

H Histogram

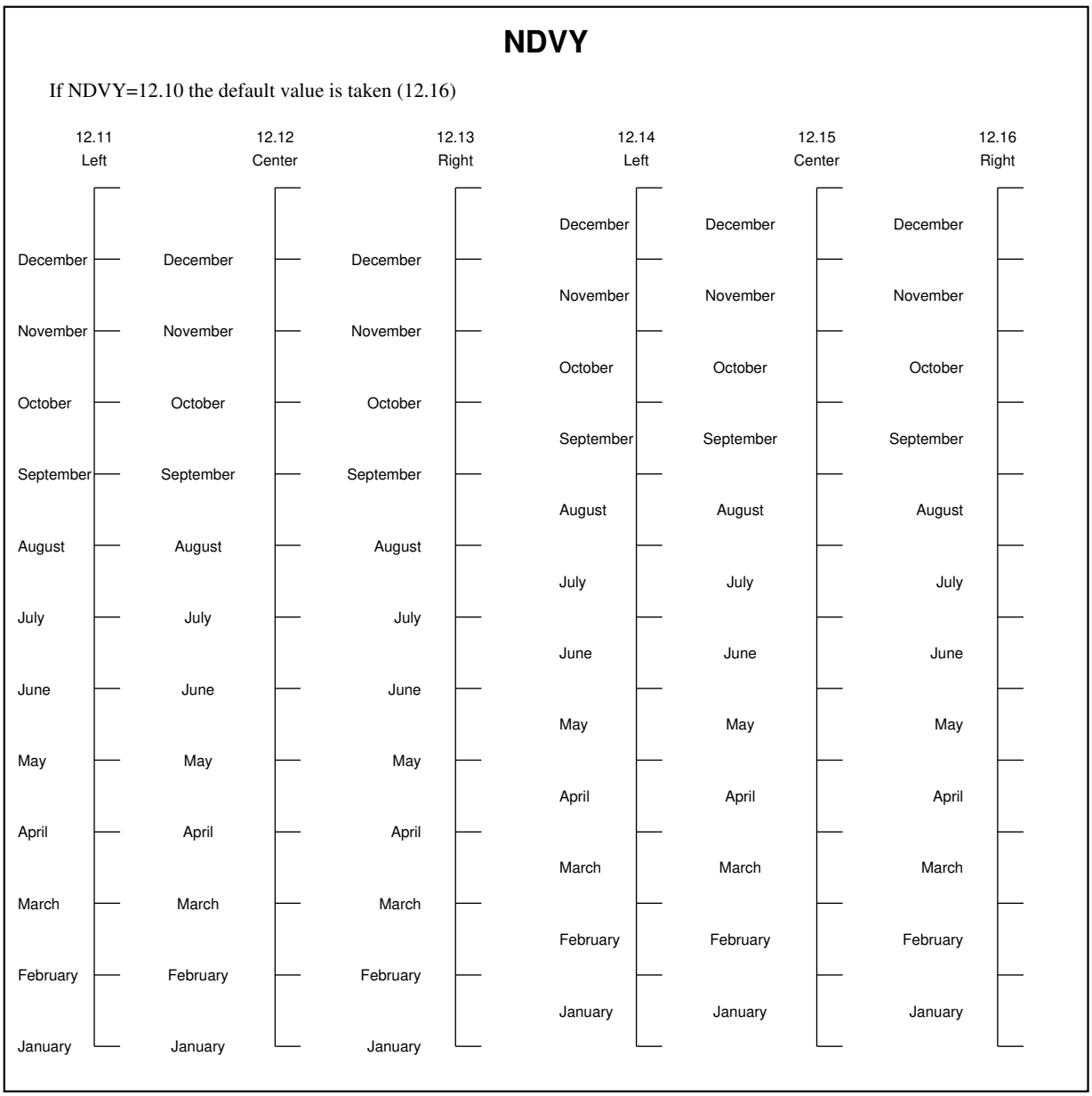


Figure 8.5: Example of labelling for vertical axes

P Page

The values given to the parameters PTYP, BTYP, HTYP, and FTYP are the HIGZ fill area interior styles. Interior style provided by the basic graphics package (i.e. GKS) can be used (cf the corresponding documentation) but in order to have the same result on all devices, numbers greater than 100 (HIGZ styles: 8.8) should be used. Figure 8.6 shows how to use the xTYP parameter.

The parameters PCOL, BCOL, HCOL and FCOL are equivalent to PTYP, BTYP, HTYP, and FTYP respectively, but instead of changing the hatch style, they change the colour of the same areas. It is possible to specify both the border and the inside color for the Histogram, Box Page, and Function (HCOL, BCOL, PCOL, FCOL).

Example of HCOL specification

```
Ex:
            +---- 1 The Histogram is filled
            |     0 Only the border is drawn
            |+--- Border color (here 2) if the histogram is filled
            ||++- Inside color (here 3) if the histogram is filled
            ||||  Border color if the histogram is not filled
            ||||
            VVVV
 SET  HCOL  1203
```

The same mechanism is also available for FCOL, BCOL and PCOL.

If PCOL, BCOL, HCOL or FCOL are between 1 and 99, then only the contour of the corresponding area is changed. If they are between 1001 and 1099, then the surface is filled with the colour determined by the corresponding fill area colour index (1 to 99). If they are between 1199 and 1999, then the surface is filled with the colour determined by the corresponding fill area colour index (1 to 99) and the border is drawn with the corresponding line color index (1 to 9).

If one of the *COL is greater than 1000 the corresponding value of the Fill Area Interior Style (for HTYP, BTYP, PTYP or FTYP) is automatically set to 1 (solid).

In addition, BCOL has two digits after the dot. The first one specifies the colour of the zone box shadowing and the second the colour of the statistic box shadowing.

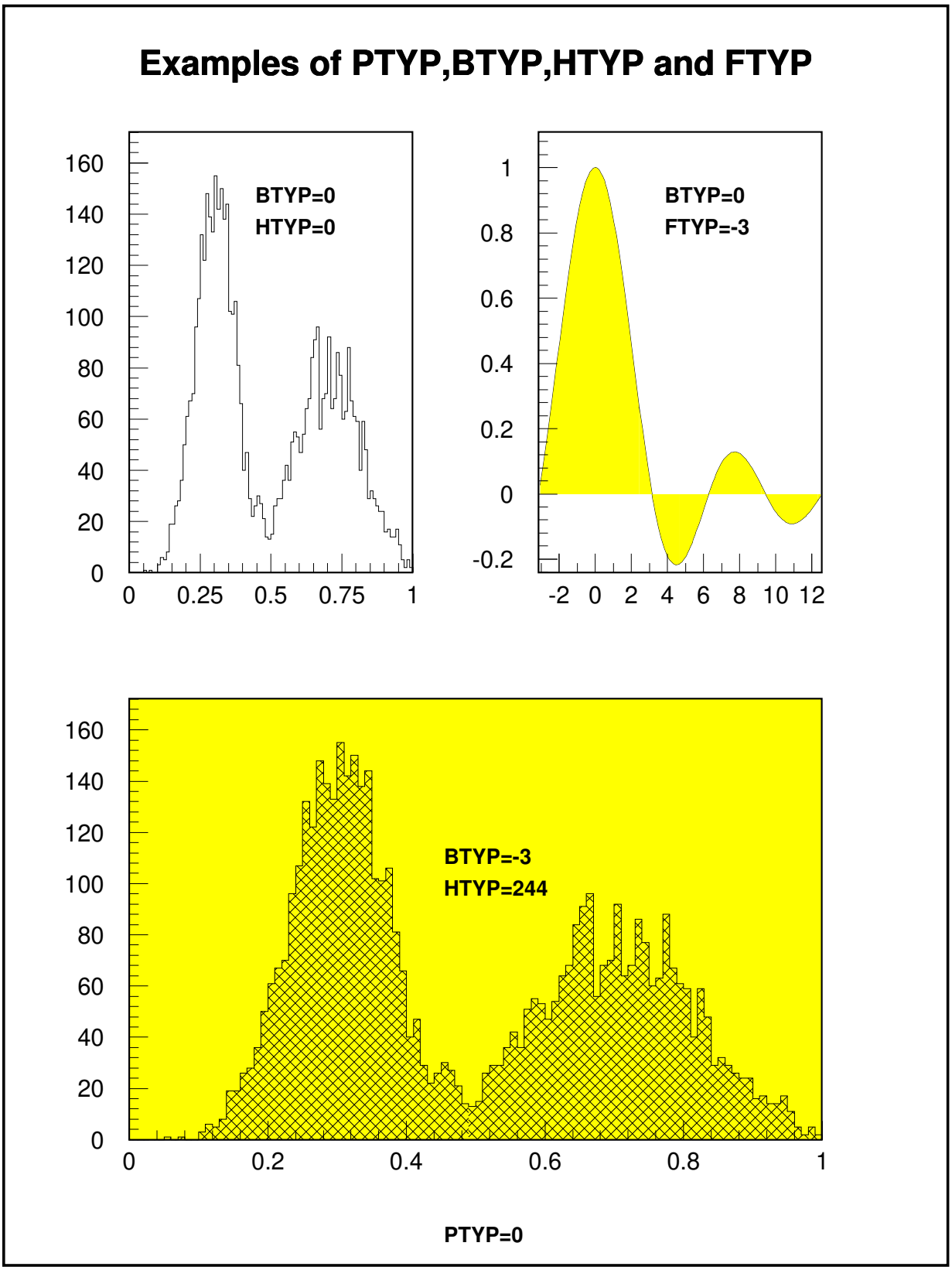


Figure 8.6: Usage of fill area types in HPLOT

8.7 Information about histograms

Four options are available to plot additional informations on HPLOT pictures: DATE, FILE, STAT and FIT.

```
PAW > OPTION DATE        | Plot date and hour on current HPLOT picture
PAW > OPTION FILE        | Plot file name of current histogram
PAW > OPTION STAT        | Plot statistics of current histogram
PAW > OPTION FIT         | Plot Fit parameters of current histogram
```

For each of these OPTION commands a corresponding SET parameter is available:

```
PAW > SET DATE i     | Default is 2
PAW > SET FILE i     | Default is 1
```

where i defines the position of the date or file name:

i = 1 : Top left corner of page/current histogram.

i = 2 : Top right corner

i = 3 : Bottom left corner

i = 4 : Bottom right corner

For example the command:

```
PAW > SET DATE 3
```

sets the position of the date to the bottom left corner of the HPLOT pictures.

```
PAW > SET STAT i     | Default is 1111
```

where i corresponds to binary status bits AOURMEI as follows:

A=1 Draw the contents of all channels

O=1 Draw number of overflows

U=1 Draw number of underflows

R=1 Draw R.M.S.

M=1 Draw mean value

E=1 Draw number of entries

I=1 Draw histogram identifier

For example the command:

```
PAW > SET STAT 10
```

sets the statistics informations to be only the number of entries.

```
PAW > SET FIT i      | Default is 101
```

where `i` corresponds to binary status bits `CEP` as follows:

`C=1` Draw χ^2

`E=1` Draw errors

`P=1` Draw fit parameters

For example to draw only the result of the χ^2 fit one would use:

```
PAW > SET FIT 100
```

For all these `OPTIONs`, the **character size** is specified with the command `SET CSIZ` and the character font used with `SET CFON`.

Fill area style, marker and line type

The Fill Area Interior Style, The Fill Area Style Index, the Marker TYPe and the Line TYPe are set respectively using the `IGSET` parameters `FAIS`, `FASI`, `MTYP` and `LTYPE`.

Example

```
PAW > IGSET FAIS 3      | Fill area are hatched
PAW > IGSET FASI 244    |   with the style index
PAW > IGSET MTYP 25     | Marker type is an empty square
PAW > IGSET LTYP 15     | Line type is dotted
```

HIGZ provides some portable fill area styles index coded using three digits `ijk` as follows:

`i`: Distance between each hatch in mm

`j`: Angle between `90` and `180` degrees

`k`: Angle between `0` and `90` degrees

These numbers are coded according to table 8.4 and examples are shown in figure 8.8.

i	Distance	j	Angle	k	Angle
		0	180°	0	0°
1	0.75mm	1	170°	1	10°
2	1.50mm	2	160°	2	20°
3	2.25mm	3	150°	3	30°
4	3.00mm	4	135°	4	45°
5	3.75mm	5	not drawn	5	not drawn
6	4.50mm	6	120°	6	60°
7	5.25mm	7	110°	7	70°
8	6.00mm	8	100°	8	80°
9	6.75mm	9	90°	9	90°

Table 8.4: Codification for the HIGZ portable fill area interior styles

Example

```
PAW > IGSET FAIS 3      | Fill area interior style is hatched
PAW > IGSET FASI 190    | Hatch type is 190
```

These commands will yield hatching with two sets of lines at 90° and 0° spaced 1 mm apart.

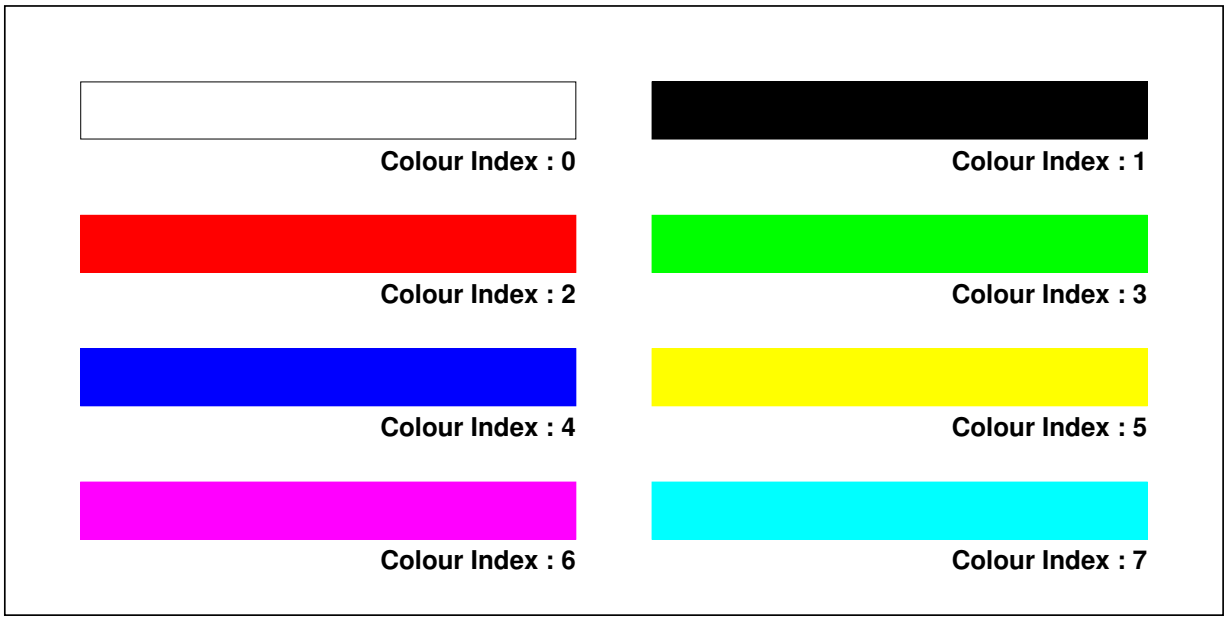


Figure 8.7: PostScript grey level simulation of the basic colours

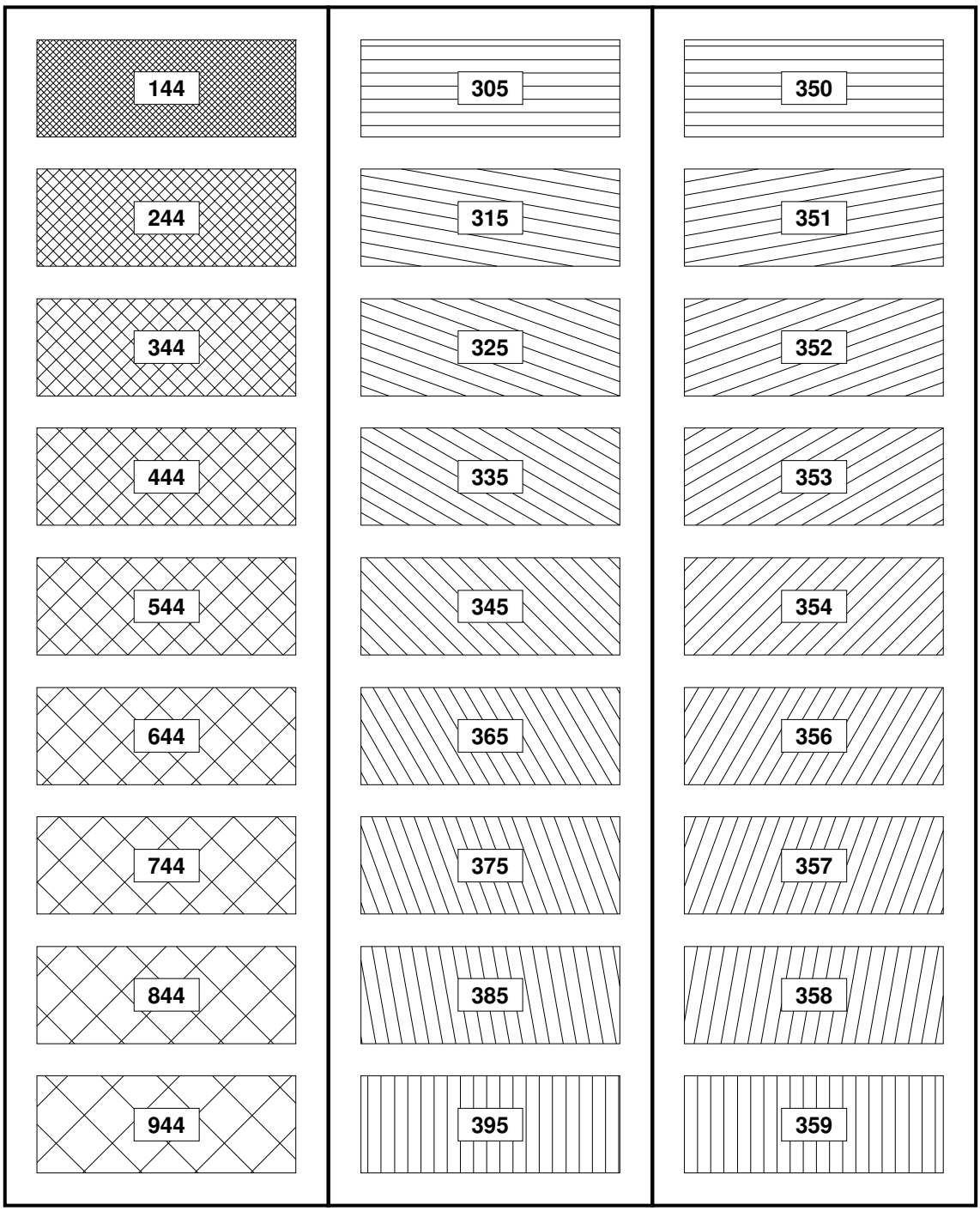


Figure 8.8: HIGZ portable hatch styles

Marker Type	Marker
31	✳
30	☆
29	★
28	✚
27	◊
26	△
25	□
24	○
23	▼
22	▲
21	■
20	●

Figure 8.9: HIGZ portable marker types

Line Index	Line Type
15	
14	
13	
12	

Figure 8.10: HIGZ portable line types

8.8 Text drawing

In PAW, text output can be produced in two ways:

1. *Automaticaly* with commands like GRAPH or HISTO/PLOT in which a lot of text is drawn: the axis labels, the histogram title, the global title, the statistics etc. . The attributes (font, colour or size) and the placement of these texts are controled with the command SET. In the rest of the chapter, the text produce *automaticaly* will be called *HPLOT text*

2. *Directly* with the commands ITX and TEXT. The attributes of ITX are controlled with the command IGSET whereas the attributes of TEXT are given with the command parameters.

Text placement

The text placement specify where the text must be drawn. For the *HPLOT text*, the text position is always in centimeters whereas for ITX or TEXT the current coordinate system is used.

HPLOT text

The possible text placements for *HPLOT text* are described in the following example:

```
PAW > SET XVAL 0.40 | distance between the Y axis and the axis values
PAW > SET YVAL 0.20 | distance between the X axis and the axis values
PAW > SET YLAB 0.80 | distance X axis to labels
PAW > SET XLAB 1.40 | distance Y axis to labels
PAW > SET YGTI 1.50 | Y position of global title
PAW > SET YHTI 1.20 | Y position  of histogram title
PAW > SET YNPG 0.60 | Y position for the page number
PAW > HISTO/PLOT 10 | the histogram 10 is drawn with previous settings
```

See figure 8.3 for more details.

ITX

In the command ITX the text position is defined with two mandatory parameters (X and Y):

```
PAW > SELNT 1         | cm coordinates
PAW > ITX 5 5 'Hello' | 'Hello' is drawn at the position (5,5)
```

TEXT

In the command TEXT the text position is defined with two mandatory parameters (X and Y):

```
PAW > SELNT 1           | cm coordinates
PAW > TEXT 5 5 'Hello' 1 | 'Hello' is drawn at the position (5,5)
```

Text size

For all the texts drawn with PAW commands, the text size is always specified in centimeters.

HPLOT text

The possible text sizes for *HPLOT text* are described in the following example:

```
PAW > SET ASIZ 0.28 | axis label size
PAW > SET CSIZ 0.28 | comment size
PAW > SET GSIZ 0.28 | global title size
PAW > SET KSIZ 0.28 | Hershey character size
PAW > SET 2SIZ 0.28 | scatter plot and table character. size
PAW > SET TSIZ 0.28 | histogram title size
PAW > SET VSIZ 0.28 | axis values size
PAW > HISTO/PLOT 10 | the histogram 10 is drawn with previous settings
```

See figure 8.3 for more details.

ITX

The text character heigh attribute for use by future invocations of `ITX` is set using the `CHHE` parameter as follows:

```
PAW > IGSET CHHE 1    | set the character heigh to 1 cm.
PAW > ITX 5 5 'Hello' | the size of 'Hello' is 1 cm.
```

TEXT

In the command `TEXT` the text size is a mandatory parameter (`SIZE`).

```
PAW > TEXT 5 5 'Hello' 1 | the size of 'Hello' is 1 cm.
```

Text orientation

The text orientation is an angle (in degrees) between the `X` axis and the text axis. By default this angle is equal to 0.

HPLOT text

Text orientation cannot be changed with some `SET` parameters for the *HPLOT text*. It is always automaticaly computed. For example in the command `ATITLE`, which draws the axis titles, the title on the `Y` axis is automaticaly drawn with an angle of 90 degrees.

ITX

The text orientation attribute for use by future invocations of `ITX` is set using the `TANG` parameter as follows:

```
PAW > IGSET TANG 90   | set the text angle to 90 degrees.
PAW > ITX 5 5 'Hello' | 'Hello' is drawn with an angle of 90 degrees.
```

TEXT

In the command TEXT the text orientation is an optional parameter (ANGLE).

```
PAW > TEXT 5 5 'Hello' ! 90 | 'Hello' is drawn with an angle of 90 degrees
```

Text alignment

The text alignment controls the placement of the character string with respect to the specified text position.

HPLOT text

Text alignment cannot be changed for the *HPLOT text*. It is automaticaly computed.

ITX

The text alignment attributes for use by future invocations of ITX are set using the TXAL parameter as follows:

```
PAW > IGSET TXAL (10*(horizontal alignment) + (vertical alignment))
```

The horizontal and vertical alignments parameters must be in the range 0-3. The horizontal alignment specifies which end of the string (or its geometric center) is aligned with the specified point given in ITX. The vertical alignment controls whether the top of tall characters (or the bottom of capital letters) line up with the specified point (see figure 8.11).

```
PAW > IGSET TXAL 23   | The horizontal and vertical alignments are centered
PAW > ITX 5 5 'Hello' | 'Hello' is drawn center adjusted
```

TEXT

In the command TEXT the text aligment is an optional parameter (CHOPT). Only the horizontal alignement can be changed among three possible values: Left, Center or Right.

```
PAW > TEXT 5 5 'Hello' 1 ! L | 'Hello' is drawn left adjusted (default)
PAW > TEXT 5 5 'Hello' 1 ! C | 'Hello' is drawn center adjusted
PAW > TEXT 5 5 'Hello' 1 ! R | 'Hello' is drawn right adjusted
```

Text colour

The text colour is define via a colour index in the colour table.

HPLOT text

```
PAW > SET XCOL 2    | X axis color
PAW > SET YCOL 3    | Y axis color
PAW > HISTO/PLOT 10 | the histogram 10 is drawn with previous settings
```

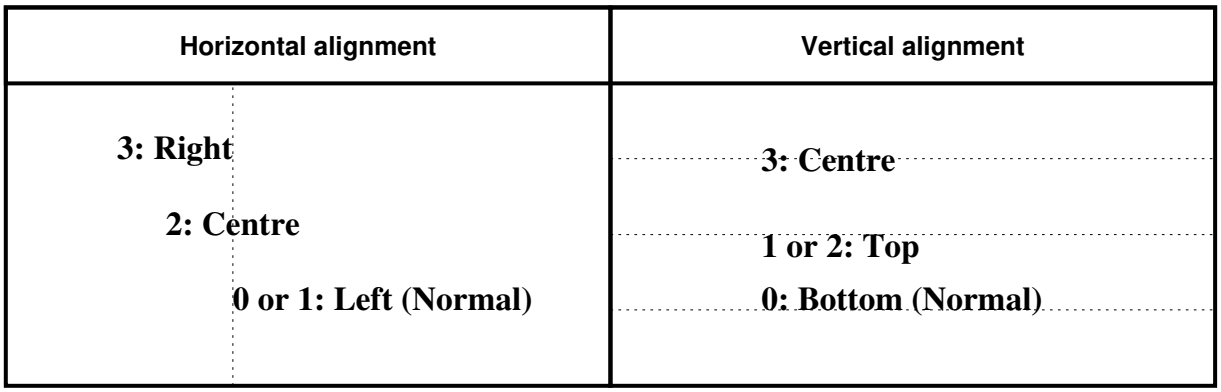


Figure 8.11: Text alignment

`ITXALH` horizontal alignment

- `0` normal (usually same as 1)
- `1` left end of string at specified point
- `2` center of string at specified point
- `3` right end of string at specified point

`ITXALV` vertical alignment

- `0` normal
- `1` top of tallest chars plus any built in spacing
- `2` top of tallest chars
- `3` halfway between 2 and 4

ITX

The text colour attribute for use by future invocations of `ITX` is set using the `TXCI` parameter as follows:

```
PAW > IGSET TXCI 3     | set the text colour to green.
PAW > ITX 5 5 'Hello' | 'Hello' is drawn in green.
```

TEXT

The text colour attribute for use by future invocations of `TEXT` is set using the `TXCI` parameter as follows:

```
PAW > IGSET TXCI 2        | set the text colour to red.
PAW > TEXT 5 5 'Hello' ! | 'Hello' is drawn in red.
```

Text font and precision

Text font selects the desired character font e.g. a roman font, a sans-serif font, etc. Text precision specifies how closely the graphics package implementation must follow the current size and orientation attributes. String (`0`) precision is most liberal (hardware), stroke (`2`) precision is most strict. Character precision is in the middle (`1`). The value of text font is dependent upon the basic graphics package used. However, font number `0`, with precision `2` is always available, independently from the basic graphics package used.Hardware characters are available with all the basic graphics packages. With X11, a large variety of font is available. They are the same as the PostScript fonts (see figure 8.15).

HPLOT text

```
PAW > SET CFON -60  | comment font is Helvetica Bold
PAW > SET GFON -20  | global title font is Times Bold
PAW > SET LFON -60  | axis labels font is Helvetica Bold
PAW > SET TFON -20  | general comments is Times Bold
PAW > SET VFON -60  | axis values font is Helvetica Bold
PAW > HISTO/PLOT 10 | the histogram 10 is drawn with previous settings
```

Note that `SET *FON ffp` set all the *HPLOT text* font to the same value `ffp`.

ITX

Text font and precision attributes for use by later invocations of `ITX` are set with `TXFP` as follows:

```
PAW > IGSET TXFP (10*(Text font) + (text precision))
```

TEXT

This command draws a software character text, independently from the basic graphics package used by HIGZ. It can produce over 300 different graphic signs. The way in which software characters are defined is via a string of valid characters, intermixed by other characters, acting as "escape" characters (e.g. a change of alphabet, upper or lower case). The string is interpreted by `TEXT` and the resulting characters are defined according to the figure 8.12, which shows the list of available software characters. This command allows the user to mix different types of characters (roman, greek, special, upper and lower case, sub and superscript). There are a total of 10 control characters.

List of escape characters and their meaning			
<	go to lower case	>	go to upper case (default)
[	go to greek (Roman = default)	]	end of greek
"	go to special symbols	#	end of special symbols
↑	go to superscript	?	go to subscript
!	go to normal level of script	&	backspace one character
$	termination character (optional)		

Note that characters can be also entered directly in lower case or upper case instead of using the control characters `<` and `>`.

The boldface characters may be simulated by setting the attributes '`PASS`' and '`CSHI`' with `IGSET`. The meaning of these attributes is the following: Every stroke used to display the character is repeated `PASS` times, at a distance (in percentage of the character height) given by `CSHI`.

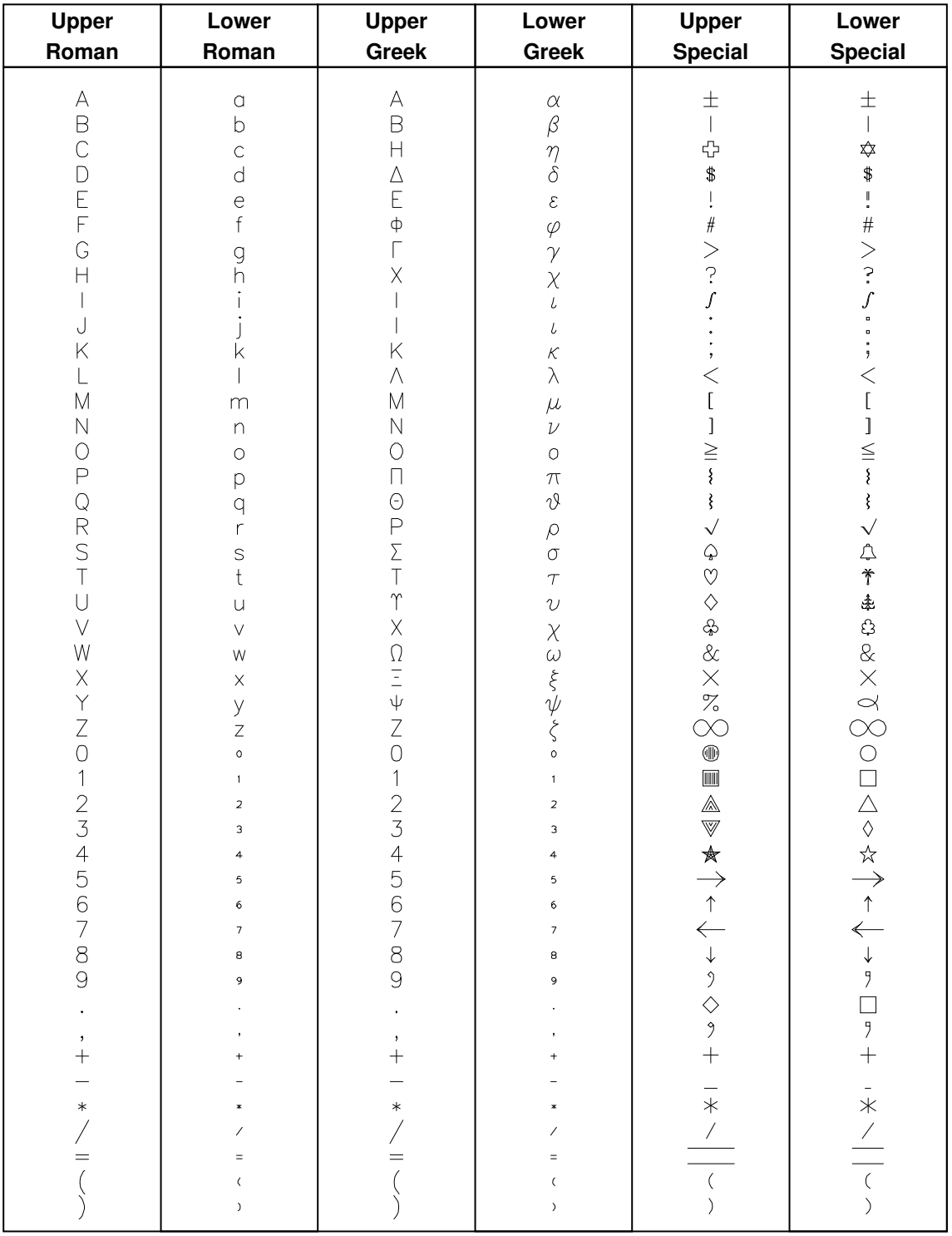


Figure 8.12: Characters available in `IGTEXT`

PostScript text fonts

PostScript files the text can be generated with PostScript fonts. The figure 8.15 shows all the PostScript fonts available on most PostScript printers. Note that the fonts `-15` to `-24` are the same than `-1` to `-14`, but they are drawn in hollow mode.

The correspondence between ASCII and ZapfDingbats font is given on figures 8.16 and 8.17. `TEXT` control characters are taken into account. In addition the character ~ switches to the ZapfDingbats character set.

List of escape characters and their meaning			
<	go to lower case (optional)	>	go to upper case (optional)
[	go to greek (Roman = default)	]	end of greek
”	go to special symbols	#	end of special symbols
~	go to ZapfDingbats	#	end of ZapfDingbats
↑	go to superscript	?	go to subscript
!	go to normal level of script	&	backspace one character
$	termination character (optional)		

The PostScript fonts can be used with precision `0` or precision `1`. On the screen, a PostScript font used with precision `1` appears like the `TEXT` characters, with precision 0 its appears as hardware character (X11 fonts). In both cases the PostScript file is the same.

Note that characters can also be entered directly in lower or upper case instead of using the escape characters < and >.

Example of PostScript text (result in figure 8.13)

```
PAW > IGSET LWID 6
PAW > BOX 0 16 0 5
PAW > IGSET CHHE 0.5
PAW > IGSET TXAL 3
PAW > IGSET TXFP -130
PAW > ITX 3 4 'K\355nstler in den gr\345\373ten st\311dten
PAW > ITX 3 3 '\253\265 l''\372uvre on conna\333t l''artisan\273
PAW > ITX 3 2 '\(proverbe fran\321ais\
PAW > ITX 3 1 '\252\241Ma\337ana\41 \322ag&\306!das&\313!\272, dit l''\323l\325ve.
```

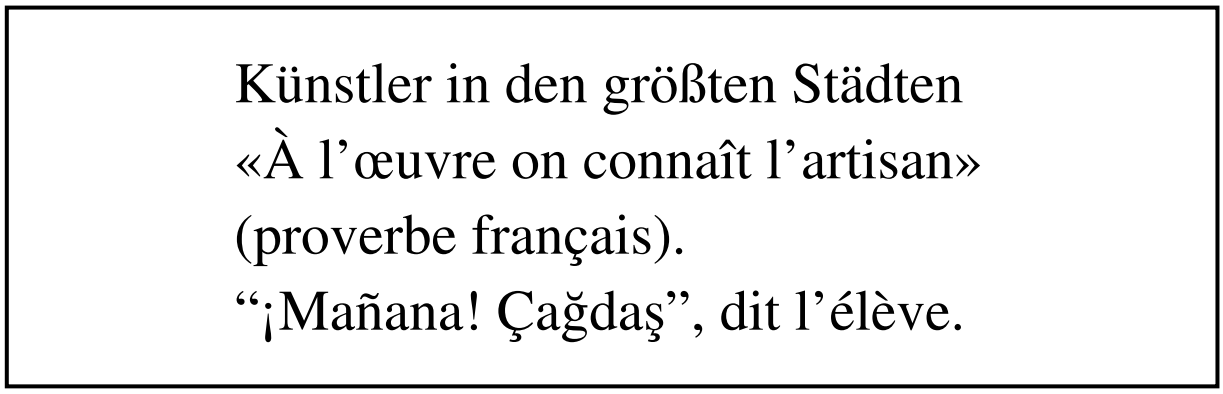


Figure 8.13: PostScript fonts usage (1).

Example of PostScript text and maths (result in figure 8.14)

```
PAW > IGSET LWID 6
PAW > BOX 0 16 0 5
PAW > IGSET CHHE 0.5
PAW > IGSET TXAL 23
PAW > IGSET TXFP -130
PAW > ITX 8 4 'e^+!e^-! "5# Z^o! "5# ll&^-!, qq&^\261!'
PAW > ITX 8 3 '| a&^[\256]! \267 b&^[\256]! | = [\345] a^i?jk!+b^kj?i'
PAW > ITX(8 2 'i ("d#?[m!y]&^\261![g^m]! + m [y]&^\261! ) = 0" r# (~r# + m^2!) [y] = 0'
PAW > ITX 8 1 'L?em! = e J^[m]?em! A?[m]! , J^[m]?em!=l&^\261![ g?m]!l , M^j?i! = [\345&?a]! A?[a! t^a]j?i! '
```

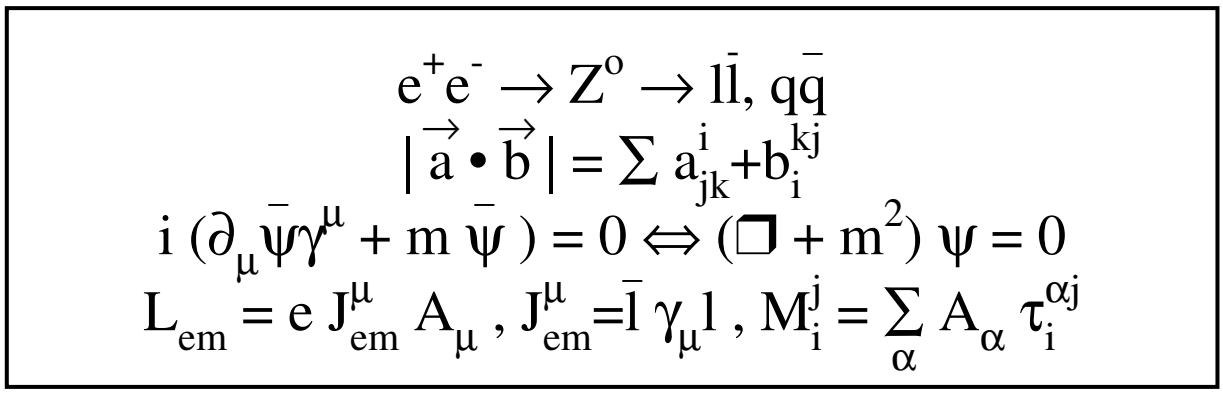


Figure 8.14: PostScript fonts usage (2).

Font/Prec	PostScript Font Style	
-1/0	*ABCDEFghijlk0123456789*	Times-Italic
-2/0	**ABCDEFghijlk0123456789**	Times-Bold
-3/0	***ABCDEFghijlk0123456789***	Times-BoldItalic
-4/0	ABCDEFghijlk0123456789	Helvetica
-5/0	*ABCDEFghijlk0123456789*	Helvetica-Oblique
-6/0	**ABCDEFghijlk0123456789**	Helvetica-Bold
-7/0	***ABCDEFghijlk0123456789***	Helvetica-BoldOblique
-8/0	ABCDEFghijlk0123456789	Courier
-9/0	*ABCDEFghijlk0123456789*	Courier-Oblique
-10/0	**ABCDEFghijlk0123456789**	Courier-Bold
-11/0	***ABCDEFghijlk0123456789***	Courier-BoldOblique
-12/0	ΑΒΗΔΕΦγχιιλκ0123456789	Symbol
-13/0	ABCDEFghijlk0123456789	Times-Roman
-14/0	✡✢✣✤✥✦✧✩✪✫✬✭✮✯✰✱✲✳✴✵✶✷✸✹✺✻✼✽✾✿❀❁❂❃❄❅❆❇❈❉❊❋❌❍❎❏❐❑❒❖❘❙❚❛❜❝❞❡❢❣❤❥❦❧❨❩❪❫❬❭❮❯❰❱❲❳❴❵	ZapfDingbats
-15/0	*ABCDEFghijlk0123456789*	Times-Italic
-16/0	**ABCDEFghijlk0123456789**	Times-Bold
-17/0	***ABCDEFghijlk0123456789***	Times-BoldItalic
-18/0	ABCDEFghijlk0123456789	Helvetica
-19/0	*ABCDEFghijlk0123456789*	Helvetica-Oblique
-20/0	**ABCDEFghijlk0123456789**	Helvetica-Bold
-21/0	***ABCDEFghijlk0123456789***	Helvetica-BoldOblique
-22/0	ΑΒΧΔΕΦγηιφλκ0123456789	Symbol
-23/0	ABCDEFghijlk0123456789	Times-Roman
-24/0	✡✢✣✤✥✦✧✩✪✫✬✭✮✯✰✱✲✳✴✵✶✷✸✹✺✻✼✽✾✿❀❁❂❃❄❅❆❇❈❉❊❋❌❍❎❏❐❑❒❖❘❙❚❛❜❝❞❡❢❣❤❥❦❧❨❩❪❫❬❭❮❯❰❱❲❳❴❵	ZapfDingbats

Figure 8.15: PostScript text fonts.

Input	Upper Roman	Upper Greek	Upper Special	Upper Zapf
A	A	A	±	✡
B	B	B	∣	✢
C	C	H	∃	✣
D	D	Δ	∀	✤
E	E	E	!	✥
F	F	Φ	#	✦
G	G	Γ	>	✧
H	H	X	?	★
I	I	I	⌡	☆
J	J	I	:	✪
K	K	K	;	✫
L	L	Λ	<	✬
M	M	M	[	✭
N	N	N	]	✮
O	O	O	≥	✯
P	P	Π	{	✰
Q	Q	Θ	}	✱
R	R	P	√	✲
S	S	Σ	♠	✳
T	T	T	♥	✴
U	U	Y	♦	✵
V	V	X	♣	✶
W	W	Ω	≤	✷
X	X	Ξ	×	✸
Y	Y	Ψ	%	✹
Z	Z	Z	∞	✺
0	0	0	⊗	✻
1	1	1	⊕	✼
2	2	2	∅	✽
3	3	3	◊	✾
4	4	4	•	✿
5	5	5	→	❀
6	6	6	↑	❁
7	7	7	←	❂
8	8	8	↓	❃
9	9	9	↔	❄
.	.	.	.	❅
,	,	,	,	❆
+	+	+	+	❇
-	-	−	∠	❈
*	*	*	〈	❉
/	/	/	÷	❊
=	=	=	≠	❋
(	(	(	≡	●
)	)	)	″	❍
{	{	{	{	■
}	}	}	}	❑
				❒
				❒
				❒
				▲
				▼
				◆
				❖
				◗
				─
				▌
				▐
				✚
				✛
				✜
				✝
				✞
				✟
				✠
				✡
				✢
				✣
				✤
				✥
				✦
				✧
				★
				✩
				✪
				✫
				✬
				✭
				✮
				✯
				✰
				✱
				✲
				✳
				✴
				✵
				✶
				✷
				✸
				✹
				✺
				✻
				✼
				✽
				✾
				✿
				❀
				❁
				❂
				❃
				❄
				❅
				❆
				❇
				❈
				❉
				❊
				❋
				●
				❍
				■

Input	Upper Roman	Upper Greek	Upper Special	Upper Zapf
\241	¡	ϒ	ϒ	❶
\242	¢	′	′	❷
\243	£	≤	≤	❸
\244	⁄	⁄	⁄	❹
\245	¥	∞	∞	❺
\246	ƒ	ƒ	ƒ	❻
\247	§	♣	♣	❼
\250	¤	♦	♦	❽
\251	'	♥	♥	❾
\252	“	♠	♠	❿
\253	«	↔	↔	➀
\254	‹	←	←	➁
\255	›	↑	↑	➂
\256	fi	→	→	➃
\257	fl	↓	↓	➄
\260	à	°	°	➅
\261	–	±	±	➆
\262	†	″	″	➇
\263	‡	≥	≥	➈
\264	·	×	×	➉
\265	À	∝	∝	➊
\266	¶	∂	∂	➋
\267	•	•	•	➌
\270	‚	÷	÷	➍
\271	„	≠	≠	➎
\272	»	≡	≡	➏
\273	…	≈	≈	➐
\274	‰	⋮	⋮	➑
\275	â	—	—	➒
\276	ª	↵	↵	➓
\277	º	ℵ	ℵ	①
\300	Á	ℑ	ℑ	②
\301	ˋ	ℜ	ℜ	③
\302	´	℘	℘	④
\303	ˆ	⊗	⊗	⑤
\304	˜	⊕	⊕	⑥
\305	¯	∅	∅	⑦
\306	˘	∩	∩	⑧
\307	˙	∪	∪	⑨
\310	¨	⊇	⊇	⑩
\311	ä	⊄	⊄	❶
\312	°	⊂	⊂	❷
\313		⊆	⊆	❸
\314	Å	∈	∈	❹
\315	˝	∉	∉	❺
\316	˛			❻
\317	ˇ			

Input	Lower Roman	Lower Greek	Lower Special	Lower Zapf
\321	ç	∇	∇	❽
\322	Ç	®	®	❾
\323	˝	©	©	❿
\324	é	™	™	➔
\325	È	∏	∏	→
\326	è	√	√	↔
\327	Ê	·	·	↕
\330	ê	¬	¬	➘
\331	Ë	∧	∧	➙
\332	ë	∨	∨	➚
\333	Í	⇔	⇔	→
\334	î	⇐	⇐	➛
\335	Î	⇑	⇑	→
\336	ï	⇒	⇒	➟
\337	Ï	⇓	⇓	➡
\340	ñ	◊	◊	➠
\341	Ñ	〈	〈	➢
\342	Æ	®	®	➢
\343	ô	©	©	➤
\344	ª	™	™	➤
\345	Ô	∑	∑	➥
\346	ö	⎛	⎛	➦
\347	Ö	⎜	⎜	➧
\350	û	⎜	⎜	➨
\351	Ł	⎜	⎜	➨
\352	Ø	⎜	⎜	➩
\353	Œ	⎜	⎜	➪
\354	º	⎜	⎜	➪
\355	Û	⎜	⎜	➫
\356	ü	⎜	⎜	➫
\357	Ù	⎜	⎜	➬
\360	ı	⎝	⎝	➭
\361	Å	⎞	⎞	➮
\362	Ã	⎟	⎟	➯
\363	ÿ	⎟	⎟	➱
\364	Ÿ	⎟	⎟	➲
\365	ı	⎟	⎟	➳
\366	á	⎟	⎟	➵
\367	Á	⎟	⎟	➸
\370	ł	⎟	⎟	➸
\371	ø	⎟	⎟	➺
\372	œ	⎟	⎟	➻
\373	ß	⎠	⎠	➼
\374	ù	⎟	⎟	➽
\375	Ù	⎟	⎟	➾
\376		⎟	⎟	➚
\377		⎭	⎭	➘

Figure 8.17: PostScript characters (2).

8.9 The HIGZ graphics editor

The HIGZ pictures in memory can be modified interactively with the HIGZ graphics editor. The command `PICT/MODIFY` invokes the HIGZ editor (see figure 8.18 for more details):

```
PAW > PICT/MODIFY PNAME
```

`PNAME` can be the complete name, the picture number in memory or ' '.

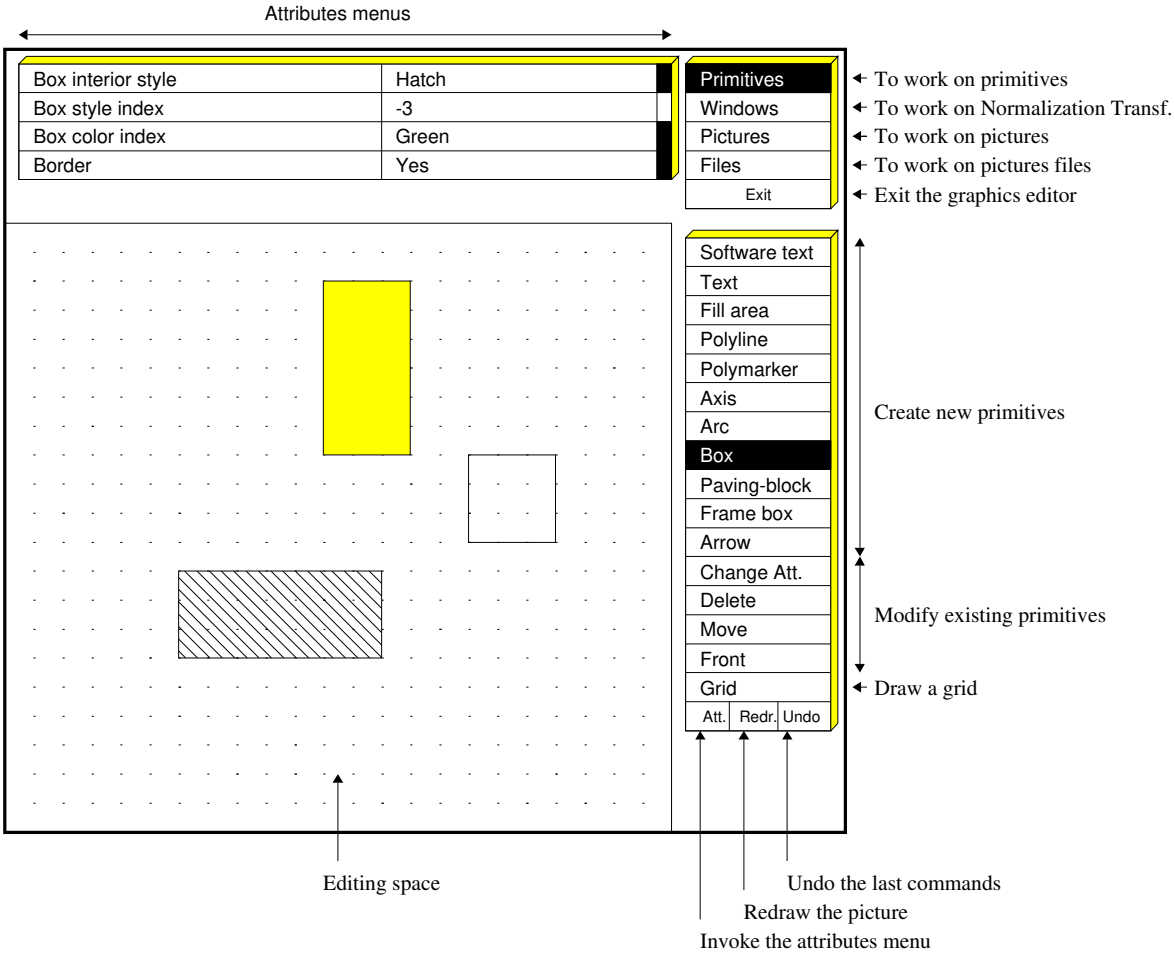


Figure 8.18: The HIGZ graphics editor

Chapter 9: Distributed PAW

With the increasing number of workstations, it happens more and more frequently that a user wants to run PAW on a mainframe or on a workstation. Several tools described in this chapter have been developed in order to use in the most convenient way all the resources available in an heteregoneous environment of workstations, superminis, data acquisition systems and mainframes.

TELNETG: A powerful terminal emulator. An alphanumeric window (line mode) is created on the local workstation (e.g. Apollo) to create a session (like with `TELNET`) on a remote computer (e.g. VAX). On the remote computer, a graphics program is run and a window is automatically created on the local workstation to receive the graphics output.

3270G Same as the `TELNETG` emulator for the case of a connection with an IBM machine in full screen mode under VM/CMS.

ZFTP The ZEBRA file transfer program optimized to transport ZEBRA RZ or FZ files between machines with different data representations.

There exists also the possibility to access files on a **remote computer** from a PAW session on a workstation. PAW can be used in a **real time** environment. Access to HBOOK histograms being filled by a different process on the same machine (Global sections on VAX) or a computer on the network (e.g. OS9 modules).

Both ZFTP and real time access to histograms on a remote computer require the implementation of a **PAW server** on this computer. The PAW server is automatically started from a PAW session, if PAW has been implemented with the relevant options (PATCHY [15] flag CZ). PAW and the PAW server must be linked with two special modules called **CZ** and **TCPAW** [16, 17].

CZ is a small FORTRAN package (about 300 lines). It provides an interface between the ZEBRA Input/Output routines and the high level transport routines of the TCPAW package.

TCPAW[16] is a networking package, written in C by Ben Segal (about 1500 lines). It provides a very simple FORTRAN-callable interface to TCP/IP services. It supports client and server modules running on UNIX, Apollo, VMS, VM/CMS and OS9 environments. Small parts of TCPAW are CERN specific but it would be perfectly possible to transport it elsewhere with minor modifications. The package currently requires the Wollongong (TWG) TCP/IP software to be present on VMS connected systems, the IBM FAL 1.2 Product on VM/CMS, and Microware TCP/IP on OS9. The UNIX systems Ultrix, CRAY Unicos, SUN OS, IBM AIX, Apollo/Aegis, Apple A/UX and HP-UX are supported as delivered.

9.1 TELNETG and 3270G

Figure 9.1 describes the functionality of these two programs. They allow to run a graphics application based on HIGZ (e.g. PAW, GEANT, etc.) on a host machine and to receive the graphics output on the local machine. TELNETG is designed to work with operating systems supporting a command line interface and 3270G for a full screen interface.

TELNETG and 3270G supports both graphics Input and Output. The graphics locator (commands LOCATE, VLOCATE, etc.) as well as the various KUIP graphics menu styles (G and GP) may be used.

Both programs exploit the fact that the HIGZ macro primitives are very compact, therefore reducing the amount of information to be sent through the network. Compared to more conventional emulators (4014, 4207, etc.) gains in speed are typically a factor of 10 when drawing one-dimensional histograms and may reach a factor 100 for two-dimensional plots (lego, surface, scatterplot).

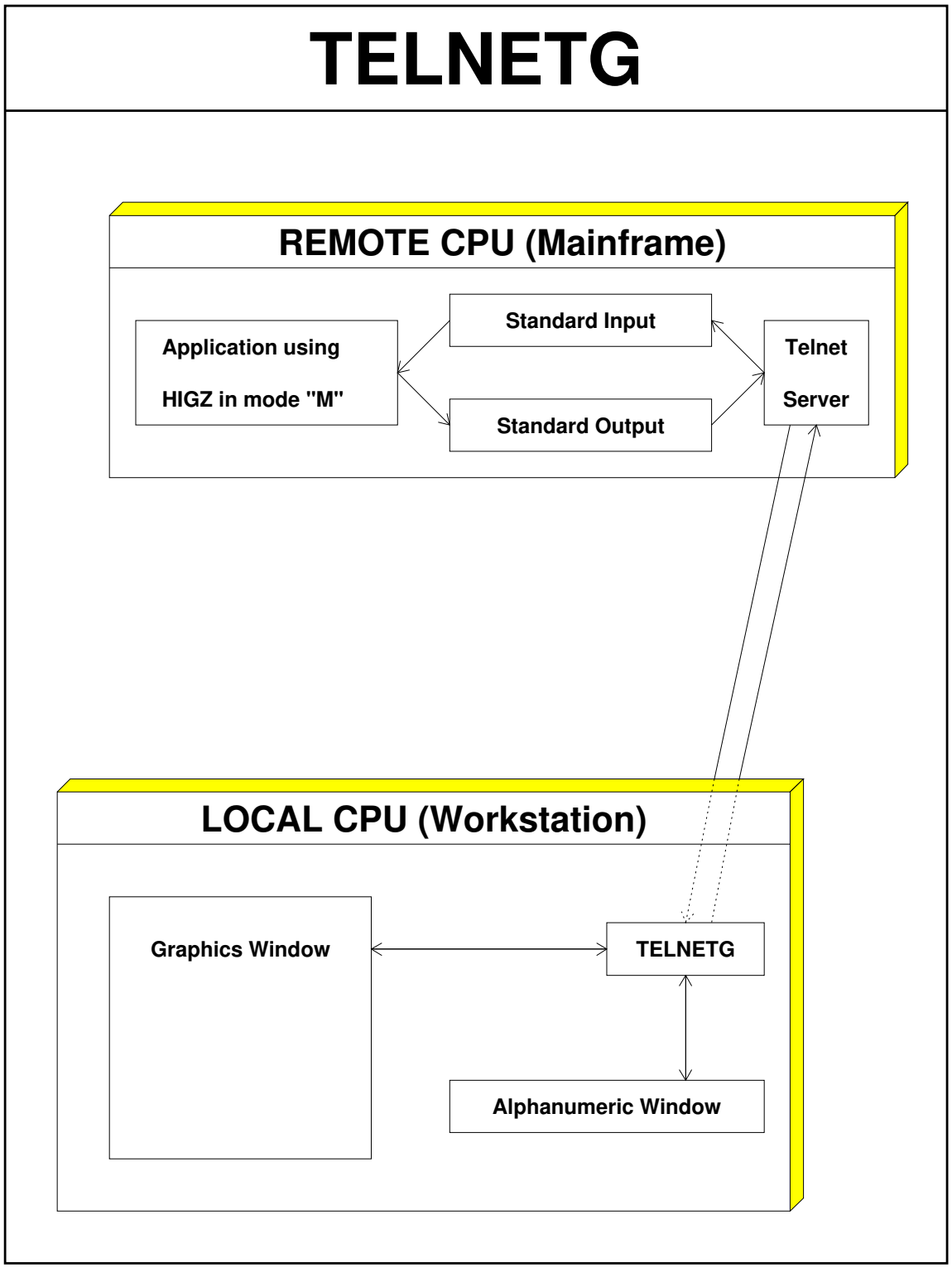


Figure 9.1: The TELNETG program

TELNETG combines a slightly modified version of the standard TELNET program written in the C language and an interface to the HIGZ system written in FORTRAN.

The following example shows how to use TELNETG from an Apollo to a VAX. The integer identifier of the workstation type must be preceded by a **minus sign** (e.g. for an Apollo DN3000):

Example of a TELNETG session

```
$ TELNETG vxcrna
Trying...
Open
         This is the CERN Central VAXcluster running VMS V5.1

Username: USERNAME
Password: PASSWORD(not echoed)
        Welcome to VAX/VMS version V5.1 on node VXCRNA
 TERMINAL TYPE <? for HELP; No default>:D1
VxCrnA$ PAW
 ******************************************************
 *                                                    *
 *            W E L C O M E    to   P A W             *
 *                                                    *
 *           Version 1.11/02  29 March 1991           *
 *                                                    *
 ******************************************************
 Workstation type (?=HELP) <CR>=7878 : -10002
 VERSION 7.4/2.6 OF GKSGRAL STARTED
 PAW > hi/plot 10           | The graphics is sent to the Apollo
 PAW > locate               | Graphics input using the Apollo mouse
```

9.2 ZFTP

The ZFTP program (ZEBRA File Transfer Program) provides the same functionality as the FTP program which is available like TELNET on all workstations and mainframes supporting TCP/IP. In addition ZFTP has been optimized to allow the transfer of ZEBRA binary files both sequential and direct access.

The direct access ZEBRA/RZ files (used for HBOOK histograms and HIGZ pictures) contain data in the local data representation. Because ZEBRA is an object oriented language supporting machine independent Input/Output, ZFTP is able to translate in flight all the ZEBRA data structures in a transparent way in the network buffers. ZFTP copies the RZ files on the local machine with the same parameters (RECL, quota, etc.) than on the remote machine. The original date and time of the objects is also preserved.

In addition to binary file transfer, ZFTP can also transfer alphanumeric text files (up to 80 characters/line). On IBM/VM-CMS, these files must be of type RECFM=F,LRECL=80.

The ZFTP user interface is based on KUIP and is the same on all systems. If several files have to be transferred (maybe on a regular basis), KUIP macros may be used. The following commands are available:

OPEN To start a communication with a remote machine.

CLOSE Close the current communication.

GETA Transfer an Alphanumeric text file from the remote machine.

PUTA Transfer an Alphanumeric text file to a remote machine.

GETRZ Transfer a RZ file from a remote machine.

PUTRZ Transfer a RZ file to a remote machine.

GETFZ Transfer a FZ file from a remote machine.

PUTFZ Transfer a FZ file to a remote machine.

RSHELL Send a command to a remote machine.

Example of a ZFTP session

```
# Start execution of the program from inside the PAW directory
$ ZFTP
ZFTP > open CERNVM                      |Starts communication with CERNVM
                                        | (prompt for username/password)
ZFTP > getrz RZFILE.DAT.D  local.dat    | Transfer IBM file "RZFILE.DAT"
                                        |  to local file "local.dat"
ZFTP > puta local.car                   | Transfer local alphanumeric file
                                        | "local.car" to IBM
                                        | IBM file name will be "LOCAL CAR A"
ZFTP > quit
```

9.3 Access to remote files from a PAW session

When running PAW, it is often necessary to access files (e.g. HBOOK files) which reside on a different computer. The ZFTP program described above can be used if a very frequent access to the file is required. A more convenient mechanism is the possibility to access the files directly. On many systems, one may now use NFS [18] for this purpose. Under some circumstances, for example if the HBOOK file

is not in exchange mode and it is to be accessed from a computer running a different operating system, an alternate approach is required. To fill this gap the PAW server is provided. This works using a conventional Client/Server model. The client (PAW) typically runs on a workstation. When the PAW command RLOGIN is invoked, a PAW server is automatically started on the remote machine, normally a mainframe or data server.

Once the `RLOGIN REMOTE` command has been executed, the PAW Current Directory is set to `//REMOTE`. The PAW client can now instruct the PAW server to attach a file using the `RSHELL` command (e.g. `rshell file pawtest.dat`). If an histogram with HBOOK ID=10 is on the remote file, than the PAW command `Histo/Plot 10` will plot this histogram on the local workstation. The histogram resides on `//PAWC` like other histograms coming from local files.

The `RSHELL` command may be used to communicate with the PAW server. The expression typed following `RSHELL` is passed to the server. The current implementation of the PAW server recognizes the commands:

`rshell file filename`	Server connects filename
`rshell cdir //lun11`	Server changes current directory
`rshell ld`	Server lists current directory
`rshell ld //`	Server lists all connected files
`rshell message`	Server pass message to operating system

Access to remote files from a workstation

```
PAW > rlogin CERNVM                          | connect to CERNVM
PAW > rshell file HRZTEST.DAT                | PAW server connects HRZTEST DAT A to //LUN11
PAW > histo/plot 10                          | plot histogram 10 from CERNVM
PAW > histo/fit 20 G                         | fit histo 20 with a gaussian and plot it
PAW > rlogin VXCRNA                          | connect to VXCRNA
PAW > rshell file DISK$DL:[PAW]HEXAM.DAT;3   | PAW server on VXCRNA connects file to //LUN11
PAW > histo/plot 110                         | plot histogram 110 from VXCRNA
PAW > rshell file HRZTEST.DAT                | PAW server on VXCRNA connects file to //LUN12
PAW > histo/plot 110 s                       | plot histogram 110 from HRZTEST.DAT
                                             | on VXCRNA on the existing picture
PAW > rshell ld //                           | list all files connected on VXCRNA
PAW > cdir //CERNVM                          | Change current PAW directory to CERNVM
PAW > histo/plot 110                         | plot histogram 110 from CERNVM
PAW > histo/plot //VXCRNA/110                | plot histogram 110 from VXCRNA
PAW > cdir //PAWC                            | current directory to local memory
PAW > histo/list                             | list all histograms in //PAWC
PAW > Histo/delete 0                         | delete all histograms in memory
PAW > hrin //VXCRNA/0                        | read all histograms from VXCRNA
                                             | file HRZTEST.DAT to //PAWC
PAW > cdir //CERNVM                          | change directory to CERNVM
PAW > rshell file NEW.DAT.D 1024 N           | creates a new file on the D disk
PAW > hrout 0                                | write all histograms from //PAWC
                                             | to CERNVM file NEW DAT D
```

9.4 Using PAW as a presenter on VMS systems (global section)

```
      PROGRAM PRODUCE
      PARAMETER MAXPAGES=100
      COMMON/PAWC/IPAWC(128*MAXPAGES)
      CHARACTER*8 GNAME
      INTEGER*4 HCREATEG
*
      GNAME='GTEST'
      WAIT_TIME=1.
      NUMEVT=1000
*.............           Create Global section
      NPAGES=HCREATEG(GNAME,IPAWC,128*MAXPAGES)
      IF(NPAGES.GT.0) THEN
         PRINT 1000,GNAME
 1000    FORMAT(' Global Section: ',A,' created')
      ELSE
         IERROR=-NPAGES
         PRINT 2000,IERROR
 2000    FORMAT(' Global Section Error', I6)
         GO TO 99
      ENDIF
      CALL HLIMIT(128*NPAGES)
*.............           Book histos.
      CALL HBOOK1(10,'Test1$',50,-4.,4.,0.)
      CALL HBOOK1(20,'Test2$',50,-4.,4.,0.)
*.............           Fill histos.
      DO 20 I=1,NUMEVT
         DO 10 J=1,100
            CALL RANNOR(A,B)
            CALL HFILL(10,A,0.,1.)
            CALL HFILL(20,B,0.,1.)
 10      CONTINUE
        CALL LIB$WAIT(WAIT_TIME)
 20   CONTINUE
*
 99   STOP
      END
```

```
$ fort produce
$ link produce,SYS$INPUT/OPTIONS,-
cern$library:packlib/lib,kernlib/lib
PSECT=PAWC,PAGE
```

```
PAW > edit produce
   macro produce ntimes=100
     nt=[ntimes]
     zone 1 2
     histo/plot 10 K
     histo/plot 20 K
   loop:
       histo/plot 10 U
       histo/plot 20 U
       wait ' ' 1
       nt=[nt] -1
       if nt>0 goto loop
   return
PAW > global GTEST
PAW > exec produce ntimes=20
```

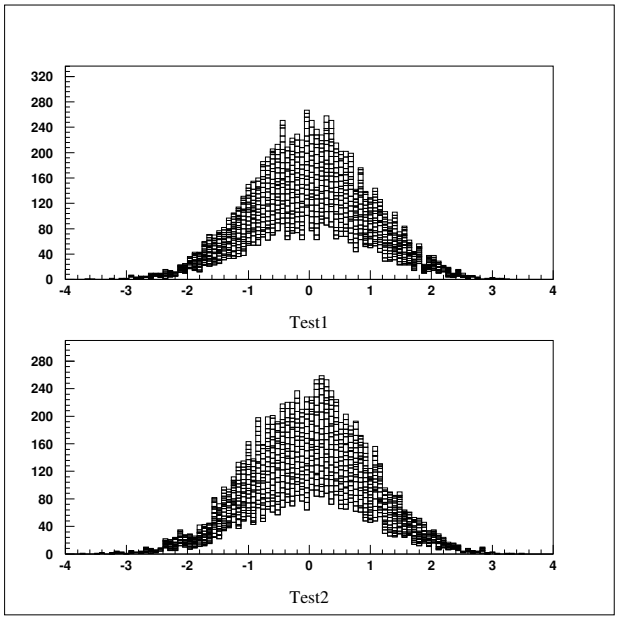


Figure 9.2: Visualise histograms in global section

In addition to the facilities described in the previous section, the standard version of PAW may be used as an online presenter on VMS systems using the mechanism of global sections. It is possible for two processes to reference the same histograms using **global sections**. For example, the first process may be a **histogram producer** (e.g. a monitoring task) and the second process **PAW**. As the histograms are being gradually filled by the first task, PAW can view them, and even reset them. To use the global sections, it is also necessary to "page align" the common which is in the global section. This is achieved in the "link step" when making the process (see example). The relevant statements are `SYS$INPUT/OPTIONS` to tell the linker that some options follow the link statement, and `PSECT=PAWC,PAGE` which is the option to page align the `/PAWC/` common.

9.5 Using PAW as a presenter on OS9 systems

The technique described in previous sections may also be used to access HBOOK histograms being filled by a monitoring task on OS9 systems from a standard PAW session running on a machine with the TCP/IP software.

```
      INDIRECT PAWC
      PROGRAM PRODUCE
*
*        Monitoring task MT1 in processor OP2.
*
      PARAMETER NWPAW=10000
      COMMON/PAWC/IPAWC(NWPAW)
*
      CALL HLIMIT(NWPAW)
*
*       Book histos.
*
      CALL HBOOK1(10,'TEST1$',50,-3.,3.,0.)
      CALL HBOOK1(20,'TEST2$',50,-3.,3.,0.)
*
*       Fill histos.
*
      NUMEVT=10000
      DO 20 I=1,NUMEVT
         DO 10 J=1,100
            CALL RANNOR(A,B)
            CALL HFILL(10,A,0.,1.)
            CALL HFILL(20,B,0.,1.)
 10      CONTINUE
 20   CONTINUE
*
 99   STOP
      END
```

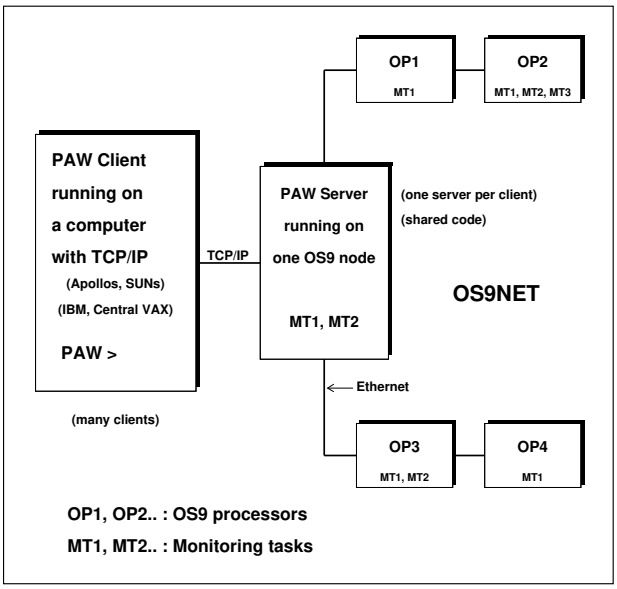


Figure 9.3: Visualising histograms on OS9 modules from PAW

Example of how to access OS9 modules from PAW

```
PAW > rlogin O-OPAL01                  | connect to an OS9 machine
PAW > rshell module OP2/MT1            | PAW server connects to OP2/MT1
                                       | (Processor OP2, Monitoring Task MT1)
PAW > histo/plot 10                    | plot histogram 10
PAW > hrin 0                           | read all histograms into //PAWC
PAW > Histo/File 1 local.dat 1024 N    | create a new file local.dat
                                       | on the client machine
PAW > hrout 0                          | save all histograms from //PAWC
                                       | to the local file
PAW > rshell module OP3/MT2            | PAW server connects to another
                                       | OS9 monitoring task
PAW > Output 56 os9.listing            | Change output file on client
PAW > rshell ldir                      | list all histograms in MT2
                                       | on file os9.listing
PAW > Output -56                       | Change output file to default (unit 6)
                                       | file os9.listing is closed
```

Part III

PAW - Reference section

Notation used in the reference section

Optional parameters are enclosed in square brackets, e.g. `[optpar]`

The **type** of a parameter is indicated following its name as follows:

`C` Character data

`I` Integer data

`R` Real (floating point) data

Supplementary information is given at the end of the line describing the parameter:

`D=` Default value
e.g. `D='S'` for Character data or `D=40` for Integer data

`R=` Range of possible values
e.g. `R=0:1` means that the variable's value lies between `0` and `1`.
`R=' ,L,P,*,+'` enumerates the possible values for the given Character variable.

Chapter 10: KUIP

Command Processor commands.

KUIP/HELP [item option]

ITEM	C	"Command or menu path" D='␣'
OPTION	C	"View mode" D='N'

Possible OPTION values are:

EDIT	The help text is written to a file and the editor is invoked,
E	Same as 'EDIT'.
NOEDIT	The help text is output on the terminal output.
N	Same as 'NOEDIT'

Give the help of a command. If ITEM is a command its full explanation is given: syntax (as given by the command USAGE), functionality, list of parameters with their attributes (prompt, type, default, range, etc.). If ITEM='/' the help for all commands is given.

If HELP is entered without parameters or ITEM is a submenu, the dialogue style is switched to 'AN', guiding the user in traversing the tree command structure.

'HELP -EDIT' (or just 'HELP -E') switches to edit mode: instead of writing the help text to the terminal output, it is written into a temporary file and the pager or editor defined by the command HOST_PAGER is invoked. (On Unix workstations the pager can be defined to display the help text asynchrously in a separated window.) 'HELP -NOEDIT' (or just 'HELP -N') switches back to standard mode. The startup value is system dependent.

KUIP/USAGE item

ITEM	C	"Command name"

Give the syntax of a command. If ITEM='/' the syntax of all commands is given.

KUIP/MANUAL item [output option]

ITEM	C	"Command or menu path"
OUTPUT	C	"Output file name" D='␣'
OPTION	C	"Text formatting system" D='␣'

Possible OPTION values are:

'␣'	plain text : plain text format
LATEX	LaTeX format (encapsulated)
TEX	LaTeX format (without header)

Write on a file the text formatted help of a command. If ITEM is a menu path the help for all commands

linked to that menu is written. If ITEM='/' the help for the complete command tree is written. If OUTPUT=' ' the text is written to the terminal.
The output file produced with option LATEX can be processed directly by LaTeX, i.e. it contains a standard header defining the meta commands used for formatting the document body. With option TEX only the document body is written into the output file which can be included by a driver file containing customized definitions of the standard meta commands. Example:

```
MANUAL / MAN.TEX LATEX
```

will produce the file MAN.TEX containg the documentation of all available commands in LaTeX format.

KUIP/EDIT `fname`

`FNAME` C "File name"

Invoke the editor on the file. The command HOST˙EDITOR can be used to define the editor.
If FNAME does not contain an extension the default filetype '.KUMAC' is supplied. The search path defined by the command DEFAULTS is used to find an already existing file. If the file does not exist it is created with the given name.

KUIP/PRINT `fname`

`FNAME` C "File name"

Send a file to the printer. The command HOST˙PRINT can be used to define the host command for printing the file depending on it file extension.

KUIP/PSVIEW `fname`

`FNAME` C "File name"

Invoke the PostScript viewer on the file. The command HOST˙PSVIEWER can be used to define the PostScript viewer.
If FNAME does not contain an extension the default filetype '.PS' is supplied.

KUIP/LAST `[ n fname ]`

`N` I "N last commands to be saved" `D=-99 R=-99:`
`FNAME` C "File name" `D='␣'`

Perform various operations with the history file.
If FNAME is not specified, the current history file is assumed by default (the startup history file name is LAST.KUMAC). To change the history file the command LAST 0 NEW-FNAME must be entered.
If N.EQ.-99 (default case) the default host editor is called to edit the current history file, containing all the commands of the session.
If N.LT.0 the last -N commands are printed on the screen. On MVS this allows to edit and resubmit commands. On workstations this allows to resubmit blocks of commands by mouse-driven cut-and-paste operations.
If N.EQ.0 the history file FNAME is rewound and set as the current one (the command LAST 0 FNAME itself is not recorded).
If N.GT.0 the last N commands of the session are saved in the current history file.
See also the command RECORDING.

KUIP/MESSAGE `[ string ]`

`STRING` C "Message string" `D='␣' Separate`

Write a message string on the terminal. A useful command inside a macro. Several message strings can be given in the same command line, each of them separated by one or more spaces (the usual parameter separator); therefore multiple blanks will be dropped and only one will be kept. If multiple blanks should not be dropped, the string must be surrounded by single quotes.

KUIP/SHELL `[ cmd ]`

`CMD` C "Shell command string" `D='␣'`

Execute a command of the host operating system. The command string is passed to the command processor defined by HOST˙SHELL. If CMD=' ' the shell is spawned as interactive subprocess. To return from the shell enter 'RETURN' (the full word, not just ⟨CR⟩) or 'exit' (depending on the operation system).

KUIP/WAIT `[ string sec ]`

`STRING` C "Message string" `D='␣'`
`SEC` R "Number of seconds" `D=0 R=0:`

Make a pause (e.g. inside a macro). Wait a given number of seconds (if SEC.GT.0) or just until ⟨CR⟩ is entered (if SEC.EQ.0). A message string is also written on the terminal before waiting.

KUIP/IDLE `sec [ string ]`

`SEC` I "Number of seconds" `R=0:`
`STRING` C "Command string" `D='␣'`

Execute a command if program is idle. The command string is executed if there was no keyboard activity during SEC seconds.

KUIP/UNITS

List all Input/Output logical units currently open. The files attached to them are also shown.

KUIP/EXIT

End of the interactive session.

KUIP/QUIT

End of the interactive session.

KUIP/FUNCTIONS

```
*** KUIP System Functions ***
```

The function name (and arguments) is literally replaced, at run-time, by its current value. At present, the following functions are available:

```
 $DATE  ....................  Current date in format DD/MM/YY
 $TIME  ....................  Current time in format HH.MM.SS
 $CPTIME  ..................  CP time elapsed since last call (in sec)
 $RTIME  ...................  Real time elapsed since last call (in sec)
 $VDIM(VNAME,IDIM)  ........  Physical length of vector VNAME
                              on dimension IDIM (1..3)
 $VLEN(VNAME,IDIM)  ........  As above, but for the logical length
                              (i.e. stripping trailing zeroes)
 $NUMVEC  ..................  Current number of vectors
 $VEXIST(VNAME)  ...........  Index of vector VNAME
                              (1..$NUMVEC or 0 if VNAME does not exist)
 $SUBSTRING(STRING,IX,NCH)  ...  STRING(IX:IX+NCH-1)
 $UPPER(STRING)  ...........  STRING changed to upper case
 $LOWER(STRING)  ...........  STRING changed to lower case
 $LEN(STRING)  .............  Length of STRING
 $INDEX(STR1,STR2)  ........  Position of first occurrence of STR2 in
STR1
 $WORDS(STRING,SEP)  .......  Number of words separated by SEP
 $WORD(STRING,K,N,SEP)  ....  Extract N words starting at word K
 $QUOTE(STRING)  ...........  Add quotes around STRING
 $UNQUOTE(STRING)  .........  Remove quotes around STRING
 $EXEC('macro args')  ......  EXITM value of EXEC call
 $DEFINED('var_name')  .....  List of defined macro variables
 $EVAL(Expression)  ........  Result of the Expression computed by KUIP
 $SIGMA(Expression)  .......  Result of the Expression computed by SIGMA
 $RSIGMA(Expression) .......  As above but a decimal point is added to
                              integer results
 $FORMAT(number,format)  ...  Format a number according to a Fortran
                              format string, e.g.
                              $FORMAT(1.5,F5.2) ==> ' 1.50'
                              $FORMAT(123,I5.5) ==> '00123'
 $ARGS  ....................  Command line at program invocation
 $KEYNUM  ..................  Address of latest clicked key in style GP
 $KEYVAL  ..................  Value of latest clicked key in style GP
 $LAST  ....................  Latest command line executed
 $ANUM  ....................  Number of aliases
 $ANAM(I)  .................  Name of I-th alias
 $AVAL(I)  .................  Value of I-th alias
 $STYLE  ...................  Current style as defined by SET/STYLE
 $OS  ......................  Operating system name, e.g. UNIX or VMS
 $MACHINE  .................  Hardware or Unix brand, e.g. VAX or HPUX
 $PID  .....................  Process ID
 $IQUEST(I)  ...............  Value of IQUEST(I) status vector
 $ENV(var)  ................  Value of environment variable
 $FEXIST(file)  ............  1 if file exists or 0 otherwise
 $SHELL(cmd,N)  ............  N'th line of shell command output (Unix
```

```
only)
 $SHELL(cmd,sep)  ............  Shell output with newlines replaced by sep
 $SHELL(cmd)  ................  Same as $SHELL(cmd,' ')

 $CALL('fun(args)')  .........  Call a Fortran REAL FUNCTION
 $ICALL('ifun(args)')  .......  Call an INTEGER FUNCTION
 $LCALL('lfun(args)')  .......  Call a LOGICAL FUNCTION and return 0 or 1
 $DCALL('dfun(args)')  .......  Call a DOUBLE PRECISION FUNCTION
 $HCDIR()  ...................  Current Hbook working directory
 $HEXIST(id)  ................  1 if histogram ID exists or 0 otherwise
 $HINFO(id,'ENTRIES')  .......  Number of entries
 $HINFO(id,'MEAN')  ..........  Mean value
 $HINFO(id,'RMS')  ...........  Standard deviation
 $HINFO(id,'EVENTS')  ........  Number of equivalent events
 $HINFO(id,'OVERFLOW')  ......  Content of overflow channel
 $HINFO(id,'UNDERFLOW')  .....  Content of underflow channel
 $HINFO(id,'MIN')  ...........  Minimum bin content
 $HINFO(id,'MAX')  ...........  Maximum bin content
 $HINFO(id,'SUM')  ...........  Total histogram content
 $HINFO(id,'NSLIX')  .........  Number of X slices
 $HINFO(id,'NSLIY')  .........  Number of Y slices
 $HINFO(id,'NBANX')  .........  Number of X bandes
 $HINFO(id,'NBANY')  .........  Number of Y bandes
 $HINFO(id,'NPROX')  .........  Projection X (0 or 1)
 $HINFO(id,'NPROY')  .........  Projection Y (0 or 1)
 $HINFO(id,'XBINS')  .........  Number of bins in X direction
 $HINFO(id,'XMIN')  ..........  Lower histogram limit in X direction
 $HINFO(id,'XMAX')  ..........  Upper histogram limit in X direction
 $HINFO(id,'YBINS')  .........  Number of bins in Y direction
 $HINFO(id,'YMIN')  ..........  Lower histogram limit in Y direction
 $HINFO(id,'YMAX')  ..........  Upper histogram limit in Y direction
 $HTITLE(id)  ................  Histogram title
 $GRAFINFO('XZONES')  ........  Number of zones in X direction
 $GRAFINFO('YZONES')  ........  Number of zones in Y direction
 $GRAFINFO('NT')  ............  Current Normalization Transformation number
 $GRAFINFO('WNXMIN')  ........  Lower X limit of window in current NT
 $GRAFINFO('WNXMAX')  ........  Upper X limit of window in current NT
 $GRAFINFO('WNYMIN')  ........  Lower Y limit of window in current NT
 $GRAFINFO('WNYMAX')  ........  Upper Y limit of window in current NT
 $GRAFINFO('VPXMIN')  ........  Lower X limit of viewport in current NT
 $GRAFINFO('VPXMAX')  ........  Upper X limit of viewport in current NT
 $GRAFINFO('VPYMIN')  ........  Lower Y limit of viewport in current NT
 $GRAFINFO('VPYMAX')  ........  Upper Y limit of viewport in current NT
 $GRAFINFO('TXALIH')  ........  Horizontal text alignment
 $GRAFINFO('TXALIV')  ........  Vertical text alignment
 $GRAFINFO('TXFONT')  ........  Text font
 $GRAFINFO('TXPREC')  ........  Text precision
```

```
 $GRAFINFO('?attr')  ..........  HPLOT/HIGZ attribute (see HELP SET for
valid names)
 $RGBINFO(icol,'R')  ..........  Weight of Red in color table
 $RGBINFO(icol,'G')  ..........  Weight of Green in color table
 $RGBINFO(icol,'B')  ..........  Weight of Blue in color table
 $CUT(n)  .....................  Cut expression $n
 $CUTEXPAND(string)  ..........  Replace $n in the (quoted) string by
$CUT(n)
```

KUIP/BUGREPORT `[ chopt ]`

`CHOPT` C "Options" `D='B'`

Possible `CHOPT` values are:

- `B` Send a bug report
- `C` Send a comment, suggestion, etc.

Email a bug report or comment to the PAW team. The local editor is invoked with a template to be filled out. After the template has been edited, version information about PAW and the operating system is appended. The user is asked for a confirmation before the report is send.
In Paw++ this command can be accessed via the "Help" menu of the "Executive Window" or the "Main Browser" (menu item "Mail Paw++ Developers").
This command is implemented only on UNIX, VMS and VM systems.

10.1 ALIAS

Operations with aliases. Aliases are defined to provide shortcut abbreviations for the input line or some part of it. When encountered on an input line an alias is replaced by its string value which can contain further aliases. (Be careful not to define recursive aliases.)
To juxtaposition aliases, a double slash can be used as concatenation sign. Inside quoted strings and for the ALIAS commands themselves the alias substitution is inhibited. Otherwise

```
ALIAS/CREATE ALPHA BETA
ALIAS/CREATE ALPHA BETA
```

whould create an recursive alias BETA and

```
ALIAS/CREATE ALPHA BETA
ALIAS/CREATE BETA GAMMA
ALIAS/DELETE ALPHA
```

would delete the alias name BETA instead of ALPHA itself.

KUIP/ALIAS/CREATE `name value [ chopt ]`

`NAME` C "Alias name"
`VALUE` C "Alias value"
`CHOPT` C "Option" `D='A'`

Possible CHOPT values are:

- A create an Argument alias
- C create a Command alias
- N No alias expansion of value

Create an alias NAME which should be substituted by VALUE. An alias name is a sequence of letters and digits starting with a letter. The underscores (’_’), the at-sign (’@’) and the dollar-sign (’$’) count as letters.
There are two types of aliases: Command aliases are recognized only if they occur in the command position, i.e. as the first token on the line. Argument aliases are recognized anywhere on the command line (except inside quoted strings) if they are surrounded by one of the following separators:

```
blank  /  ,  =  :  .  %  ’  (  )
```

Also switch ON the alias translation, i.e. ALIAS/TRANSLATION ON. If CHOPT=’C’ then the alias is a command alias, i.e. an alias that will only be translated when it is the first token on a command line. Example:

```
Alias/Create GG Graph/Struct/Scratch
Alias/Create FF File1/Name1/Name2
GG FF/ID
```

is equivalent to

```
Graph/Struct/Scratch File1/Name1/Name2/ID
```

```
Alias/Create LS DIR C
```

is equivalent to

```
DIR
```

only when LS is the first token on a command line. In the following case LS will not be translated

```
SHELL LS
```

Aliases occuring inside an value are expanded indepedent whether the value is enclosed by quotes. The option -N allows to suppress this implicit alias expansion.

KUIP/ALIAS/LIST [name]

NAME C “Alias name wildcard” D=’*’

List all aliases matching the wildcard (names and values).

KUIP/ALIAS/DELETE name

NAME C “Alias name wildcard” Loop

Delete the definition of aliases matching the wildcard. NAME=’*’ deletes all aliases.

KUIP/ALIAS/TRANSLATION [option]

OPTION C “Option” D=’ON’

Possible OPTION values are:

?	show current setting
ON	switch alias translation ON
OFF	switch alias translation OFF

Switch ON/OFF the alias translation. If OFF, alias definitions are not used in parsing the command lines. It is automatically switched ON when an alias is created. If OPTION='?' the current value is shown. The startup value is OFF.

10.2 SET˙SHOW

Set or show various KUIP parameters and options.

KUIP/SET˙SHOW/STYLE [option sgylen sgsize sgyspa sgbord wktype]

OPTION	C	"Option" D='?'
SGYLEN	R	"max Y LENgth of each menu item box" D=0.025 R=0.005:0.25
SGSIZE	R	"space available for the application" D=0.8 R=0:0.90
SGYSPA	R	"max Y length of space between menus" D=0.02 R=-0.5:0.50
SGBORD	R	"X or Y border for menus" D=0.015 R=0:0.25
WKTYPE	I	"Graphics workstation type" D=0

Possible OPTION values are:

?	show current style
C	Command line : select Command line input
AN	Menu with Numbers : select general Alpha menu (with Numbers)
AL	Menu with Letters : select general Alpha menu (with Letters)
G	Graphics menu hardware : select Graphics menu (with hardware character fonts)
GW	Graphics menu shadowed : select Graphics menu (with shadowed Width effect)
GS	Graphics menu Software : select Graphics menu (with Software character fonts)
GP	Panel keys : select Graphics menu (with Panel keys only, i.e. no command tree menu)
XM	Motif/X11 : select Motif/X11 interface

Select the user dialogue style (or working mode). The startup value is 'C' (command mode). The current value is returned by the system function $STYLE.

The G-styles are only available if the application program is calling KUWHAG instead of KUWHAT. When one of these options is choosen the remaining parameters control the geometrical layout of the menus on the screen and the graphics workstation type (in case HIGZ was not initialized).

Style 'XM' is only available if the program is calling KUWHAM. In that case switching to other styles is not possible.

KUIP/SET˙SHOW/PANEL line [gkey]

LINE	R	"Line number" D=0
GKEY	C	"Graphics key value(s)" D='␣'

Set up a panel of graphics keys. The "panel interface" is available in "STYLE GP" and in KUIP/Motif (not in the basic command mode). N.B. in "STYLE GP" only one panel of commands can be set up, whereas in KUIP/Motif there is no limitation.

Examples:

```
PANEL  0                        | reset the panel (in memory)
PANEL  2 A/L QUIT V/L           | initialize line 2 with 3 graphics keys,
                                  respectively A/L, QUIT, V/L
PANEL  2 A/L ' '  V/L ' ' ' '   | initialize line 2 with 5 graphics keys,
                                  and fill 1st and 3rd keys
PANEL  2.04 MESSAGE             | initialize 4th key of 2nd line to MESSAGE
PANEL  2.04                     | clear 4th key of 2nd line
PANEL -2.08                     | initialize line 2 with 8 graphics keys
PANEL -6.16                     | initialize line 6 with 16 graphics keys
```

Note that the key number on the right of the decimal point must always be defined with two digits.

Keys ending with a minus sign make an additional request of keyboard input; the complete command line will be the key text, with a blank at the place of the minus, concatenated with the additional keyboard input. Example:

```
PANEL 1.03 'VEC/PRI-'        | entering VAB will execute VEC/PRI VAB.
```

Keys ending with a double minus sign behave as above but no blank is put at the place of the double minus. Example:

```
PANEL 1.03 'VEC/PRI V--'     | entering AB will execute VEC/PRI VAB
```

The dollar sign inside a key is replaced by additional keyboard input. Example:

```
PANEL 1.03 'VEC/PRI V($)'    | entering 11:20 will execute VEC/PRI V(11:20)
```

In KUIP/Motif there are 2 addditinal commands in order to display or to close one panel:

```
PANEL  0 D [title] [geometry]
PANEL  0 C [title]
```

Examples:

```
- PANEL  0 D 'This is my first panel' 500x300+500+600
```

displays the panel which has been set in memory by the key definition, and sets the title to “This is my first panel”, the window size to “500x300” (WxH) and the window position to “500 600” in x and y. If no title and/or no geometry is specified one is given by default.

```
- PANEL  0 C 'This is my first panel'
```

closes (destroys and erases from the screen) the panel whith title “This is my first panel”. If no title is specified the last created panel is closed by default.

KUIP/SET˙SHOW/NEWPANEL `line col title width height xpos ypos`

LINE	I	"Number of lines" D=5 R=1:30
COL	I	"Number of columns" D=5 R=1:30
TITLE	C	"Panel Title" D='New Panel'
WIDTH	I	"Panel width (in pixels)" D=300 R=10:
HEIGHT	I	"Panel height (in pixels)" D=300 R=10:
XPOS	I	"X Position (in pixels)" D=0 R=0:
YPOS	I	"Y Position (in pixels)" D=0 R=0:

Set up a new panel with empty keys (to be filled interactively).

KUIP/SET˙SHOW/COMMAND `[ chpath ]`

CHPATH	C	"Path name for command line" D='␣'

Set a filter for the parsing of command lines. If it has been called, it means that whenever a command line is entered, if and only if it is not an existing command (not just ambiguous), it is inserted into the CHPATH string, with $n (n=1..9) being replaced by the n-th token of the command (tokens are separated by spaces), or $* being replaced by the whole command line. Examples:

```
COMMAND 'V/CR $*(10)'
AA                      =>   V/CR AA(10)
BB                      =>   V/CR BB(10)
V/LIST                  =>   V/LIST

COMMAND 'VECTOR/PLOT $1 555 $2'
AA E                    =>   VECTOR/PLOT AA 555 E
BB                      =>   VECTOR/PLOT BB 555

COMMAND                 =>   shows its current value
COMMAND *               =>   reset (equivalent to COMMAND $*)
```

Note that COMMAND and subsequent command lines can be used inside macros, excepted when producing macro statements (like EXEC, IF, GOTO, etc.). For example, the above examples would work also inside macros, while COMMAND 'EXEC $*' or COMMAND 'GOTO $1' will not.

KUIP/SET˙SHOW/APPLICATION `path [ cmdex ]`

PATH	C	"Application name" D='␣'
CMDEX	C	"Exit command" D='EXIT'

Set the application name. This means that all input lines will be concatenated to the string PATH (until the command specified by the parameter CMDEX is executed, which resets the application to the null string). The value of CMDEX may be specified if the default value EXIT has to be changed (i.e. because already used by the application). APPLICATION can also be inserted in a macro: in this case at least 4 characters must be specified (i.e. APPL).

KUIP/SET˙SHOW/ROOT [path]

PATH C "Root directory" D='/'

Set the root for searching commands. If PATH='?' the current root is shown. This allows to access commands regardless of possible ambiguities with different menus. Commands are first searched starting from the current root: if a command is found it is executed. Only if a command is not found a second pass of search is done, starting now from the top root of the command tree (i.e. '/').

KUIP/SET˙SHOW/TIMING [option]

OPTION C "Option" D='ON'

Possible OPTION values are:

ON
OFF
ALL

Set ON/OFF/ALL the timing of commands. If ON, the real time and the CPU time for the latest executed command (or macro) are presented. If ALL, the time is shown for each command being executed within a macro. The startup value is OFF.

KUIP/SET˙SHOW/PROMPT prompt

PROMPT C "Prompt string" D='␣'

Set the prompt string for the command mode dialogue. If PROMPT is blank the current prompt is left unchanged. If PROMPT contains the character sequence '[]' the current command number is inserted between the square brackets.

KUIP/SET˙SHOW/BREAK [option]

OPTION C "Option" D='ON'

Possible OPTION values are:

ON
OFF
TB
?

Set ON/OFF the break handling. If OPTION='?' the current value is shown. The startup value is ON.

Hitting the keyboard interrupt (CTRL/C on VMS or CTRL/Q on the Apollo) under break ON condition, the current command or macro execution will be interrupted and the user will get again the application prompt.

BREAK TB switch ON the traceback of the routines called, with their line numbers, when an error occurs.

This allows the detection of the routines which provoked the error.

KUIP/SET˙SHOW/COLUMNS `[ ncol ]`

`NCOL` I "Number of columns for terminal output" `D=80 R=-1:`

Set the maximum number of columns for terminal output. If NCOL=0 the current number of columns is shown. If NCOL=-1 the current number of columns is taken from the environment variable COLUMNS. If COLUMNS is undefined the startup value is 80.

KUIP/SET˙SHOW/RECORDING `[ nrec ]`

`NREC` I "Rate for recording on history file" `D=25 R=0:`

Set the recording rate for the history file. Every NREC commands of the session the current history file is updated. If NREC=0 the history is not kept at all (i.e. the file is not written). See also the command LAST.

KUIP/SET˙SHOW/HOST˙EDITOR `[ editor top left width height dxpad dypad npads ]`

`EDITOR`	C	"Host editor command" `D='?'`
`TOP`	I	"Top position of the edit window" `D=20 R=0:`
`LEFT`	I	"Left position of the edit window" `D=20 R=0:`
`WIDTH`	I	"Width of the edit window" `D=0 R=0:`
`HEIGHT`	I	"Height of the edit window" `D=0 R=0:`
`DXPAD`	I	"X offset for help PAD windows" `D=30 R=0:`
`DYPAD`	I	"Y offset for help PAD windows" `D=20 R=0:`
`NPADS`	I	"Maximum number of shifted pads" `D=4 R=1:`

Set the host command to invoke the editor. The EDIT command will invoke this editor. If EDITOR='?' the current host editor command is shown.

On Apollo the special value EDITOR='DM' invoke Display Manager pads. The special values EDITOR='WINDOW' and 'PAD' can be used to specify the window positions (in pixel units). 'WINDOW' defines the parameters for edit pads, while 'PAD' defines the parameters for read-only pads (e.g. used by 'HELP -EDIT').

On VMS the special values EDITOR='EDT' and 'TPU' invoke the callable editors. The startup time is considerably lower compared to spawning the editor as a subprocess. The callable EDT has one disadvantage though: after an error, e.g. trying to edit a file in a non-existing directory, subsequent calls will always fail. The TPU call can be augmented by command line options, e.g.

```
HOST_EDITOR TPU/DISP=DECW    | DECwindow interface to EVE
```

On Unix a variety of editors are available, e.g.

```
HOST_EDITOR vi
HOST_EDITOR 'emacs -geometry 80x48'
```

On Unix workstations it is possible to do asynchronous editing via the KUIP edit server, i.e. to start an editor in a separate window while the application can continue to receive commands. In order to do that the following conditions must be fulfilled:

```
- The KUIP edit server 'kuesvr' must be found in the search path.
- The editor command set by HOST_EDITOR must end with an ampersand ('&').
- The environment variable 'DISPLAY' must be set.
```

The ampersand flags your intention to use the edit server if possible. If the edit server cannot be used the ampersand will be ignored, i.e. even with

```
HOST_EDITOR 'vi &'
```

the KUIP/EDIT command will block until the editor terminates if either the 'kuesvr' is not available or 'DISPLAY' is undefined. When using the edit server the editor command is expected to create its own window. 'vi' being a frequent choice, the above command is automatically interpreted as

```
HOST_EDITOR 'xterm -e vi &'
```

The startup value can be defined by the environment variable 'EDITOR'. Otherwise it is set to a system dependent default: 'DM' (Apollo), 'EDT' (VMS), 'XEDIT' (VM/CMS), 'vi' (Unix).

KUIP/SET˙SHOW/HOST˙PAGER `[ pager ]`

`PAGER` C "Host pager command" `D='?'`

Set the host command to view a file in read-only mode. If OPTION='?' the current host pager command is shown. The 'HELP -EDIT' command will invoke this pager, e.g.

```
HOST_PAGER more
```

On Unix workstations the pager can be asynchronous by creating a separate window, e.g.

```
HOST_PAGER 'xterm -e view &'
HOST_PAGER 'ved &'
```

On Apollo the special value PAGER='DM' defines the use of Display Manager read-only pads. The pad positions can be adjusted by the HOST˙EDITOR command.

The startup value can be defined by the environment variables 'KUIPPAGER' or 'PAGER'. If neither of them is defined the value set by the HOST˙EDITOR command is used. On VAX/VMS the startup value is 'TYPE/PAGE'.

KUIP/SET˙SHOW/HOST˙PRINTER `[ command filetype ]`

`COMMAND` C "Host printer command" `D='?'`
`FILETYPE` C "File extension" `D='␣'`

Set the host commands for printing files with KUIP/PRINT. The KUIP/PRINT command will use the host command matching the file extension or use the default command defined for FILETYPE=' '.

If COMMAND='?' the currently set commands are shown. If COMMAND=' ' the currently defined command is delete. The command string can contain '$*' and '$-' to indicate the position where the file name with/without file extension should be inserted. For example,

```
MANUAL / refman.tex latex
HOST_PRINTER 'latex $* ; dvips $-' .tex
KUIP/PRINT refman.tex
```

invokes the shell command 'latex refman.tex ; dvips refman'. The predefined defaults are not guaranteed to work since the actual print commands are very much installation dependent.

KUIP/SET˙SHOW/HOST˙PSVIEWER `[ psviewer ]`

`PSVIEWER` C "Host PostScript Viewer command" `D='?'`

Set the host command to invoke the PostScript Viewer. The PSVIEW command will invoke this PostScript Viewer. If PSVIEWER='?' then the current viewer command is shown.

The startup value can be defined by the environment variables 'KUIPPSVIEWER' or 'PSVIEWER'.

On Unix workstations it is by default set to 'ghostview'. On VAX/VMS the default commands is 'VIEW/FORM=PS/INTERFA

KUIP/SET˙SHOW/HOST˙SHELL `[ shell ]`

`SHELL` C "Host shell command" `D='?'`

Set the default host shell invoked by the KUIP/SHELL command. If OPTION='?' the current host shell is shown. The startup value is taken from the 'SHELL' environment variable.

KUIP/SET˙SHOW/RECALL˙STYLE `[ option ]`

`OPTION` C "Command recall and editing style" `D='?'`

Possible `OPTION` values are:

`?`	show current setting
`KSH`	Korn shell : Emacs like command line editing
`KSHO`	Korn shell + Overwrite : like 'KSH' but overwrite instead of insert mode
`DCL`	VAX/VMS DCL : DCL command line editing
`DCLO`	VAX/VMS DCL + Overwrite : like 'DCL' but overwrite instead of insert mode
`NONE`	disable command line editing

Set the command recall and editing style. If OPTION='?' the current style is shown. The startup value is 'DCL' on VAX/VMS, 'NONE' on Cray and Apollo DM pads, and 'KSH' on other systems.

If the terminal emulator returns ANSI escape sequences (hpterm doesn't!) the up/down arrow keys can be used to recall items from the command history list and the left/right arrow keys to move the cursor.

'KSH' style provides the following control keys for editing:

```
^A/^E   : Move cursor to beginning/end of the line.
^F/^B   : Move cursor forward/backward one character.
^D      : Delete the character under the cursor.
^H, DEL : Delete the character to the left of the cursor.
^K      : Kill from the cursor to the end of line.
```

```
^L       : Redraw current line.
^O       : Toggle overwrite/insert mode. Text added in overwrite mode
           (including yanks) overwrites existing text, while insert mode
           does not overwrite.
^P/^N    : Move to previous/next item on history list.
^R/^S    : Perform incremental reverse/forward search for string on
           the history list.  Typing normal characters adds to the
           current search string and searches for a match.  Typing
           ^R/^S marks the start of a new search, and moves on to
           the next match.  Typing ^H or DEL deletes the last
           character from the search string, and searches from the
           starting location of the last search.
           Therefore, repeated DELs appear to unwind to the match
           nearest the point at which the last ^R or ^S was typed.
           If DEL is repeated until the search string is empty the
           search location begins from the start of the history
           list. Typing ESC or any other editing character accepts
           the current match and loads it into the buffer,
           terminating the search.
^T       : Toggle the characters under and to the left of the cursor.
^U       : Kill from the prompt to the end of line.
^Y       : Yank previously killed text back at current location.
           Note that this will overwrite or insert, depending on
           the current mode.
TAB      : By default adds spaces to buffer to get to next TAB stop
           (just after every 8th column).
LF, CR   : Returns current buffer to the program.
```

'DCL' style provides the following control keys for editing:

```
BS/^E    : Move cursor to beginning/end of the line.
^F/^D    : Move cursor forward/backward one character.
DEL      : Delete the character to the left of the cursor.
^A       : Toggle overwrite/insert mode.
^B       : Move to previous item on history list.
^U       : Delete from the beginning of the line to the cursor.
TAB      : Move to next TAB stop.
LF, CR   : Returns current buffer to the program.
```

KUIP/SET˙SHOW/VISIBILITY `cmd [ chopt ]`

`CMD` C "Command name" `D='␣'`
`CHOPT` C "?, OFF, ON" `D='?'`

Possible `CHOPT` values are:

```
?
```

```
OFF
ON
```

Set or show the visibility attributes of a command.
If CHOPT='OFF':

```
 - the command it is not executable anymore
 - STYLE G draws a shadowed box on the command
 - HELP may be still requested on the command
```

The startup value is ON.

KUIP/SET˙SHOW/DOLLAR `[ option ]`

`OPTION` C "Substitution of environment variables" `D='?'`

Possible `OPTION` values are:

`?`	show current setting
`ON`	enable substitution
`OFF`	disable substitution

Set or show the status of environment variable substitution.
This command allows to enable/disable the interpretation of environment variables in command lines. The startup value is 'ON', i.e. "$var" is substituted by the variable value.
Note that the system function "$ENV(var)" allows using environment variables even for 'DOLLAR OFF'
.

KUIP/SET˙SHOW/FILECASE `[ option ]`

`OPTION` C "Case conversion for filenames" `D='?'`

Possible `OPTION` values are:

`?`	show current setting
`KEEP`	filenames are kept as entered on the command line
`CONVERT`	filenames are case converted
`RESTORE`	restore previous FILECASE setting

Set or show the case conversion for filenames.
This command has only an effect on Unix systems to select whether filenames are kept as entered on the command line. The startup value is 'CONVERT', i.e. filenames are converted to lowercase.
On other systems filenames are always converted to uppercase.
The 'RESTORE' option set the conversion mode to the value effective before the last FILECASE KEEP/CONVERT command. E.g. the sequence

```
FILECASE KEEP; EDIT Read.Me; FILECASE RESTORE
```

forces case sensitivity for the EDIT command and restores the previous mode afterwards.

KUIP/SET˙SHOW/LCDIR `[ directory ]`

`DIR*ECTORY` C "Directory name" `D='␣'`

Set or show the local working directory.

The current working directory is set to the given path name or the current directory is shown.

To show the current directory used LCDIR without argument. 'LCDIR ' switches to the home directory. 'LCDIR .' switches back to the working directory at the time the program was started.

Chapter 11: MACRO

Macro Processor commands.

MACRO/EXEC mname [margs]

MNAME C "Macro name"
MARGS C "Macro arguments" D='␣' Separate

Execute the command lines contained in the macro MNAME. As a file can contain several macros, the character '#' is used to select a particular macro inside a file as explained below.

If MNAME does not contain the character '#', the file MNAME.KUMAC is searched and the first macro is executed (it may be an unnamed macro if a MACRO statement is not found as first command line in the file).

If MNAME is of the form FILE#MACRO, the file named FILE.KUMAC is searched and the macro named MACRO is executed.

Examples:

```
EXEC ABC   to exec first (or unnamed) macro of file ABC.KUMAC
EXEC ABC#M to exec macro M of file ABC.KUMAC
```

The command MACRO/DEFAULTS can be used to define a directory search path for macro files.

MACRO/LIST [mname]

MNAME C "Macro name pattern" D='␣'

List all macros in the search path defined by MACRO/DEFAULTS. Macros are files with the extension KUMAC. MNAME may be specified to restrict the list to the macros containing such a string in the first part of their name. For example,

```
MACRO/LIST ABC
```

will list only macros starting with ABC.

MACRO/TRACE [option level]

OPTION C "Option" D='ON'
LEVEL C "Level" D='␣'

Possible OPTION values are:

```
ON
OFF
```

Possible LEVEL values are:

```
'␣'
```

```
TEST
WAIT
FULL
DEBUG
```

Set ON/OFF the trace of commands during macro execution. If TRACE='ON' the next command is written on the terminal before being executed. If LEVEL='TEST' the command is only echoed but not executed. If LEVEL='WAIT' the command WAIT is automatically inserted after the execution of each command. The startup values are OPTION='OFF' and LEVEL=' '.

MACRO/DEFAULTS `[ path option ]`

```
PATH    C  "Search path for macro files"  D='?'
OPTION  C  "Automatic EXEC"  D='?'
```

Possible `OPTION` values are:

`?`	show current setting
`Command`	search for commands only
`C`	same as 'Command'
`Auto`	search for commands before macros
`A`	same as 'Auto'
`AutoReverse`	search for macros before commands
`AR`	same as 'AutoReverse'

Set or show MACRO search attributes.

On Unix and VMS systems PATH defines a comma separated list of directories in which the commands KUIP/EDIT, MACRO/EXEC, and MACRO/LIST search for macro files. For example,

```
MACRO/DEFAULT '.,macro,~/macro'          | Unix
MACRO/DEFAULT '[],[.macro],[macro]'      | VMS
```

defines to search files first in the current directory, then in the subdirectory 'macro' of the current directory, and last the subdirectory 'macro' of the home directory.

On VM/CMS system PATH defines a comma separated list of filemodes. E.g.

```
MACRO/DEFAULT '*'        | search all disks
MACRO/DEFAULT 'A,C'      | search only disks A and C
```

If PATH='?' the currently defined search path is shown. If PATH='.' the search path is undefined, i.e. files are search for in the current directory (A-disk on VM/CMS) only. The startup value is PATH='.'.

The search path is not applied if the file specification already contains an explicit directory path or if it starts with a '-' character (which is stripped off).

OPTION allows to define whether macros can be invoked by their name only without prepending the KUIP/EXEC command:

```
DEFAULT -Command
CMD                     | CMD must be a command
DEFAULT -Auto
CMD                     | if CMD is not a command try EXEC CMD
DEFAULT -AutoReverse
CMD                     | try EXEC CMD first; if not found try command CMD
```

The startup value is 'Command' (also reset by PATH='.').

Important note:

Inside macros the DEFAULT -A (or -AR) logic is disabled, i.e. DEFAULT -C is always assumed.

MACRO/DATA

Application command to store immediate data into a file. Example:

```
Application DATA vec.dat
1  2  3
4  5  6
7  8  9
vec.dat
vec/read x,y,z vec.dat
```

11.1 GLOBAL

Operations on global variables.

MACRO/GLOBAL/CREATE `name [ value text ]`

`NAME` C "Variable name" `Loop`
`VALUE` C "Initial value" `D='␣'`
`TEXT` C "Comment text" `D='␣'`

Create a global variable.

If used inside a macro the variable [name] is declared as global.

MACRO/GLOBAL/IMPORT `name`

`NAME` C "Variable name" `Loop`

Import global variables.

If used inside a macro the variables listed are declared as global. The name may contain '*' as a wildcard matching any sequence of characters.

MACRO/GLOBAL/DELETE `name`

`NAME` C "Variable name" `Loop`

Delete global variables.

The global variables listed are deleted. The name may contain '*' as a wildcard matching any sequence of characters.

MACRO/GLOBAL/LIST `[ name file ]`

`NAME` C "Variable name" `D='*'`
`FILE` C "Output file" `D='␣'`

List global variables.

If a file name is specified the output is the list of GLOBAL/CREATE commands to define the selected global variables. The default file extension is .kumac.

11.2 SYNTAX

Explanation of KUIP macro syntax.

A macro is a set of command lines stored in a file, which can be created and modified with any text editor.

In addition to all available KUIP commands the special "macro statements" listed below are valid only inside macros. Note that the statement keywords are fixed. Aliasing such as "ALIAS/CREATE jump GOTO" is not allowed.

11.2.1 Expressions

Explanation of KUIP expression syntax.

KUIP has a built-in parser for different kinds of expressions: arithmetic expressions, boolean expressions, string expressions, and "garbage expressions".

MACRO/SYNTAX/Expressions/Arithmetic

Explanation of arithmetic expression syntax.

The syntactic elements for building arithmetic expressions are:

```
expr ::=  number
        | vector-name                (for scalar vectors)
        | vector-name(expr)
        | vector-name(expr,expr)
        | vector-name(expr,expr,expr)
        | [variable-name]            (if value is numeric or
                                      the name of a scalar vector)
        | [variable-name](expr...)   (if value is a vector name)
        | alias-name                 (if value is numeric constant)
        | $system-function(...)
        | - expr
        | expr + expr
        | expr - expr
        | expr * expr
        | expr / expr
        | (expr)
        | ABS(expr)
        | INT(expr)
        | MOD(expr,expr)
```

They can be used in the macro statements DO, FOR, and EXITM, in macro variable assignments, as system function arguments where a numeric value is expected, or as the argument to the $EVAL function.

Note that all arithmetic operations are done in floating point, i.e., "5/2" becomes "2.5". If a floating point result appears in a place where an integer is expected, for example as an index, the value is truncated.

MACRO/SYNTAX/Expressions/Boolean

Explanation of Boolean expression syntax.

Boolean expressions can only be used in the macro statements IF, WHILE, and REPEAT. The possible syntactic elements are shown below.

```
 bool  ::= expr rel-op expr
         | string eq-op string
         | expr eq-op string
         | .NOT. bool
         | bool .AND. bool
         | bool .OR. bool
         | ( bool )

rel-op ::= .LT. | .LE. | .GT. | .GE.
         |  <   |  <=  |  >   |  >=
         | eq-op

eq-op  ::= .EQ. | .NE.
         |  =   | <>
```

MACRO/SYNTAX/Expressions/String

Explanation of string expression syntax.

String expressions can be used in the macro statements CASE, FOR, and EXITM, in macro variable assignments, as system function arguments where a string value is expected, or as the argument to the $EVAL function. They may be constructed from the syntactic elements shown below.

```
string ::= quoted-string
         | unquoted-string
         | string // string            (concatenation)
         | expr // string              (expr represented as string)
         | [variable-name]
         | alias-name
         | $system-function(...)
```

MACRO/SYNTAX/Expressions/Garbage

Explanation of "garbage" expression syntax.

Expressions which do not satisfy any of the other syntax rules we want to call "garbage" expressions. For example,

```
s = $OS$MACHINE
```

is not a proper string expression. Unless they appear in a macro statement where specifically only an arithmetic or a boolean expression is allowed, KUIP does not complain about these syntax errors. Instead the following transformations are applied:

- alias substitution
- macro variable replacement; values containing a blank character are implicitly quoted
- system function calls are replaced one by one with their value provided that the argument is a syntactically correct expression
- string concatenation

11.2.2 Variables

Explanation of KUIP macro variables.
Macro variables do not have to be declared. They become defined by an assignment statement,

```
name = expression
```

The right-hand side of the assignment can be an arithmetic expression, a string expression, or a garbage expression (see MACRO/SYNTAX/Expressions). The expression is evaluated and the result is stored as a string (even for arithmetic expressions).
A variable value can be used in other expressions or in command lines by enclosing the name in square brackets, [name]. If the name enclosed in brackets is not a macro variable then no substitution takes place.

MACRO/SYNTAX/Variables/Numbered

Accessing macro arguments.
The EXEC command can pass arguments to a macro. The arguments are assigned to the numbered variables [1], [2], etc., in the order given in the EXEC command. The name of the macro, including the file specification, is assigned to [0].
A numbered variable cannot be redefined, i.e., an assignment such as "1 = foo" is illegal. See MACRO/SYNTAX/SHIFT.

MACRO/SYNTAX/Variables/Special

Predefined special macro variables.
For each macro the following special variables are always defined:

```
[0]     Fully qualified name of the macro.
[#]     Number of macro arguments
[*]     List of all macro arguments, separated by blanks
[@]     EXITM return code of the last macro called by
        the current one.  The value is "0" if the last
        macro did not supply a return code or no macro
        has been called yet.
```

As for numbered variables these names cannot be used on the left-hand side of an assignment. The values or [#] and [*] are updated by the SHIFT statement.

MACRO/SYNTAX/Variables/Indirection

Referencing a macro variable indirectly.

Macro variables can be referenced indirectly. If the variable [name] contains the name of another variable the construct

```
[%name]
```

is substituted by that other variable's value. For example, this is another way to traverse the list of macro arguments:

```
DO i=1,[#]
  arg = [%i]
  ...
ENDDO
```

There is only one level of indirection, i.e., the name contained in "name" may not start with another "%".

MACRO/SYNTAX/Variables/Global

Declaring a global variable.

```
EXTERN name ...
```

The variable names listed in the EXTERN statement are declared as global variables. If a name has not been defined with the GLOBAL/CREATE command, it is created implicitly and initialized to the empty string. The name list may contain wildcards, for example

```
EXTERN *
```

makes all defined global variables visible.

MACRO/SYNTAX/Variables/READ

Reading a variable value from the keyboard.

```
READ name  [ prompt ]
```

Variable values can be queried from the user during macro execution. The READ statement prompts for the variable value. If name is already defined the present value will be proposed as default.

MACRO/SYNTAX/Variables/SHIFT

Manipulation numbered variables.

The only possible manipulation of numbered variables is provided by the SHIFT statement which copies [2] into [1], [3] into [2], etc., and discards the value of the last defined numbered variable. For example, the construct

```
WHILE [1] <> ' ' DO
  arg = [1]
  ...
  SHIFT
ENDDO
```

allows to traverse the list of macro arguments.

11.2.3 Definitions

Statements for defining macros.

MACRO/SYNTAX/Definitions/MACRO

Defining a macro.
A .kumac file may contain several macros. An individual macro has the form

```
MACRO macro-name [ parameter-list ]
   statements
RETURN
```

Each statement is either a command line or one of the macro constructs described in this section (MACRO/SYNTAX). For the first macro in the file the MACRO header can be omitted. For the last macro in the file the RETURN trailer may be omitted. Therefore a .kumac file containing only commands (like the LAST.KUMAC) already constitutes a valid macro.

MACRO/SYNTAX/Definitions/RETURN

Ending a macro definition

```
RETURN [ value ]
```

The RETURN statement flags the end of the macro definition and not the end of macro execution, i.e., the construct

```
IF ... THEN
  RETURN          | error!
ENDIF
```

is illegal. See MACRO/SYNTAX/EXITM.
The value is stored into the variable [@] in the calling macro. If no value is given it defaults to zero.

MACRO/SYNTAX/Definitions/EXITM

Terminate macro execution and return to calling macro.

```
EXITM [ value ]
```

In order to return from a macro prematurely the EXITM statement must be used. The value is stored into the variable [@] in the calling macro. If no value is given it defaults to zero.

MACRO/SYNTAX/Definitions/STOPM

Terminate macro execution and return to command line prompt.

```
STOPM
```

The STOPM statement unwinds nested macro calls and returns to the command line prompt.

MACRO/SYNTAX/Definitions/ENDKUMAC

Ignore rest of KUMAC file.
A logical "end of file" marker. The KUIP parser will not read any part of a .kumac file which appears after the "ENDKUMAC" command.

11.2.4 Branching

Macro statements for general flow control.

MACRO/SYNTAX/Branching/CASE

Select one of many branches.

```
CASE expression IN
(label)  statement  [ statements ]
...
(label)  statement  [ statements ]
ENDCASE
```

The CASE switch evaluates the string expression and compares it one by one against the label lists until the first match is found. If a match is found the statements up to the next label are executed before skipping to the statement following the ENDCASE. None of the statements are executed if there is no match with any label.

Each label is a string constant and the comparison witht the selection expression is case-sensitive. If the same statement sequence should be executed for distinct values a comma-separated list of values can be used.

The "*" character in a label item acts as wildcard matching any string of zero or more characters, i.e., "(*)" constitutes the default label.

MACRO/SYNTAX/Branching/GOTO˙and˙IF˙GOTO

Unconditional and conditional branching.

```
GOTO label
```

The simplest form of flow control is provided by the GOTO statement which continues execution at the statement following the target "label:". If the jump leads into the scope of a block statement, for example a DO-loop, the result is undefined.

The target may be given by a variable containing the actual label name.

```
IF expression GOTO label
```

This old-fashioned construct is equivalent to

```
IF expression THEN
   GOTO label
ENDIF
```

MACRO/SYNTAX/Branching/IF˙THEN

Conditional execution of statement blocks.

```
IF expression THEN
   statements
ELSEIF expression THEN
   statements
...
ELSEIF expression THEN
   statements
ELSE
   statements
ENDIF
```

The general IF construct executes the statements following the first IF/ELSEIF clause for with the boolean expression is true and then continues at the statement following the ENDIF. The ELSEIF clause can be repeated any number of times or can be omitted altogether. If none of the expressions is true, the statements following the optional ELSE clause are executed.

MACRO/SYNTAX/Branching/ON˙ERROR

Installing an error handler.

Each command returns a status code which should be zero if the operation was successful or non-zero if any kind of error condition occurred. The status code can be tested by $IQUEST(1) system function.

```
ON ERROR GOTO label
```

installs an error handler which tests the status code after each command and branches to the given label when a non-zero value is found. The error handler is local to each macro.

```
ON ERROR EXITM  [ expression ]
```

and

```
ON ERROR STOPM
```

are short-hand notations for a corresponding EXITM or STOPM statement at the targat label.

```
ON ERROR CONTINUE
```

continues execution with the next command independent of the status code. This is the initial setting when entering a macro.

```
OFF ERROR
```

An error handler can be deactivated by this statement.

```
ON ERROR
```

An error handler can be reactivated by this statement.

11.2.5 Looping

Macro statements for construction loops.

MACRO/SYNTAX/Looping/DO

Loop incrementing a loop counter.

```
DO loop = start_expr, finish_expr  [, step_expr ]
   statements
ENDDO
```

The step size (setp˙expr) defaults to "1". The arithmetic expressions involved can be floating point values but care must be taken of rounding errors.

Note that "DO i=1,0" results in zero iterations and that the expressions are evaluated only once.

MACRO/SYNTAX/Looping/FOR

Loop over items in an expression list.

```
FOR name IN expr_1 [ expr_2 ... expr_n ]
   statements
ENDFOR
```

In a FOR-loop the number of iterations is determined by the number of items in the blank-separated expression list. The expression list must not be empty. One by one each expression evaluated and assigned to the variable name before the statements are executed.

The expressions can be of any type: arithmetic, string, or garbage expressions, and they do not need to be all of the same type. In general each expression is a single list item even if the result contains blanks.

The variable [*] is treated as a special case being equivalent to the expression list "[1] [2] ... [n]" which allows yet another construct to traverse the macro arguments:

```
FOR arg IN [*]
   ...
ENDFOR
```

MACRO/SYNTAX/Looping/REPEAT

Loop until condition becomes true.

```
REPEAT
   statements
UNTIL expression
```

The body of a REPEAT-loop is executed at least once and iterated until the boolean expression evaluates to true.

MACRO/SYNTAX/Looping/WHILE

Loop while condition is true.

```
WHILE expression DO
   statements
ENDWHILE
```

The WHILE-loop is iterated while the boolean expression evaluates to true. The loop body is not executed at all if the boolean expression is false already in the beginning.

MACRO/SYNTAX/Looping/BREAKL

Terminate a loop.

```
BREAKL [ level ]
```

Allows to terminate a loop prematurely. The BREAKL continues executing after the end clause of a DO, FOR, WHILE, or REPEAT block, where "level" indicates how many nested constructs to terminate. The default value level=1 terminates the innermost loop construct.

MACRO/SYNTAX/Looping/NEXTL

Continue with next loop iteration.

```
NEXTL [ level ]
```

Allows to continue with the next loop iteration without executing the rest of the loop body. Execution continues just before the end clause of a DO, FOR, WHILE, or REPEAT block, where "level" indicates how many nested blocks to skip. The default value level=1 skips to the end of the innermost loop construct.

Chapter 12: VECTOR

Vector Processor commands. Vectors are equivalent to FORTRAN 77 arrays and they use the same notation except when omitting indexes (see last line below). Up to 3 dimensions are supported. Examples:

```
Vec(20) (mono-dimensional with 20 elements)
```

may be addressed by:

```
Vec          for all elements
Vec(13)      for element 13-th
Vec(12:)     for elements 12-th to last
Vec(:10)     for elements first to 10-th
Vec(5:8)     for elements 5-th to 8-th

Vec(3,100) (2-dimensional with 3 columns by 100 rows):
```

may be addressed by:

```
Vec(2,5:8)   for elements 5-th to 8-th in 2-nd column
Vec(2:3,5:8) for elements 5-th to 8-th in 2-nd to 3-rd columns
Vec(2,5)     for element 5-th in 2-nd column
Vec(:,3)     for all elements in 3-rd row
Vec(2)       for all elements in 2-nd column (SPECIAL CASE)
```

The latest line shows the special (and non-standard with FORTRAN 77) notation such that missing indexes are substituted to the right.

An 'invisible' vector called '?', mono-dimensional and of length 100, is always present. Is is used for communicating between user arrays and KUIP vectors, being equivalenced with the real array VECTOR(100) in the labeled common block /KCWORK/.

VECTOR/CREATE `vname [ type values ]`

`VNAME`	C	"Vector name(length)"
`TYPE`	C	"Vector type" `D='R'`
`VALUES`	C	"Value list" `D='␣' Separate Vararg`

Possible `TYPE` values are:

```
R
I
```

Create a vector named VNAME (elements are set to zero). The dimensions are taken from the name, for example VEC(20), VEC(3,100), VEC(2,2,10). Up to 3 dimensions are supported. Dimensions which are not specified are taken to 1, for example VEC(10) —⟩ VEC(10,1,1) and VEC —⟩ VEC(1,1,1). The vector may be of type Real or Integer. A vector is filled at the same time if parameters are given after the TYPE:

```
VEC/CREATE V(10) R 1 2 3 4 5 66 77 88 99 111
```

```
VEC/CREATE W(20) R 1 2 3
```

In the last example only the first three elements are filled. Vector elements may be changed later with the command VECTOR/INPUT.

If many equal values have to be entered consecutively, one can specify just one value and precede it by a repetition factor and an asterisk. Example:

```
VEC/CREATE Z(20) R 5*1 2 4*3   --->   VEC/CREATE Z(20) R 1 1 1 1 1 2 3 3 3
3
```

Enter HELP VECTOR for more information on vector addressing.

VECTOR/LIST

List all vectors (name, dimensions, type).

VECTOR/DELETE `vlist`

`VLIST` C "Vector list" `D='␣' Loop`

Delete from memory all vectors in the list VLIST. The vectors are separated in the list by a comma and embedded blanks are not allowed. An asterisk at the end of VLIST acts as wild-card:

```
VEC/DEL AB*          --->  deletes all vectors starting by AB
VEC/DEL *            --->  deletes all vectors
```

VECTOR/COPY `vnam1 vnam2`

`VNAM1` C "Source vector name"
`VNAM2` C "Destination vector name"

Copy a vector into another one. Mixed vector type copy is supported (e.g. Integer —⟩ Real and viceversa). If VNAM2 does not exist it is created with the required dimensions, not necessarily the same as the source vector if a sub-range was specified. For example, if A is a 3 x 100 vector and B does not exist, COPY A(2,11:60) B will create B as a 50 elements mono-dimensional vector; a special (and non-standard with FORTRAN 77) notation is used such that, still using the above vectors, COPY A(2,1:100) B and COPY A(2) B have the same effect.

Note that VECTOR/COPY does not allow a range for the destination vector not specifying consecutive elements (i.e. along the first dimension):

```
VEC/COPY V(5)      W(3,4)     | O.K.
VEC/COPY V1(2:3,5) V2(4:5,9)  | O.K.
VEC/COPY V1(5,2:3) V2(4:5,9)  | O.K.
VEC/COPY V1(3,3:4) V2(4,4:5)  | NOT allowed
VEC/COPY V1(2:3,5) V2(2,4:5)  | NOT allowed
```

Enter HELP VECTOR for more information on vector addressing.

VECTOR/INPUT vname [values]

VNAME	C	"Vector name"
VALUES	C	"Value list" D='␣' Separate Vararg

Enter values into a vector from the terminal. Example:

```
VEC/INPUT V(6:10) 1.1 2.22 3.333 4.4444 5.55555
```

If many equal values have to be entered consecutively, one can specify just one value and precede it by a repetition factor and an asterisk. Example:

```
VEC/INPUT V 5*1 2 4*3   --->   VEC/INPUT V 1 1 1 1 1 2 3 3 3 3
```

Enter HELP VECTOR for more information on vector addressing.

VECTOR/PRINT vname [dense]

VNAME	C	"Vector name"
DENSE	I	"Output density" D=1 R=0,1,2

Write to the terminal the content of a vector. Enter HELP VECTOR for more information on vector addressing.

If DENSE.EQ.0 the output is one vector element per line. If DENSE.EQ.1 the output for a sequence of identical vector elements is compressed to two lines stating the start and end indices. If DENSE.EQ.2 the output for a sequence of identical vector elements is compressed to a single line.

VECTOR/READ vlist fname [format opt match]

VLIST	C	"Vector list"
FNAME	C	"File name" D='␣'
FORMAT	C	"Format" D='␣'
OPT	C	"Options" D='OC'
MATCH	C	"Matching pattern" D='␣'

Possible OPT values are:

```
OC
O
'␣'
C
```

Enter values into vector(s) from a file. A format can be specified, e.g. FORMAT='F10.5,2X,F10.5', or the free format is used if FORMAT is not supplied.

If vector(s) are not existing they will be created of the size as read from the file.

Vectors in the list VLIST are separated by a comma and embedded blanks are not allowed. If subscripts are present in vector names, the smallest one is taken.

OPT is used to select between the following options:

```
'OC'   file is Opened, read and then Closed (default case)
'O'    file is Opened and then read (left open for further reading)
' '    file is read (already open, left so for further reading)
'C'    file is read and then Closed (already open)
```

If the character 'Z' is present in OPT, the vector elements equal to zero after reading are set to the latest non-zero element value (for example reading 1 2 3 0 0 4 0 5 will give 1 2 3 3 3 4 4 5).

MATCH is used to specify a pattern string, restricting the vector filling only to the records in the file which verify the pattern. Example of patterns:

```
 /string/      match a string (starting in column 1)
-/string/      do not match a string (starting in column 1)
 /string/(n)   match a string, starting in column n
 /string/(*)   match a string, starting at any column
```

Enter HELP VECTOR for more information on vector addressing.

VECTOR/WRITE `vlist [ fname format chopt ]`

`VLIST`	C	"Vector list"
`FNAME`	C	"File name" `D='␣'`
`FORMAT`	C	"Format" `D='5(1X,G13.7)'`
`CHOPT`	C	"Options" `D='OC'`

Possible `CHOPT` values are:

```
OC
O
'␣'
C
```

Write to a file the content of vector(s). If FNAME=' ' the content is written to the terminal. A format can be specified, e.g. FORMAT='F10.5,2X,F10.5', or the default one is used if FORMAT is not supplied.

Vectors in the list VLIST are separated by a comma and embedded blanks are not allowed. If subscripts are present in vector names, the smallest one is taken.

CHOPT is used to select between the following options:

```
'OC'   file is Opened, written and then Closed (default case)
'O'    file is Opened and then written (left open for further writing)
' '    file is written (already open, left so for further writing)
'C'    file is written and then Closed (already open)
```

Enter HELP VECTOR for more information on vector addressing.

VECTOR/DRAW `vname [ id chopt ]`

VNAME	C	"Vector name"
ID	C	"Histogram Identifier" D='12345'
CHOPT	C	"Options" D='␣'

Possible CHOPT values are:

'␣'	Draw an histogram.
C	Draw a smooth curve.
S	Superimpose plot on top of existing picture.
+	Add contents of ID to last plotted histogram.
B	Select Bar chart format.
L	Connect channels contents by a line.
P	Draw the current polymarker at each channel.
*	Draw a * at each channel.

Draw vector VNAME interpreting it as a histogram. Optionally save the contents in histogram ID.

VECTOR/HFILL vname id

VNAME	C	"Vector name"
ID	C	"Histogram Identifier"

Fill the existing histogram ID with vector VNAME. Note that the command VECTOR/PLOT can automatically book, fill and plot the contents of a vector.

VECTOR/PLOT vname [id chopt]

VNAME	C	"Vector name"
ID	C	"Histogram Identifier" D='12345'
CHOPT	C	"Options" D='␣'

Possible CHOPT values are:

'␣'	Draw an histogram.
C	Draw a smooth curve.
S	Superimpose plot on top of existing picture.
+	Add contents of ID to last plotted histogram.
B	Select Bar chart format.
L	Connect channels contents by a line.
P	Draw the current polymarker at each channel.
*	Draw a * at each channel.

Each element of VNAME is used to fill an histogram which is automatically booked with 100 channels and then plotted. If VNAME has the form VNAME1%VNAME2 then a scatter-plot of vector VNAME1 versus VNAME2 is plotted. If ID is given different of 12345, then a 2-Dim histogram is created with

40 bins by 40 bins and filled. One can use the command VECTOR/HFILL to fill an already existing histogram.

VECTOR/FIT `x y ey func [ chopt np par step pmin pmax errpar ]`

X	C	"Vector of X coordinates"
Y	C	"Vector of Y coordinates"
EY	C	"Vector of errors on Y" D='?'
FUNC	C	"Function name"
CHOPT	C	"Character options" D='␣'
NP	I	"Number of parameters" D=0 R=0:20
PAR	C	"Vector of parameters"
STEP	C	"Vector of steps size"
PMIN	C	"Vector of lower bounds"
PMAX	C	"Vector of upper bounds"
ERRPAR	C	"Vector of errors on parameters"

Possible CHOPT values are:

'␣'	Do the fit, plot the result and print the parameters.
O	Do not plot the result of the fit. By default the fitted function is drawn unless the option 'N' below is specified.
N	Do not store the result of the fit bin by bin with the histogram. By default the function is calculated at the middle of each bin and the fit results stored with the histogram data structure.
Q	Quiet mode. No print
V	Verbose mode. Results after each iteration are printed By default only final results are printed.
B	Some or all parameters are bounded. The vectors STEP,PMIN,PMAX must be specified. Default is: All parameters vary freely.
L	Use Log Likelihood. Default is chisquare method.
D	The user is assumed to compute derivatives analytically using the routine HDERIV. By default, derivatives are computed numerically.
W	Sets weights equal to 1. Default weights taken from the square root of the contents or from HPAKE/HBARX (PUT/ERRORS).
M	The interactive Minuit is invoked.
E	Performs a better Error evaluation (MIGRAD + HESSE + MINOS).
Z	FUNC is the user fitting model

Fit a user defined function to the points defined by the two vectors X and Y and the vector of associated errors EY. See command Histo/Fit for explanation of parameters. Note that if option 'W' is specified or EY='?' (default), the array EY is ignored. Option 'L' is not available.

```
When option 'Z' is given, FUNC is the user fitting model.
```

```
FUNC is a subroutine with the calling sequence:
    Subroutine FUNC(N,X,Y,EY,NPAR,IFLAG,NPFITS)
where
    - X(N),Y(N),EY(N) are the input vectors,
    - NPAR the number of parameters
    - NPFITS is an output parameter = Number of points used in the fit
The user must declare the common/HCFITD/FITPAD(24),FITFUN in FUNC
```

12.1 OPERATIONS

Simple arithmetic operations between vectors. In all the operations only the minimum vector length is considered, i.e. an operation between a vector A of dimension 10 and a vector B of dimension 5 will involve the first 5 elements in both vectors. If the destination vector does not exist, it is created with the same length as the source vector.

VECTOR/OPERATIONS/VBIAS `vnam1 bias vnam2`

`VNAM1`	C	"Source vector name"
`BIAS`	R	"Bias value"
`VNAM2`	C	"Destination vector name"

VNAM2(I) = BIAS + VNAM1(I)

VECTOR/OPERATIONS/VSCALE `vnam1 scale vnam2`

`VNAM1`	C	"Source vector name"
`SCALE`	R	"Scale factor"
`VNAM2`	C	"Destination vector name"

VNAM2(I) = SCALE * VNAM1(I)

VECTOR/OPERATIONS/VADD `vnam1 vnam2 vnam3`

`VNAM1`	C	"First source vector name"
`VNAM2`	C	"Second source vector name"
`VNAM3`	C	"Destination vector name"

VNAM3(I) = VNAM1(I) + VNAM2(I)

VECTOR/OPERATIONS/VMULTIPLY `vnam1 vnam2 vnam3`

`VNAM1`	C	"First source vector name"
`VNAM2`	C	"Second source vector name"
`VNAM3`	C	"Destination vector name"

VNAM3(I) = VNAM1(I) * VNAM2(I)

VECTOR/OPERATIONS/VSUBTRACT `vnam1 vnam2 vnam3`

`VNAM1`	C	"First source vector name"
`VNAM2`	C	"Second source vector name"
`VNAM3`	C	"Destination vector name"

VNAM3(I) = VNAM1(I) - VNAM2(I)

VECTOR/OPERATIONS/VDIVIDE `vnam1 vnam2 vnam3`

`VNAM1`	C	"First source vector name"
`VNAM2`	C	"Second source vector name"
`VNAM3`	C	"Destination vector name"

VNAM3(I) = VNAM1(I) / VNAM2(I) (or 0 if VNAM2(I)=0)

Chapter 13: HISTOGRAM

Manipulation of histograms, Ntuples. Interface to the HBOOK package.

HISTOGRAM/FILE `lun fname [ lrecl chopt ]`

`LUN`	I	"Logical unit number" `R=0:128`
`FNAME`	C	"File name"
`LRECL`	I	"Record length in words" `D=1024`
`CHOPT`	C	"Options" `D='␣'`

Possible `CHOPT` values are:

`'␣'`	Existing file is opened (read mode only).
`N`	A new file is opened.
`U`	Existing file is opened to be modified.
`D`	Reset lock.

Open an HBOOK direct access file. If LUN is 0 the next free logical unit will be used. If LRECL is 0 the system will determine the correct record length of an existing file.

HISTOGRAM/LIST `[ chopt ]`

`CHOPT`	C	"Options" `D='␣'`

Possible `CHOPT` values are:

`'␣'`	List histograms and Ntuples in the current directory.
`I`	A verbose format is used (HINDEX), (only for //PAWC).
`S`	List with histograms sorted by increasing IDs.

List histograms and Ntuples in the current directory.

HISTOGRAM/DELETE `id`

`ID`	C	"Histogram Identifier" `Loop`

Delete histogram/Ntuple ID in Current Directory (memory). If ID=0 delete all histograms and Ntuples. To delete histograms in disk files use command HIO/HSCRATCH.

HISTOGRAM/PLOT `[ id chopt ]`

`ID`	C	"Histogram Identifier" `Loop Minus`
`CHOPT`	C	"Options" `D='␣' Minus`

Possible `CHOPT` values are:

'␣'	Draw the histogram.
C	Draw a smooth curve.
S	Superimpose plot on top of existing picture.
+	Add contents of ID to last plotted histogram.
-	Substract contents of ID to last plotted histogram.
+-	Draw the delta with the last plotted histogram.
B	Select Bar chart format.
L	Connect channels contents by a line.
P	Draw the current polymarker at each channel or cell.
*	Draw a * at each channel.
K	Must be given if option 'U' is given later.
U	Update channels modified since last call.
E	Draw error bars and current marker.
E0	Draw error bars without symbols clipping.
E1	Draw small lines at the end of the error bars.
E2	Draw error rectangles.
E3	Draw a filled area through the end points of the vertical error bars.
E4	Draw a smoothed filled area through the end points of the vertical error bars.
A	Axis labels and tick marks are not drawn.
BOX	Draw 2-Dim with proportional boxes.
COL	Draw 2-Dim with a color table.
Z	Used with COL or SURF, it draws the color map.
SURF	Draw as a surface plot (angles are set via the command angle).
SURF1	Draw as a surface with color levels
SURF2	Same as SURF1 but without cell lines.
SURF3	Same as SURF but with the contour plot (in color) on top.
SURF4	Draw as a surface with Gouraud shading.
LEGO	Draw as a lego plot (angles are set via the command angle).
LEGO1	Draw lego plot with light simulation.
LEGO2	Draw lego plot with color levels.
BB	Suppress the Back Box on 3D plots.
FB	Suppress the Front Box on 3D plots.
CONT	Draw 2-Dim as a contour plot (15 levels).
TEXT	Draw 2-Dim as a table.
CHAR	Draw 2-Dim with characters (a la HBOOK).
HIST	Draw only histogram (no errors or associated function).
FUNC	Draw only the associated function (not the histogram).
CYL	Cylindrical coordinates for 3D plots.
POL	Polar coordinates for 3D plots.
SPH	Spherical coordinates for 3D plots.

PSD Pseudo-rapidity/phi coordinates for 3D plots.

Plot a single histogram or a 2-Dim projection. If ID=0 or ID=* all the histograms in the current directory are plotted. Each plotted histogram will start either a new picture or a new zone in the current picture.

```
Histogram subranges can be specified in 2 different ways:
 1- h/pl id(ic1:ic2) with ic1 and ic2 integers means plot
                     from channel ic1 to channel ic2
 2- h/pl id(x1:x2)   with x1 and x2 reals (with a .) means plot
                     from channel corresponding to x1
 Note that the mixed mode h/pl id(x1:ic2) is also accepted
 This subrange works also for 2-DIM cases.
 Ex: Histo/plot 10(25:1.) or Histo/plot 20(4:18,0.:0.5).
```

A specific histogram cycle can be accessed:

```
 PAW > h/pl id;nc | cycle number nc is used (default is highest cycle)
```

1 Dim histograms could be plotted with option LEGO or SURF. In this case the angles are THETA=1 and PHI=-1. When option 'E' is used, the marker type can be changed with SET MTYP, the marker size with SET KSIZ, the marker color with SET PMCI.

```
To plot projection X of ID type
  PAW > HI/PLOT ID.PROX
To plot band 1 in Y of ID type
  PAW > HI/PLOT ID.BANY.1
To plot slice 3 in Y of ID type
  PAW > HI/PLOT ID.SLIY.3
```

In addition to the Cartesian coordinate systems, Polar, cylindrical, spherical, pseudo-rapidity/phi coordinates are available for LEGO and SURFACE plots, including stacked lego plots. For example:

```
PAW > Histo/plot 10+20+30  LEGO1,CYL | stacked cylindrical lego plot
PAW > Histo/plot 10+20+30  LEGO1,POL |         polar
PAW > Histo/plot 10+20+30  LEGO1,SPH |         spherical
PAW > Histo/plot 10+20+30  LEGO1,PSD |         pseudo-rapidity/phi
```

Note that the viewing angles may be changed via the command ANGLES. The axis, the front box, and the back box can be suppressed on 3D plots with the options 'A', 'FB' and 'BB'.

HISTOGRAM/ZOOM `[ id chopt icmin icmax ]`

ID	C	"Histogram Identifier" Loop Minus
CHOPT	C	"Options" D='␣'
ICMIN	I	"First channel" D=1
ICMAX	I	"Last channel" D=9999

Possible CHOPT values are:

’␣’	Plot the zoomed histogram.
C	Draw a smooth curve.
S	Superimpose plot on top of existing picture.
+	Add contents of ID to last plotted histogram.
B	Select Bar chart format.
L	Connect channels contents by a line.
P	Draw the current polymarker at each channel.
*	Draw a * at each channel.

Plot a single histogram between channels ICMIN and ICMAX. Each plotted histogram will start either a new picture or a new zone in the current picture. If no parameters are given to the command, then the system waits for two points using the graphics cursor. To quit ZOOM, click the right button of the mouse or CRTL/E.

HISTOGRAM/MANY˙PLOTS `idlist`

IDLIST	C	“List of histogram Identifiers” Vararg

Plot one or several histograms into the same plot. Plotted histograms are superimposed on the same zone of the picture.

HISTOGRAM/PROJECT `id`

ID	C	“Histogram Identifier” Loop

Fill all booked projections of a 2-Dim histogram. Filling is done using the 2-D contents of ID.

HISTOGRAM/COPY `id1 id2 [ title ]`

ID1	C	“First histogram Identifier”
ID2	C	“Second histogram Identifier” Loop
TITLE	C	“New title” D=’␣’

Copy a histogram (not Ntuple) onto another one. Bin definition, contents, errors, etc. are preserved. If TITLE is not given, ID2 has the same title as ID1.

HISTOGRAM/FIT `id func [ chopt np par step pmin pmax errpar ]`

ID	C	“Histogram Identifier”
FUNC	C	“Function name” D=’␣’
CHOPT	C	“Options” D=’␣’
NP	I	“Number of parameters” D=0 R=0:34
PAR	C	“Vector of parameters”
STEP	C	“Vector of steps size”
PMIN	C	“Vector of lower bounds”
PMAX	C	“Vector of upper bounds”
ERRPAR	C	“Vector of errors on parameters”

Possible CHOPT values are:

'␣'	Do the fit, plot the result and print the parameters.
0	Do not plot the result of the fit. By default the fitted function is drawn unless the option 'N' below is specified.
N	Do not store the result of the fit bin by bin with the histogram. By default the function is calculated at the middle of each bin and the fit results stored with the histogram data structure.
Q	Quiet mode. No print
V	Verbose mode. Results after each iteration are printed By default only final results are printed.
B	Some or all parameters are bounded. The vectors STEP,PMIN,PMAX must be specified. Default is: All parameters vary freely.
L	Use Log Likelihood. Default is chisquare method.
D	The user is assumed to compute derivatives analytically using the routine HDERIV. By default, derivatives are computed numerically.
W	Sets weights equal to 1. Default weights taken from the square root of the contents or from HPAKE/HBARX (PUT/ERRORS). If the L option is given (Log Likelihood), bins with errors=0 are excluded of the fit.
M	The interactive Minuit is invoked. (see Application HMINUIT below).
E	Performs a better Error evaluation (MIGRAD + HESSE + MINOS).
U	User function value is taken from /HCFITD/FITPAD(24),FITFUN.
K	Keep the settings of Application HMINUIT for a subsequent command.

Fit a user defined (and parameter dependent) function to a histogram ID (1-Dim or 2-Dim) in the specified range. FUNC may be:

```
A- The name of a file which contains the user defined
   function to be minimized. Function name and file name
   must be the same. For example file FUNC.FOR is:
     FUNCTION FUNC(X)   or FUNC(X,Y) for a 2-Dim histogram
     COMMON/PAWPAR/PAR(2)
     FUNC=PAR(1)*X +PAR(2)*EXP(-X)
     END
    Ex: His/fit 10 func.for ! 5 par

   When the option U is given, the file FUNC.FOR should look like:
     FUNCTION FUNC(X)   or FUNC(X,Y) for a 2-Dim histogram
     DOUBLE PRECISION FITPAD(24),FITFUN
     COMMON/HCFITD/FITPAD,FITFUN
     FITFUN=FITPAD(1)*X +FITPAD(2)*EXP(-X)
     FUNC=FITFUN
     END

B- One of the following keywords (1-Dim only):
   G : to fit Func=par(1)*exp(-0.5*((x-par(2))/par(3))**2)
   E : to fit Func=exp(par(1)+par(2)*x)
```

```
      Pn: to fit Func=par(1)+par(2)*x+par(3)*x**2.....+par(n+1)*x**n
       Ex: His/fit 10 g

   C- A combination of the keywords in B with the 2 operators + or *.
      Ex: His/Fit 10 p4+g ! 8 par
          His/Fit 10 p2*g+g ! 9 par
        Note that in this case, the order of parameters in PAR must
        correspond to the order of the basic functions.
        For example, in the first case above, par(1:5) apply to
        the polynomial of degree 4 and par(6:8) to the gaussian while
        in the second case par(1:3) apply to the polynomial of degree 2,
        par(4:6) to the first gaussian and par(7:9) to the second gaussian.
        Blanks are not allowed in the expression.
```

For cases A and C, before the execution of this command, the vector PAR must be filled (via Vector/Input) with the initial values. For case B, if NP is set to 0, then the initial values of PAR will be calculated automatically. After the fit, the vector PAR contains the new values of parameters. If the vector ERRPAR is given, it will contain the errors on the fitted parameters. A bin range may be specified with ID.

```
 Ex. Histo/Fit 10(25:56).
```

When the Histo/it command is used in a macro, it might be convenient to specify MINUIT directives in the macro itself via the Application HMINUIT as described in this example:

```
Macro fit
Application HMINUIT exit
name 1 par_name1
name 2 par_name2
migrad
improve
exit
Histo/fit id fitfun.f M
Return
```

13.1 2D˙PLOT

Plotting of 2-Dim histograms in various formats.

HISTOGRAM/2D˙PLOT/LEGO `[ id theta phi chopt ]`

`ID`	C	“Histogram Identifier” `Loop`
`THETA`	R	“Angle THETA in degrees” `D=30.`
`PHI`	R	“Angle PHI in degrees” `D=30.`
`CHOPT`	C	“Options” `D='␣'`

Possible CHOPT values are:

- '␣' Hidden line algorithm is used.
- 1 Hidden surface algorithm is used. The colour of the lego is given by SET HCOL CI where CI is a colour index. For the top and the sides of the lego the same hue is used but with a different light.
- 2 Hidden surface algorithm is used. The colour of each bar changes according to the value of Z. It is possible to change the set of colours used with SET HCOL c.L where L define a palette of colours given by the command ATT/PALETTE.

Draw a lego plot from 2-Dim or 1-Dim histograms. It is also possible to produce stacked lego plots. A stacked lego plot consists of a superimposition of several histograms, whose identifiers are given in the command LEGO separated by the character "+".

```
PAW > LEGO ID1+ID2+ID3    | Maximum number of ID's is 10. The colours of
                          | each IDn is given by the command ATT/PALETTE
```

Examples:

```
PAW > SET HCOL 2          | The colour the histogram is 2 (red)
PAW > LEGO 20             | Display a lego with lines
PAW > LEGO 20 ! ! 1       | Display a lego with different lights
PAW > LEGO 20 ! ! 2       | Display a lego with colours
PAW > PALETTE  1 3 2 3 4  | Create the palette number 1 with 3
                          | elements: 2,3
PAW > SET HCOL 0.1        | The subsequent stack lego plots will use list 1
PAW > LEGO 10+20+30       | Plot a stack of lego plots with lines
PAW > LEGO 10+20+30 ! ! 1 | Plot a stack of lego plots with light
```

Notes: - The commands OPTION BAR, SET BARW and SET BARO act on lego plots

```
        - The options 1 and 2 must be used only on selective erase
          devices.
```

HISTOGRAM/2D˙PLOT/SURFACE [id theta phi chopt]

ID	C	"Histogram Identifier" Loop
THETA	R	"Angle THETA in degrees" D=30.
PHI	R	"Angle PHI in degrees" D=30.
CHOPT	C	"Options" D='␣'

Possible CHOPT values are:

- '␣' Hidden line algorithm is used.
- 1 Hidden surface algorithm is used and each cell is filled with a colour corresponding to the Z value (or grey scale with PostScript). It is possible to change the set of colours used with SET HCOL ic.L where L define a palette of colours given by the command ATT/PALETTE.

2 Similar to option '1' except that the cell lines are not drawn. This is very useful to draw contour plots with colours if THETA=90 and PHI=0.

3 Surface is drawn with a contour plot in color on top. The contour plot is drawn with the colors defined with the command PALETTE.

4 Surface is drawn with Gouraud shading.

Draw a surface plot from 2-Dim or 1-Dim histograms. With this command it is possible to draw color contour plots:

```
PAW > ATT/PAL 1 3 2 3 4   | Define the palette 1 with 3 elements
PAW > SET HCOL 0.1        | Set the list 1 as colours for histograms
PAW > SET NDVZ 4          | Set the number of Z divisions to 4
PAW > SURF id 90 0 2      | Draw the contour
```

Note: - The options 1 to 4 must be used only on selective erase devices.

HISTOGRAM/2D˙PLOT/CONTOUR `[ id nlevel chopt param ]`

`ID`	C	"Histogram Identifier" `Loop`
`NLEVEL`	I	"Number of contour lines" `D=10`
`CHOPT`	C	"Options" `D='1'`
`PARAM`	C	"Vector of contour levels"

Possible `CHOPT` values are:

0 Use colour to distinguish contours.

1 Use line style to distinguish contours.

2 Line style and colour are the same for all contours.

3 The contour is drawn with filled colour levels. The levels are equidistant. The color indices are taken in the current palette (defined with the command PALETTE). If the number of levels (NLEVEL) is greater than the number of entries in the current palette, the palette is explore again from the beginning in order to reach NLEVEL.

S Superimpose plot on top of existing picture.

Draw a contour plot from a 2-Dim histogram. If PARAM is not given, contour levels are equidistant. If given, the vector PARAM may contain up to 50 values.

13.2 CREATE

Creation ("booking") of HBOOK objects in memory.

HISTOGRAM/CREATE/1DHISTO `id title ncx xmin xmax [ valmax ]`

ID	C	"Histogram Identifier" Loop
TITLE	C	"Histogram title" D='␣'
NCX	I	"Number of channels" D=100
XMIN	R	"Low edge" D=0.
XMAX	R	"Upper edge" D=100.
VALMAX	R	"Maximum bin content" D=0.

Create a one dimensional histogram. The contents are set to zero. If VALMAX=0, then a full word is allocated per channel, else VALMAX is used as the maximum bin content allowing several channels to be stored into the same machine word.

HISTOGRAM/CREATE/PROFILE id title ncx xmin xmax ymin ymax [chopt]

ID	C	"Histogram Identifier"
TITLE	C	"Histogram title" D='␣'
NCX	I	"Number of channels" D=100
XMIN	R	"Low edge in X" D=0.
XMAX	R	"Upper edge in X" D=100.
YMIN	R	"Low edge in Y" D=-1.E30
YMAX	R	"Upper edge in Y" D=1.E30
CHOPT	C	"Options" D='␣'

Possible CHOPT values are:

'␣'	Error on mean
S	Spread option

Create a profile histogram. Profile histograms accumulate statistical quantities of a variable y in bins of a variable x. The contents are set to zero.

HISTOGRAM/CREATE/BINS id title ncx xbins [valmax]

ID	C	"Histogram Identifier"
TITLE	C	"Histogram title" D='␣'
NCX	I	"Number of channels" D=100
XBINS	C	"Vector of NCX+1 low-edges"
VALMAX	R	"Maximum bin content" D=0.

Create a histogram with variable size bins. The low-edge of each bin is given in vector XBINS (NCX+1) values. The contents are set to zero. See 1DHISTO for VALMAX.

HISTOGRAM/CREATE/2DHISTO id title ncx xmin xmax ncy ymin ymax [valmax]

ID	C	"Histogram Identifier" Loop
TITLE	C	"Histogram title" D='␣'
NCX	I	"Number of channels in X" D=40
XMIN	R	"Low edge in X" D=0.
XMAX	R	"Upper edge in X" D=40.
NCY	I	"Number of channels in Y" D=40
YMIN	R	"Low edge in Y" D=0.
YMAX	R	"Upper edge in Y" D=40.
VALMAX	R	"Maximum bin content" D=0.

Create a two dimensional histogram. The contents are set to zero. See 1DHISTO for VALMAX.

HISTOGRAM/CREATE/PROX id

ID	C	"Histogram (2-Dim) Identifier" Loop

Create the projection onto the x axis. The projection is not filled until the Histo/Project command is executed.

```
To plot projection X of ID type:
  PAW > HI/PLOT ID.PROX
```

HISTOGRAM/CREATE/PROY id

ID	C	"Histogram (2-Dim) Identifier" Loop

Create the projection onto the y axis. The projection may be filled with Histo/Project.

```
To plot projection Y of ID type:
  PAW > HI/PLOT ID.PROY
```

HISTOGRAM/CREATE/SLIX id nslices

ID	C	"Histogram (2-Dim) Identifier" Loop
NSLICES	I	"Number of slices"

Create projections onto the x axis, in y-slices. The projection may be filled with Histo/Project.

```
To plot slice 3 in X of ID type:
  PAW > HI/PLOT ID.SLIX.3
```

HISTOGRAM/CREATE/SLIY id nslices

ID	C	"Histogram (2-Dim) Identifier" Loop
NSLICES	I	"Number of slices"

Create projections onto the y axis, in x-slices. The projection may be filled with Histo/Project.

```
To plot slice 2 in Y of ID type:
  PAW > HI/PLOT ID.SLIY.2
```

HISTOGRAM/CREATE/BANX `id ymin ymax`

`ID`	C	"Histogram (2-Dim) Identifier" `Loop`
`YMIN`	R	"Low edge in Y"
`YMAX`	R	"Upper edge in Y"

Create a projection onto the x axis, in a band of y. Several bands can be defined on the one histogram. The projection may be filled with Histo/Project.

```
To plot band 1 in X of ID type:
  PAW > HI/PLOT ID.BANX.1
```

HISTOGRAM/CREATE/BANY `id xmin xmax`

`ID`	C	"Histogram (2-Dim) Identifier" `Loop`
`XMIN`	R	"Low edge in X"
`XMAX`	R	"Upper edge in X"

Create a projection onto the y axis, in a band of x. Several bands can be defined on the one histogram. The projection may be filled with Histo/Project.

```
To plot band 1 in Y of ID type:
  PAW > HI/PLOT ID.BANY.1
```

HISTOGRAM/CREATE/TITLE˙GLOBAL `[ chtitl chopt ]`

`CHTITL`	C	"Global title" `D='␣'`
`CHOPT`	C	"Options" `D='␣'`

Possible `CHOPT` values are:

`'␣'`	The global title is plotted at the top of each picture.
`U`	If the option 'UTIT' is on, a user title is plotted at the bottom of each histogram.

Set the global title. The size and the Y position of the global title may be changed by the commands SET GSIZ and SET YGTI respectively. The size and the Y position of the user title may be changed by the commands SET TSIZ and SET YHTI respectively.

13.3 HIO

Input/Output operations of histograms.

HISTOGRAM/HIO/HRIN `id [ icycle iofset ]`

`ID`	C	"Histogram Identifier" `Loop`
`ICYCLE`	I	"Cycle number" `D=999`
`IOFSET`	I	"Offset" `D=0`

Read histogram/Ntuple ID from the current directory on direct access file to memory. An identical histogram is created but with an ID equal to that of the original histogram plus the offset IOFSET. Identifier may be '0' or '*' (for all histograms). If ICYCLE $\rangle$ 1000 and ID=0 read all histograms in all subdirectories as well. If IOFSET = 99999 then the contents of histogram ID on the disk file are added to the current histogram in memory if it exists. For example to add all histograms from FILE1 and FILE2 in memory, the sequence of commands can be:

```
PAW > Histo/File 1 FILE1
PAW > Hrin 0
PAW > Histo/File 2 FILE2
PAW > Hrin 0 ! 99999
```

HISTOGRAM/HIO/HROUT `id [ chopt ]`

`ID`	C	"Histogram Identifier" `Loop`
`CHOPT`	C	"Options" `D='␣'`

Possible `CHOPT` values are:

`'␣'`	Write histo/Ntuple ID from memory to current directory.
`T`	Writes all histograms in subdirectories as well.

Write histo/Ntuple ID from memory to current directory. Identifier may be '0' or '*' (for all histograms).

HISTOGRAM/HIO/HSCRATCH `id`

`ID`	C	"Histogram Identifier" `Loop`

Delete histogram ID in Current Directory on disk. If ID='0' or '*' delete all histograms. To delete histograms in memory use command HISTO/DELETE.

HISTOGRAM/HIO/HFETCH `id fname`

`ID`	C	"Histogram Identifier"
`FNAME`	C	"File name"

Fetch histogram ID from file FNAME. FNAME has been created by the old version of HBOOK3 (Unformatted).

HISTOGRAM/HIO/HREAD `id fname`

`ID`	C	"Histogram Identifier"
`FNAME`	C	"File name"

Read histogram ID from file FNAME. FNAME has been created by the old version of HBOOK3 (Formatted).

HISTOGRAM/HIO/PRINT id [chopt]

ID C "Histogram Identifier" Loop
CHOPT C "Options" D='␣'

Possible CHOPT values are:

'␣' Print histograms.
S Only statistics (Number of entries, mean, RMS, underflow, overflow) are printed.

Print histograms (line-printer format) on screen. The command OUTPUT˙LP may be used to change the output file.

HISTOGRAM/HIO/DUMP id

ID C "Histogram Identifier" Loop

Dump the histogram ZEBRA data structure on the terminal.

HISTOGRAM/HIO/OUTPUT˙LP [lun fname]

LUN I "Logical unit number" D=6
FNAME C "File name" D='␣'

Change the HBOOK "line printer" file name. If FNAME=' ' then OUTPUT is appended to an already opened file on unit LUN. If LUN is negative, the file is closed and subsequent output is directed to unit 6.

HISTOGRAM/HIO/GLOBAL˙SECT gname

GNAME C "Global section name" D='␣'

Map the global section GNAME (VAX only). The current directory is changed to //GNAME.

HISTOGRAM/HIO/GRESET id

ID C "Histogram Identifier"

Reset histogram ID in the global section.

13.4 OPERATIONS

Histogram operations and comparisons.

HISTOGRAM/OPERATIONS/ADD id1 id2 id3 [c1 c2 option]

ID1 C "First histogram Identifier"
ID2 C "Second histogram Identifier"
ID3 C "Result histogram Identifier"
C1 R "Scale factor for ID1" D=1.
C2 R "Scale factor for ID2" D=1.
OPTION C "Option" D='␣'

Possible OPTION values are:

'␣'
E

Add histograms: ID3 = C1*ID1 + C2*ID2. Applicable to 1-Dim and 2-Dim histograms. See command HRIN to add histograms with same IDS from different files. If option 'E' is set, error bars are calculated for ID3.

HISTOGRAM/OPERATIONS/SUBTRACT id1 id2 id3 [c1 c2 option]

ID1	C	"First histogram Identifier"
ID2	C	"Second histogram Identifier"
ID3	C	"Result histogram Identifier"
C1	R	"Scale factor for ID1" D=1.
C2	R	"Scale factor for ID2" D=1.
OPTION	C	"Option" D='␣'

Possible OPTION values are:

'␣'
E

Subtract histograms: ID3 = C1*ID1 - C2*ID2. Applicable to 1-Dim and 2-Dim histograms. If option 'E' is set, error bars are calculated for ID3.

HISTOGRAM/OPERATIONS/MULTIPLY id1 id2 id3 [c1 c2 option]

ID1	C	"First histogram Identifier"
ID2	C	"Second histogram Identifier"
ID3	C	"Result histogram Identifier"
C1	R	"Scale factor for ID1" D=1.
C2	R	"Scale factor for ID2" D=1.
OPTION	C	"Option" D='␣'

Possible OPTION values are:

'␣'
E

Multiply histogram contents: ID3 = C1*ID1 * C2*ID2. Applicable to 1-Dim and 2-Dim histograms. If option 'E' is set, error bars are calculated for ID3.

HISTOGRAM/OPERATIONS/DIVIDE id1 id2 id3 [c1 c2 option]

ID1	C	"First histogram Identifier"
ID2	C	"Second histogram Identifier"
ID3	C	"Result histogram Identifier"
C1	R	"Scale factor for ID1" D=1.
C2	R	"Scale factor for ID2" D=1.
OPTION	C	"Option" D='␣'

Possible OPTION values are:

'␣'
E

Divide histograms: ID3 = C1*ID1 / C2*ID2. Applicable to 1-Dim and 2-Dim histograms. If option 'E' is set, error bars are calculated for ID3.

HISTOGRAM/OPERATIONS/RESET id [title]

ID	C	"Histogram Identifier" Loop
TITLE	C	"New title" D='␣'

Reset contents and errors of an histogram. Bin definition is not modified.

HISTOGRAM/OPERATIONS/DIFF id1 id2 [chopt]

ID1	C	"First Histogram Identifier"
ID2	C	"Second Histogram Identifier"
CHOPT	C	"Options" D='D'

Possible CHOPT values are:

'␣'	The comparison is done only on the shape of the two histograms.
N	Include also comparison of the relative normalization of the two histograms, in addition to comparing the shapes. PROB is then a combined confidence level taking account of absolute contents.
D	Debug printout, produces a blank line and two lines of information at each call, including the ID numbers, the number of events in each histogram, the PROB value, and the maximum Kolmogorov distance between the two histograms. For 2-Dim histograms, there are two Kolmogorov distances (see below). If 'N' is specified, there is a third line of output giving the PROB for shape alone, and for normalization.
O	Overflow, requests that overflow bins be taken into account.
U	Underflow, requests that underflow bins be taken into account.
L	Left: include x-underflows
R	Right: include x-overflows
T	Top: include y-overflows
B	Bottom: include y-underflows
F1	Histogram 1 has no error (is a function)
F2	Histogram 2 has no error (is a function)

Test of compatibility for two 1-Dim histograms ID1 and ID2. A probability PROB is calculated as a number between zero and one, where PROB near one indicates very similar histograms, and PROB near zero means that it is very unlikely that the two arose from the same parent distribution. For two histograms sampled randomly from the same distribution, PROB will be (approximately) uniformly distributed between 0 and 1. See discussion in HBOOK manual under "HDIFF- Statistical Considerations". By default (if no options are selected with CHOPT) the comparison is done only on the shape of the two histograms, without consideration of the difference in numbers of events, and ignoring all underflow and overflow bins.

HISTOGRAM/OPERATIONS/SORT `id [ chopt ]`

`ID`	C	"Histogram Identifier" `Loop`
`CHOPT`	C	"Options" `D='XA'`

Possible `CHOPT` values are:

- `X` X-axis is being treated.
- `Y` Y-axis is being treated.
- `Z` Z-axis is being treated.
- `A` Alphabetically.
- `E` Reverse alphabetical order.
- `D` By increasing channel contents.
- `V` By decreasing channel contents.

Sort the alphanumeric labels of the histogram ID according to the value of CHOPT.

HISTOGRAM/OPERATIONS/SMOOTH `id [ option sensit smooth ]`

`ID`	C	"Histogram or Ntuple Identifier" `Minus`
`OPTION`	C	"Options" `D='2M'`
`SENSIT`	R	"Sensitivity parameter" `D=1. R=0.3:3.`
`SMOOTH`	R	"Smoothness parameter" `D=1. R=0.3:3.`

Possible `OPTION` values are:

- `0` Replace original histogram by smoothed.
- `1` Replace original histogram by smoothed.
- `2` Store values of smoothed function and its parameters without replacing the original histogram (but see note below) - the smoothed function can be displayed at editing time - see HISTOGRAM/PLOT.
- `M` Invoke multiquadric smoothing (see HBOOK routine HQUAD).
- `Q` Invoke the 353QH algorithm (see HBOOK routine HSMOOF).
- `S` Invoke spline smoothing.
- `V` Verbose (default for all except 1-D histogram).
- `N` Do not plot the result of the fit.
- `F` Write Fortran77 function to HQUADF.DAT (multiquadric only)

Smooth a histogram or "simple" ntuple. ("simple" = 1, 2, or 3 variables.)

For multiquadric smoothing, SENSIT controls the sensitivity to statistical fluctuations. SMOOTH controls the (radius of) curvature of the multiquadric basis functions.

Notes:

1) The multiquadric basis functions are SQRT(R**2+D**2), where R is the distance from the "centre", and D is a scale parameter and also the curvature at the "centre". "Centres" are located at points where the 2nd differential or Laplacian of event density is statistically significant.

2) The data must be statistically independent, i.e. events (weighted or unweighted) drawn randomly from a parent probability distribution or differential cross-section.

For spline smoothing, SENSIT and SMOOTH control the no. of knots (= 10 * SENSIT) and degree of splines (= SMOOTH + 2) (thus if SENSIT and SMOOTH are at their default values a 10-knot cubic spline is used).

Notes:

1) The spline option ALWAYS replaces the contents of a 2-D histogram. (Also chi-squared is unavailable in this case.)

2) Use the SPLINE command for more flexibility.

HISTOGRAM/OPERATIONS/SPLINE `id [ isel knotx kx ]`

`ID`	C	"Histogram Identifier"
`ISEL`	I	"Option flag" `D=2`
`KNOTX`	I	"Number of knots" `D=10`
`KX`	I	"Degree of the spline" `D=3`

Smooth 1-Dim or 2-Dim histogram ID using B-splines. If ID is a 1-Dim histogram then:

```
ISEL = 0,1 replace original histogram by smoothed.
     = 2   superimpose as a function when editing.
```

If ID is a 2-Dim histogram then original contents are replaced.

HISTOGRAM/OPERATIONS/PARAM `id [ isel r2min maxpow ]`

`ID`	C	"Histogram Identifier"
`ISEL`	I	"Control word" `D=11`
`R2MIN`	R	"Min correlation coefficient" `D=1.`
`MAXPOW`	I	"Max degree of polynomials" `D=5 R=1:20`

Perform a regression on contents of the 1-Dim histogram ID. Find the best parameterization in terms of elementary functions (regressors). See HBOOK guide HPARAM. Control word ISEL=1000*T +100*W +10*S +P

```
S = 1 resulting parametric fit superimposed on histogram
    0 no superposition
P = 0 minimal output: the residual sum of squares is printed
    1 normal output: in addition, the problem characteristics and
      options are printed; also the standard deviations and
```

```
        confidence intervals of the coefficients.
      2 extensive output: the results of each iteration are printed
        with the normal output.
W = 0 weights on histogram contents are already defined via HBARX
      or HPAKE. If not they are taken to be equal to the
      square-root of the contents.
      1 weights are equal to 1.
T = 0 monomials will be selected as the elementary functions
      1 Chebyshev polynomials with a definition region: [-1,1]
      2 Legendre polynomials with a definition region: [-1,1]
      3 shifted Chebyshev polynomials with a definition region: [0,1]
      4 Laguerre polynomials with a definition region: [0,+infinite]
      5 Hermite polynomials with a definition region: [-inf,+inf]
```

The FORTRAN code of the parameterization is written onto the file FPARAM.DAT.

HISTOGRAM/OPERATIONS/HSETPR `param value`

PARAM	C	"Parameter name" D='FEPS'
VALUE	R	"Parameter value" D=0.001

Set various parameters for command PARAM.

13.5 GET˙VECT

Fill a vector from values stored in HBOOK objects.

HISTOGRAM/GET˙VECT/CONTENTS `id vname`

ID	C	"Histogram Identifier"
VNAME	C	"Vector name"

Get contents of histogram ID into vector VNAME.

HISTOGRAM/GET˙VECT/ERRORS `id vname`

ID	C	"Histogram Identifier"
VNAME	C	"Vector name"

Get errors of histogram ID into vector VNAME.

HISTOGRAM/GET˙VECT/FUNCTION `id vname`

ID	C	"Histogram Identifier"
VNAME	C	"Vector name"

Get function associated to histogram ID into vector VNAME.

HISTOGRAM/GET˙VECT/ABSCISSA `id vname`

`ID`	C	"Histogram Identifier"
`VNAME`	C	"Vector name"

Get values of center of bins abscissa into vector VNAME.

HISTOGRAM/GET˙VECT/REBIN `id x y ex ey [ n ifirst ilast chopt ]`

`ID`	C	"Histogram Identifier"
`X`	C	"Name of vector X"
`Y`	C	"Name of vector Y"
`EX`	C	"Name of vector EX"
`EY`	C	"Name of vector EY"
`N`	I	"Number of elements to fill" `D=100`
`IFIRST`	I	"First bin" `D=1`
`ILAST`	I	"Last bin" `D=100`
`CHOPT`	C	"Option" `D='␣'`

Possible `CHOPT` values are:

`N` Do not normalize values in Y

The specified channels of the 1-Dim histogram ID are cumulated (rebinned) into new bins. The final contents of the new bin is the average of the original bins by default. If the option N is given, the final contents of the new bin is the sum of the original bins. Get contents and errors into vectors, grouping bins. Bin width and centers are also extracted. Allow to combine 2, 3 or more bins into one.

```
E.g.:  REBIN 110 X Y EX EY 25 11 85
        will group by 3 channels 11 to 85  and return
        new abscissa, contents and errors.
        Errors in X are equal to 1.5*BINWIDTH.
N.B.:
       REBIN ID X Y EX EY  is a convenient way to return in
       one call abscissa, contents and errors for 1-Dim histogram.
       In this case the errors in X are equal to 0.5*BINWIDTH.
```

13.6 PUT˙VECT

Replace histogram contents with values in a vector.

HISTOGRAM/PUT˙VECT/CONTENTS `id vname`

`ID`	C	"Histogram Identifier"
`VNAME`	C	"Vector name"

Replace contents of histogram with values of vector VNAME.

HISTOGRAM/PUT_VECT/ERRORS `id vname`

`ID`	C	"Histogram Identifier"
`VNAME`	C	"Vector name"

Replace errors of histogram with values of vector VNAME.

13.7 SET

Set histogram attributes.

HISTOGRAM/SET/MAXIMUM `id vmax`

`ID`	C	"Histogram Identifier"	`Loop`
`VMAX`	R	"Maximum value"	

Set the maximum value on the Y axis. To select again an automatic scale, just set VMAX less then the minimum.

HISTOGRAM/SET/MINIMUM `id vmin`

`ID`	C	"Histogram Identifier"	`Loop`
`VMIN`	R	"Minimum value"	

Set the minimum value on the Y axis. To select again an automatic scale, just set VMIN greater then the maximum.

HISTOGRAM/SET/NORMALIZE_FACTOR `id [ xnorm ]`

`ID`	C	"Histogram Identifier"	
`XNORM`	R	"Normalization factor"	`D=1`

Set the contents/errors normalization factor. Only valid for histograms (1-Dim). (does not change contents, only presentation).

HISTOGRAM/SET/SCALE_FACTOR_2D `id [ xscale ]`

`ID`	C	"Histogram Identifier"	
`XSCALE`	R	"Scale factor"	`D=0`

Set the scale factor for histograms (2-Dim).

HISTOGRAM/SET/IDOPT `id option`

`ID`	C	"Histogram Identifier"
`OPTION`	C	"Options"

Set options for histogram ID. (* means default).

Possible OPTION values are:

SETD*	Set all options to the default values
SHOW	Print all the options currently set
BLAC	1 Dim histogram printed with X characters
CONT*	1 Dim histogram is printed with the contour option
STAR	1 Dim histogram is printed with a * at the Y value
SCAT*	Print a 2 Dim histogram as a scatter-plot
TABL	Print a 2 Dim histogram as a table
PROE*	Plot errors as the error on mean of bin in Y for profile histograms
PROS	Plot errors as the Spread of each bin in Y for profile histograms
STAT	Mean value and RMS computed at filling time
NSTA*	Mean value and RMS computed from bin contents only
ERRO	Errors bars printed as SQRT(contents)
NERR*	Do not print print error bars
INTE	Print the values of integrated contents bin by bin
NINT*	Do not print integrated contents
LOGY	1 Dim histogram is printed in Log scale in Y
LINY*	1 Dim histogram is printed in linear scale in Y
PCHA*	Print channel numbers
NPCH	Do not print channel numbers
PCON*	Print bin contents
NPCO	Do not print bin contents
PLOW*	Print values of low edge of the bins
NPLO	Do not print the low edge
PERR	Print the values of the errors for each bin
NPER*	Do not print the values of the errors
PFUN	Print the values of the associated function bin by bin
NPFU*	Do not print the values of the associated function
PHIS*	Print the histogram profile
NPHI	Do not print the histogram profile
PSTA*	Print the values of statistics (entries,mean,RMS,etc.)
NPST	Do not print values of statistics
ROTA	Print histogram rotated by 90 degrees
NROT*	Print histogram vertically
1EVL	Force an integer value for the steps in the Y axis
AEVL*	Steps for the Y axis are automatically computed
2PAG	Histogram is printed over two pages
1PAG*	Histogram is printed in one single page
AUTO*	Automatic scaling

Chapter 14: FUNCTION

Operations with Functions. Creation and plotting.

FUNCTION/FUN1 id ufunc ncx xmin xmax [chopt]

ID	C	"Histogram Identifier"
UFUNC	C	"Name of the function"
NCX	I	"Number of channels" D=100 R=1:
XMIN	R	"Low edge" D=0.
XMAX	R	"Upper edge" D=100.
CHOPT	C	"Options" D='P'

Possible CHOPT values are:

P	The function is drawn.

Create a one dimensional histogram and fill the bins with the values of a (single-valued) function. The function UFUNC may be given in two ways:

-An expression of the variable x in case of a simple function.

```
Ex: FUN1  10 sin(x)/x  100 0 10
```

-UFUNC is the name of a COMIS function in a text file with the name UFUNC.FTN or UFUNC.FOR or UFUNC FORTRAN (Apollo, VAX, IBM).

FUNCTION/FUN2 id ufunc ncx xmin xmax ncy ymin ymax [chopt]

ID	C	"Histogram (2-Dim) Identifier"
UFUNC	C	"Name of the function"
NCX	I	"Number of channels in X" D=40 R=1:
XMIN	R	"Low edge in X" D=0.
XMAX	R	"Upper edge in X" D=40.
NCY	I	"Number of channels in Y" D=40 R=1:
YMIN	R	"Low edge in Y" D=0.
YMAX	R	"Upper edge in Y" D=40.
CHOPT	C	"Options" D='S'

Possible CHOPT values are:

'␣'	Create the histogram.
S	The function is drawn as a surface.
L	The function is drawn as a lego plot.
C	The function is drawn as a contour plot.

Create a two dimensional histogram and fill the bins with the values of a (two-valued) function. The function UFUNC may be given in two ways:
-An expression of the variables x and y in case of a simple function.

```
Ex: FUN2  10 abs(sin(x**2+y**2)) 40 -2 2 40 -2 2 C
```

-UFUNC is the name of a COMIS function in a text file with the name UFUNC.FTN or UFUNC.FOR or UFUNC FORTRAN (Apollo, VAX, IBM).

FUNCTION/DRAW ufunc [chopt]

UFUNC C "Name of function"
CHOPT C "Options" D='␣'

Draw the function UFUNC in the current ranges specified by the command: RANGE XLOW XUP YLOW YUP ZLOW ZUP and with THETHA and PHI angles specified by the command ANGLE THETA PHI. The number of points to evaluate the function between XLOW, XUP YLOW, YUP, and ZLOW, ZUP can be changed by the command POINTS NPX NPY NPZ.
The function UFUNC may be given in two ways: - As an expression of the variables X, Y, Z in the case of a

```
simple function.
Ex:
PAW >  FUN/DRAW X*Y*Z          | equivalent to :
PAW >  FUN/DRAW X*Y*Z=0
PAW >  FUN/DRAW X**2+Y**2+Z**2=1
PAW >  FUN/DRAW X**2+Y**2=1-Z**2
```

- As a COMIS function in a text file with the name UFUNC.FTN or

```
UFUNC.FOR or UFUNC FORTRAN (Apollo, VAX, IBM).
Ex:
The file FTEST.FOR contains:

FUNCTION FTEST(X,Y,Z)
IF(X.LE.0..AND.Y.LE.0.)THEN
  FTEST=(X+0.5)**2+(Y+0.5)**2+(Z+0.5)**2-0.2
ELSE
  FTEST=(X-0.5)**2+(Y-0.5)**2+(Z-0.5)**2-0.1
ENDIF
END

  PAW > RANGE -1 1 -1 1 -1 1  | Define the range as a cube between -1 1 in
the 3
                              directions
  PAW > POINTS 20 20 20       | FUN/DRAW will use 20 points in the 3
directions
  PAW > FUN/DRAW FTEST.FOR    | Draw 2 spheres centered on (-0.5,-0.5,-0.5)
                              and (0.5,0.5,0.5) with the radius SQRT(0.2)
                              and SQRT(0.1)
```

FUNCTION/PLOT ufunc xlow xup [chopt]

UFUNC	C	"Name of function"
XLOW	R	"Lower limit"
XUP	R	"Upper limit"
CHOPT	C	"Options" D='C'

Possible CHOPT values are:

C	Draw a smooth curve.
S	Superimpose plot on top of existing picture.
+	Add contents of ID to last plotted histogram.
L	Connect channel contents by a line.
P	Draw the current polymarker at each channel.
*	Draw a * at each channel.

Plot single-valued function UFUNC between XLOW and XUP. The function UFUNC may be given in two ways:

-An expression of the variable x in case of a simple function.

```
Ex: FUN/PLOT   sin(x)/x  0 10
```

-UFUNC is the name of a COMIS function in a text file with the name UFUNC.FTN or UFUNC.FOR or UFUNC FORTRAN (Apollo, VAX, IBM). For example, if the file FTEST.FOR contains:

```
FUNCTION FTEST(X)
FTEST=SIN(X)*EXP(-0.1*X)
END
```

Then, FUN/PLOT FTEST.FOR 0 10, will interpret the Fortran code in the file FTEST.FOR and draw the function for x between 0 and 10.

The number of points to evaluate the function between XLOW and XUP can be changed by the command /FUN/POINTS. Only 1-Dim functions are supported. For 2-Dim use FUN2.

FUNCTION/POINTS [npx npy npz]

NPX	I	"Number of points on X axis" D=20 R=2:1000
NPY	I	"Number of points on Y axis" D=20 R=2:1000
NPZ	I	"Number of points on Z axis" D=20 R=2:1000

Change the number of points to be used by FUN/DRAW and FUN/PLOT. Note that the default for NPX is 20 for 3-Dim plots (FUN/DRAW) but it is 100 for 1-Dim plots (FUN/PLOT).

FUNCTION/RANGE [xlow xup ylow yup zlow zup]

```
XLOW   R   "X Lower limit"  D=-1.
XUP    R   "X Upper limit"  D=1.
YLOW   R   "Y Lower limit"  D=-1.
YUP    R   "Y Upper limit"  D=1
ZLOW   R   "Z Lower limit"  D=-1.
ZUP    R   "Z Upper limit"  D=1.
```

Change the range used by FUN/DRAW.

FUNCTION/ANGLE `[ theta phi ]`

```
THETA  R   "Angle THETA in degrees"  D=30.
PHI    R   "Angle PHI in degrees"  D=30.
```

Change the angle used by FUN/DRAW and HISTO/PLOT.

Chapter 15: NTUPLE

Ntuple creation and related operations.

NTUPLE/CREATE idn title nvar chrzpa nprime varlist

IDN	C	"Ntuple Identifier"
TITLE	C	"Ntuple title" D='␣'
NVAR	I	"Number of variables" D=1 R=1:512
CHRZPA	C	"RZ path" D='␣'
NPRIME	I	"Primary allocation" D=1000
VARLIST	C	"Names of the NVAR variables" Vararg

Create a Row˙Wise˙Ntuple. (See below how to create a Column˙Wise˙Ntuple). The Ntuple may be created either purely in memory or possibly using an automatic overflow to an RZ file. Memory allocation works in the following way. If CHRZPA = ' ', then a bank of NPRIME words is created. When the space in this bank is exhausted at filling time, a new linear structure of length NPRIME is created and this process will be repeated should the structure become exhausted. If CHRZPA contains the top directory name of an already existing RZ file (as declared with HISTO/FILE), then a bank of length NPRIME is also created, but at filling time, this bank is moved to the RZ file when full, and then it is overwritten by any new entries. The Ntuple can be filled by calling HFN from an interactively defined subroutine called by the command NTUPLE/LOOP or by NTUPLE/READ. The number of variables per data point is given in the parameter NVAR.

To create a Column˙Wise˙Ntuple, create a file, eg. newnt.f with:

```
      Subroutine Newnt
      character*8 mother,in1,in2
      common/ntupc/mother,in1,in2
      common/ntupr/xover
      lin=41
      lout=42
      id=1
      open(unit=lin,file='datafile.dat',status='old')
      call hropen(lout,'NTUPLE','New_Ntuple.hbook','N',1024,istat)
      call hbnt(id,'New Ntuple',' ')
      call hbname(id,'ntupr',xover,'XOVER')
      call hbnamc(id,'ntupc',mother,'MOTHER:c*8,in1:c*8,in2:c*8')
   10 read(lin,1000,end=20,err=20)xover,mother,in1,in2
```

1000 format(e15.7,2x,a,7x,a,7x,a)

```
      call hfnt(1)
      go to 10
   20 call hrout(id,icycle,' ')
      call hrend('NTUPLE')
      close (lin)
      close (lout)
      end
```

and then call this routine via the CALL command:

```
PAW > call newnt.f
```

NTUPLE/LIST

List all Ntuples in the Current Directory. Note that the command HISTO/LIST lists all histograms and Ntuples in the Current Directory.

NTUPLE/PRINT `idn`

`IDN` C "Ntuple Identifier"

Print a summary about Ntuple IDN. Number of entries, variables names and limits are listed.

NTUPLE/HMERGE `outfile infiles`

`OUTFILE` C "Output file name" `D='␣'`
`INFILES` C "Input file names" `D='␣' Vararg`

Merge HBOOK files containing histograms and/or ntuples. Ntuples are merged and histograms with the same ID are added. The INFILES are merged into a new file OUTFILE. If OUTFILE already exists, it is overwritten.

NTUPLE/DUPLICATE `id1 id2 [ newbuf title option ]`

`ID1` C "Source Ntuple"
`ID2` I "New Ntuple"
`NEWBUF` I "Buffer size" `D=-1`
`TITLE` C "Title of ID2" `D='␣'`
`OPTION` C "Options" `D='A'`

Possible `OPTION` values are:

`'␣'`
`A` Set the Addresses of variables in common /PAWCR4,etc/.
`M` Create ID2 as a Memory resident Ntuple.
`'␣'` Copy ID1 structure in ID2. Reset addresses of variables.

The structure of Ntuple ID1 is duplicated in a new ntuple ID2. This command is useful when one wants to create an ntuple with the same variables but only a subset of the events. NEWBUF is the buffer size for ID2. If NEWBUF⟨0 the buffer size of ID1 is taken. If NEWBUF=0 the current buffer size is taken (10000 words for RWNs). NEWBUF⟩0 will be the new buffer size. If TITLE=' ' ID2 has the same title as ID1. In case of a disk-resident ntuple (default), ID2 is created into the current working directory which must be open in WRITE mode.

```
Example of use:
  Macro Dup
```

```
Histo/file 1 source.hbook
Histo/file 2 New.hbook ! N
Ntuple/Dupl //lun1/10 20
Nt/loop //lun1/10  duplic.f
Hrout 20
Return

File duplic.f:
      real function duplic(dum)
      include ?
*-* The call to HGNT is only necessary for CWNs
*-* For RWNs, replace HFNT by HFN(20,xvar) where xvar is the name
*-* of the first variable in /PAWCR4/
      if(some_condition)then
         call hgnt(10,idnevt,ierr)
         call hfnt(20)
      endif
      duplic=1.
      end
```

NTUPLE/RECOVER `idn`

`IDN`	I	"Ntuple Identifier"

To recover Ntuple ID. If the job producing the Ntuple crashed or the header was not stored correctly in the file with HROUT, RECOVER will scan the Ntuple to rebuild the header table and recompute the number of entries. The file on which the Ntuple resides must be open in Update mode.

NTUPLE/SCAN `idn [ uwfunc nevent ifirst option varlis ]`

`IDN`	C	"Ntuple Identifier"
`UWFUNC`	C	"User cut function" `D='0'`
`NEVENT`	I	"Number of events" `D=99999999`
`IFIRST`	I	"First event" `D=1`
`OPTION`	C	"Options" `D='␣'`
`VARLIS`	C	"Names of the NVARS variables to scan" `D='␣' Vararg`

Possible `OPTION` values are:

`'␣'`	
`S`	Graphical scan (spider plot).
`'␣'`	Alphanumeric output of the Ntuple.
`A`	Used with "S" it displays the average spider.

Scan the entries of an Ntuple subject to user cuts. Scan the variables for NEVENT events starting at IFIRST, requiring that the events satisfy cut UWFUNC. In the case of Alphanumeric output Up to 8 vari-

ables may be scanned, the default is to scan the first 8 variables.

When the option S (Spider plot) is specified, each event is presented in a graphical form (R versus PHI plot) to give a multi dimensional view of the event. Each variable is represented on a separate axis with a scale ranging from the minimum to the maximum value of the variable. A line joins all the current points on every axis where each point corresponds to the current value of the variable. When the HCOL parameter is specified (eg SET HCOL 1002) a fill area is drawn.

VARLIS may contain a list of the original variables, expressions of the original variables or/and ranges of variables. A range can be given in the following form:

```
:          means all variables (default).
var1:var2  means from variable var1 to variable var2 included.
var1:      means from variable var1 to the last.
:var2      means from variable 1 to variable var2
```

For example, if IDN=30 has the 3 variables X,Y,Z,U,V,W one can do:

```
PAW > scan 30
PAW > scan 30 option=s
      each event is drawn as a spider plot.
PAW > scan 30 option=sa
      each event is drawn as a spider plot and the average spider
      plot is also drawn.
PAW > scan 30 option=s  X:Z W
PAW > scan 30 z>10
PAW > scan 30 z>10 ! ! ! z abs(x) y+z x func.for
      where func.for is a COMIS function returning an expression
      of the original variables. This function func.for may be
      generated automatically by the PAW command:
PAW > uwfunc 30 func.for
```

NTUPLE/LOOP idn uwfunc [nevent ifirst]

IDN	C	"Identifier of Ntuple"
UWFUNC	C	"Selection function or cut identifier" D='␣'
NEVENT	I	"Number of events" D=99999999
IFIRST	I	"First event" D=1

Invoke the selection function UWFUNC for each event starting at event IFIRST. In UWFUNC, the user can fill one or several histograms previously booked. The loop will be terminated if UWFUNC returns a negative value. For more information about UWFUNC, see command NTUPLE/PLOT.

NTUPLE/MERGE idn1 idn2 [uwfunc nevent ifirst]

IDN1	C	"Identifier of first Ntuple"
IDN2	C	"Identifier of second Ntuple"
UWFUNC	C	"Selection function or cut identifier" D='␣'
NEVENT	I	"Number of events" D=99999999
IFIRST	I	"First event" D=1

Merge two Disk-Resident Row-Wise-Ntuples. Invoke the selection function UWFUNC for each of the NEVENT events starting at event IFIRST of Ntuple IDN1. Suppose you have 4 files containing Ntuple ID=10 and you want to merge the 4 files into the file 4, the sequence is:

```
PAW >Histo/file 1 file1
PAW >Histo/file 2 file2
PAW >Histo/file 3 file3
PAW >Histo/file 4 file4 1024 U
PAW >Ntuple/Merge //lun1/10 //lun4/10
PAW >Ntuple/Merge //lun2/10 //lun4/10
PAW >Ntuple/Merge //lun3/10 //lun4/10
PAW >Ntuple/plot 10.x .........
```

Only the events with UWFUNC⟩0 are appended to IDN2. IDN2 may be empty. Note that the Ntuple variables may be redefined inside UWFUNC. For more information about UWFUNC, see command NTUPLE/PLOT. Note that this command cannot be used for memory resident ntuples or CWNs. Use instead the command HMERGE.

NTUPLE/PROJECT idh idn [uwfunc nevent ifirst]

IDH	C	"Identifier of histogram to fill"
IDN	C	"Identifier of Ntuple"
UWFUNC	C	"Selection function or cut identifier" D='␣'
NEVENT	I	"Number of events" D=99999999
IFIRST	I	"First event" D=1

Project an Ntuple onto a 1-Dim or 2-Dim histogram, possibly using a selection function or predefined cuts. IDN may be given as IDN or IDN.X , IDN.Y%X , IDN.1, IDN.2%1. Y%X means variable Y of Ntuple IDN versus variable X. For more information about UWFUNC, see command NTUPLE/PLOT. The histogram IDH is not reset before filling. This allows several PROJECTs from different Ntuples.

NTUPLE/READ idn fname [format chopt nevent]

IDN	C	"Ntuple Identifier"
FNAME	C	"File name"
FORMAT	C	"Format" D='*'
CHOPT	C	"Options" D='␣'
NEVENT	I	"Number of events" D=1000000

Read Ntuple values from the alphanumeric file FNAME with the format specifications in FORMAT. Before executing this command, the Ntuple IDN must have been created with the command Ntuple/Create.

NTUPLE/PLOT idn [uwfunc nevent ifirst nupd option idh]

IDN	C	"Ntuple Identifier"
UWFUNC	C	"Selection function" D='0'
NEVENT	I	"Number of events" D=99999999
IFIRST	I	"First event" D=1
NUPD	I	"Frequency to update histogram" D=100000000
OPTION	C	"Options" D='␣'
IDH	I	"Identifier of histogram to fill" D=1000000

Possible OPTION values are:

'␣'	
C	Draw a smooth curve.
S	Superimpose plot on top of existing picture.
+	Add contents of IDN to last plotted ntuple.
B	Bar chart format.
L	Connect channels contents by a line.
P	Draw the current polymarker at each channel or cell.
*	Draw a * at each channel.
U	Update channels modified since last call.
E	Compute (HBARX) and draw error bars with current marker.
A	Axis labels and tick marks are not drawn.
'␣'	Draw the ntuple as an histogram.
PROF	Fill a Profile histogram (mean option).
PROFS	Fill a Profile histogram (spread option).
PROFI	Fill a Profile histogram (integer spread option).

Project and plot an Ntuple as a (1-Dim or 2-Dim) histogram with automatic binning (ID=1000000), possibly using a selection algorithm. See parameter CHOPT in command HISTO/PLOT to have more details on the possible OPTION.

```
IDN may be given as IDN
                    IDN.X
                    IDN.Y%X
                    IDN.1
                    IDN.2%1
                    IDN.expression1
                    IDN.expression1%expression2
```

Y%X means a scatter-plot Y(I) versus X(I) where I is the event number. 2%1 means a scatter-plot variable 2 versus variable 1. In this example, X and Y are the names of the variables 1 and 2 respectively. Expression 1 is any numerical expression of the Ntuple variables. It may include a call to a COMIS function.

```
UWFUNC may have the following forms:
```

```
1- UWFUNC='0' or missing (only IDN given). No selection is applied.
2- UWFUNC is a CUT or combination of valid CUTS created by the
   command NTUPLE/CUTS. Ex:
          UWFUNC=$1            means use cut $1
          UWFUNC=$1.AND.$2
          UWFUNC=.NOT.($1.AND.$2)
          UWFUNC=($1.OR.$2).AND.$3
3- UWFUNC is a FORTRAN expression
   Ex:    X>3.14.AND.(Y<Z+3.15)
4- UWFUNC is a variable name or an arithmetic expression
   Ex:   NT/PLOT 30.X Y  weight of each event is variable Y
         NT/PLOT 30.X X**2+Y**2
5- UWFUNC is the name of a selection function in a text file with
   the name UWFUNC.FTN, UWFUNC.FOR, UWFUNC FORTRAN (Apollo, VAX, IBM).
```

The command UWFUNC may be used to generate automatically this function. For example if IDN=30 is an Ntuple with 3 variables per event and 10000 events, then

```
NTUPLE/PLOT 30.X SELECT.FOR
```

will process the 10000 events of the Ntuple IDN=30. For each event, the function SELECT is called. It returns the weight of the event. Example:

```
FUNCTION SELECT(X)
DIMENSION X(3)
IF(X(1)**1+X(2)**2.LT.1.5)THEN
   SELECT=0.
ELSE
   SELECT=1.
ENDIF
END
```

The file SELECT.FOR (VAX), SELECT.FTN (Apollo) or SELECT FORTRAN (IBM) can be edited from PAW using the command EDIT. Note that if the suffix (.FTN, .FORTRAN or .FOR) is omitted, then COMIS will start from the precompiled version in memory and not from the file. Results of a selection can be saved in a MASK (See NTUPLE/MASK).

```
Ex: NT/PLOT 30.X Z<0.4>>MNAME(4)
    means mark bit 4 in mask MNAME for all events satisfying
    the condition Z<0.4
```

A MASK may also be given as input to a selection expression.

```
Ex:  NT/PLOT 30.X MNAME(4).and.Z<0.4
     means all events satisfying bit 4 of MNAME AND Z<0.4
```

It is possible to plot expressions of the original variables.

```
Ex 1:  NT/PLOT 30.SIN(X)%SQRT(Y**2+Z**2)  Z<0.4
    plots a scatter-plot of variable U versus V for all events
    satisfying the condition Z<0.4. U and V are defined as being
    U=SIN(X) and V=SQRT(X**2+Y**2)
Ex 2:  NT/PLOT 30.FUNC.FTN(X)%(SIN(Y)+3.)  Z<0.2.and.TEST.FTN>6
    plots a scatter-plot of variable U versus V for all events
    satisfying the condition (Z<0.2 and the result of the COMIS
    function TEST.FTN >6). U and V are defined as being
    U=Result of the COMIS function FUNC.FTN,  V=SIN(Y)+3.
```

The default identifier of the histogram being filled is IDH=1000000. At the next invocation of this command, it will be overwritten. If either NEVENT or IFIRST or NUPD are negative, then the identifier of the histogram being filled will be taken as IDH=-NEVENT or IDH=-IFIRST or IDH=-NUPD. IDH may have been created with H/CREATE. Before filling IDH, the contents of IDH are reset if IDH already exists. Use NTUPLE/PROJECT to cumulate several passes into IDH. Note that IDH not equal to 1000000 is a convenient way to force user binning. Every NUPD events, the current status of the histogram is displayed.

NTUPLE/CHAIN `[ cname entry ]`

CNAME	C	"Chain Name" D='␣'
ENTRY	C	"Chain Member(s) | -P Path" D='␣' Vararg

Using the chain command one can build logical Ntuples of unlimited size. The chain command creates an Ntuple chain CNAME and add member(s) ENTRY. If the chain already exists the member is simply added. More than one member may be specified at a time. A chain can contain three different type of members: files, logical units and other chains. The member type is deduced from the format of the member. Entries containing the characters . / : ; $ are considered to be files, entries like //LUN4 are assumed to be logical units and all other type of entries are chains. Chain names must be unique. After a chain has been defined it can be traversed, by all Ntuple commands (NT/PLOT, NT/PROJ, NT/LOOP), by changing the current working directory to the chain: CD //CNAME. A member may be deleted from a chain by preceding it by a - sign. A complete chain can be deleted by preceding the chain name by a -. All chains can be deleted by giving a - as chain name. Not specifying any parameters results in the listing of all defined chains. A chain tree will be printed by appending a ⟩ character to the chain name. The path of all chain members, from chain CNAME downwards, can be changed by specifying a chain path. This is done by giving a chain name followed by the -P option and a path specification. The chain path will be pre-pended to the member names. Chains down the tree can override a path specified higher up in the tree.

```
Examples of chain (Ntuple tree) definition:
CHAIN   Year93  Jan Feb March April May ...
CHAIN   Jan     Week1 Week2 Week3 Week4
CHAIN   Week1   file1.hbook file2.hbook ...
CHAIN   Week2   file3.hbook file4.hbook ...
CD //Jan
NT/PLOT 10.e     ; loop over all files in chains Week1, Week2, Week3, ...
CD //Year93      ; loop over all files in chains Jan, Feb, March, ...
CHAIN Year93 -P /user/delphi   ; all files from chain Year93 downward will
```

```
                                    be changed to /user/delphi/file1.hbook,
...
 CHAIN Year93>    ; print the chain tree Year93
 CHAIN -Feb       ; delete chain Feb
 CHAIN Jan -file3.hbook   ; delete file3.hbook from chain Jan
```

NTUPLE/DRAW `idn [ value option ]`

`IDN`	C	"Ntuple Identifier"
`VALUE`	C	"Isosurface value (for 3-D)" `D='0'`
`OPTION`	C	"Options" `D='␣'`

Draw a simple ntuple (1, 2 or 3 variables). For simple ntuples, with 1, 2 or 3 variables per event, this command will draw a histogram with HPLOT options. If the ntuple has an associated functional representation, as the result, e.g., of using SMOOTH, it will also draw the function. No selections are allowed.

For 3-variable ntuples which have been SMOOTHed, give a VALUE for the isosurface of event density. If VALUE=0, an isosurface value half way between the minimum and maximum fitted smoothing function values will be used.

NTUPLE/WAVE `idn [ lun ]`

`IDN`	C	"Ntuple Identifier"
`LUN`	I	"Logical unit no." `D=-1`

Produce a formatted file suitable for Wavefront's Data Visualiser. Only for simple 3-variable ntuples which have been SMOOTHed. A file with logical unit no. LUN must previously have been opened with the FORTRAN/FILE command.

NTUPLE/CUTS `cutid [ option fname wkid ]`

`CUTID`	C	"Cut identifier"
`OPTION`	C	"Options" `D='P' Minus`
`FNAME`	C	"File name" `D='␣'`
`WKID`	I	"Workstation identifier" `D=1`

Possible `OPTION` values are:

- `G` Define a new cut CUTID using graphics input on the latest 1-Dim or 2-Dim projection of the Ntuple. For a 1-Dim projection, give 2 points cutmin,cutmax. For a 2-Dim projection, give up to 20 points to delimit the selected area. The polygon will automatically be closed by PAW.
- `X` Same as G but with a tracking cross cursor.
- `P` Print definition of cut CUTID.
- `-` Reset cut CUTID.
- `R` Read definition of cut CUTID from file FNAME.
- `W` Write definition of cut CUTID on file FNAME (text file).
- `D` Draw cut contour.

Define the CUTID with the format $nn. nn is an integer between 1 and 99. This cut can then be used in subsequent commands NTUPLE/PLOT, PROJECT.

```
OPTION='expression'  allows to define the cut CUTID. For example
the command:
 PAW > CUTS $1 X<0.8.and.Y<SQRT(X)
defines the cut $1.
```

Note that CUTID=$0 means all cuts except for 'G' option. When option G is selected, graphical cuts are only operational for plots of the original Ntuple variables, not for expressions of these variables. WKID allows to define in which window the locator is performed (option 'G' or 'X' only).

NTUPLE/CSELECT [chopt csize]

CHOPT	C	"Options" D='N'
CSIZE	R	"Comment size" D=0.28

Possible CHOPT values are:

'␣'	Comment is left adjusted to the current zone
R	Comment is right adjusted to the current zone
C	Comment is centered to the current zone
B	Comment is drawn below the top zone line
N	All subsequent NTUPLE/PLOT commands will print the selection mechanism with the options specified in CHOPT.

To write selection mechanism as a comment on the picture. By default, the comment is drawn left justified above the top zone line. Example:

```
CSEL          All coming NT/PLOT commands will draw a comment
              of size CSIZE=0.28cm Left justified.
CSEL NRB 0.4  All coming NT/PLOT commands will draw a comment
              of size 0.4 cm Right justified Below the top line.
CSEL  CB      Draw previous selection mechanism Centered Below
              the top zone line.
```

The Global title font (SET GFON) with precision 1 is used to draw the text.

NTUPLE/MASK mname [chopt number]

MNAME	C	"Mask name"
CHOPT	C	"Options" D='␣'
NUMBER	I	"Bit number" D=0

Possible CHOPT values are:

'␣'	Existing mask on file MNAME.MASK is attached for READ only.

U	Existing mask on file MNAME.MASK is attached for UPDATE.
N	A new mask on file MNAME.MASK is created for NUMBER events.
P	The comments for all active bits is printed.
C	Mask is closed.
R	Reset bit number NUMBER.If NUMBER=99, resets all bits.

Perform Operations with masks. A mask is a direct-access file with the name MNAME.MASK. It must contain as many 32 bit words as there are events in the associated Ntuple. Masks are interesting when only a few events of a Ntuple are selected with a time consuming selection algorithm. For example if the command:

```
NT/PLOT 30.X  Z<0.4.AND.SELECT.FTN>>MNAME(6)
```

then for all events in Ntuple 30 satisfying the condition above, the bit 6 in the corresponding mask words will be set. One can then use the mask as selection mechanism. Example:

```
NT/PLOT 30.X MNAME(6)
```

will produce the same results than the NT/PLOT command above, but will be much faster if only a small fraction of all the events is selected. MASKS are automatically saved across PAW sessions on files. Example:

```
MASK  TEST N 10000
   creates a new mask on file TEST.MASK with enough words to
   process a Ntuple with 10000 events
MASK  TEST UP
   opens an existing mask for update and
   prints the active selection bits with explanation
```

NTUPLE/UWFUNC idn fname [chopt]

IDN	C	"Ntuple Identifier"
FNAME	C	"File name"
CHOPT	C	"Options" D='␣'

Possible CHOPT values are:

'␣'	Generate the FORTRAN skeleton of a selection function.
E	Present the selection function in the local editor.
P	Code to print events is generated (not valid for new Ntuples).
T	Names of the Ntuple variables are generated in DATA statements (not valid for new Ntuples).

To generate the FORTRAN skeleton of a selection function or the INCLUDE file with the columns declaration.
A FORTRAN function is generated if the FNAME is of the form, xxx.f, xxx.for, xxx.fortran. Otherwise

an INCLUDE file is generated. Example: If Ntuple ID=30 has variable names [X,Y,Z,ETOT,EMISS,etc] then:
NTUPLE/UWFUNC 30 SELECT.FOR will generate the file SELECT.FOR with:

```
FUNCTION SELECT(XDUMMY)
COMMON/PAWIDN/IDNEVT,VIDN1,VIDN2,VIDN3,X,Y,Z,ETOT,EMISS,etc
SELECT=1.
END
```

Then using the command EDIT one can modify this file which could then look something like (IDNEVT is the event number):

```
FUNCTION SELECT(XDUMMY)
COMMON/PAWIDN/IDNEVT,VIDN1,VIDN2,VIDN3,X,Y,Z,ETOT,EMISS,etc
IF(X**2+Y**2.GT.Z**2.OR.ETOT.GT.20.)THEN
   SELECT=1.
ELSE
   SELECT=0.
ENDIF
END
```

If in a subsequent command NTUPLE/PLOT, the selection function SELECT is used, then:

```
If NTUPLE/PLOT 30.ETOT SELECT.FOR
   VIDN1=ETOT
If NTUPLE/PLOT 30.SQRT(X**2+Y**2)%(ETOT-EMISS)
   VIDN1=ETOT-EMISS
   VIDN2=SQRT(X**2+Y**2)
```

NTUPLE/UWFUNC 30 SELECT.INC will generate an include file. This include file may be referenced in a selection function in the following way:

```
FUNCTION SELECT(XDUMMY)
include 'select.inc'
SELECT=1.
IF(X.LE.Y)SELECT=0.
END
```

NTUPLE/LINTRA `idn [ chopt nevent ifirst nvars varlis ]`

`IDN`	C	"Ntuple Identifier"
`CHOPT`	C	"Options" `D='␣'`
`NEVENT`	I	"Number of events" `D=99999999`
`IFIRST`	I	"First event" `D=1`
`NVARS`	I	"Number of the most significant variables " `D=20 R=0:20`
`VARLIS`	C	"Names of the NVARS most significant variables "

Possible `CHOPT` values are:

- `N` The variables are normalized. This option is useful in the case the ranges of variables are very different
- `P` Print more results about the analysis

Data reduction on Ntuple. The method used is the PRINCIPAL COMPONENTS ANALYSIS. The Principal Components Analysis method consists in applying a linear transformation to the original variables of a ntuple. This transformation is described by an orthogonal matrix and is equivalent to a rotation of the original space to a new set of coordinates vectors, which hopefully provide easier identification and dimensionality reduction. This matrix is real positive definite and symmetric and has all its eigenvalues greater than zero. Among the family of all complete orthonormal bases, the basis formed by the eigenvectors of the covariance matrix and belonging to the largest eigenvalues corresponds to the most significant features for the description of the original ntuple. Reduction of the variables for NEVENT events starting at IFIRST The default is to take all the 20 first variables. This command creates a file : -⟩ XTOXSI.FORTRAN or xtoxsi.for,xtoxsi.ftn. This file contains a Fortran function which computes the new variables. These new variables can be visualized in PAW with for example:

```
PAW > Ntuple/plot id.xtoxsi.ftn(1)
PAW > Ntuple/plot id.xtoxsi.ftn(1)%xtoxsi.ftn(3)
```

NTUPLE/VMEM `[ mxsize ]`

`MXSIZE` I "Maximum size of dynamic memory buffer in MBytes" `D=-1 R=-2:128`

Change or show the size of the dynamic memory buffer used to store Ntuple columns during Ntuple analysis. The default is 10 MB. Giving a value of 0 turns the buffer facility off. The upper limit is 128 MB, but be sure you have enough swap space and realize that when the buffer is swapped to disk you loose part of the benefit of the buffer facility (which is to reduce the number of disk accesses). Omitting the argument or specifying -1 will show you the current upper limit and used and free space. Giving -2 shows which columns are currently stored in memory.

Chapter 16: GRAPHICS

Interface to the graphics packages HPLOT and HIGZ.

GRAPHICS/SET [chatt value]

CHATT C "Attribute name" D='SHOW'
VALUE R "Attribute value" D=0

Set a specific HPLOT attribute. If CHATT='SHOW', print defaults and current values for all attributes. If CHATT='*', restore default values for all attributes. If VALUE=0, the attribute is set to its default value.

HPLSET : Current values in use

Parameter	Current value	Default value	Explanation
XSIZ	20.00	20.00	Size along X
YSIZ	20.00	20.00	Size along Y
XMGL	2.00	2.00	X MarGin Left
XMGR	2.00	2.00	X MarGin Right
XLAB	1.40	1.40	distance y axis to LABel
XVAL	.40	.40	distance y axis to axis VALues
XTIC	.30	.30	X axis TICk marks length
YMGL	2.00	2.00	Y MarGin Low
YMGU	2.00	2.00	Y MarGin Up
YLAB	.80	.80	distance x axis to LABel
YVAL	.20	.20	distance x axis to axis VALues
YTIC	.30	.30	Y axis TICk marks length
YNPG	.60	.60	Y position for Number of PaGe
YGTI	1.50	1.50	Y position of Global TItle
YHTI	1.20	1.20	Y position of Histogram TItle
SMGR	.00	.00	Stat MarGin Right (%)
SMGU	.00	.00	Stat MarGin Up (%)
KSIZ	.28	.28	Hershey charact. (HPLKEY) SIZe
GSIZ	.28	.28	Global title SIZe
TSIZ	.28	.28	histogram Title SIZe
ASIZ	.28	.28	Axis label SIZe
CSIZ	.28	.28	Comment and stat SIZe
PSIZ	.28	.28	Page number SIZe
VSIZ	.28	.28	axis Values SIZe
SSIZ	.28	.28	aSterisk SIZe (for functions)
2SIZ	.28	.28	scatter-plot & table char. SIZe
XWIN	2.00	2.00	X space between WINdows
YWIN	2.00	2.00	Y space between WINdows
HMAX	.90	.90	Histogram MAXimum for scale
PASS	1.00	1.00	number of PASS for characters

CSHI	.03	.03	Character SHIft between 2 pass
BARO	.25	.25	BAR histogram Offset (%)
BARW	.50	.50	BAR histogram Width (%)
DASH	.15	.15	length of basic DASHed segment
DMOD	1	1	Dash MODe (or type) for lines
GRID	3	3	GRID line type
DATE	2	2	DATE position
FILE	1	1	FILE name position
STAT	1111	1111	STAT values to be plotted
FIT	101	101	FIT values to be plotted
HTYP	0	0	Histogram fill area TYPe
BTYP	0	0	Box fill area TYPe
PTYP	0	0	Picture fill area TYPe
FTYP	0	0	Function fill area TYPe
HCOL	.00	1.00	Histogram fill area COLor
BCOL	1.00	1.00	Box fill area and shading COLor
PCOL	1	1	Picture fill area COLor
FCOL	1	1	Function fill area COLor
XCOL	1	1	X axis COLor
YCOL	1	1	Y axis COLor
HWID	1	1	Histogram line WIDth
BWID	1	1	Box line WIDth
PWID	1	1	Picture line WIDth
FWID	1	1	Function line WIDth
XWID	1	1	X ticks WIDth
YWID	1	1	Y ticks WIDth
TFON	2	2	Text (and Title) FONT and PREC
GFON	2	2	Global title FONT and PREC
VFON	2	2	axis Values FONT and PREC
LFON	2	2	axis Labels FONT and PREC
CFON	2	2	Comment FONT and PREC
NDVX	10510.00	10510.00	Number of DIVisions for X axis
NDVY	10510.00	10510.00	Number of DIVisions for Y axis
NDVZ	10510.00	10510.00	Number of DIVisions for Z axis
FPGN	1	1	First PaGe Number
ERRX	.50	.50	ERRor on X (% of bin width)
1DEF	0	0	1D Plot Option
2DEF	0	0	2D Plot Option

IGSET : Current values in use

Parameter	Current value	Default value	Explanation
FAIS	0	0	Fill area interior style
FASI	1	1	Fill area style index

LTYP	1	1	Line type
BASL	.150	.010	Basic segment length (NDC)
LWID	1.000	1.000	Line width
MTYP	1	1	Marker type
MSCF	1.000	1.000	Marker scale factor
PLCI	1	1	Polyline color index
PMCI	1	1	Polymarker color index
FACI	1	1	Fill area color index
TXCI	1	1	Text color index
TXAL	0 0	0 0	Text alignment
CHHE	.280	.010	Character height
TANG	.000	.000	Text angle
TXFP	0 2	0 2	Text font and precision
PICT	1	1	Current automatic number
BORD	0	0	Border flag
PASS	1	1	Number of pass in IGTEXT
CSHI	.030	.020	IGTEXT shift
LASI	.018	.018	Label axis size
LAOF	.013	.013	Label axis offset
TMSI	.019	.019	Tick marks size
AWLN	.000	.000	Axis wire lenght
BARO	.250	.250	Offset of IGHIST (IGRAPH) bars
BARW	.500	.500	Width of IGHIST (IGRAPH) bars
NCOL	8	8	Number of COLors
CLIP	1	1	Clipping mode
NLIN	40	40	Number of line for 3D shapes
AURZ	0	0	Automatic saving flag
DIME	2	2	Dimension used (2D or 3D)

GRAPHICS/OPTION [choptn]

CHOPTN C "Option name" D='SHOW'

Set general plotting options for HPLOT. If CHOPTN='SHOW' print all current and default options. If CHOPTN='*', restore all default options.

HPLOPT : Option values			
Current	Default	Alternative	Explanation
VERT	VERT	HORI	VERTical or HORIzontal orientation of paper
NEAH	NEAH	EAH	Error bars And Histogram are plotted (if both are present)
NCHA	NCHA	CHA	scatter plots drawn with dots

				(NCHA) or 1 char./bin (CHA)
NAST	NAST	AST	functions drawn with (AST)	
				or without (NAST) asterisks
SOFT	SOFT	HARD	SOFTware or HARDware characters	
				are used
NSQR	NSQR	SQR	size is set to the largest	
				square (SQR)
HTIT	HTIT	UTIT	HBOOK TITle (HTIT)	
				or User TITle (UTIT) is printed
TAB	TAB	NTAB	table printed as TABles (TAB)	
				or scatter plots (NTAB)
BOX	BOX	NBOX	a box is (BOX) or is not (NBOX)	
				drawn around picture
NTIC	NTIC	TIC	cross-wires are drawned (TIC)	
				or not (NTIC) on each plot
NSTA	NSTA	STA	STAtistics are printed (STA)	
				or not (NSTA) on each plot
NFIT	NFIT	FIT	FIT parameters are printed	
				or not (NFIT) on each plot
NZFL	NZFL	ZFL	picture is (ZFL) or is not	
				(NZFL) put in Z data base
NPTO	NPTO	PTO	PTO (Please Turn Over)	
				(NPTO)
NBAR	NBAR	BAR	BAR charts for histogram	
				(NBAR)
DVXR	DVXR	DVXI	Integer (DVXI) or Real (DVXR)	
				divisions for X axis
DVYR	DVYR	DVYI	Integer (DVYI) or Real (DVYR)	
				divisions for Y axis
NGRI	NGRI	GRID	GRID or not grid (NGRI)	
				on X and Y axis
NDAT	NDAT	DATE	DATE is printed (DATE)	
				or not (NDAT) on each plot
NFIL	NFIL	FILE	FILE name is printed (FILE)	
				or not (NFIL) on each plot
A4	A4	A0/6	page format for the plotter	
				(A0,A1,A2,A3,A4,A5,A6)
NOPG	NOPG	P	page number is (P)	
				or is not (NOPG) printed
LINY	LINY	LOGY	LINear or LOGarithmic scale	
				in Y
LINX	LINX	LOGX	LINear or LOGarithmic scale	
				in X
LINZ	LINZ	LOGZ	LINear or LOGarithmic scale	
				in Z (Lego or Surface)
NHST	HSTA	HNST	Filling statistics (HSTA)	

```
|             |             |               | (HNST)                            |
+-------------+-------------+---------------+-----------------------------------+
```

GRAPHICS/METAFILE [lun metafl chmeta]

LUN	I	"Logical unit number" D=0
METAFL	I	"Metafile ID" D=0
CHMETA	C	"Metafile name" D='␣'

Set the metafile logical unit and metafile type. This command controls the destination of the subsequent graphics output. Example:

```
LUN =-10 output only on metafile opened on unit 10;
LUN =  0 output only on screen;
LUN = 10 output on both screen and metafile opened on unit 10;
```

Use the command FORTRAN/FILE to open a new file, FORTRAN/CLOSE to close it. Note that PAW opens the file PAW.METAFILE on the unit 10 at initialization time.

```
METAFL=   4 Appendix E GKS.
METAFL=-111 HIGZ/PostScript (Portrait).
METAFL=-112 HIGZ/PostScript (Landscape).
METAFL=-113 HIGZ/Encapsulated PostScript.
METAFL=-114 HIGZ/PostScript Color (Portrait).
METAFL=-115 HIGZ/PostScript Color (Landscape).
METAFL=-777 HIGZ/LaTex Encapsulated.
METAFL=-778 HIGZ/LaTex.
```

The PostScript metafile types have the following format:

```
                    -[Format][Nx][Ny][Type]
Where:
```

[Format] Is an integer between 0 and 99 which defines the format of the

```
paper. For example if Format=3 the paper is in the standard
A3 format. Format=4 and Format=0 are the same and
define an A4 page.
The A0 format is selected by Format=99.
The US format Letter is selected by Format=100.
The US format Legal is selected by Format=200.
The US format Ledger is selected by Format=300.
```

[Nx, Ny] Specify respectively the number of zones on the x and y axis.

```
Nx and Ny are integers between 1 and 9.
```

[Type] Can be equal to:

```
          1: Portrait mode with a small margin at the bottom of the page.
          2: Landscape mode with a small margin at the bottom of the page.
          4: Portrait mode with a large margin at the bottom of the page.
          5: Landscape mode with a large margin at the bottom of the page.
             The large margin is useful for some PostScript printers (very
             often for the colour printers) as they need more space to grip
             the paper for mechanical reasons. Note that some PostScript
             colour printers can also use the so called "special A4" format
             permitting the full usage of the A4 area; in this case larger
             margins are not necessary and {\tt Type}=1 or 2 can be used.
          3: Encapsulated PostScript. This Type permits the generation of
             files which can be included in other documents, for example
             in LaTeX files. Note that with this Type, Nx and Ny must always
             be equal to 1, and Format has no meaning. The size of the
picture
             must be specified by the user via the SIZE command. Therefore
             the workstation type for Encapsulated PostScript is -113. For
             example if the name of an encapsulated PostScript file is
             example.eps, the inclusion of this file into a LaTeX file will
             be possible via (in the LaTeX file):
                     \begin{figure}
                      \epsffile{example.eps}
                      \caption{Example of Encapsulated PostScript in LaTeX.}
                      \label{EXAMPLE}
                     \end{figure}
```

With Type=1,2,4 and 5 the pictures are centered on the page, and the usable area on paper is proportional to the dimensions of A4 format. Examples: -111 or -4111 defines an A4 page not divided. -6322 define an A6 landscape page divided in 3 columns and 2 rows.

```
+-------+-------+-------+
|   1   |   2   |   3   |
+-------+-------+-------+
|   4   |   5   |   6   |
+-------+-------+-------+
```

The first picture will be drawn in the area 1. After each clear the screen, the graphics output will appear in the next area in the order defined above. If a page is filled, a new page is used with the same grid. Note that empty pages are not printed in order to save paper. Ignoring formats smaller than A12, the total number of possible different PostScript workstation types is: 4x9x9x13+1 = 4213 !

GRAPHICS/WORKSTATION `iwkid [ chopt iwtyp ]`

`IWKID` I "Workstation ID" `D=1 Loop`
`CHOPT` C "Options" `D='OA'`
`IWTYP` I "Workstation type" `D=1`

Possible `CHOPT` values are:

- `O` Open a new workstation
- `C` Close a workstation
- `A` Activate a workstation
- `D` Deactivate a workstation
- `L` Give the list of open workstations

To create/delete workstations or change status.

```
IWKID > 0  Do the action specified by CHOPT on the
           workstation identified by IWKID.
IWKID = 0  Do the action specified by CHOPT on all
           workstations.
IWKID < 0  Do the action specified by CHOPT on the
           workstation identified by -IWKID and the
           complementary action on all the others.
```

GRAPHICS/SLIDE

Invoke the SLIDE package.

16.1 MISC

Miscellaneous HPLOT functions.

GRAPHICS/MISC/NEXT

Clear the screen. Initialize a new HIGZ picture if option ZFL or ZFL1 has been selected. Select the Normalization Transformation number 1 (cm).

GRAPHICS/MISC/CLR

Clear the screen.

GRAPHICS/MISC/LOCATE `[ ntpri chopt wkid ]`

`NTPRI` C "Transformation with highest priority" `D='-1'`
`CHOPT` C "Options" `D='R'`
`WKID` I "Workstation identifier" `D=1`

Possible `CHOPT` values are:

- `R` Request mode is used to locate the points (default)
- `S` Sample mode is used to locate the points
- `I` Integrate an histogram between 2 bins
- `+` Use the tracking cross (default is cross-hair)
- `T` The output is done on the terminal.

Locate points on the screen using the graphics cursor and output coordinates on terminal. Control is returned when the BREAK (right) mouse button is clicked (or CRTL/E) or when 20 points are located. The optional parameter NTPRI may be specified to locate a point in the specific transformation number NTPRI. NTPRI=-1 (default) means that all the histogram transformation numbers (10, 20, etc.) have priority on transformation number 1. WKID allows to define in which window the locator is performed.

```
Note: With the Motif version of PAW the locator is automatically
```

invoke when the mouse cursor enter the window.

GRAPHICS/MISC/VLOCATE `vecx vecy [ chopt ntpri wkid ]`

`VECX`	C	"Vector for coordinates X"
`VECY`	C	"Vector for coordinates Y"
`CHOPT`	C	"Options" `D='␣' Minus`
`NTPRI`	I	"Transformation with highest priority" `D=-1`
`WKID`	I	"Workstation identifier" `D=1`

Possible `CHOPT` values are:

`'␣'`	Use the cross-hair
`+`	Use the tracking cross
`-`	Use the rubber line
`L`	Connect points by a polyline
`P`	Draw the current polymarker at each point
`*`	Draw a * at each point
`S`	Sample mode is used. Allows to see the coordinates of point before clicking

Locate a set of points using the graphics cursor. Return corresponding coordinates in vectors X and Y. If vectors X or Y do not exist, they are automatically created. Control is returned when the point is outside picture limits or when the BREAK (right) mouse button is clicked (or CRTL/E). The optional parameter NTPRI may be specified to locate a point in the specific transformation number NTPRI (see LOCATE). WKID allows to define in which window the locator is performed.

GRAPHICS/MISC/HMOVE

Change the contents of a histogram channel using the cursor. Position the cursor to the channel to be changed, trigger graphics input, position the cursor to the new channel value (a rubber band box is used to visualize the change), trigger graphics input to fix the new value.

16.2 VIEWING

To define Normalization transformations. Either automatically (ZONE and SIZE) or 'by hand' (SVP, SWN and SELNT).

GRAPHICS/VIEWING/ZONE `[ nx ny ifirst chopt ]`

NX	I	"Number of divisions along X" D=1
NY	I	"Number of divisions along Y" D=1
IFIRST	I	"First division number" D=1
CHOPT	C	"Option" D='␣'

Possible CHOPT values are:

'␣'

S Redefine zones on current picture

'␣' Define the zones for all subsequent pictures.

Subdivide the picture into NX by NY zones, starting at zone IFIRST (count along X first).

GRAPHICS/VIEWING/SIZE [xsize ysize]

XSIZE	R	"Size along X" D=20.
YSIZE	R	"Size along Y" D=20.

Set the size of the picture. On the terminal, the pictures will have the ratio YSIZE/XSIZE, and, if a metafile is produced, pictures will be YSIZE by XSIZE cm. This command sets the parameters for the normalization transformation number 1 to [0-XSIZE], [0-YSIZE].

GRAPHICS/VIEWING/SVP nt x1 x2 y1 y2

NT	I	"Normalization transformation number"
X1	R	"Low X of viewport in NDC" D=0 R=0:1
X2	R	"High X of viewport in NDC" D=1 R=0:1
Y1	R	"Low Y of viewport in NDC" D=0 R=0:1
Y2	R	"High Y of viewport in NDC" D=1 R=0:1

Set the viewport of the normalization transformation NT in the Normalized Device Coordinates (NDC). Note that the command SELNT should be invoke in order to validate the viewport parameters.

GRAPHICS/VIEWING/SWN nt x1 x2 y1 y2

NT	I	"Normalize transformation number"
X1	R	"Low X of window in WC" D=0
X2	R	"High X of window in WC" D=20
Y1	R	"Low Y of window in WC" D=0
Y2	R	"High Y of window in WC" D=20

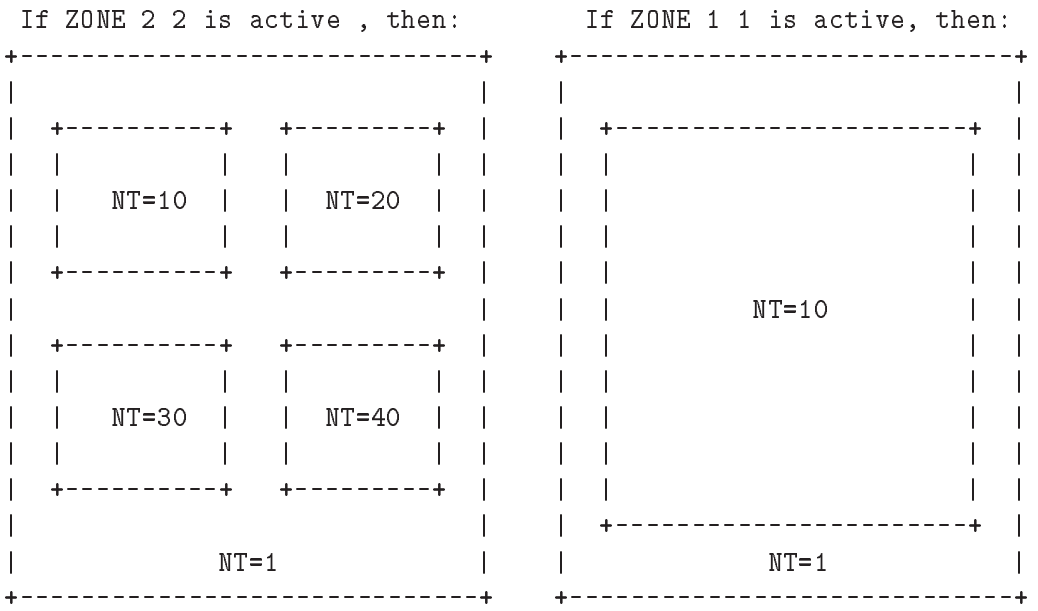
Set the window of the normalization transformation NT in World Coordinates (WC). Note that the command SELNT should be invoke in order to validate the window parameters. For example:

```
PAW > Nul 0 1 -1 1      | Draw an empty frame (0,1)x(-1,1)
PAW > Line 0 0 1 1      | Draw a line in (0,1)x(-1,1)
PAW > Swn 10 0 10 0 10  | Change the coordinates to (0,10)x(0,10)
PAW > Selnt 10          | Activate the coordinates (0,10)x(0,10)
PAW > Line 0 0 1 1      | Draw a line in (0,10)x(0,10)
```

GRAPHICS/VIEWING/SELNT `nt`

`NT` I "Normalization transformation number"

Select a normalization transformation number.



16.3 PRIMITIVES

Call HIGZ drawing primitives

GRAPHICS/PRIMITIVES/PLINE `n x y`

`N` I "Number of points"
`X` C "Vector name for X coordinates"
`Y` C "Vector name for Y coordinates"

Draw a polyline of N points X,Y in the current Normalization transformation. The PLINE attributes can be changed with the command SET.

Example:

```
   SET * ; OPT *          | Reset the defaults
   NUL -1 1 0 1           | Draw a frame (cf HELP NULL)
* Create vector X and Y (cf HELP SIGMA)
   SIGMA X=ARRAY(100,-1#1)
   SIGMA Y=X*X
   SET PLCI 4             | The line color is blue
   SET LWID 6             | The line width is 6
   SET LTYP 2             | The line type is dashed
   PLINE 100 X Y          | Draw a 100 points line
```

GRAPHICS/PRIMITIVES/LINE `x1 y1 x2 y2`

`X1` R "X first coordinate"
`Y1` R "Y first coordinate"
`X2` R "X second coordinate"
`Y2` R "Y second coordinate"

Draw a line connecting points (X1,Y1) and (X2,Y2) in the current Normalization transformation. This command is kept for backward compatibility. It has a reverse calling sequence compare to BOX or ARROW and it doesn't take LOG scales into account. It is recommended to use DLINE instead. The LINE attributes can be changed with the command SET.

Example:

```
SET * ; OPT *          | Reset the defaults
NUL 0 5 0 5            | Draw a frame (cf HELP NULL)
SET PLCI 2             | The line color is red
SET LWID 6             | The line width is 6
SET LTYP 3             | The line type is dotted
LINE 0 0 5 5           | Draw a line
```

GRAPHICS/PRIMITIVES/DLINE `x1 x2 y1 y2`

`X1` R "X first coordinate"
`X2` R "X second coordinate"
`Y1` R "Y first coordinate"
`Y2` R "Y second coordinate"

Draw a line connecting points (X1,Y1) and (X2,Y2) in the current Normalization transformation taking care of logarithmic scales. The DLINE attributes can be changed with the command SET.

Example:

```
SET * ; OPT *          | Reset the defaults
OPTION LOGY            | Log scale on the Y axis.
NUL 0 5 1 100          | Draw a frame (cf HELP NULL)
SET PLCI 2             | The line color is red
SET LWID 6             | The line width is 6
SET LTYP 1             | The line type is solid
DLINE 0 5 1 10         | Draw a line
```

GRAPHICS/PRIMITIVES/FAREA `n x y`

`N` I "Number of points"
`X` C "Vector name for X coordinates"
`Y` C "Vector name for Y coordinates"

Fill the area defined by the N points X,Y in the current Normalization transformation. The FAREA attributes can be changed with the command SET.

Example:

```
   SET * ; OPT *          | Reset the defaults
   NUL -1.1 1.1 -1.1 1.1  | Draw a frame (cf HELP NULL)
* Create vector X and Y (cf HELP SIGMA)
   SIGMA X=ARRAY(100,-3.14#3.14)
   SIGMA Y=SIN(X)*COS(X)
   SIGMA X=COS(X)
   SET FACI 2             | The fill area color is red
   SET FAIS 1             | The fill area interior style is solid
   FAREA 100 X Y          | Draw a 100 points line
   SET FACI 1             | The fill area color is black
   SET FAIS 0             | The fill area interior style is hollow
   FAREA 100 X Y          | Draw a 100 points line
   SET FAIS 3             | The fill area interior style is hatched
   SET FASI 245           | Defines the type of hatches
   FAREA 100 X Y          | Draw a 100 points line
```

GRAPHICS/PRIMITIVES/PMARKER `n x y`

`N` I "Number of points"
`X` C "Vector name for X coordinates"
`Y` C "Vector name for Y coordinates"

Draw polymarkers at the N points X,Y in the current Normalization transformation. The PMARKER attributes can be changed with the command SET.

Example:

```
   SET * ; OPT *          | Reset the defaults
   NUL -3.2 3.2 -1 1      | Draw a frame (cf HELP NULL)
* Create vector X and Y (cf HELP SIGMA)
   SIGMA X=ARRAY(100,-3.14#3.14)
   SIGMA Y=SIN(X)*COS(X)
   SET PMCI 6             | The marker color is magenta
   SET MTYP 3             | The marker type is *
   SET MSCF 2             | The marker size is 2
   PMARKER 100 X Y        | Draw a 100 points polymarker
```

GRAPHICS/PRIMITIVES/BOX `x1 x2 y1 y2`

`X1` R "X coordinate of first corner"
`X2` R "X coordinate of second corner"
`Y1` R "Y coordinate of first corner"
`Y2` R "Y coordinate of second corner"

Draw and fill a box with the current fill area and line attributes. Use the current Normalization transformation. The BOX attributes can be changed with the command SET.

Example:

```
SET * ; OPT *          | Reset the defaults
NULL 0 10 0 10         | Draw a frame (cf HELP NULL)
SET FAIS 0             | Fill area interior style hollow
BOX 1 3 1 3            | Draw a box
SET FAIS 1             | Fill area interior style solid
BOX 1 3 3 5            | Draw a box
SET FAIS 3             | Fill area interior style hatched
SET FASI 245           | Changes the type of hatches
BOX 1 3 5 7            | Draw a box
SET FASI 3             | Changes the type of hatches
BOX 3 5 5 7            | Draw a box
SET BORD 1             | The border is requested
SET PLCI 2             | Line color is red
SET FASI 4             | Changes the type of hatches
BOX 5 7 5 7            | Draw a box
```

GRAPHICS/PRIMITIVES/FBOX `x1 x2 y1 y2 x3 x4 y3 y4`

`X1`	R	"X coord of first corner of ext box"
`X2`	R	"X coord of second corner of ext box"
`Y1`	R	"Y coord of first corner of ext box"
`Y2`	R	"Y coord of second corner of ext box"
`X3`	R	"X coord of first corner of int box"
`X4`	R	"X coord of second corner of int box"
`Y3`	R	"Y coord of first corner of int box"
`Y4`	R	"Y coord of second corner of int box"

Draw and fill a frame (2 nested boxes) with the current fill area and line attributes. Use the current Normalization transformation. The FBOX attributes can be changed with the command SET.

Example:

```
SET * ; OPT *          | Reset the defaults
NULL 0 10 0 10         | Draw a frame (cf HELP NULL)
SET FAIS 3             | Fill area interior style hatched
SET FASI 3             | Changes the type of hatches
SET FACI 2             | Fill are color is red
SET PLCI 4             | Line color is blue
SET LWID 8             | The line width is 8
SET BORD 1             | The border is requested
FBOX 1 9 1 9 3 7 3 7   | Draw a frame box
```

GRAPHICS/PRIMITIVES/ARROW `x1 x2 y1 y2 [ size ]`

X1	R	"X coordinate of start point"
X2	R	"X coordinate of end point"
Y1	R	"Y coordinate of start point"
Y2	R	"Y coordinate of end point"
SIZE	R	"Arrow size" D=0.4

Draw an arrow Use the current Normalization transformation. The ARROW attributes can be changed with the command SET. ARROW and LINE attributes are the same.

```
(X1,Y1) ----> (X2,Y2) if SIZE>0.
(X1,Y1) <---> (X2,Y2) if SIZE<0.
```

Example:

```
SET * ; OPT *          | Reset the defaults
NULL 0 10 0 7          | Draw a frame (cf HELP NULL)
ARROW 1 9 1 1 .2       | Draw a simple arrow (left to right)
ARROW 9 1 2 2 .4       | Draw a simple arrow (right to left)
ARROW 1 9 3 3 -.8      | Draw a double arrow
SET PLCI 2             | Arrow color is red
ARROW 1 9 4 4 -.8      | Draw a double arrow
SET LWID 8             | Arrow line width is 8
ARROW 1 9 5 5 -.8      | Draw a double arrow
SET LTYP 3             | Arrow line type is dotted
ARROW 1 9 6 6 -.8      | Draw a double arrow
```

GRAPHICS/PRIMITIVES/HELIX [x1 y1 x2 y2 r wi phi]

X1	R	"X coordinate of the begin of helix" D=0.
Y1	R	"Y coordinate of the begin of helix" D=0.
X2	R	"X coordinate of the end of helix" D=10.
Y2	R	"Y coordinate of the end of helix" D=10.
R	R	"Radius of helix" D=.3
WI	R	"Number of turns " D=1.
PHI	R	"Projection angle " D=15.

Draw an helix with the current line attributes. Use the current Normalization transformation. Feynman graph: gluon phi = 30, photon phi = 0.
Example:

```
SET * ; OPT *            | Reset the defaults
NUL 0 10 0 10 'AB'       | Draw a frame (cf HELP NULL)
HELIX 1 1 3 3 ! 10 !     | Draw an helix
SET LWID 8               | Helix line width is 8
HELIX 3 3 7 7 1  5 !     | Draw an helix
SET PLCI 2               | Arrow color is red
SET LTYP 2               | Helix line type is dashed
HELIX 7 7 10 10 .2 5 10  | Draw an helix
```

GRAPHICS/PRIMITIVES/ARCHELIX `[ x1 y1 x2 y2 r wi phi rl ]`

X1	R	"X coordinate of the begin of helix" D=0.
Y1	R	"Y coordinate of the begin of helix" D=0.
X2	R	"X coordinate of the end of helix" D=10.
Y2	R	"Y coordinate of the end of helix" D=10.
R	R	"Radius of helix" D=.3
WI	R	"Number of turns " D=1.
PHI	R	"Projection angle " D=30.
RL	R	"Radius of loop " D=15.

Draw an archelix with the current line attributes. Use the current Normalization transformation. Feynman graph: gluon phi = 30, photon phi = 0.
Example:

```
SET * ; OPT *               | Reset the defaults
NUL 0 10 0 10 'AB'          | Draw a frame (cf HELP NULL)
ARCHELIX 1 1 3 3 ! 9 ! 1    | Draw an helix
SET LWID 8                  | Helix line width is 8
ARCHELIX 3 3 7 7 ! 9 ! 1    | Draw an helix
SET PLCI 2                  | Arrow color is red
SET LTYP 2                  | Helix line type is dashed
ARCHELIX 7 7 10 10 ! 9 ! 3  | Draw an helix
```

GRAPHICS/PRIMITIVES/ARLINE `[ x1 y1 x2 y2 h ]`

X1	R	"X coordinate of the begin" D=0.
Y1	R	"Y coordinate of the begin" D=0.
X2	R	"X coordinate of the end" D=10.
Y2	R	"Y coordinate of the end" D=10.
H	R	"arrow size" D=.5

Draw a line with arrow in middle (fermion line) with the current line and fill area attributes. Use the current Normalization transformation.
Example:

```
SET * ; OPT *          | Reset the defaults
NULL 0 10 0 6          | Draw a frame (cf HELP NULL)
ARLINE 1 1 9 1 .2      | Draw a arrow line (left to right)
ARLINE 9 2 1 2 .4      | Draw a arrow line (right to left)
SET PLCI 2             | Arrow color is red
SET FAIS 1             | Fill area interior style solid
ARLINE 9 3 1 3 .4      | Draw a arrow line (right to left)
SET LWID 8             | Arrow line width is 8
SET FACI 4             | The fill area color is blue
ARLINE 9 4 1 4 .4      | Draw a arrow line (right to left)
SET LTYP 3             | Arrow line type is dotted
ARLINE 9 5 1 5 .4      | Draw a arrow line (right to left)
```

GRAPHICS/PRIMITIVES/FPOINT `[ x y r ]`

X	R	"X" D=0.
Y	R	"Y" D=0.
R	R	"Radius" D=.5

Draw a filled point (vertex) with the current fill area attributes. Use the current Normalization transformation.

Example:

```
SET * ; OPT *          | Reset the defaults
NULL 0 10 0 10         | Draw a frame (cf HELP NULL)
SET FAIS 1             | Fill area interior style solid
FPOINT 5 1 .1          | Draw a filled point
FPOINT 5 3 .2          | Draw a filled point
FPOINT 5 5 .3          | Draw a filled point
SET FACI 4             | The fill area color is blue
FPOINT 5 7 .4          | Draw a filled point
FPOINT 5 9 .5          | Draw a filled point
```

GRAPHICS/PRIMITIVES/AXIS `x0 x1 y0 y1 wmin wmax ndiv [ chopt ]`

X0	R	"X axis origin in WC"
X1	R	"X end axis in WC"
Y0	R	"Y axis origin in WC"
Y1	R	"Y end axis in WC"
WMIN	R	"Lowest value for labels"
WMAX	R	"Highest value for labels"
NDIV	I	"Number of divisions" D=510
CHOPT	C	"Options" D='␣' Minus

Possible CHOPT values are:

'␣'	Draw an axis with default values.
G	Logarithmic scale, default is linear.
B	Blank axis. Useful to superpose axis.
U	Unlabeled axis, default is labeled.
+	Tick marks are drawn on Positive side. (default)
-	Tick marks are drawn on the negative side.
=	Tick marks are drawn on Equal side
P	Labels are drawn Parallel to the axis
O	Labels are drawn Orthogonal to the axis (Top to Down).
o	Labels are drawn Orthogonal to the axis (Down to Top).
R	labels are Right adjusted on tick mark.

L	labels are Left adjusted on tick mark.
C	labels are Centered on tick mark.
M	In the Middle of the divisions.
Y	Direction of labels DOWN . Default is RIGHT
.	Dot obligatory
T	Alphanumeric labels .
S	Tick marks Size
H	Labels Height
D	Distance labels-axis
N	No bining optimization
I	Integer labeling

Draw an axis in the current Normalization transformation.

```
      NDIV=N1 + 100*N2 + 10000*N3
      N1, N2, N3 = Number of 1st, 2nd, 3rd divisions respectively, eg:.
          NDIV=0 --> no tick marks.
          NDIV=2 --> 2 divisions, one tick mark in the middle
                     of the axis.
Orientation of tick marks on axis: Tick marks are normally drawn
on the positive side of the axis.However, if X0=X1, then Negative .
      CHOPT='+': tick marks are drawn on Positive side. (default)
      CHOPT='-': tick marks are drawn on the negative side.
      i.e: '+-' --> tick marks are drawn on both sides of the axis.
Position of labels on axis: Labels are normally drawn on side
opposite to tick marks.However:
      CHOPT= '='   on Equal side
Orientation of labels on axis: Labels are normally drawn
parallel to the axis. However if X0=X1, then Orthogonal
                              if Y0=Y1, then Parallel
      CHOPT= 'P' : Parallel to the axis
      CHOPT= 'O' : Orthogonal to the axis (Top to Down).
      CHOPT= 'O' : Orthogonal to the axis (Down to Top).
Position of labels on tick marks: Labels are centered on
tick marks. However , if X0=X1, then they are right adjusted.
      CHOPT='R': labels are Right adjusted on tick mark.
                 (default is centered)
      CHOPT='L': labels are Left adjusted on tick mark.
      CHOPT='C': labels are Centered on tick mark.
      CHOPT='M': In the Middle of the divisions.
Direction of labels: Default is RIGHT
      CHOPT='Y':   Down
Format of labels: Blank characters are stripped, and then the
label is correctly aligned. The dot,if last character of the
string, is also stripped, unless
```

```
      CHOPT='.'     Dot obligatory
In the following, we have some parameters, like
tick marks length and characters height (in percentage
of the length of the axis).The default values are as follows:
   Primary tick marks: 3.0 %
   Secondary tick marks: 1.5 %
   Third order tick marks: .75 %
   Characters height for labels: 2%
   Characters spacing (related to height): 40%
   Labels offset: 4.0 %
Type of labels: Labels are normally numeric . However, alphanumeric
labels can be drawn (see command LABEL).
      CHOPT='T':   Alphanumeric labels .
Intrinsic parameters: These values can be changed with the command
SET. The default value is used unless the corresponding option is
selected by CHOPT:
      CHOPT='D' The distance between the labels and the axis
                (the offset) is given by the preceding command
                SET with the parameter LAOF.
      CHOPT='H' The size (height) of the labels is given by the
                preceding command SET with the parameter LASI.
      CHOPT='S' The size of the tick marks is given by the preceding
                command SET with the parameter TMSI.
Axis bining optimization: By default the axis bining is optimized .
      CHOPT='N': No bining optimization
      CHOPT='I': Integer labeling
```

Example:

```
SET * ; OPT *                          | Reset the defaults
NUL 0 12 0 12 'A'                      | Draw a frame (cf HELP NULL)
AXIS 1  11  1  1  0  100  510 'A'      | Axis with arrow
AXIS 1  11  3  3  1  10000  510 'G'    | LOG axis
LABEL 1 11 a b c d e f g h i j k       | define alphanumeric labels
AXIS 1  11  5  5  0  12  11 'NATY'     | alphanumeric labeling
AXIS 1  11  6  6  -100  0  510 'A'
AXIS 11  1  7  7  -100  0  810 'A+-'   | Double side tick marks
AXIS 1  11  8  11  0  1234567  615 'A' | exponent is required
```

GRAPHICS/PRIMITIVES/ARC `x1 y1 r1 [ r2 phimin phimax ]`

X1	R	"X coordinate of centre"
Y1	R	"Y coordinate of centre"
R1	R	"Inner radius"
R2	R	"Outer radius" D=-1.
PHIMIN	R	"Minimum angle" D=0.
PHIMAX	R	"Maximum angle" D=360.

Draw an arc of circle with the current fill area and line attributes. Use the current Normalization transformation. If R1 is not equal to R2 the area between the two arcs of radius R1 and R2 is filled according to the current fill area attributes. The border is never drawn unless the interior style is hollow or the command SET BORD 1 has been called. If R1 is equal to R2 a polyline is drawn.

Example:

```
SET * ; OPT *          | Reset the defaults
NULL 0 20 0 20 'AB'    | Draw a frame (cf HELP NULL)
SET PLCI 2             | Line color is red
SET LWID 6             | Line width is 6
ARC  5  5 4 4 ! !      | Draw an circle
ARC  5 15 4 4 30 260   | Draw an arc of circle
SET FAIS 3             | Fill area with hatches
SET FASI 3             | Type of hatches
ARC 15 15 1 4 ! !      | Draw an arc
SET BORD 1             | Border is requested
ARC 15  5 1 4 30 !     | Draw an arc
```

GRAPHICS/PRIMITIVES/PIE `x0 y0 radius n values [ chopt iao ias iac ]`

`X0`	R	"X coordinate of centre of the pie"
`Y0`	R	"Y coordinate of centre of the pie"
`RADIUS`	R	"Radius of the pie chart"
`N`	I	"Number of values"
`VALUES`	C	"Vector name for N values"
`CHOPT`	C	"Options" `D='␣'`
`IAO`	C	"Name of vector with offsets" `D='␣'`
`IAS`	C	"Name of vector with styles" `D='␣'`
`IAC`	C	"Name of vector with colors" `D='␣'`

Possible `CHOPT` values are:

`'␣'`	Draw a Pie Chart with default values.
`C`	Colours array is present.
`L`	Alphanumeric labels are required.
`O`	Offset array is present.
`N`	The label of each slice will be the corresponding numeric value in array VALUES.
`P`	The label of each slice will be in expressed in percentage.
`S`	Style array is present.
`H`	Force the labels size to be the current character height. Without this option the labels size is computed automatically.
`R`	Draw the labels aligned on the radius of each slice.

Draw a pie chart in the current Normalization transformation.

Example:

```
   SET * ; OPT *                                  | Reset the defaults
   NULL 0 20 0 20 'AB'                            | Draw a frame
   LABEL 1 5 'Lab1' 'Lab2' 'Lab3' 'Lab4' 'Lab5'   | define labels
* Initialize vectors
   V/CRE VWS(5) R 28.3 18.6 16.9 13.5 22.7
   V/CRE OFFSET(5)  R 2*0. 2*20. 0.
   V/CRE COLOUR(5)  R 2 3 4 5 6
   SET FAIS 1                                     | Fill solid
   SET BORD 1                                     | Draw the border
   PIE 10. 10. 7.  5 VWS 'L' OFFSET ! COLOUR      | Draw the pie chart
```

GRAPHICS/PRIMITIVES/TEXT `x y text size [ angle chopt ]`

`X`	R	"X coordinate"
`Y`	R	"Y coordinate"
`TEXT`	C	"Text to be drawn"
`SIZE`	R	"Text size" `D=0.3`
`ANGLE`	R	"Comment angle" `D=0`
`CHOPT`	C	"Justification option" `D='L'`

Possible `CHOPT` values are:

- `L` Text is Left justified.
- `C` Text is Centered.
- `R` Text is Right justified.

Draw text at position X,Y in the current normalization transformation using the software font IGTEXT. SIZE is always given in centimeters (as defined by the command SIZE). A boldface effect can be obtained using the parameters PASS and CSHI of the command SET. The text color can be changed by SET TXCI. Example:

```
   SET * ; OPT *                                  | Reset the defaults
   NULL 0 10 0 10                                 | Draw a frame
   TEXT 5 1 'Left justified' .5 0. L
   TEXT 5 2 'Centered' .5 0. C
   TEXT 5 3 'Right justified' .5 0. R
   TEXT 5 4 '-- 30 degrees' .5 30. L
   TEXT 5 4 '-- 60 degrees' .5 60. L
   TEXT 5 4 '-- 90 degrees' .5 90. L
   TEXT 5 4 '-- 120 degrees' .5 120. L
   TEXT 5 4 '-- 150 degrees' .5 150. L
   TEXT 5 8 'Some Greek ... [a, b, c, d]' .5 0. C
   Set PASS 7                                     | Number of passes
   TEXT 5 9 'Bold TEXT' .5 0. C
```

GRAPHICS/PRIMITIVES/ITX `x y text`

X	R	"X coordinate"
Y	R	"Y coordinate"
TEXT	C	"Text to be drawn"

Draw text at position X,Y in the current Normalization transformation, using the current font parameters. The font and the precision can be changed by SET TXFP. The character size can be changed by SET CHHE. The text color can be changed by SET TXCI. The text orientation can be changed with SET TXAL. The text angle can be changed by SET TANG.
Example:

```
SET * ; OPT *                          | Reset the defaults
NULL 0 10 0 6                          | Draw a frame
SET TXFP -20                           | Times bold
SET CHHE .5                            | Text size 0.5 cm
SET TXAL 10                            | Horizontal align. Left
ITX 5 1 'Left justified'
SET TXAL 20                            | Horizontal align. Center
ITX 5 2 'Centered'
SET TXAL 30                            | Horizontal align. Right
ITX 5 3 'Right justified'
SET TXAL 12                            | Vertical align. Top
ITX .2 4 'Top justified'
SET TXAL 13                            | Vertical align. Middle
ITX .2 5 'Middle justified'
SET TXAL  0                            | Default align.
SET TANG 30                            | Angle 30 degrees
ITX 5 4 '-- 30 degrees --'
```

GRAPHICS/PRIMITIVES/LABELS `labnum nlabs chlabs`

LABNUM	I	"Label identifier" D=1 R=1:9
NLABS	I	"Number of labels" D=0 R=0:50
CHLABS	C	"List of labels" D='␣' Vararg

Define a list of alphanumeric labels to be used by subsequent commands such as PIE and AXIS. The position of the labels on the axis may be changed with SET NDVX (NDVY).
Example:

```
SET * ; OPT *                  | Reset the defaults
ZONE 1 3
LABEL 1 3 AAAAA BBBBB CCCCC    | Define labels
SET NDVX 3.15                  | 3 div, lab id 1, 5=center on bin
NULL 0 10 0 1                  | Draw a frame
SET NDVX 3.11                  | 3 div, lab id 1, 1=center on tick
NULL 0 10 0 1                  | Draw a frame
SET NDVX 3.18                  | 3 div, lab id 1, 8=bottom -> up
NULL 0 10 0 1                  | Draw a frame
```

A full description of the possible alignments is given in the PAW manual (see NDVX in the index).

GRAPHICS/PRIMITIVES/PAVE `x1 x2 y1 y2 [ dz isbox isfram chopt ]`

X1	R	"X bottom left corner of box"
X2	R	"X top right corner of box"
Y1	R	"Y bottom left corner of box"
Y2	R	"Y top right corner of box"
DZ	R	"Box width" D=0.4
ISBOX	I	"Box style" D=0
ISFRAM	I	"Frame style" D=5
CHOPT	C	"Option" D='TR'

Possible CHOPT values are:

TR	Top and Right frame are drawn
TL	Top and Left frame
BR	Bottom and Right frame
BL	Bottom and Left frame
L	Left frame only
R	Right frame only
T-	Top frame only pointing left
B-	Bottom frame only pointing left
S	Shadow mode
K	Key mode

Draw a paving-block (box with 3D effect). ISBOX (ISFRAM) may be 1000+ICOLOR where ICOLOR is the color index of the box (frame), otherwise the style index. If ISBOX (ISFRAM) = 0, only the box contour is drawn with the current polyline attributes.

Example:

```
SET * ; OPT *                          | Reset the defaults
NULL 0 10 0 10                         | Draw a frame
PAVE 1 4 1 4 ! ! 1001 CHOPT=TRS
PAVE 5 9 1 4 ! ! 1001 CHOPT=BLS
PAVE 1 4 5 9 ! !    3 CHOPT=TR
PAVE 5 9 5 9 ! !    3 CHOPT=BL
```

GRAPHICS/PRIMITIVES/HIST `n x y [ chopt ]`

N	I	"Number of values"
X	C	"Vector name for X coordinates"
Y	C	"Vector name for Y coordinates"
CHOPT	C	"Options" D='AHW'

Possible CHOPT values are:

- A X and Y axes are drawn (default).
- H An histogram is drawn as a contour (default).
- W The Window/Viewport parameters are automatically computed from the X and Y values (default).
- R The histogram is Rotated, i.e. the values in X are used for the ordinate and the values in Y for the abscissa (default is the contrary). If option R is selected (and option 'N' is not selected), the user must give: 2 values for Y (Y(1)=YMIN and Y(2)=YMAX) N values for X, one for each bin. Otherwise the user must give: N values for Y, one for each bin. 2 values for X (X(1)=XMIN and X(2)=XMAX) If option 'N' is selected see below.
- N Non equidistant bins (default is equidistant). The arrays X and Y must be dimensioned as follows: If option R is not selected (default) then give: (N+1) values for X (limits of bins). N values for Y, one for each bin. Otherwise give: (N+1) values for Y (limits of bins). N values for X, one for each bin.
- F The area delimited by the histogram is filled according to the fill area interior style and the fill area style index or colour index. Contour is not drawn unless CHOPT='H' is also selected.
- C A Smooth curve is drawn across points at the centre of each bin of the histogram.
- L A straight Line is drawn across points at the centre of each bin of the histogram.
- * A star is plotted at the center of each bin of the histogram.
- P Idem as '*' but with the current marker.
- B A Bar chart with equidistant bins is drawn as fill areas. (Contours are drawn). The bar origin and the bar width can be controlled by the routine SET using the options BARO and BARW respectively.

Draw an histogram defined by arrays X and Y. The number of components needed in vectors X and/or in Y may be dependent upon the value of CHOPT (see options 'R' and 'N'). To set Log scales in X and/or Y, use OPT LOGX/LOGY. Note that when an option is specified, it is also necessary to specify the options 'W' or 'HW' in order to start a new zone or/and draw the axes.

Example

```
   SET * ; OPT *                          | Reset the defaults
   Zone 1 2
* This command needs vectors
   V/CREATE Y(10) r 1 2 3 4 5 5 4 3 2 1
   V/CREATE X(11) r 1 2 4 6 8 10 15 16 20 21 30
   HIST 10 X Y 'WH'                       | Equidistant bins
   HIST 10 X Y 'HWN'                      | Non Equidistant bins
```

GRAPHICS/PRIMITIVES/GRAPH n x y [chopt]

N	I	"Number of values"
X	C	"Vector name for X coordinates"
Y	C	"Vector name for Y coordinates"
CHOPT	C	"Options" D='ALW'

Possible CHOPT values are:

- A X and Y axes are drawn (default).
- L Every point is connected with a straight line. (default)
- W The Window/Viewport parameters are automatically computed from the X and Y values (default).
- C The values in Y are plotted in the form of a smooth curve. A Spline approximation algorithm is used.
- F A fill area is drawn. If the option 'CF' is used the contour of the fill area is smooth. The border of the fill area is drawn if the command SET BORD 1 has been typed. The fill area type may be changed via the SET parameters FASI and FASI
- R The graph is Rotated, i.e. the values in X are used for the ordinate and the values in Y for the abscissa (default is the contrary).
- B A Bar chart with equidistant bins is drawn as fill areas. (Contours are drawn). The bar origin and the bar width can be controlled by the routine SET using the options BARO and BARW respectively.
- * A star is plotted at every point.
- P A marker is plotted at every point, according to current marker type and polymarker colour index.

Draw a curve through a set of points. To set Log scales in X and/or Y, use OPT LOGX/LOGY. Note that when an option is specified, it is also necessary to specify the options 'AW' or 'ALW' in order to start a new zone or/and draw the axes.

Example

```
   SET * ; OPT *                          | Reset the defaults
   ZONE 1 2
* This command needs vectors
   V/CREATE Y(10) r 1 2 3 4 5 5 4 3 2 1
   V/CREATE X(11) r 1 2 4 6 8 10 15 16 20 21
   GRAPH 10 X Y 'WC*L'                    | Draw an "open" graph
   SET FAIS 3                             | Interior style: hatched
   SET FASI 245                           | Define hatches type
   SET BORD 1                             | Border requested
   NULL 0 22 0 6                          | define new scales
   GRAPH 10 X Y 'CF*'                     | Draw an "closed" graph
```

16.4 ATTRIBUTES

Change HIGZ attributes.

GRAPHICS/ATTRIBUTES/COLOR˙TABLE `icol [ red green blue ]`

ICOL	I	"Color Index" D=1
RED	R	"Weight of red" D=0. R=0.:1.
GREEN	R	"Weight of green" D=0. R=0.:1.
BLUE	R	"Weight of blue" D=0. R=0.:1.

Define the color ICOL.

GRAPHICS/ATTRIBUTES/PALETTE `palnb [ nel list ]`

PALNB	I	"Palette number" D=0 R=0:9
NEL	I	"Number of elements in the palette" D=0 R=0:50
LIST	I	"List of the palette elements" D=0

Define a palette of attributes. The palette number is used in the command SET. The command SET HCOL 0.1 defines the palette number 1 as colour indices used by the command LEGO in case of stacked lego plots and plotting of SURFACE with options 1 or 2, LEGO with option 2 and CONTOUR with option 3.
By default the palettes are initialized with 6 elements: 2,3,4,5,6,7.
If the number of elements (NEL) is equal to 0 (default), the palette is filled automatically according to the number of colours defined with the command SET NCOL:

```
a) If NCOL is smaller or equal to 8, the palette is filled with a
   subset of the 8 basic colours.
   Examples:
   PAW > SET NCOL 8        | Define the number of colours
   PAW > PALETTE 1         | The palette 1 is filled with
                           | 8 elements: 0,5,7,3,6,2,4,1
   PAW > SET NCOL 4        | Define the number of colours
   PAW > PALETTE 1         | The palette 1 is filled with
                           | 4 elements: 0,5,7,3

b) If NCOL is greater than 8, the palette is filled
   with colours varying continuously from blue to red. This is
   called a "geographical palette".
   Examples:
   PAW > SET NCOL 16     | Define the number of colours
   PAW > PALETTE 1         | Fill palette 1 with 8 elements
                           | (8,9,10,11,12,13,14,15) varying
                           | continuously from blue to red
   Note that after the command SET NCOL, the color indices from
   8 to NCOL are set with gray levels. The command PALETTE 1
   reset the same indices with a "geographical palette" varying
   continuously from blue to red.
```

16.5 HPLOT

Draw various HPLOT objects (symbols, errors, key, etc.).

GRAPHICS/HPLOT/SYMBOLS `x y n [ isymb ssize ]`

X	C	"Vector of X coordinates"
Y	C	"Vector of Y coordinates"
N	I	"Number of points" D=1
ISYMB	I	"Symbol number" D=24
SSIZE	R	"Symbol size" D=0.28

Draw the same symbol at several points x,y in the current normalization transformation.

GRAPHICS/HPLOT/ERRORS `x y ex ey n [ isymb ssize chopt ]`

X	C	"Vector of X coordinates"
Y	C	"Vector of Y coordinates"
EX	C	"Vector of X error bars"
EY	C	"Vector of Y error bars"
N	I	"Number of points" D=1
ISYMB	I	"Symbol number" D=24
SSIZE	R	"Symbol size" D=0.28
CHOPT	C	"Options" D='␣'

Possible CHOPT values are:

'␣'	Coordinates are expressed in histogram coordinates (of the last drawn histogram). Error bars are drawn.
C	Coordinates are expressed in centimeters.
W	A new window is defined and axis are drawn.
1	Draw small lines at the end of the error bars.
2	Draw error rectangles.
3	Draw a filled area through the end points of the vertical error bars.
4	Draw a smoothed filled area through the end points of the vertical error bars.
0	Turn off the symbols clipping.

Draw (according to the CHOPT value) a series of points using a symbol and error bars in horizontal and vertical direction in the current normalization transformation. By default, the symbols are not drawn if they are on the edges of the plot: the option '0' allows to turn off this symbols clipping. If ISYMB = 0 or SSIZE = 0. no symbol is drawn. Note that the options can be cumulated.

GRAPHICS/HPLOT/AERRORS `x y exl exu eyl eyu n [ isymb ssize chopt ]`

X	C	"Vector of X coordinates"
Y	C	"Vector of Y coordinates"
EXL	C	"Vector of X error bars (Low)"
EXU	C	"Vector of X error bars (Up)"
EYL	C	"Vector of Y error bars (Low)"

EYU	C	"Vector of Y error bars (Up)"
N	I	"Number of points" D=1
ISYMB	I	"Symbol number" D=24
SSIZE	R	"Symbol size" D=0.28
CHOPT	C	"Options" D='␣'

Possible CHOPT values are:

'␣'	Coordinates are expressed in histogram coordinates (of the last drawn histogram). Error bars are drawn.
C	Coordinates are expressed in centimeters.
W	A new window is defined and axis are drawn.
1	Draw small lines at the end of the error bars.
2	Draw error rectangles.
3	Draw a filled area through the end points of the vertical error bars.
4	Draw a smoothed filled area through the end points of the vertical error bars.
0	Turn off the symbols clipping.

Draw (according to the CHOPT value) a series of points using a symbol and asymmetric error bars in horizontal and vertical direction in the current normalization transformation. By default, the symbols are not drawn if they are on the edges of the plot: the option '0' allows to turn off this symbols clipping. If ISYMB = 0 or SSIZE = 0. no symbol is drawn. Note that the options can be cumulated.

GRAPHICS/HPLOT/KEY x y [isymb text]

X	R	"X coordinate of comment"
Y	R	"Y coordinate of comment"
ISYMB	I	"Symbol number" D=24
TEXT	C	"Legend" D='␣'

Draw one symbol and its explanation (legend) at a point x,y in the current normalization transformation.

GRAPHICS/HPLOT/TICKS [chopt xval yval]

CHOPT	C	"Options" D='␣'
XVAL	R	"X position" D=1.E30
YVAL	R	"Y position" D=1.E30

Possible CHOPT values are:

'␣'	Tick marks are drawn on the edges of the picture
X	Cross-wire drawn perpendicular to the X-axis
Y	Cross-wire drawn perpendicular to the Y-axis
A	Value drawn Above cross-wire
B	Value drawn Below cross-wire

L Value drawn Left of cross-wire
R Value drawn Right of cross-wire

Draw 'cross-wires' on a picture, optionally with tick marks and values. Cross-wires are lines perpendicular to the X and/or Y axis.

```
XVAL intersection on the X-axis
YVAL intersection on the Y-axis
```

The values of XVAL are always histogram coordinates. The tick marks will be drawn on both side of the cross wire, unless the cross-wires are requested on the boundary of the box surrounding the histogram (i.e. at the extreme limits of the drawn histogram). In this case tick marks will only be drawn inside the box. The options 'A' and 'B' (for Above and Below) refer only to the cross-wire perpendicular to the Y axis. In each case only one cross-wire will be drawn. Similarly 'L' and 'R' (Left and Right) refer only to the cross-wires perpendicular to the X-axis. It is possible to redefine the length of tick marks on the X or Y axis with SET XTIC or SET YTIC. The position of the axis values may be changed with SET XVAL or SET YVAL.

GRAPHICS/HPLOT/ATITLE `[ xtit ytit ztit ]`

`XTIT` C "X Axis title" `D='␣'`
`YTIT` C "Y Axis title" `D='␣'`
`ZTIT` C "Z Axis title" `D='␣'`

Draw axis titles on the axes of the present plot zone.

GRAPHICS/HPLOT/GRID

Draw a grid in cm.

GRAPHICS/HPLOT/NULL `[ xmin xmax ymin ymax chopt ]`

`XMIN` R "Low range in X" `D=0.`
`XMAX` R "High range in X" `D=1.`
`YMIN` R "Low range in Y" `D=0.`
`YMAX` R "High range in Y" `D=1.`
`CHOPT` C "Options" `D='␣'`

Possible `CHOPT` values are:

'␣' Draw a frame box only.
S Redefine the scale for the current zone.
A Axis labels and tick marks are not drawn.
B The box is not drawn.

Draw a frame box. If XMIN, XMAX, etc. are given, draw a frame box with the window coordinates set to XMIN, XMAX, YMIN, YMAX. Axis labels and tick marks are drawn by default.

Chapter 17: PICTURE

Creation and manipulation of HIGZ pictures.

PICTURE/FILE `lun fname [ lrecl chopt ]`

`LUN`	I	"Logical unit number" `R=1:128`
`FNAME`	C	"File name"
`LRECL`	I	"Record length in words" `D=1024`
`CHOPT`	C	"Options" `D='␣'`

Possible `CHOPT` values are:

`'␣'`	Existing file is opened.
`N`	A new file is opened.
`U`	Existing file is modified.
`A`	Automatic saving.

Open a HIGZ direct access picture file. If CHOPT='AU' or 'AN', pictures will be automatically saved on the direct access file. This automatic saving facility can be switched off using SET AURZ 0.

PICTURE/LIST

List all the HIGZ pictures currently stored in memory.

PICTURE/CREATE `pname`

`PNAME`	C	"Picture name" `Loop`

Create a new picture, named PNAME, in memory. Note that all commands which start a new picture (clear workstation) automatically create pictures named PICT1, PICT2, etc. if the command OPTION ZFL or OPTION ZFL1 has been executed.

PICTURE/DELETE `pname`

`PNAME`	C	"Picture name" `D='␣' Loop`

Delete the picture PNAME from memory. PNAME='*' means all pictures.

PICTURE/SCRATCH `pname [ icycle ]`

`PNAME`	C	"Picture name" `D='␣' Loop`
`ICYCLE`	I	"Cycle number " `D=9999`

Delete the picture PNAME from current directory on disk.

PICTURE/PLOT `[ pname ]`

`PNAME`	C	"Picture name" `D='␣' Loop`

Plot the picture PNAME. PNAME=' ' means the current picture. PNAME='*' means all pictures.

PICTURE/MODIFY `[ pname chopt ]`

`PNAME` C "Picture name" `D='␣'`
`CHOPT` C "Options" `D='␣'`

Possible `CHOPT` values are:

- `S` Software characters are used for the text in menus.
- `A` The option shadow is used.

Edit the picture PNAME. PNAME=' ' means the current picture. This command is only available on workstations.

PICTURE/MERGE `pname [ x y scale chopt ]`

`PNAME` C "Picture name"
`X` R "X coordinates (NDC) where to draw PNAME" `D=0`
`Y` R "Y coordinates (NDC) where to draw PNAME" `D=0`
`SCALE` R "Scale factor" `D=1.`
`CHOPT` C "Options" `D='␣'`

Possible `CHOPT` values are:

- `'␣'` Merge the picture PNAME with the current picture.
- `D` Picture PNAME is displayed during merging.

Add the picture PNAME to the current picture.

PICTURE/COPY `pname1 pname2`

`PNAME1` C "Picture name"
`PNAME2` C "New picture name" `Loop`

Copy a picture.

PICTURE/RENAME `pname1 pname2`

`PNAME1` C "Old picture name"
`PNAME2` C "New picture name"

Rename a picture.

PICTURE/PRINT `[ file ]`

`FILE` C "File name" `D='␣'`

Print the current picture. The current picture is transformed into a printable file. The file type is defined according to the extension of the file name i.e.

```
FILE = filename.ps    A PostScript file is generated (-111)
FILE = filename.eps   A Encapsulated PostScript file
                      is generated (-113)
FILE = filename.tex   A LaTex file is generated (-778)
```

Do HELP META for details about the metafile types. Note that a new picture is automatically created for each new plot if the OPTION ZFL1 is on.

If FILE=HIGZPRINTER or FILE=' ' the PostScript file paw.ps (-111) is generated and the operating system command defined by the environment variable HIGZPRINTER is executed.

The environment variable HIGZPRINTER should be defined as follow:

```
On UNIX sytems:
       setenv HIGZPRINTER 'lp -dprinter_name paw.ps'
  or
       export HIGZPRINTER='lp -dprinter_name paw.ps'
On VAX/VMS sytems:
       HIGZPRINTER == "XPRINT paw.ps /PRINTER=printer_name"
On CERNVM:
       setenv HIGZPRINTER 'XPRINT PAW PS (PR printer_name'
```

Note that if the environment variable HIGZPRINTER is not defined the file paw.ps is created but not printed.

PICTURE/IZOUT `[ pname ]`

`PNAME` C "Picture name" `D='␣' Loop`

Write the picture PNAME to a direct access picture file (see command PICTURE/FILE). PNAME=' ' means the current picture. PNAME='*' means all pictures.

PICTURE/IZIN `pname [ icycle ]`

`PNAME` C "Picture name" `Loop`
`ICYCLE` I "Cycle number " `D=9999`

Read picture into memory from a direct access picture file. (see command PICTURE/FILE). PNAME='*' means all pictures.

PICTURE/IZPICT `pname [ chopt ]`

`PNAME` C "Picture name"
`CHOPT` C "Options" `D='M'`

Possible `CHOPT` values are:

`M` Make a new picture in memory with name PNAME. An empty structure is created in memory and becomes the current picture. If PNAME = ' ', the picture is automatically named as PICTnnn, where the starting value of nnn is either 0 (default), or the value assigned by SET to the parameter PICT.

- `D` Display the picture PNAME in memory.
- `S` Scratch the picture PNAME from memory. If PNAME = ' ' the current picture is scratched.
- `N` The picture following the current picture in memory becomes the current picture. If the current picture is the last one in memory, the first picture in memory becomes the current picture.
- `L` Give the list of the pictures in memory, following the sequence of their storage in memory.
- `F` The First picture in memory becomes the current picture.
- `P` Print the picture data structure. Useful to debug programs.
- `C` Set Current picture. All calls to HIGZ graphic functions are stored in the current structure according to the option selected be IGZSET.

Perform various operations on a picture. PNAME=' ' means the current picture. PNAME='*' means all pictures.

PICTURE/SWITCH `[ chopt ]`

`CHOPT` C "Options" `D='G'`

Possible `CHOPT` values are:

- `G` graphics output only.
- `Z` Graphics primitives stored in ZEBRA memory only.

Set the graphics switch to control plotting output to terminal (G) and/or picture in memory (Z).

PICTURE/IGSET `[ chatt value ]`

`CHATT` C "Attribute name" `D='SHOW'`
`VALUE` R "Attribute value" `D=0.`

Set a HIGZ attribute. If CHATT='SHOW' print default and current values for all attributes. If CHATT='*' restore default values for all attributes. If VALUE=0, the attribute is set to its default value.

```
+-------------------------------------------------------------------------+
|                        IGSET : Current values in use                    |
+-------------+---------------+---------------+-------------------------+
|  Parameter  | Current value | Default value |          Explanation          |
+-------------+---------------+---------------+-------------------------+
|    FAIS     |       0       |        0      | Fill area interior style      |
|    FASI     |       1       |        1      | Fill area style index         |
|    LTYP     |       1       |        1      | Line type                     |
|    BASL     |        .150   |         .010  | Basic segment length (NDC)    |
|    LWID     |       1.000   |        1.000  | Line width                    |
|    MTYP     |       1       |        1      | Marker type                   |
|    MSCF     |       1.000   |        1.000  | Marker scale factor           |
```

PLCI	1	1	Polyline color index
PMCI	1	1	Polymarker color index
FACI	1	1	Fill area color index
TXCI	1	1	Text color index
TXAL	0 0	0 0	Text alignment
CHHE	.280	.010	Character height
TANG	.000	.000	Text angle
TXFP	0 2	0 2	Text font and precision
PICT	1	1	Current automatic number
BORD	0	0	Border flag
PASS	1	1	Number of pass in IGTEXT
CSHI	.030	.020	IGTEXT shift
LASI	.018	.018	Label axis size
LAOF	.013	.013	Label axis offset
TMSI	.019	.019	Tick marks size
AWLN	.000	.000	Axis wire lenght
BARO	.250	.250	Offset of IGHIST (IGRAPH) bars
BARW	.500	.500	Width of IGHIST (IGRAPH) bars
NCOL	8	8	Number of COLors
CLIP	1	1	Clipping mode
NLIN	40	40	Number of line for 3D shapes
AURZ	0	0	Automatic saving flag
DIME	2	2	Dimension used (2D or 3D)

Chapter 18: ZEBRA

Interfaces to the ZEBRA RZ, FZ and DZ packages.

18.1 RZ

ZEBRA/RZ package: direct access Input/Output.

ZEBRA/RZ/FILE `lun fname [ lrecl chopt ]`

`LUN`	I	"Logical unit number" `R=1:128`
`FNAME`	C	"File name"
`LRECL`	I	"Record length in WORDS" `D=1024`
`CHOPT`	C	"Options" `D='␣'`

Possible `CHOPT` values are:

`'␣'`	Read only mode.
`U`	Update mode.

Open an existing direct access file.

ZEBRA/RZ/MAKE `lun fname [ lrecl nrec nwkey chform chtags ]`

`LUN`	I	"Logical unit number" `R=1:128`
`FNAME`	C	"File name"
`LRECL`	I	"Record length in WORDS" `D=1024`
`NREC`	I	"Number of records" `D=1000`
`NWKEY`	I	"Number of words per Key" `D=1`
`CHFORM`	C	"Key format" `D='I'`
`CHTAGS`	C	"List of Tags" `D='HBOOK-ID'`

Possible `CHFORM` values are:

`I`
`B`
`A`
`H`

Open a new direct access file.

ZEBRA/RZ/MDIR `chdir [ nwkey chform chtags ]`

```
CHDIR   C  "Directory name"
NWKEY   I  "Number of words per Key"  D=1
CHFORM  C  "CHFORM"  D='I'
CHTAGS  C  "List of Tags"  D='HBOOK-ID'
```

Create a new RZ directory below the current directory.

ZEBRA/RZ/DDIR `chdir`

```
CHDIR  C  "Directory name"
```

Delete the directory CHDIR from the current directory.

ZEBRA/RZ/LDIR `[ chpath chopt ]`

```
CHPATH  C  "Path name"  D='␣'
CHOPT   C  "Options"  D='␣'
```

Possible `CHOPT` values are:

`'␣'`	List contents of a directory.
`A`	List all the Ntuple extensions.
`T`	List a directory Tree.

List contents of a directory (memory or disk). To list all RZ files currently opened, type 'LD //'. Note that if the Current Directory is //PAWC, this command uses the same format as HISTO/LIST.

ZEBRA/RZ/CDIR `[ chpath chopt ]`

```
CHPATH  C  "Path name"  D='␣'
CHOPT   C  "Options"  D='␣'
```

Change the current working directory (CWD). IF CHPATH is given make it the new CWD. Otherwise, print the pathname of the CWD.

```
Ex.  CD dir1         ; make DIR1 the new CWD
     CD //file1/dir2 ; make //FILE1/DIR2 the new CWD
     CD              ; print the name of the CWD
```

ZEBRA/RZ/PURGE `[ keep ]`

```
KEEP  I  "Number of cycles to be kept"  D=1
```

Purge an RZ directory.

ZEBRA/RZ/LOCK `[ chlock ]`

```
CHLOCK  C  "Lock identifier"  D='RZFILE'
```

Lock an RZ directory.

ZEBRA/RZ/FREE `[ chlock ]`

`CHLOCK` C "Lock identifier" `D='RZFILE'`

Free an RZ directory.

ZEBRA/RZ/STAT `chpath`

`CHPATH` C "Name of top directory"

Print space statistics for an RZ file.

18.2 FZ

ZEBRA/FZ package: sequential access Input/Output.

ZEBRA/FZ/FILE `lun fname [ lrecl chopt ]`

`LUN` I "Logical unit number" `R=1:128`
`FNAME` C "File name"
`LRECL` I "Record length in words" `D=900`
`CHOPT` C "Options" `D='IX'`

Possible `CHOPT` values are:

- `I` Input file.
- `O` Output file.
- `X` Binary exchange mode.
- `A` Alphanumeric exchange mode.

Open an FZ sequential formatted or unformatted file.

ZEBRA/FZ/TOFZ `lun [ chopt ]`

`LUN` I "Logical unit number of FZ file" `R=1:128`
`CHOPT` C "Options" `D='␣'`

Copy the current directory tree onto an FZ file.

ZEBRA/FZ/FRFZ `lun [ chopt ]`

`LUN` I "Logical unit number of FZ file" `R=1:128`
`CHOPT` C "Options" `D='␣'`

Copy the FZ file into the current directory tree.

ZEBRA/FZ/TOALPHA `fname`

`FNAME` C "Name of the FZ text file"

Copy the current directory tree onto a FZ file. An alphanumeric format is used. The file FNAME can be exchanged between different machines.

ZEBRA/FZ/FRALPHA `fname`

`FNAME` C "Name of the FZ text file"

Copy the FZ alphanumeric file into the current directory.

18.3 DZ

ZEBRA/DZ package: debugging.

ZEBRA/DZ/SHOW `name [ number chopt ]`

`NAME` C "Bank name"
`NUMBER` I "Bank number" `D=1`
`CHOPT` C "Options" `D='BSV'`

Possible `CHOPT` values are:

- `B` Print the bank.
- `S` Print the bank contents from left to right Sideways with up to ten elements per line.
- `V` Print the vertical (down) structure.
- `D` Print the bank contents from top to bottom Downwards with five elements per line.
- `L` Print the linear structure.
- `Z` Print the data part of each bank in hexadecimal format

Display the contents of a bank or a data structure identified by its NAME and NUMBER. The output format of the data part is controlled by the internal or external I/O characteristic.

ZEBRA/DZ/SURV `name [ number ]`

`NAME` C "Bank name"
`NUMBER` I "Bank number" `D=1`

Print a survey of the structure identified by NAME, NUMBER.

ZEBRA/DZ/SNAP `[ idiv chopt ]`

`IDIV` I "Division number " `D=2 R=0:24`
`CHOPT` C "Options" `D='M'`

Possible `CHOPT` values are:

- `M` Print Map entry for each bank
- `E` Extend map entry to dump all links of each bank (otherwise only as many links as will fit on a line)
- `F` Full. Dump all active banks, links and data
- `K` Kill. Dropped banks to be treated as active (dropped banks are not normally dumped under D or F option)

- `L` Dump all Link areas associated with the store
- `W` Dump the Working space, links and data
- `Z` Dump the information in hexadecimal.

Snap of one or more divisions. Provides a snapshot of one or more divisions in a ZEBRA store. The kind of information provided is controlled by CHOPT.

ZEBRA/DZ/VERIFY `[ idiv chopt ]`

```
IDIV    I   "Division number "  D=0 R=0:24
CHOPT   C   "Options"  D='CLSU'
```

Possible `CHOPT` values are:

- `C` Check chaining of banks only
- `L` Check validity of the structural links (implies 'C')
- `S` Check the store parameters
- `U` Check the validity of the up and origin (implies 'C')
- `F` Errors are considered fatal and generate a call to ZFATAL

Check the structure of one or more ZEBRA divisions. The verification detail depends on the settings in CHOPT.

ZEBRA/DZ/STORE `[ ixstor ]`

```
IXSTOR  I   "Store number"  D=0 R=0:24
```

Display the structure of the ZEBRA store IXSTOR. Output the parameters characterizing the store, followed by a list of all divisions and all link areas associated with the store in question.

Chapter 19: FORTRAN

Interface to MINUIT, COMIS, SIGMA and FORTRAN Input/Output.

FORTRAN/HMINUIT

To input commands for Interactive MINUIT in a macro. Example:

```
Application HMINUIT EXIT
SET EPS 1.E-14
MIGRAD
SET PRIN 2
MINOS
EXIT
Histo/fit 10 g m
```

FORTRAN/COMIS

Invoke the COMIS FORTRAN interpreter. COMIS allows to execute FORTRAN routines without re-compiling and relinking. It communicates with PAW commands through vectors and functions. COMIS has its PAW-independent command structure. Example in command mode:

```
PAW > Comis
CS >    do 10 i=1,10
MND>       x=sqrt(i)*10.
MND>       print *,i,x
MND> 10 continue
MND>    END
CS > quit
PAW >
```

COMIS code may be inserted into a macro. Example:

```
Vector/Create Y(10) r 1 2 3 4 5 6 7 8 9 10
*
*  In the following COMIS code, the statement "Vector Y" declares
*  to COMIS an existing KUIP vector. KUIP dimension is assumed.
*  The statement "Vector X(10)" creates a new KUIP vector.
*  (Note that SUBROUTINEs must be declared before the MAIN program)
*  (KUIP vectors cannot be created into the MAIN program)
*
APPLIcation COMIS QUIT
      SUBROUTINE DEMO
      Vector Y
      Vector X(10)
      do 10 i=1,10
         XX=i
         X(i)=Y(i)*sqrt(XX)*10.
  10  CONTINUE
```

```
      END
      CALL DEMO
      END
QUIT
Vector/print X       | Print KUIP vector created by COMIS
```

FORTRAN/CALL urout

UROUT C "User routine"

Execute the routine UROUT. UROUT may be a routine compiled and linked with PAW. For example : CALL HPRINT(10).

UROUT may also be the name of a file which can be edited interactively with the command EDIT. For example if file UROUT.FOR contains:

```
      SUBROUTINE UROUT(N)
      SUM=0.
      DO 10 I=1,N
        SUM=SUM+I
   10 CONTINUE
      PRINT *,SUM
      END
```

Then one can type CALL UROUT.FOR(10). The routine UROUT may also contain references to the library routines mentioned below.

The functions $CALL, $ICALL, and $DCALL allow to call REAL, INTEGER, and DOUBLE PRECISION functions, respectively. The function call must be enclosed in quotes, for example:

```
      $CALL('fun.f(1.5)')
```

with file fun.f containing

```
      FUNCTION FUN(X)
      FUN=X**2
      END
```

The following routines from the CERN Program Library can be called:

```
From HBOOK:
   HBOOK1,HBOOK2,HBOOKN,HFILL,HF1,HPRINT,HDELET,HRESET
   HFITGA,HFITPO,HFITEX,HPROJ1,HPROJ2,HFN,HGFIT,HRENID
   HROPEN,PAOPEN,PACLOS,PAREAD,PAWRIT,HCDIR,HGIVEN,HKIND
   HTITLE,HBFUN1,HBFUN2,HRNDM1,HRNDM2,HBARX,HBARY,HDIFFB
   HPAK,HPAKE,HUNPAK,HGIVE,HGN,HGNF,HGNPAR,HF2,HFF1,HFF2
   HRIN,HROUT,HI,HIE,HIX,HIJ,HIF,HIDALL,HNOENT,HX,HXY
   HTITLE,HCOPY,HSTATI,HBPROF,HOPERA,HIDOPT,HDERIV,HBAR2
   HMAXIM,HMINIM,HMAX,HMIN,HSUM,HNORMA,HMCINI,HMCMLL
   HEXIST,HREND,HRGET,HRPUT,HSCR,HFIND,HCX,HCXY,HLABEL
```

```
   HBPROX,HBPROY,HBANDX,HBANDY,HBSLIX,HBSLIY,HPROF2
   HBOOKB,HBSTAT,HDIFF,HUNPKE,HREBIN,HERROR,HGNTB,HSTAF
   HOUTPU,HERMES,HISTDO,HFUNC,HXI,HIJXY,HXYIJ,HLPOS,HFC1
   HSPLI1,HSPLI2,HMDIR,HLDIR,HLOCAT,HFITH,HFITV,HFINAM
   HBNT,HBNAME,HBNAMC,HFNT,HFNTB,HGNT,HGNTF,HGNTV,HBSET
   HRENAME,HNTDUP
From HPLOT:
   HPLOT,HPLSYM,HPLERR,HPLEGO,HPLNT,HPLSUR,HPLSOF,HPLFRA
   HPLABL,HPLSET,HPLGIV,HPLOC,HPLTOC,HPLNEW,HPLOPT
From ZEBRA:
   MZSTOR,MZDIV,MZLINK,MZWORK,MZBOOK,MZDROP,MZPUSH
   MZWIPE,MZGARB,MZFORM,LZFIND,LZFID,DZSHOW,DZVERI
   FZIN,FZOUT,FZFILE,FZENDI,FZENDO
   RZCDIR,RZLDIR,RZFILE,RZEND,RZIN,RZOUT,RZVIN,RZVOUT
   RZOPEN,RZIODO,RZCLOS,RZQUOT
From KUIP:
   KUGETV,KUDPAR,KUVECT,KILEXP,KUTIME,KUEXEL,KUPROS
   KUNWG,KUCMD,KUGUID,KUNDPV,KUPAR,KUPVAL,KUACT
From HIGZ:
   IPL,IPM,IFA,IGTEXT,IGBOX,IGAXIS,IGPIE,IGRAPH,IGHIST
   IGARC,IGLBL,IGRNG,IGMETA,IGSA,IGSET,IRQLC,IRQST,ISCR
   ISELNT,ISFAIS,ISFASI,ISLN,ISMK,ISVP,ISWN,ITX,ICLRWK
   IGPAVE,IGTERM,ISFACI,IGHTOR,IGONT
From KERNLIB:
   VZERO,UCOPY,RANNOR,LENOCC,SBITO,SBIT1,SBYT
   JBIT,JBYT,UCTOH,UHTOC,CLTOU,CUTOL,ERF,ERFC,FREQ,GAMMA
   PROB,DENLAN,DSTLAN,DIFLAN,XM1LAN,XM2LAN,RANLAN
   RNDM,RDMIN,RDMOUT,SORTZV,CSF77

The following common blocks may be referenced:
  /PAWC/, /QUEST/, /KCWORK/, /PAWPAR/, /PAWIDN/
  /HCFITS/, /HCFITD/, /RZCLUN/
```

FORTRAN/LOOP `ntimes urout`

NTIMES	I	"Number of calls" D=1
UROUT	C	"User routine"

The routine UROUT is called NTIMES times. See command CALL for explanation of UROUT.

FORTRAN/FILE `lun fname [ status ]`

LUN	I	"Logical unit number"
FNAME	C	"File name"
STATUS	C	"File status" D='DONTKNOW'

Possible STATUS values are:

OLD	Open existing file for reading.
APPEND	Open existing file and position at EOF.
NEW	Create new file; error if already existing.
UNKNOWN	Open existing or create new file.
DONTKNOW	Like UNKNOWN except on VMS opens highest cycle.

Open a FORTRAN formatted text file. UNKNOWN opens a file for write access without flagging an error if the file already exists. On VMS a new cycle is created. DONTKNOW is the same as UNKNOWN except on VMS where the highest cycle is opened. This option should be used if it is not yet known whether the file will be read or written.

FORTRAN/CLOSE lun

LUN I "Logical unit number" R=0:128

Close the file on unit LUN. If the file has been opened with HISTO/FILE, PICTURE/FILE, etc, then before closing the unit, PAW will close correctly the file with CALL HREND or FZENDI(O), ICLWK, etc. Giving 0 as unit will close all open files.

FORTRAN/REWIND lun

LUN I "Logical unit number" R=1:128

Rewind the file on unit LUN.

FORTRAN/SIGMA [expr]

EXPR C "Expression" D='␣'

Invoke the SIGMA package. SIGMA is an array manipulation package using its own vector-oriented language, outside the PAW command conventions. SIGMA may be invoked in one of the three following ways:

```
1- Using the KUIP $SIGMA function. Example:
   PAW > Vector/Create x(10) r 1 2 3 4 5 6 7 8 9 10
   PAW > Graph 10 x $sigma(sqrt(x))

2- Using the SIGMA command. Example:
   PAW > sigma x=array(10,1#10)
   PAW > sigma y=sqrt(x)
   PAW > Graph 10 x y

3- Using the APPLication command. Example:
   PAW > APPLication SIGMA
   SIGMA > x=array(10,1#10)
   SIGMA > y=sqrt(x)
   SIGMA > exit
   PAW > Graph 10 x y
```

Chapter 20: NETWORK

To access files on remote computers. To send messages to a remote process (ZEBRA server)

NETWORK/RLOGIN `host`

`HOST` C "Host name" `D='␣'`

Start a communication with a remote machine HOST. Current Directory will be changed to //HOST.

NETWORK/RSHELL `message`

`MESSAGE` C "Message to remote host" `D='␣'`

Send MESSAGE to current remote host. Note that the Current Directory must be //HOST (see RLOGIN). Some PAW commands (Histo/Plot, Histo/List) can communicate directly with HOST.

20.1 PIAF

To establish and control the connection to the Piaf server. The Parallel Interactive Analysis Facility (Piaf) is a cluster of 5 high-performance HP workstations.

A locally running PAW session (client) connected to the Piaf server can access Hbook RZ files stored on the server side in a transparent way. Commands with high CPU or I/O requirements, e.g. NT/PLOT and NT/PROJECT are processed by the server and only the resulting histograms etc. are sent back to the client.

In order to use the Piaf server the PAW client must have been compiled with the communications option CZ using TCP/IP as transport protocol.

NETWORK/PIAF/CONNECT `[ server node ]`

`SERVER` C "Server name" `D='piaf'`
`NODE` C "Front-end node" `D='128.141.201.28'`

Establish a connection to the Piaf server. Subsequent HISTO/FILE commands can refer to files on the server using path names '//piaf/file.hbook'.

NETWORK/PIAF/STAGE `source [ target option ]`

`SOURCE` C "Source file identifier"
`TARGET` C "Target file name" `D='␣'`
`OPTION` C "Options" `D='␣'`

Possible `OPTION` values are:

`N` NoWait. Submit the request to the staging system and return immediately.

Stage an Ntuple file on the Piaf server. The source file identifier can be the name of a local file on the client system, a Fatmen path, or a tape identifier. If the target file name is not specified it is constructed from the source identifier.

Unless the option N is used the STAGE command waits until the staging is completed and the file is ready to be used.

NETWORK/PIAF/GET `remote [ local format recl ]`

`REMOTE`	C	"Remote file name"
`LOCAL`	C	"Local file name" `D='␣'`
`FORMAT`	C	"Text or binary" `D='RZ'`
`RECL`	I	"Record length in bytes" `D=0 R=0:`

Possible `FORMAT` values are:

`T`	Text file.
`RZ`	Zebra RZ file in exchange format.
`BIN`	Binary file with record length given by RECL.

Copy a file from the Piaf server to the client system. If not specified the local file name will be same as the remote file name. RECL needs to be specified only for BIN format. For IBM only: A text file with RECL=0 is written in V-format. Otherwise it is written in F-format with the given LRECL.

NETWORK/PIAF/PUT `local [ remote format ]`

`LOCAL`	C	"Local file name"
`REMOTE`	C	"Remote file name" `D='␣'`
`FORMAT`	C	"Text or binary" `D='RZ'`

Possible `FORMAT` values are:

`T`	Text file.
`RZ`	Zebra RZ file in exchange format.
`BIN`	Binary file.

Copy a file from the client system to the Piaf server. If not specified the remote file name will be same as the local file name. Note for VMS: Avoid text files with variable record length. Use Stream˙LF format instead.

NETWORK/PIAF/LS `[ files ]`

`FILES`	C	"File pattern" `D='␣'`

List files stored on the Piaf server.

NETWORK/PIAF/CAT `file`

`FILE`	C	"File name"

Print a Piaf file on the terminal.

NETWORK/PIAF/RM `file`

`FILE` C "File name"

Delete a Piaf file.

NETWORK/PIAF/MV `from to`

`FROM` C "Old file name"
`TO` C "New file name"

Rename a Piaf file.

NETWORK/PIAF/CP `from to`

`FROM` C "Old file name"
`TO` C "New file name"

Copy a Piaf file to a new file.

NETWORK/PIAF/PWD

Show current Piaf working directory.

NETWORK/PIAF/MKDIR `dir`

`DIR` C "Directory name"

Create a new directory on Piaf.

NETWORK/PIAF/RMDIR `dir`

`DIR` C "Directory name"

Delete a directory on Piaf.

NETWORK/PIAF/MESSAGE `mess`

`MESS` C "Message"

Send a message to Piaf.

NETWORK/PIAF/STATUS

Inquire the status of the Piaf server.

NETWORK/PIAF/MODE `[ option ]`

`OPTION` C "Processing mode" `D='?'`

Possible `OPTION` values are:

- `?` Inquire the current mode.
- `SEQ` Set sequential processing mode.
- `PAR` Set parallel processing mode.

Inquire or change the processing mode of the Piaf server. In parallel mode the Piaf server uses slave servers to process Ntuple requests on all available machines in parallel.

With certain types of COMIS selection functions, e.g. when reading from an external file for each event, parallel processing is not possible. The Piaf server should be switched to sequential mode, i.e. the master server alone processes the Ntuple request.

NETWORK/PIAF/LOGLEVEL `level`

`LEVEL` I "Log level" `D=0`

Set the level of diagnostic output from the Piaf server.

NETWORK/PIAF/DISCONNECT

Close the connection to the Piaf server.

Chapter 21: OBSOLETE

Obsolete commands

21.1 GRAPHICS

21.1.1 ATTRIBUTES

OBSOLETE/GRAPHICS/ATTRIBUTES/SMK `[ mkt ]`

`MKT` I "Marker type" `D=1`

Set the marker type. Obsolete command use SET MTYP

OBSOLETE/GRAPHICS/ATTRIBUTES/SLN `[ iln ]`

`ILN` I "Line style" `D=1 R=1:`

Set the line style. Obsolete command use SET LTYP.

OBSOLETE/GRAPHICS/ATTRIBUTES/SFAIS `[ ints ]`

`INTS` I "Fill area interior style" `D=0 R=0:3`

Set the fill area interior style. Obsolete command use SET FAIS.

OBSOLETE/GRAPHICS/ATTRIBUTES/SFASI `[ styli ]`

`STYLI` I "Fill area style index" `D=1`

Set the fill area style index. Obsolete command use SET FASI.

OBSOLETE/GRAPHICS/ATTRIBUTES/SFACI `[ ifaci ]`

`IFACI` I "Fill area color index" `D=1`

Set the fill area color index. Obsolete command use SET FACI.

OBSOLETE/GRAPHICS/ATTRIBUTES/SPLCI `[ iplci ]`

`IPLCI` I "Polyline color index" `D=1`

Set the polyline color index. Obsolete command use SET PLCI.

OBSOLETE/GRAPHICS/ATTRIBUTES/SPMCI `[ ipmci ]`

`IPMCI` I "Polymarker color index" `D=1`

Set the polymarker color index. Obsolete command use SET PMCI.

OBSOLETE/GRAPHICS/ATTRIBUTES/STXCI `[ itxci ]`

`ITXCI` I "Text color index" `D=1`

Set the text color index. Obsolete command use SET TXCI.

OBSOLETE/GRAPHICS/ATTRIBUTES/STXFP `[ ifont iprec ]`

`IFONT` I "Font number" `D=0`
`IPREC` I "Font precision" `D=2`

Set text font and precision. Obsolete command use SET TXFP.

OBSOLETE/GRAPHICS/ATTRIBUTES/SCHH `[ chh ]`

`CHH` R "Character height" `D=0.28`

Set the character height. Obsolete command use SET CHHE.

OBSOLETE/GRAPHICS/ATTRIBUTES/SLWSC `[ lw ]`

`LW` R "Line width" `D=1 R=1:`

Set the line width scale factor. Obsolete command use SET LWID.

Appendix A: PAW tabular overview

Table A.1: Alphabetical list of PAW commands

Calling sequence	Page
1DHISTO (HISTOGRAM/CREATE/1DHISTO) id title ncx xmin xmax [valmax]	375
2DHISTO (HISTOGRAM/CREATE/2DHISTO) id title ncx xmin xmax ncy ymin ymax [valmax]	376
ABSCISSA (HISTOGRAM/GET_VECT/ABSCISSA) id vname	386
ADD (HISTOGRAM/OPERATIONS/ADD) id1 id2 id3 [c1 c2 option]	380
AERRORS (GRAPHICS/HPLOT/AERRORS) x y exl exu eyl eyu n [isymb ssize chopt]	431
ANGLE (FUNCTION/ANGLE) [theta phi]	392
APPLICATION (KUIP/SET_SHOW/APPLICATION) path [cmdex]	340
ARC (GRAPHICS/PRIMITIVES/ARC) x1 y1 r1 [r2 phimin phimax]	423
ARCHELIX (GRAPHICS/PRIMITIVES/ARCHELIX) [x1 y1 x2 y2 r wi phi rl]	420
ARLINE (GRAPHICS/PRIMITIVES/ARLINE) [x1 y1 x2 y2 h]	420
ARROW (GRAPHICS/PRIMITIVES/ARROW) x1 x2 y1 y2 [size]	418
ATITLE (GRAPHICS/HPLOT/ATITLE) [xtit ytit ztit]	433
AXIS (GRAPHICS/PRIMITIVES/AXIS) x0 x1 y0 y1 wmin wmax ndiv [chopt]	421
Arithmetic (MACRO/SYNTAX/Expressions/Arithmetic)	351
BANX (HISTOGRAM/CREATE/BANX) id ymin ymax	378
BANY (HISTOGRAM/CREATE/BANY) id xmin xmax	378
BINS (HISTOGRAM/CREATE/BINS) id title ncx xbins [valmax]	376
BOX (GRAPHICS/PRIMITIVES/BOX) x1 x2 y1 y2	417
BREAK (KUIP/SET_SHOW/BREAK) [option]	341
BREAKL (MACRO/SYNTAX/Looping/BREAKL)	359
BUGREPORT (KUIP/BUGREPORT) [chopt]	336
Boolean (MACRO/SYNTAX/Expressions/Boolean)	352
CALL (FORTRAN/CALL) urout	445
CASE (MACRO/SYNTAX/Branching/CASE)	356
CAT (NETWORK/PIAF/CAT) file	449
CDIR (ZEBRA/RZ/CDIR) [chpath chopt]	440
CHAIN (NTUPLE/CHAIN) [cname entry]	400
CLOSE (FORTRAN/CLOSE) lun	447
CLR (GRAPHICS/MISC/CLR)	412
COLOR_TABLE (GRAPHICS/ATTRIBUTES/COLOR_TABLE) icol [red green blue]	430
COLUMNS (KUIP/SET_SHOW/COLUMNS) [ncol]	342
COMIS (FORTRAN/COMIS)	444
COMMAND (KUIP/SET_SHOW/COMMAND) [chpath]	340
CONNECT (NETWORK/PIAF/CONNECT) [server node]	448
CONTENTS (HISTOGRAM/GET_VECT/CONTENTS) id vname	385
CONTENTS (HISTOGRAM/PUT_VECT/CONTENTS) id vname	386
CONTOUR (HISTOGRAM/2D_PLOT/CONTOUR) [id nlevel chopt param]	375
COPY (HISTOGRAM/COPY) id1 id2 [title]	371
COPY (PICTURE/COPY) pname1 pname2	435
COPY (VECTOR/COPY) vnam1 vnam2	361
CP (NETWORK/PIAF/CP) from to	450
CREATE (KUIP/ALIAS/CREATE) name value [chopt]	336
CREATE (MACRO/GLOBAL/CREATE) name [value text]	350
CREATE (NTUPLE/CREATE) idn title nvar chrzpa nprime varlist	393
CREATE (PICTURE/CREATE) pname	434
CREATE (VECTOR/CREATE) vname [type values]	360
CSELECT (NTUPLE/CSELECT) [chopt csize]	402
CUTS (NTUPLE/CUTS) cutid [option fname wkid]	401
DATA (MACRO/DATA)	350
DDIR (ZEBRA/RZ/DDIR) chdir	440

Table A.1: Overview of PAW command sequences (continued)

Calling sequence	Page
DEFAULTS (MACRO/DEFAULTS) [path option]	349
DELETE (HISTOGRAM/DELETE) id	368
DELETE (KUIP/ALIAS/DELETE) name	337
DELETE (MACRO/GLOBAL/DELETE) name	350
DELETE (PICTURE/DELETE) pname	434
DELETE (VECTOR/DELETE) vlist	361
DIFF (HISTOGRAM/OPERATIONS/DIFF) id1 id2 [chopt]	382
DISCONNECT (NETWORK/PIAF/DISCONNECT)	451
DIVIDE (HISTOGRAM/OPERATIONS/DIVIDE) id1 id2 id3 [c1 c2 option]	381
DLINE (GRAPHICS/PRIMITIVES/DLINE) x1 x2 y1 y2	416
DO (MACRO/SYNTAX/Looping/DO)	358
DOLLAR (KUIP/SET_SHOW/DOLLAR) [option]	346
DRAW (FUNCTION/DRAW) ufunc [chopt]	390
DRAW (NTUPLE/DRAW) idn [value option]	401
DRAW (VECTOR/DRAW) vname [id chopt]	363
DUMP (HISTOGRAM/HIO/DUMP) id	380
DUPLICATE (NTUPLE/DUPLICATE) id1 id2 [newbuf title option]	394
EDIT (KUIP/EDIT) fname	332
ENDKUMAC (MACRO/SYNTAX/Definitions/ENDKUMAC)	355
ERRORS (GRAPHICS/HPLOT/ERRORS) x y ex ey n [isymb ssize chopt]	431
ERRORS (HISTOGRAM/GET_VECT/ERRORS) id vname	385
ERRORS (HISTOGRAM/PUT_VECT/ERRORS) id vname	387
EXEC (MACRO/EXEC) mname [margs]	348
EXIT (KUIP/EXIT)	333
EXITM (MACRO/SYNTAX/Definitions/EXITM)	355
FAREA (GRAPHICS/PRIMITIVES/FAREA) n x y	416
FBOX (GRAPHICS/PRIMITIVES/FBOX) x1 x2 y1 y2 x3 x4 y3 y4	418
FILE (FORTRAN/FILE) lun fname [status]	446
FILE (HISTOGRAM/FILE) lun fname [lrecl chopt]	368
FILE (PICTURE/FILE) lun fname [lrecl chopt]	434
FILE (ZEBRA/FZ/FILE) lun fname [lrecl chopt]	441
FILE (ZEBRA/RZ/FILE) lun fname [lrecl chopt]	439
FILECASE (KUIP/SET_SHOW/FILECASE) [option]	346
FIT (HISTOGRAM/FIT) id func [chopt np par step pmin pmax errpar]	371
FIT (VECTOR/FIT) x y ey func [chopt np par step pmin pmax errpar]	365
FOR (MACRO/SYNTAX/Looping/FOR)	358
FPOINT (GRAPHICS/PRIMITIVES/FPOINT) [x y r]	421
FRALPHA (ZEBRA/FZ/FRALPHA) fname	442
FREE (ZEBRA/RZ/FREE) [chlock]	441
FRFZ (ZEBRA/FZ/FRFZ) lun [chopt]	441
FUN1 (FUNCTION/FUN1) id ufunc ncx xmin xmax [chopt]	389
FUN2 (FUNCTION/FUN2) id ufunc ncx xmin xmax ncy ymin ymax [chopt]	389
FUNCTION (HISTOGRAM/GET_VECT/FUNCTION) id vname	385
FUNCTIONS (KUIP/FUNCTIONS)	333
GET (NETWORK/PIAF/GET) remote [local format recl]	449
GLOBAL_SECT (HISTOGRAM/HIO/GLOBAL_SECT) gname	380
GOTO_and_IF_GOTO (MACRO/SYNTAX/Branching/GOTO_and_IF_GOTO)	356
GRAPH (GRAPHICS/PRIMITIVES/GRAPH) n x y [chopt]	428
GRESET (HISTOGRAM/HIO/GRESET) id	380
GRID (GRAPHICS/HPLOT/GRID)	433
Garbage (MACRO/SYNTAX/Expressions/Garbage)	352
Global (MACRO/SYNTAX/Variables/Global)	354
HELIX (GRAPHICS/PRIMITIVES/HELIX) [x1 y1 x2 y2 r wi phi]	419

Table A.1: Overview of PAW command sequences (continued)

Calling sequence	Page
HELP (KUIP/HELP) [item option]	331
HFETCH (HISTOGRAM/HIO/HFETCH) id fname	379
HFILL (VECTOR/HFILL) vname id	364
HIST (GRAPHICS/PRIMITIVES/HIST) n x y [chopt]	427
HMERGE (NTUPLE/HMERGE) outfile infiles	394
HMINUIT (FORTRAN/HMINUIT)	444
HMOVE (GRAPHICS/MISC/HMOVE)	413
HOST_EDITOR (KUIP/SET_SHOW/HOST_EDITOR) [editor top left width height dxpad dypad npads]	342
HOST_PAGER (KUIP/SET_SHOW/HOST_PAGER) [pager]	343
HOST_PRINTER (KUIP/SET_SHOW/HOST_PRINTER) [command filetype]	343
HOST_PSVIEWER (KUIP/SET_SHOW/HOST_PSVIEWER) [psviewer]	344
HOST_SHELL (KUIP/SET_SHOW/HOST_SHELL) [shell]	344
HREAD (HISTOGRAM/HIO/HREAD) id fname	379
HRIN (HISTOGRAM/HIO/HRIN) id [icycle iofset]	378
HROUT (HISTOGRAM/HIO/HROUT) id [chopt]	379
HSCRATCH (HISTOGRAM/HIO/HSCRATCH) id	379
HSETPR (HISTOGRAM/OPERATIONS/HSETPR) param value	385
IDLE (KUIP/IDLE) sec [string]	333
IDOPT (HISTOGRAM/SET/IDOPT) id option	387
IF_THEN (MACRO/SYNTAX/Branching/IF_THEN)	356
IGSET (PICTURE/IGSET) [chatt value]	437
IMPORT (MACRO/GLOBAL/IMPORT) name	350
INPUT (VECTOR/INPUT) vname [values]	362
ITX (GRAPHICS/PRIMITIVES/ITX) x y text	426
IZIN (PICTURE/IZIN) pname [icycle]	436
IZOUT (PICTURE/IZOUT) [pname]	436
IZPICT (PICTURE/IZPICT) pname [chopt]	436
Indirection (MACRO/SYNTAX/Variables/Indirection)	354
KEY (GRAPHICS/HPLOT/KEY) x y [isymb text]	432
LABELS (GRAPHICS/PRIMITIVES/LABELS) labnum nlabs chlabs	426
LAST (KUIP/LAST) [n fname]	332
LCDIR (KUIP/SET_SHOW/LCDIR) [directory]	347
LDIR (ZEBRA/RZ/LDIR) [chpath chopt]	440
LEGO (HISTOGRAM/2D_PLOT/LEGO) [id theta phi chopt]	373
LINE (GRAPHICS/PRIMITIVES/LINE) x1 y1 x2 y2	416
LINTRA (NTUPLE/LINTRA) idn [chopt nevent ifirst nvars varlis]	404
LIST (HISTOGRAM/LIST) [chopt]	368
LIST (KUIP/ALIAS/LIST) [name]	337
LIST (MACRO/GLOBAL/LIST) [name file]	351
LIST (MACRO/LIST) [mname]	348
LIST (NTUPLE/LIST)	394
LIST (PICTURE/LIST)	434
LIST (VECTOR/LIST)	361
LOCATE (GRAPHICS/MISC/LOCATE) [ntpri chopt wkid]	412
LOCK (ZEBRA/RZ/LOCK) [chlock]	440
LOGLEVEL (NETWORK/PIAF/LOGLEVEL) level	451
LOOP (FORTRAN/LOOP) ntimes urout	446
LOOP (NTUPLE/LOOP) idn uwfunc [nevent ifirst]	396
LS (NETWORK/PIAF/LS) [files]	449
MACRO (MACRO/SYNTAX/Definitions/MACRO)	355
MAKE (ZEBRA/RZ/MAKE) lun fname [lrecl nrec nwkey chform chtags]	439
MANUAL (KUIP/MANUAL) item [output option]	331
MANY_PLOTS (HISTOGRAM/MANY_PLOTS) idlist	371

Table A.1: Overview of PAW command sequences (continued)

Calling sequence	Page
MASK (NTUPLE/MASK) mname [chopt number]	402
MAXIMUM (HISTOGRAM/SET/MAXIMUM) id vmax	387
MDIR (ZEBRA/RZ/MDIR) chdir [nwkey chform chtags]	439
MERGE (NTUPLE/MERGE) idn1 idn2 [uwfunc nevent ifirst]	396
MERGE (PICTURE/MERGE) pname [x y scale chopt]	435
MESSAGE (KUIP/MESSAGE) [string]	333
MESSAGE (NETWORK/PIAF/MESSAGE) mess	450
METAFILE (GRAPHICS/METAFILE) [lun metafl chmeta]	410
MINIMUM (HISTOGRAM/SET/MINIMUM) id vmin	387
MKDIR (NETWORK/PIAF/MKDIR) dir	450
MODE (NETWORK/PIAF/MODE) [option]	450
MODIFY (PICTURE/MODIFY) [pname chopt]	435
MULTIPLY (HISTOGRAM/OPERATIONS/MULTIPLY) id1 id2 id3 [c1 c2 option]	381
MV (NETWORK/PIAF/MV) from to	450
NEWPANEL (KUIP/SET_SHOW/NEWPANEL) line col title width height xpos ypos	340
NEXT (GRAPHICS/MISC/NEXT)	412
NEXTL (MACRO/SYNTAX/Looping/NEXTL)	359
NORMALIZE_FACTOR (HISTOGRAM/SET/NORMALIZE_FACTOR) id [xnorm]	387
NULL (GRAPHICS/HPLOT/NULL) [xmin xmax ymin ymax chopt]	433
Numbered (MACRO/SYNTAX/Variables/Numbered)	353
ON_ERROR (MACRO/SYNTAX/Branching/ON_ERROR)	357
OPTION (GRAPHICS/OPTION) [choptn]	408
OUTPUT_LP (HISTOGRAM/HIO/OUTPUT_LP) [lun fname]	380
PALETTE (GRAPHICS/ATTRIBUTES/PALETTE) palnb [nel list]	430
PANEL (KUIP/SET_SHOW/PANEL) line [gkey]	338
PARAM (HISTOGRAM/OPERATIONS/PARAM) id [isel r2min maxpow]	384
PAVE (GRAPHICS/PRIMITIVES/PAVE) x1 x2 y1 y2 [dz isbox isfram chopt]	427
PIE (GRAPHICS/PRIMITIVES/PIE) x0 y0 radius n values [chopt iao ias iac]	424
PLINE (GRAPHICS/PRIMITIVES/PLINE) n x y	415
PLOT (FUNCTION/PLOT) ufunc xlow xup [chopt]	391
PLOT (HISTOGRAM/PLOT) [id chopt]	368
PLOT (NTUPLE/PLOT) idn [uwfunc nevent ifirst nupd option idh]	397
PLOT (PICTURE/PLOT) [pname]	434
PLOT (VECTOR/PLOT) vname [id chopt]	364
PMARKER (GRAPHICS/PRIMITIVES/PMARKER) n x y	417
POINTS (FUNCTION/POINTS) [npx npy npz]	391
PRINT (HISTOGRAM/HIO/PRINT) id [chopt]	380
PRINT (KUIP/PRINT) fname	332
PRINT (NTUPLE/PRINT) idn	394
PRINT (PICTURE/PRINT) [file]	435
PRINT (VECTOR/PRINT) vname [dense]	362
PROFILE (HISTOGRAM/CREATE/PROFILE) id title ncx xmin xmax ymin ymax [chopt]	376
PROJECT (HISTOGRAM/PROJECT) id	371
PROJECT (NTUPLE/PROJECT) idh idn [uwfunc nevent ifirst]	397
PROMPT (KUIP/SET_SHOW/PROMPT) prompt	341
PROX (HISTOGRAM/CREATE/PROX) id	377
PROY (HISTOGRAM/CREATE/PROY) id	377
PSVIEW (KUIP/PSVIEW) fname	332
PURGE (ZEBRA/RZ/PURGE) [keep]	440
PUT (NETWORK/PIAF/PUT) local [remote format]	449
PWD (NETWORK/PIAF/PWD)	450
QUIT (KUIP/QUIT)	333
RANGE (FUNCTION/RANGE) [xlow xup ylow yup zlow zup]	391

Table A.1: Overview of PAW command sequences (continued)

Calling sequence	Page
READ (MACRO/SYNTAX/Variables/READ)	354
READ (NTUPLE/READ) idn fname [format chopt nevent]	397
READ (VECTOR/READ) vlist fname [format opt match]	362
REBIN (HISTOGRAM/GET_VECT/REBIN) id x y ex ey [n ifirst ilast chopt]	386
RECALL_STYLE (KUIP/SET_SHOW/RECALL_STYLE) [option]	344
RECORDING (KUIP/SET_SHOW/RECORDING) [nrec]	342
RECOVER (NTUPLE/RECOVER) idn	395
RENAME (PICTURE/RENAME) pname1 pname2	435
REPEAT (MACRO/SYNTAX/Looping/REPEAT)	358
RESET (HISTOGRAM/OPERATIONS/RESET) id [title]	382
RETURN (MACRO/SYNTAX/Definitions/RETURN)	355
REWIND (FORTRAN/REWIND) lun	447
RLOGIN (NETWORK/RLOGIN) host	448
RM (NETWORK/PIAF/RM) file	450
RMDIR (NETWORK/PIAF/RMDIR) dir	450
ROOT (KUIP/SET_SHOW/ROOT) [path]	341
RSHELL (NETWORK/RSHELL) message	448
SCALE_FACTOR_2D (HISTOGRAM/SET/SCALE_FACTOR_2D) id [xscale]	387
SCAN (NTUPLE/SCAN) idn [uwfunc nevent ifirst option varlis]	395
SCHH (OBSOLETE/GRAPHICS/ATTRIBUTES/SCHH) [chh]	453
SCRATCH (PICTURE/SCRATCH) pname [icycle]	434
SELNT (GRAPHICS/VIEWING/SELNT) nt	415
SET (GRAPHICS/SET) [chatt value]	406
SFACI (OBSOLETE/GRAPHICS/ATTRIBUTES/SFACI) [ifaci]	452
SFAIS (OBSOLETE/GRAPHICS/ATTRIBUTES/SFAIS) [ints]	452
SFASI (OBSOLETE/GRAPHICS/ATTRIBUTES/SFASI) [styli]	452
SHELL (KUIP/SHELL) [cmd]	333
SHIFT (MACRO/SYNTAX/Variables/SHIFT)	354
SHOW (ZEBRA/DZ/SHOW) name [number chopt]	442
SIGMA (FORTRAN/SIGMA) [expr]	447
SIZE (GRAPHICS/VIEWING/SIZE) [xsize ysize]	414
SLIDE (GRAPHICS/SLIDE)	412
SLIX (HISTOGRAM/CREATE/SLIX) id nslices	377
SLIY (HISTOGRAM/CREATE/SLIY) id nslices	377
SLN (OBSOLETE/GRAPHICS/ATTRIBUTES/SLN) [iln]	452
SLWSC (OBSOLETE/GRAPHICS/ATTRIBUTES/SLWSC) [lw]	453
SMK (OBSOLETE/GRAPHICS/ATTRIBUTES/SMK) [mkt]	452
SMOOTH (HISTOGRAM/OPERATIONS/SMOOTH) id [option sensit smooth]	383
SNAP (ZEBRA/DZ/SNAP) [idiv chopt]	442
SORT (HISTOGRAM/OPERATIONS/SORT) id [chopt]	383
SPLCI (OBSOLETE/GRAPHICS/ATTRIBUTES/SPLCI) [iplci]	452
SPLINE (HISTOGRAM/OPERATIONS/SPLINE) id [isel knotx kx]	384
SPMCI (OBSOLETE/GRAPHICS/ATTRIBUTES/SPMCI) [ipmci]	452
STAGE (NETWORK/PIAF/STAGE) source [target option]	448
STAT (ZEBRA/RZ/STAT) chpath	441
STATUS (NETWORK/PIAF/STATUS)	450
STOPM (MACRO/SYNTAX/Definitions/STOPM)	355
STORE (ZEBRA/DZ/STORE) [ixstor]	443
STXCI (OBSOLETE/GRAPHICS/ATTRIBUTES/STXCI) [itxci]	452
STXFP (OBSOLETE/GRAPHICS/ATTRIBUTES/STXFP) [ifont iprec]	453
STYLE (KUIP/SET_SHOW/STYLE) [option sgylen sgsize sgyspa sgbord wktype]	338
SUBTRACT (HISTOGRAM/OPERATIONS/SUBTRACT) id1 id2 id3 [c1 c2 option]	381
SURFACE (HISTOGRAM/2D_PLOT/SURFACE) [id theta phi chopt]	374

Table A.1: Overview of PAW command sequences (continued)

Calling sequence	Page
SURV (ZEBRA/DZ/SURV) name [number]	442
SVP (GRAPHICS/VIEWING/SVP) nt x1 x2 y1 y2	414
SWITCH (PICTURE/SWITCH) [chopt]	437
SWN (GRAPHICS/VIEWING/SWN) nt x1 x2 y1 y2	414
SYMBOLS (GRAPHICS/HPLOT/SYMBOLS) x y n [isymb ssize]	431
Special (MACRO/SYNTAX/Variables/Special)	353
String (MACRO/SYNTAX/Expressions/String)	352
TEXT (GRAPHICS/PRIMITIVES/TEXT) x y text size [angle chopt]	425
TICKS (GRAPHICS/HPLOT/TICKS) [chopt xval yval]	432
TIMING (KUIP/SET_SHOW/TIMING) [option]	341
TITLE_GLOBAL (HISTOGRAM/CREATE/TITLE_GLOBAL) [chtitl chopt]	378
TOALPHA (ZEBRA/FZ/TOALPHA) fname	441
TOFZ (ZEBRA/FZ/TOFZ) lun [chopt]	441
TRACE (MACRO/TRACE) [option level]	348
TRANSLATION (KUIP/ALIAS/TRANSLATION) [option]	337
UNITS (KUIP/UNITS)	333
USAGE (KUIP/USAGE) item	331
UWFUNC (NTUPLE/UWFUNC) idn fname [chopt]	403
VADD (VECTOR/OPERATIONS/VADD) vnam1 vnam2 vnam3	366
VBIAS (VECTOR/OPERATIONS/VBIAS) vnam1 bias vnam2	366
VDIVIDE (VECTOR/OPERATIONS/VDIVIDE) vnam1 vnam2 vnam3	367
VERIFY (ZEBRA/DZ/VERIFY) [idiv chopt]	443
VISIBILITY (KUIP/SET_SHOW/VISIBILITY) cmd [chopt]	345
VLOCATE (GRAPHICS/MISC/VLOCATE) vecx vecy [chopt ntpri wkid]	413
VMEM (NTUPLE/VMEM) [mxsize]	405
VMULTIPLY (VECTOR/OPERATIONS/VMULTIPLY) vnam1 vnam2 vnam3	366
VSCALE (VECTOR/OPERATIONS/VSCALE) vnam1 scale vnam2	366
VSUBTRACT (VECTOR/OPERATIONS/VSUBTRACT) vnam1 vnam2 vnam3	367
WAIT (KUIP/WAIT) [string sec]	333
WAVE (NTUPLE/WAVE) idn [lun]	401
WHILE (MACRO/SYNTAX/Looping/WHILE)	359
WORKSTATION (GRAPHICS/WORKSTATION) iwkid [chopt iwtyp]	411
WRITE (VECTOR/WRITE) vlist [fname format chopt]	363
ZONE (GRAPHICS/VIEWING/ZONE) [nx ny ifirst chopt]	413
ZOOM (HISTOGRAM/ZOOM) [id chopt icmin icmax]	370

Bibliography

[1] CERN. *COMIS – Compilation and Interpretation System, nProgram Library L210*, January 1994.

[2] CN/ASD Group. *HBOOK Users Guide (Version 4.21), nProgram Library Y250*. CERN, January 1994.

[3] CN/ASD Group. *HIGZ/HPLOT Users Guide, nProgram Library Q120 and Y251*. CERN, 1993.

[4] CN/ASD Group. *KUIP – Kit for a User Interface Package, nProgram library I202*. CERN, January 1994.

[5] CN/ASD Group. *MINUIT – Users Guide, nProgram Library D506*. CERN, 1993.

[6] CN/ASD Group. *PAW users guide, nProgram Library Q121*. CERN, October 1993.

[7] CN/ASD Group and J. Zoll/ECP. *ZEBRA Users Guide, nProgram Library Q100*. CERN, 1993.

[8] L. Lamport. *LaTeX A Document Preparation System (2nd Edition)*. Addison-Wesley, 1994.

[9] Adobe. *PostScript Language Manual (Second Edition)*. Addison Wesley, 1990.

[10] R. Bock et al. *HIGZ Users Guide, nProgram Library Q120*. CERN, 1991.

[11] R. Brun and H. Renshall. *HPLOT users guide, nProgram Library Y251*. CERN, 1990.

[12] F. James. *Interpretation of the errors on parameters as given by MINUIT, nSupplement to "CERN Program Library Long writeup D506"*. CERN, 1978.

[13] F. James. Determining the statistical Significance of experimental Results. Technical Report DD/81/02 and CERN Report 81–03, CERN, 1981.

[14] W. T. Eadie, D. Drijard, F. James, M. Roos, and B. Sadoulet. *Statistical Methods in Experimental Physics*. North-Holland, 1971.

[15] H. J. Klein and J. Zoll. *PATCHY Reference Manual, nProgram Library L400*. CERN, 1988.

[16] B. Segal. *The TCPAW package*. CERN, 1989.

[17] R. Brun and B. Segal. *A distributed Physics Analysis workbench*. CERN, 1989.

[18] Sun Microsystems. *Network File System Version 2*. Sun Microsystems, 1987.

Index

*
    IGSET parameter, 294
***P
    OPTION parameter, 295
**P
    OPTION parameter, 295
*COL
    SET parameter, 302
*P
    OPTION parameter, 295
[*], 204, 210
[0], 204
[1], 201, 204, 210
[#], 204
[@], 200, 204
OBSOLETE, 235, 453
RETURN, 40
$SIGMA, 239
1DHISTO (HISTOGRAM/CREATE/1DHISTO), **375**
2DHISTO (HISTOGRAM/CREATE/2DHISTO), **376**
2SIZ
    SET parameter, 297
3270G, 321

A0
    OPTION parameter, 295
A1
    OPTION parameter, 295
A2
    OPTION parameter, 295
A3
    OPTION parameter, 295
A4
    OPTION parameter, 295
A5
    OPTION parameter, 295
A6
    OPTION parameter, 295
abbreviation, 9, 22
ABSCISSA (HISTOGRAM/GET_VECT/ABSCISSA), **386**
active picture, 287
ADD (HISTOGRAM/OPERATIONS/ADD), **380**
addressing of vectors, 237
AERRORS (GRAPHICS/HPLOT/AERRORS), **431**
alias, 9, 182
ALIAS/CREATE, 182–184
alldef.kumac, 31
alphanumeric
    labels, 299
ANGLE (FUNCTION/ANGLE), **392**
ANY, 241
 ANY (SIGMA), **242**
Apollo, 15
APPLICATION (KUIP/SET_SHOW/APPLICATION), **340**
APPLICATION, 198, 199, 236
application SIGMA, 239
ARC (GRAPHICS/PRIMITIVES/ARC), **423**
arc
    border, 294
ARCHELIX (GRAPHICS/PRIMITIVES/ARCHELIX), **420**
Arithmetic (MACRO/SYNTAX/Expressions/Arithmetic), **351**
ARLINE (GRAPHICS/PRIMITIVES/ARLINE), **420**
ARRAY, 236
array, 236
    filling, 240
    in SIGMA, 240
 ARRAY (SIGMA), **240**
ARROW (GRAPHICS/PRIMITIVES/ARROW), **418**
ASIZ
    SET parameter, 296
AST
    OPTION parameter, 295
AST
    OPTION parameter, 295
asterisk size (for functions), 297
ATITLE (GRAPHICS/HPLOT/ATITLE), **433**
ATITLE, 310
attribute, 293
AURZ
    IGSET parameter, 294
    SET parameter, 291
automatic
    storage of pictures, 291
automatic naming of pictures, 294
AWLN
    IGSET parameter, 294

AXIS (GRAPHICS/PRIMITIVES/AXIS), **421**
AXIS, 299
axis
  divisions, 300
  labels
    font and precision, 297
    size, 297
  labels offset, 294
  labels size, 294
  tick marks size, 294
  title, 130
  values
    font and precision, 297
    size, 297

backspace, 313, 315
band, 14
BANX (HISTOGRAM/CREATE/BANX), **378**
BANY (HISTOGRAM/CREATE/BANY), **378**
BAR
  OPTION parameter, 295
bar
  chart, 296
  histogram
    offset, 297
    width, 297
BAR
  OPTION parameter, 295
bar charts, 136
BARO
  IGSET parameter, 294
  SET parameter, 296
BARW
  IGSET parameter, 294
  SET parameter, 296
bash shell, 6
basic operator in SIGMA, 240
BASL
  IGSET parameter, 294
batch, 3, 16
BCOL
  SET parameter, 296, 302
binning
  alphanumeric, 136
  automatic, 128
  user defined, 128
BINS (HISTOGRAM/CREATE/BINS), **376**
book histogram, 13
Boolean (MACRO/SYNTAX/Expressions/Boolean), **352**
boolean value in SIGMA, 240
BORD
  IGSET parameter, 294
BOX (GRAPHICS/PRIMITIVES/BOX), **417**
box
  around picture, 296
  border, 294
  fill area
    colour, 297
BOX
  OPTION parameter, 295
BREAK (KUIP/SET_SHOW/BREAK), **341**
BREAKL (MACRO/SYNTAX/Looping/BREAKL), **359**
BREAKL, 199, 211
Browsable, 214, 218
Browsable window, 214, 231
Browser, 214
Browser initialization, 218
BTYP
  SET parameter, 296, 302
BUGREPORT (KUIP/BUGREPORT), **336**
BWID
  SET parameter, 296

CALL (FORTRAN/CALL), **445**
CASE (MACRO/SYNTAX/Branching/CASE), **356**
CASE, 209
CASE, 199
CAT (NETWORK/PIAF/CAT), **449**
CDF (Command Definition File), 214, 216, 218, 222, 225
CDF Command Definition File, 9
CDIR (ZEBRA/RZ/CDIR), **440**
CDIR, 253, 287
CERN Program Library
  NEW, 15
  OLD, 15
  PRO, 15
CERNLIB, 18
CFON
  SET parameter, 296
CHA

OPTION parameter, 295
CHA
OPTION parameter, 295
CHAIN (NTUPLE/CHAIN), **400**
change directory, 252
character
escape, 313
key size, 297
shift, 297
CHHE
IGSET parameter, 294
SET parameter, 310
chisquare, 12
client, 327
CLOSE (FORTRAN/CLOSE), **447**
CLR (GRAPHICS/MISC/CLR), **412**
cmd1, 179
cmd2, 179
cmd3, 179
CMS, 15
CMZ, 233
COLOR_TABLE (GRAPHICS/ATTRIBUTES/COLOR_TABLE), **430**
colour, 293, 300, 302
COLUMNS (KUIP/SET_SHOW/COLUMNS), **342**
COMIS, 12, 46, 53, 64, 66, 80, 98, 146, 189, 190, 192, 236, 238, 266
COMIS (FORTRAN/COMIS), **444**
COMMAND (KUIP/SET_SHOW/COMMAND), **340**
command
abbreviation, 9, 22
definition file (CDF), 9
parameter, 40
mandatory, 22
optional, 22
search path, 15
structure, 22
visibility, 171
Command Argument Panel, 216, 217, 224
comment
and statistic size, 297
font and precision, 297
common /PAWC/, 252
components
of PAW, 9
CONNECT (NETWORK/PIAF/CONNECT), **448**
CONTENTS (HISTOGRAM/GET_VECT/CONTENTS), **385**
CONTENTS (HISTOGRAM/PUT_VECT/CONTENTS), **386**
CONTOUR (HISTOGRAM/2D_PLOT/CONTOUR), **375**
control operator in SIGMA, 240
coordinate systems
cylindrical, 104
polar, 104
pseudo rapidy, 104
spherical, 104
COPY (HISTOGRAM/COPY), **371**
COPY (PICTURE/COPY), **435**
COPY (VECTOR/COPY), **361**
correlation, 12
CP (NETWORK/PIAF/CP), **450**
CREATE (KUIP/ALIAS/CREATE), **336**
CREATE (MACRO/GLOBAL/CREATE), **350**
CREATE (NTUPLE/CREATE), **393**
CREATE (PICTURE/CREATE), **434**
CREATE (VECTOR/CREATE), **360**
create
vector, 236
cross-wires, 296
CSELECT (NTUPLE/CSELECT), **402**
CSHI
IGSET parameter, 294, 313
SET parameter, 296
CSIZ
SET parameter, 296
current
directory, 252
picture, 287
cut, 8, 13, 261, 263
graphical, 264
Cut Editor, 21
CUTS (NTUPLE/CUTS), **401**
CZ, 321

DASH
SET parameter, 296
dash mode for lines, 297
DATA (MACRO/DATA), **350**
data structure, 252
DATE
OPTION parameter, 304

SET parameter, 296, 304
date, 304
and hour on pictures, 296, 304
position, 297
DDIR (ZEBRA/RZ/DDIR), **440**
DECNET, 15, 321
default setting, 9
DEFAULTS (MACRO/DEFAULTS), **349**
DEL, 241
DELETE (HISTOGRAM/DELETE), **368**
DELETE (KUIP/ALIAS/DELETE), **337**
DELETE (MACRO/GLOBAL/DELETE), **350**
DELETE (PICTURE/DELETE), **434**
DELETE (VECTOR/DELETE), **361**
DEL (SIGMA), **242**
delta function, 242
DI3000, 11
dialogue style, 9
DIFF, 243
DIFF (HISTOGRAM/OPERATIONS/DIFF), **382**
DIFF, 241
DIFF (SIGMA), **243**
diologue
style, 9
directory
PAWC, 92
change, 252
current, 92, 252
ZEBRA, 9
DISCONNECT (NETWORK/PIAF/DISCONNECT), **451**
display, 15
distance
x axis
to labels, 297
to to axis values, 297
y axis
to labels, 297
to to axis values, 297
DIVIDE (HISTOGRAM/OPERATIONS/DIVIDE), **381**
divisions, 300
DLINE (GRAPHICS/PRIMITIVES/DLINE), **416**
DMOD
SET parameter, 296
DO (MACRO/SYNTAX/Looping/DO), **358**
DO, 199
DOLLAR (KUIP/SET_SHOW/DOLLAR), **346**
Domain, 15
DRAW (FUNCTION/DRAW), **390**
DRAW (NTUPLE/DRAW), **401**
DRAW (VECTOR/DRAW), **363**
driver, 15
DST, 12, 251, 254
Data Summary Tape, 12
DUMP (HISTOGRAM/HIO/DUMP), **380**
DUPLICATE (NTUPLE/DUPLICATE), **394**
DVXI
OPTION parameter, 295
DVXR
OPTION parameter, 295
DVYI
OPTION parameter, 295
DVYR
OPTION parameter, 295

EAH
OPTION parameter, 295
EDIT (KUIP/EDIT), **332**
EDIT, 184, 232, 267
editor, 320
EDM, 281, 283
ELSE, 199
emacs, 6
Encapsulated PostScript, 285
ENDCASE, 209
ENDKUMAC (MACRO/SYNTAX/Definitions/ENDKUMAC), **355**
ENDKUMAC, 198, 199
error
bars, 296
ERRORS (GRAPHICS/HPLOT/ERRORS), **431**
ERRORS (HISTOGRAM/GET_VECT/ERRORS), **385**
ERRORS (HISTOGRAM/PUT_VECT/ERRORS), **387**
errors on fitted parameters, 271
ERRX
SET parameter, 296
event, 13
exchange input/output, 12
exclamation mark character
place-holder, 22
EXEC (MACRO/EXEC), **348**
EXEC, 171, 198–204, 288

Executive Window, 18, 20, 215, 218, 219, 224, 227, 230, 231
EXIT (KUIP/EXIT), **333**
EXITM (MACRO/SYNTAX/Definitions/EXITM), **355**
EXITM, 189, 199, 200, 212

FACI
  IGSET parameter, 294
FAIS
  IGSET parameter, 294
  SET parameter, 305
FAREA (GRAPHICS/PRIMITIVES/FAREA), **416**
FASI
  IGSET parameter, 294
  SET parameter, 305
FBOX (GRAPHICS/PRIMITIVES/FBOX), **418**
FCOL
  SET parameter, 296, 302
Feynman diagrams, 156
FILE (FORTRAN/FILE), **446**
FILE (HISTOGRAM/FILE), **368**
FILE (PICTURE/FILE), **434**
FILE (ZEBRA/FZ/FILE), **441**
FILE (ZEBRA/RZ/FILE), **439**
FILE
  OPTION parameter, 304
  SET parameter, 296, 304
file name
  on pictures, 296, 304
  position, 297
FILECASE (KUIP/SET_SHOW/FILECASE), **346**
FILECASE, 173
fill
  area, 300
    interior style, 305
    style index, 305
  histogram, 13
  vector, 236
fill area
  colour index, 294
  interior style, 294
  style index, 294
first page number, 297
FIT (HISTOGRAM/FIT), **371**
FIT (VECTOR/FIT), **365**
FIT
  OPTION parameter, 295, 304
  SET parameter, 305
fit, 12, 13, 270
  parameters on pictures, 296, 304
  values to be plotted, 297
  vector, 238
FIT
  OPTION parameter, 295
  SET parameter, 296
font, 293
  PostScript, 315
  text, 312
fonts, 306
FOR (MACRO/SYNTAX/Looping/FOR), **358**
FOR, 199
FORTRAN, 444–448
FPGN
  SET parameter, 296
FPOINT (GRAPHICS/PRIMITIVES/FPOINT), **421**
FRALPHA (ZEBRA/FZ/FRALPHA), **442**
FREE (ZEBRA/RZ/FREE), **441**
FRFZ (ZEBRA/FZ/FRFZ), **441**
FTP, 324
FTYP
  SET parameter, 296, 302
FUN1 (FUNCTION/FUN1), **389**
FUN2 (FUNCTION/FUN2), **389**
FUNCTION, 389–393
FUNCTION (HISTOGRAM/GET_VECT/FUNCTION), **385**
function, 13, *see* sstem function185
  drawing
    one-dimensional, 56, 58
    three-dimensional, 66
    two-dimensional, 62
  fill area
    colour, 297
    type, 297
  in SIGMA, 241
  line width, 297
  range, 66
FUNCTIONS (KUIP/FUNCTIONS), **333**
FWID
  SET parameter, 296

Garbage (MACRO/SYNTAX/Expressions/Garbage), **352**
GDDM, 15
GDDM (IBM), 11
GET (NETWORK/PIAF/GET), **449**
GFON
  SET parameter, 296
GKS, 11, 15, 26, 285
GL (Silicon Graphics), 11
Global (MACRO/SYNTAX/Variables/Global), **354**
global
  section, 253, 321, 326
  title
    font and precision, 297
    size, 297
GLOBAL/CREATE, 206
GLOBAL/IMPORT, 206, 207
GLOBAL_SECT (HISTOGRAM/HIO/GLOBAL_SECT), **380**
GMR3D (Apollo), 11
GOTO, 199
GOTO_and_IF_GOTO (MACRO/SYNTAX/Branching/GOTO_and_IF_GOTO), **356**
GPR, 15
GPR (Apollo), 11
GRAPH, 238
GRAPH (GRAPHICS/PRIMITIVES/GRAPH), **428**
GRAPH, 309
graphical
  cut, 264
  output, 238
GRAPHICS, 406–434
graphics
  editor, 320
  terminal, 15
Graphics Window, 18, 20, 21
Greek letters, 313, 315
GRESET (HISTOGRAM/HIO/GRESET), **380**
GRID (GRAPHICS/HPLOT/GRID), **433**
GRID
  OPTION parameter, 296
  SET parameter, 296
grid, 296
  line type, 297
GRPLOT, 285
GSIZ
  SET parameter, 296

HARD
  OPTION parameter, 295
hardware characters, 296
hatch style, 305, 307
HBOOK, 9, 42, 80, 84, 88, 122, 124, 136, 140, 251, 266, 295
  Title, 296
HCDIR, 252, 253
HCOL
  SET parameter, 296, 302
HDERIV, 271
HELIX (GRAPHICS/PRIMITIVES/HELIX), **419**
HELP, 15, 23
HELP (KUIP/HELP), **331**
HELP, 171
HELP FUNCTIONS, 185
HESSE, 272
HFCNH, 270
HFCNV, 270
HFETCH (HISTOGRAM/HIO/HFETCH), **379**
HFILL (VECTOR/HFILL), **364**
HFITH, 270
HFITV, 270
HIDOPT, 295
HIFIT, 280
HIGZ, 11, 18, 26, 152, 191, 252, 266, 284, 286, 291, 293, 295
  G mode, 285
  graphics editor, 320
  Z mode, 285, 287
HIST, 238
HIST (GRAPHICS/PRIMITIVES/HIST), **427**
HIST/PLOT, 288
HISTO/FIL, 183
HISTO/PLOT, 175, 179, 309
HISTOFILE, 258
HISTOGRAM, 368–389
histogram, 7, 13, 251
  1D, 7
  2D, 8
  archiving, 88
  booking, 13
  contour, 102

non equidistant, 102
creation, 80
file, 80, 84
subdirectories, 88
fill area
colour, 297
type, 297
filling, 13, 80
fit, 90
line width, 297
list, 84, 86
maximum, 42
maximum for scale, 297
minimum, 42
operations, 92
graphical, 96, 98
plot, 84
presentation, 300
profile, 140
project, 138
stacked lego plots, 110
subrange, 108, 110
title size, 297
two-dimensional representations, 100
update, 96
Histogram Style Panel, 18, 20
HISTOGRAM/PLOT, 285
history file, 9
HLIMIT, 252
HLOGAR, 295
HMAX
SET parameter, 296
HMERGE (NTUPLE/HMERGE), **394**
HMINUIT (FORTRAN/HMINUIT), **444**
HMOVE (GRAPHICS/MISC/HMOVE), **413**
HORI
OPTION parameter, 295
host, 15
HOST_EDITOR (KUIP/SET_SHOW/HOST_EDITOR), **342**
HOST_EDITOR, 232, 234
HOST_PAGER (KUIP/SET_SHOW/HOST_PAGER), **343**
HOST_PRINTER (KUIP/SET_SHOW/HOST_PRINTER), **343**
HOST_PSVIEWER (KUIP/SET_SHOW/HOST_PSVIEWER), **344**
HOST_SHELL (KUIP/SET_SHOW/HOST_SHELL), **344**
HOST_SHELL, 187, 232
HPLOPT, 296
HPLOT, 9, 191, 251, 266, 284, 291, 293
HPLOT/E, 185
HREAD (HISTOGRAM/HIO/HREAD), **379**
HRFILE, 252
HRIN (HISTOGRAM/HIO/HRIN), **378**
HRIN, 252
HROUT (HISTOGRAM/HIO/HROUT), **379**
HROUT, 252
HSCRATCH (HISTOGRAM/HIO/HSCRATCH), **379**
HSETPR (HISTOGRAM/OPERATIONS/HSETPR), **385**
HTABLE, 295
HTIT
OPTION parameter, 295
HTYP
SET parameter, 296, 302
HWID
SET parameter, 296

IBM, 15
IBM 3192G graphics terminal, 15
IDLE (KUIP/IDLE), **333**
IDOPT (HISTOGRAM/SET/IDOPT), **387**
IF, 199
IF_THEN (MACRO/SYNTAX/Branching/IF_THEN), **356**
IGSET, 40
IGSET ( ), **293**
IGSET (PICTURE/IGSET), **437**
IGSET
*, 294
AURZ, 294
AWLN, 294
BARO, 294
BARW, 294
BASL, 294
BORD, 294
CHHE, 294
CSHI, 294, 313
FACI, 294
FAIS, 294

FASI, 294
LAOF, 294
LASI, 294
LTYP, 294
LWID, 294
MSCF, 294
MTYP, 294
PASS, 294, 313
PICT, 294
PLCI, 294
PMCI, 294
SHOW, 294
TANG, 294
TMSI, 294
TXAL, 294
TXCI, 294
TXFP, 294
IGSET, 293, 294, 305, 309, 313
IGTEXT, 314
IMPORT (MACRO/GLOBAL/IMPORT), **350**
Indirection (MACRO/SYNTAX/Variables/Indirection), **354**
initialisation, 17
INPUT (VECTOR/INPUT), **362**
Input Pad, 18, 20, 218–221, 230
input/output, 12
integer or real divisions on axis, 296
interactive, 3
IQUEST, **187**
IQUEST(1), 187, 211
ITX (GRAPHICS/PRIMITIVES/ITX), **426**
ITX, 309–313
IZIN (PICTURE/IZIN), **436**
IZOUT (PICTURE/IZOUT), **436**
IZPICT, 287
IZPICT (PICTURE/IZPICT), **436**

KERNLIB, 266
KEY (GRAPHICS/HPLOT/KEY), **432**
KEY, 296
KSIZ
SET parameter, 296
KUGETI, 194
KUGETR, 194
KUGETV, 192
KUIP, 9, 148, 170–175, 178–180, 182, 185, 190, 192, 193, 196–199, 203, 205, 214, 217–219, 221–225, 231–233, 235, 252, 266, 331–348
vector, 238
KUIP/EDIT, 233
KUIP/Motif, 214, 215, 218, 219, 221–225, 228–231, 234
KUVECT, 192
KUWHAM, 229

label, 299
text justification, 300
label:, 199
LABELS (GRAPHICS/PRIMITIVES/LABELS), **426**
LABELS, 299
LAOF
IGSET parameter, 294
LASI
IGSET parameter, 294
LAST (KUIP/LAST), **332**
LAST, 180
LaTeX
PostScript, 286
LCDIR (KUIP/SET_SHOW/LCDIR), **347**
LDIR (ZEBRA/RZ/LDIR), **440**
LDIR, 258
LEGO (HISTOGRAM/2D_PLOT/LEGO), **373**
length of
basic dashed segment, 297
X axis, 297
Y axis, 297
LFON
SET parameter, 296
library functions in SIGMA, 249
limits on fitted parameters, 271
LINE (GRAPHICS/PRIMITIVES/LINE), **416**
line
type, 305, 308
width, 300
linear scale, 296
lines, 293
LINTRA (NTUPLE/LINTRA), **404**
LINX
OPTION parameter, 295
LINY

OPTION parameter, 295
LINZ
OPTION parameter, 295
LIST (HISTOGRAM/LIST), **368**
LIST (KUIP/ALIAS/LIST), **337**
LIST (MACRO/GLOBAL/LIST), **351**
LIST (MACRO/LIST), **348**
LIST (NTUPLE/LIST), **394**
LIST (PICTURE/LIST), **434**
LIST (VECTOR/LIST), **361**
LOCATE (GRAPHICS/MISC/LOCATE), **412**
LOCK (ZEBRA/RZ/LOCK), **440**
logarithmic scale, 296
on lego plots, 106
logical operator in SIGMA, 240
LOGLEVEL (NETWORK/PIAF/LOGLEVEL), **451**
LOGX
OPTION parameter, 295
LOGY
OPTION parameter, 295
LOGZ
OPTION parameter, 295
LOOP (FORTRAN/LOOP), **446**
LOOP (NTUPLE/LOOP), **396**
lower case letters, 313, 315
LS, 243
LS (NETWORK/PIAF/LS), **449**
LS, 241
LS (SIGMA), **243**
LTYP
IGSET parameter, 294
LTYPE
SET parameter, 305
//LUN1, 253
LVMAX, 241
LVMAX (SIGMA), **244**
LVMIM, 241
LVMIN (SIGMA), **244**
LWID
IGSET parameter, 294

MACRO, 348–360
MACRO (MACRO/SYNTAX/Definitions/MACRO), **355**
MACRO, 198, 199, 202, 203
macro, 9, 14
conditional statement, 48
flow control, 48
indexed positional parameters, 60
loop, 46
parameter, 9
parameter list, 60
variable, 46
macro statements, 198, 199
flow control, 208
macro variable, 178
argument count, *see* [#]
argument list, *see* [*]
file name, *see* [0]
indirection, 206
numbered, *see* [1]
return code, *see* [@]
special, 204
undefined, 202, 203
MACRO/DEFAULT, 171
Macros, 9
Main Browser, 18, 214–216, 218, 219, 222
MAKE (ZEBRA/RZ/MAKE), **439**
making slides, 161
mandatory parameter, 22
Mandelbrot distribution, 64
MANUAL (KUIP/MANUAL), **331**
MANY_PLOTS (HISTOGRAM/MANY_PLOTS), **371**
marker
type, 305, 308
MASK (NTUPLE/MASK), **402**
MASK, 262
mask, 8, 13, 261, 263
match, 54
MAX, 241
MAXIMUM (HISTOGRAM/SET/MAXIMUM), **387**
MAX (SIGMA), **245**
MAXV, 241
MAXV (SIGMA), **245**
MDIR (ZEBRA/RZ/MDIR), **439**
menu, 22
MERGE (NTUPLE/MERGE), **396**
MERGE (PICTURE/MERGE), **435**
MESSAGE (KUIP/MESSAGE), **333**
MESSAGE (NETWORK/PIAF/MESSAGE), **450**
MESSAGE, 177, 189
METAFILE (GRAPHICS/METAFILE), **410**

METAFILE, 286
metafile, 8, 14, 26, 285
MIGRAD, 271, 272
MIN, 241
minimisation, 12, 270
MINIMUM (HISTOGRAM/SET/MINIMUM), **387**
 MIN (SIGMA), **245**
MINUIT, 12, 270
MINV, 241
 MINV (SIGMA), **245**
MIPS, 3
MKDIR (NETWORK/PIAF/MKDIR), **450**
MODE (NETWORK/PIAF/MODE), **450**
mode
  HIGZ
    G mode, 285
    Z mode, 285, 287
MODIFY (PICTURE/MODIFY), **435**
MODIFY, 320
Motif, 9, 18, 214
MSCF
  IGSET parameter, 294
MTYP
  IGSET parameter, 294
  SET parameter, 305
MULTIPLY (HISTOGRAM/OPERATIONS/MULTIPLY), **381**
MV (NETWORK/PIAF/MV), **450**

NAST
  OPTION parameter, 295
native input/output, 12
NBAR
  OPTION parameter, 295
NBOX
  OPTION parameter, 295
NCHA
  OPTION parameter, 295
NCO, 241
 NCO (SIGMA), **246**
NDAT
  OPTION parameter, 296
NDVX
  SET parameter, 296, 300
NDVY
  SET parameter, 296
NDVZ
  SET parameter, 297
NEAH
  OPTION parameter, 295
NETWORK, 448–452
NEWPANEL (KUIP/SET_SHOW/NEWPANEL), **340**
NEXT (GRAPHICS/MISC/NEXT), **412**
NEXTL (MACRO/SYNTAX/Looping/NEXTL), **359**
NEXTL, 199, 211
NFIL
  OPTION parameter, 296
NFIT
  OPTION parameter, 295
NGRI
  OPTION parameter, 296
NOPG
  OPTION parameter, 295
NORMALIZE_FACTOR (HISTOGRAM/SET/NORMALIZE_FACTOR), **387**
NPTO
  OPTION parameter, 295
NSQR
  OPTION parameter, 295
NSTA
  OPTION parameter, 295
NTAB
  OPTION parameter, 295
NTCUT, 263, 264
NTCUTS, 262
NTIC
  OPTION parameter, 295
NTMASK, 263
NTPLOT, 263
NTUPLE, 393–406
Ntuple, 8, 13, 251, 260
  cut, 261
  mask, 261
  weight, 261
ntuple
  and vector, 142
  chain, 144
  creation
    CWN, 124
    RWN, 122
  cuts, 134, 136
  loop, 134, 142

masks, 134
print, 122
CWN, 127
RWN, 127
profile histogram, 140
project, 128, 138
read
CWN, 124
RWN, 122
scan, 130, 132
selection criteria, 130
Ntuple Viewer, 18, 21
NTUPLEPLOT, 261
NULL (GRAPHICS/HPLOT/NULL), **433**
number of
divisions for
X axis, 297
Y axis, 297
passes for software characters, 297
Numbered (MACRO/SYNTAX/Variables/Numbered), **353**
NZFL
OPTION parameter, 295

Object window, 214, 230
OBSOLETE, 452
OFF ERROR, 199, 213
ON ERROR, 199, 213
ON ERROR CONTINUE, 199
ON ERROR EXITM, 199
ON ERROR GOTO, 199, 212
ON ERROR STOPM, 199
ON_ERROR (MACRO/SYNTAX/Branching/ON_ERROR), **357**
operating system, 9
operation on vectors, 237
operator in SIGMA, 240
OP (SIGMA), **242**
OPTION ( ), **293**
OPTION (GRAPHICS/OPTION), **408**
OPTION
***P, 295
**P, 295
*P, 295
A0, 295
A1, 295
A2, 295
A3, 295
A4, 295
A5, 295
A6, 295
AST , 295
AST, 295
BAR , 295
BAR, 295
BOX , 295
CHA , 295
CHA, 295
DATE, 304
DVXI, 295
DVXR, 295
DVYI, 295
DVYR, 295
EAH, 295
FILE, 304
FIT , 295
FIT, 295, 304
GRID, 296
HARD, 295
HORI, 295
HTIT, 295
LINX, 295
LINY, 295
LINZ, 295
LOGX, 295
LOGY, 295
LOGZ, 295
NAST, 295
NBAR, 295
NBOX, 295
NCHA, 295
NDAT, 296
NEAH, 295
NFIL, 296
NFIT, 295
NGRI, 296
NOPG, 295
NPTO, 295
NSQR, 295
NSTA, 295
NTAB, 295
NTIC, 295

NZFL, 295
PTO , 295
PTO, 295
SOFT, 295
SQR, 295
STA , 295
STAT, 304
STA, 295
TAB , 295
TIC , 295
TIC, 295
UTIT, 295
VERT, 295
ZFL , 295
ZFL1, 295
ZFL, 295
OPTION, 285, 288, 293, 304, 305
optional parameter, 22
ORDER, 241
ORDER (SIGMA), **246**
OS9, 327
module, 253, 321
OSI, 321
OUTPUT_LP (HISTOGRAM/HIO/OUTPUT_LP), **380**

page
format, 296
number, 296
number size, 297
PALETTE (GRAPHICS/ATTRIBUTES/PALETTE), **430**
PAWMAIN, 252
PANEL (KUIP/SET_SHOW/PANEL), **338**
PANEL, 224
panel
menu, 22
PANEL interface, 218, 221–223, 225
paper orientation, 296
PARAM (HISTOGRAM/OPERATIONS/PARAM), **384**
parameter, 9
errors (fit), 271
PASS
IGSET parameter, 294, 313
SET parameter, 297
path, 15
PAVE (GRAPHICS/PRIMITIVES/PAVE), **427**
PAW, 30, 31, 36, 42, 46, 80, 98, 136, 146, 156, 169, 214, 216, 229, 270
access, 15
entities, 26
initialisation, 17
object, 26
server, 321, 327
structure, 9
PAW (Physics Analysis Workstation), 18
PAW++, 18, 20, 21
PAW++ Locate, 21
/PAWC/ common, 252
/PAWC/ common, 252, 253
//PAWC directory, 253
PAWINT, 252
PAWLOGON, 15–17
PCOL
SET parameter, 297, 302
PG terminal type, 15
PICT
IGSET parameter, 294
PICT/LIST, 287
PICTURE, 434–439
picture, 8, 14, 286, 296
fill area
colour, 297
type, 297
line width, 297
print, 165
PICTURE/CREATE, 287
PICTURE/FILE, 291
PICTURE/PRINT, 288
PIE, 238
PIE (GRAPHICS/PRIMITIVES/PIE), **424**
PIE, 299
place-holder
exclamation mark character, 22
PLCI
IGSET parameter, 294
PLINE (GRAPHICS/PRIMITIVES/PLINE), **415**
PLOT
commands, 26
PLOT (FUNCTION/PLOT), **391**
PLOT (HISTOGRAM/PLOT), **368**
PLOT (NTUPLE/PLOT), **397**
PLOT (PICTURE/PLOT), **434**

PLOT (VECTOR/PLOT), **364**
PLOTHIS, 253
PMARKER (GRAPHICS/PRIMITIVES/PMARKER), **417**
PMCI
  IGSET parameter, 294
POINTS (FUNCTION/POINTS), **391**
polyline
  colour index, 294
  type, 294
  width, 294
polymarker
  colour index, 294
  scale factor, 294
  type, 294
PostScript, 14, 26, 152, 164, 285
  colour printers, 285
  fonts, 315
    Courier, 315
    Courier-Bold, 315
    Courier-BoldOblique, 315
    Courier-Oblique, 315
    Helvetica, 315
    Helvetica-Bold, 315
    Helvetica-BoldOblique, 315
    Helvetica-Oblique, 315
    Symbol, 315
    Times-Bold, 315
    Times-BoldItalic, 315
    Times-Italic, 315
    Times-Roman, 315
    ZapfDingbats, 315
  special A4, 285
precision
  text, 312
prefix SIGMA, 239
presenter, 326, 327
PRINT
  commands, 26
PRINT (HISTOGRAM/HIO/PRINT), **380**
PRINT (KUIP/PRINT), **332**
PRINT (NTUPLE/PRINT), **394**
PRINT (PICTURE/PRINT), **435**
PRINT (VECTOR/PRINT), **362**
PROD, 241
PROFILE (HISTOGRAM/CREATE/PROFILE), **376**
 PROF (SIGMA), **247**
PROJECT (HISTOGRAM/PROJECT), **371**
PROJECT (NTUPLE/PROJECT), **397**
projection, 13
PROMPT (KUIP/SET_SHOW/PROMPT), **341**
PROX (HISTOGRAM/CREATE/PROX), **377**
PROY (HISTOGRAM/CREATE/PROY), **377**
PSIZ
  SET parameter, 297
PSVIEW (KUIP/PSVIEW), **332**
PTO
  OPTION parameter, 295
PTO
  OPTION parameter, 295
PTO (Please Turn Over), 296
PTYP
  SET parameter, 297, 302
pull-down menu, 22
PURGE (ZEBRA/RZ/PURGE), **440**
PUT (NETWORK/PIAF/PUT), **449**
put
  contents, 42
PWD (NETWORK/PIAF/PWD), **450**
PWID
  SET parameter, 297

QUAD, 241
 QUAD (SIGMA), **247**
QUEST, *see* IQUEST
QUIT (KUIP/QUIT), **333**

RANGE (FUNCTION/RANGE), **391**
READ (MACRO/SYNTAX/Variables/READ), **354**
READ (NTUPLE/READ), **397**
READ (VECTOR/READ), **362**
READ, 199, 201
real time, 253
REBIN (HISTOGRAM/GET_VECT/REBIN), **386**
RECALL, 182
RECALL_STYLE (KUIP/SET_SHOW/RECALL_STYLE), **344**
RECORDING (KUIP/SET_SHOW/RECORDING), **342**
RECORDING, 180
RECOVER (NTUPLE/RECOVER), **395**
remote
  access, 258, 321
  file, 324

login, 324, 327
shell, 324, 327
RENAME (PICTURE/RENAME), **435**
REPEAT (MACRO/SYNTAX/Looping/REPEAT), **358**
REPEAT, 199
replay, 11
RESET (HISTOGRAM/OPERATIONS/RESET), **382**
RETURN (MACRO/SYNTAX/Definitions/RETURN), **355**
RETURN, 198–200
REWIND (FORTRAN/REWIND), **447**
RLOGIN, 324, 327
RLOGIN (NETWORK/RLOGIN), **448**
RM (NETWORK/PIAF/RM), **450**
RMDIR (NETWORK/PIAF/RMDIR), **450**
ROOT (KUIP/SET_SHOW/ROOT), **341**
RSHELL, 324, 327
RSHELL (NETWORK/RSHELL), **448**
RZ file, 12

SCALE_FACTOR_2D (HISTOGRAM/SET/SCALE_FACTOR_2D), **387**
SCAN (NTUPLE/SCAN), **395**
SCAN, 260
scatter plot
and table character size, 297
table, 251
SCHH (OBSOLETE/GRAPHICS/ATTRIBUTES/SCHH), **453**
SCRATCH (PICTURE/SCRATCH), **434**
selection
function, 261, 263, 266
SELNT (GRAPHICS/VIEWING/SELNT), **415**
server, 327
SET, 40
SET ( ), **293**
SET (GRAPHICS/SET), **406**
SET
*COL, 302
2SIZ, 297
ASIZ, 296
AURZ, 291
BARO, 296
BARW, 296
BCOL, 296, 302
BTYP, 296, 302
BWID, 296
CFON, 296
CHHE, 310
CSHI, 296
CSIZ, 296
DASH, 296
DATE, 296, 304
DMOD, 296
ERRX, 296
FAIS, 305
FASI, 305
FCOL, 296, 302
FILE, 296, 304
FIT , 296
FIT, 305
FPGN, 296
FTYP, 296, 302
FWID, 296
GFON, 296
GRID, 296
GSIZ, 296
HCOL, 296, 302
HMAX, 296
HTYP, 296, 302
HWID, 296
KSIZ, 296
LFON, 296
LTYPE, 305
MTYP, 305
NDVX, 296, 300
NDVY, 296
NDVZ, 297
PASS, 297
PCOL, 297, 302
PSIZ, 297
PTYP, 297, 302
PWID, 297
SMGR, 297
SMGU, 297
SSIZ, 297
STAT, 297, 304
TANG, 310
TFON, 297
TSIZ, 297
TXAL, 311
TXCI, 312

TXFP, 313
VFON, 297
VSIZ, 297
XCOL, 297
XLAB, 297
XMGL, 297
XMGR, 297
XSIZ, 297
XTIC, 297
XVAL, 297
XWID, 297
XWIN, 297
YCOL, 297
YGTI, 297
YHTI, 297
YLAB, 297
YMGL, 297
YMGU, 297
YNPG, 297
YSIZ, 297
YTIC, 297
YVAL, 297
YWID, 297
YWIN, 297
SET, 285, 293, 300, 304, 305, 309, 310
SET , 293
SET/APPLICATION, 198, 200
SET/COMMAND, 172, 235
SET/DOLLAR, 185
SET/ROOT, 235
SET/VISIBILITY, 171
SFACI (OBSOLETE/GRAPHICS/ATTRIBUTES/SFACI), **452**
SFAIS (OBSOLETE/GRAPHICS/ATTRIBUTES/SFAIS), **452**
SFASI (OBSOLETE/GRAPHICS/ATTRIBUTES/SFASI), **452**
SHELL (KUIP/SHELL), **333**
SHELL, 231, 232, 288
shell
bash, 6
tcsh, 6
SHIFT (MACRO/SYNTAX/Variables/SHIFT), **354**
SHIFT, 199, 204
SHOW (ZEBRA/DZ/SHOW), **442**
SHOW
IGSET parameter, 294
SIGMA, 12, 44, 46, 50, 102, 146, 150, 189, 190, 192, 236, 237, 239–250
$SIGMA, 239
access, 239
APPLication SIGMA, 239
array, 240
filling, 240
structure, 240
basic operator, 240
boolean value, 240
control operator, 240
function, 241
library functions, 249
logical operator, 240
prefix SIGMA, 239
vector, 240
SIGMA (FORTRAN/SIGMA), **447**
SIZE (GRAPHICS/VIEWING/SIZE), **414**
SIZE, 286
slice, 14
SLIDE (GRAPHICS/SLIDE), **412**
SLIX (HISTOGRAM/CREATE/SLIX), **377**
SLIY (HISTOGRAM/CREATE/SLIY), **377**
SLN (OBSOLETE/GRAPHICS/ATTRIBUTES/SLN), **452**
SLWSC (OBSOLETE/GRAPHICS/ATTRIBUTES/SLWSC), **453**
SMGR
SET parameter, 297
SMGU
SET parameter, 297
SMK (OBSOLETE/GRAPHICS/ATTRIBUTES/SMK), **452**
SMOOTH (HISTOGRAM/OPERATIONS/SMOOTH), **383**
SMOOTH, 175
SNAP (ZEBRA/DZ/SNAP), **442**
SOFT
OPTION parameter, 295
software
characters, 296
SORT (HISTOGRAM/OPERATIONS/SORT), **383**
Special (MACRO/SYNTAX/Variables/Special), **353**
special symbols, 25, 313, 315

SPLCI (OBSOLETE/GRAPHICS/ATTRIBUTES/SPLCI), **452**
SPLINE (HISTOGRAM/OPERATIONS/SPLINE), **384**
SPMCI (OBSOLETE/GRAPHICS/ATTRIBUTES/SPMCI), **452**
SQR
  OPTION parameter, 295
SSIZ
  SET parameter, 297
STA
  OPTION parameter, 295
STA
  OPTION parameter, 295
STAGE (NETWORK/PIAF/STAGE), **448**
STAT (ZEBRA/RZ/STAT), **441**
STAT
  OPTION parameter, 304
  SET parameter, 297, 304
statistic
  analysis, 12
  parameters on pictures, 296, 304
  values to be plotted, 297
STATUS (NETWORK/PIAF/STATUS), **450**
STOPM (MACRO/SYNTAX/Definitions/STOPM), **355**
STOPM, 199, 200, 212
STORE (ZEBRA/DZ/STORE), **443**
STRING, 177
String (MACRO/SYNTAX/Expressions/String), **352**
structure of PAW, 9
STXCI (OBSOLETE/GRAPHICS/ATTRIBUTES/STXCI), **452**
STXFP (OBSOLETE/GRAPHICS/ATTRIBUTES/STXFP), **453**
STYLE (KUIP/SET_SHOW/STYLE), **338**
style, 8
STYLE G, 171
style of dialogue, 9
subscript, 313, 315
SUBTRACT (HISTOGRAM/OPERATIONS/SUBTRACT), **381**
SUMV, 241
 SUMV (SIGMA), **248**
superscript, 313, 315
SURFACE (HISTOGRAM/2D_PLOT/SURFACE), **374**
SURV (ZEBRA/DZ/SURV), **442**
SVP (GRAPHICS/VIEWING/SVP), **414**
SWITCH
  Z, 287
SWITCH (PICTURE/SWITCH), **437**
SWN (GRAPHICS/VIEWING/SWN), **414**
SYMBOLS (GRAPHICS/HPLOT/SYMBOLS), **431**
symbols, 25
system function, 178, 185
  $ANAM, **186**
  $ANUM, **186**
  $ARGS, **187**
  $AVAL, **186**
  $CPTIME, 187, **187**
  $DATE, **187**
  $DEFINED, **187**, 205, 206
  $ENV, **187**
  $EVAL, 189, **189**, 190, 194
  $EXEC, **189**
  $FEXIST, **187**
  $FORMAT, **190**
  $INDEX, **187**
  $INLINE, 191, **191**, 204
  $IQUEST, **187**, 212
  $KEYNUM, **186**
  $KEYVAL, **186**
  $LAST, **186**
  $LEN, 187, **187**
  $LOWER, **187**
  $MACHINE, 187, **187**, 188
  $NUMVEC, **186**, 192
  $OS, 187, **187**, 188
  $PID, **187**
  $QUOTE, **188**, 189
  $RSIGMA, 190, **190**
  $RTIME, 187, **187**
  $SHELL, 187, **187**
  $SIGMA, **189**, 190, 193
  $STYLE, **186**
  $SUBSTRING, **187**
  $TIME, **187**
  $UNQUOTE, **189**, 197
  $UPPER, **187**
  $VDIM, 186, **186**
  $VEXIST, **186**
  $VLEN, **186**

$WORDS, **188**
$WORD, **188**
arguments, 185
name separators, 185

TAB
OPTION parameter, 295
TANG
IGSET parameter, 294
SET parameter, 310
TCP/IP, 15, 258, 321, 327
TCPAW, 321
tcsh shell, 6
Tektronix, 15
TELNET, 321, 324
TELNETG, 321
termination character, 313, 315
TEXT (GRAPHICS/PRIMITIVES/TEXT), **425**
TEXT, 294, 309–313, 315
text, 102
(and title) font and precision, 297
alignment, 294
horizontal, 311
vertical, 311
angle, 294
character height, 294
colour index, 294
data, 26
font, 294, 312
precision, 294, 312
width, 294
text alignment, 312
TFON
SET parameter, 297
TIC
OPTION parameter, 295
TIC
OPTION parameter, 295
tick marks, 300
TICKS (GRAPHICS/HPLOT/TICKS), **432**
TIMING (KUIP/SET_SHOW/TIMING), **341**
title, 104
title font and precision, 297
TITLE_GLOBAL (HISTOGRAM/CREATE/TITLE_GLOBAL), **378**
TMSI
IGSET parameter, 294
tn3270, 15
TOALPHA (ZEBRA/FZ/TOALPHA), **441**
TOFZ (ZEBRA/FZ/TOFZ), **441**
TRACE (MACRO/TRACE), **348**
Transcript Pad, 18, 20, 218–221, 230
TRANSLATION (KUIP/ALIAS/TRANSLATION), **337**
TSIZ
SET parameter, 297
TXAL
IGSET parameter, 294
SET parameter, 311
TXCI
IGSET parameter, 294
SET parameter, 312
TXFP
IGSET parameter, 294
SET parameter, 313

UNITS (KUIP/UNITS), **333**
Unix, 6
unix, 15
UNTIL, 199
upper case letters, 313, 315
USAGE (KUIP/USAGE), **331**
USAGE, 172
USAGE command, 25
user
title, 296
UTIT
OPTION parameter, 295
UWFUNC (NTUPLE/UWFUNC), **403**
UWFUNC, 178, 266

VADD (VECTOR/OPERATIONS/VADD), **366**
VAX, 15, 321
VAX/VMS, 326
Vaxstation, 15
VBIAS (VECTOR/OPERATIONS/VBIAS), **366**
VDIVIDE (VECTOR/OPERATIONS/VDIVIDE), **367**
VECDEF, 192
VECTOR, 360–368
VECTOR, 236
vector, 8, 14, 236
address, 237
and COMIS, 142
and ntuple, 142

arithmetic, 237, 240
create, 36, 236
delete, 36
dimensions, 38
draw, 36, 38, 40, 42
fill, 236
fit, 50
graph, 36, 158
hfill, 42
in SIGMA, 240
input, 36
operations, 44, 240
plot, 42
read, 46
using match, 54
subranges, 38
write, 38
VECTOR/CREATE, 192
VECTOR/LIST, 192
VECTOR/READ, 192
VECTOR/WRITE, 192
VEFIT, 280
VERIFY (ZEBRA/DZ/VERIFY), **443**
version, 15
VERT
OPTION parameter, 295
VFON
SET parameter, 297
VISIBILITY (KUIP/SET_SHOW/VISIBILITY), **345**
VISIBILITY, 171
VLOCATE (GRAPHICS/MISC/VLOCATE), **413**
VM-CMS, 15
VMAX, 241
VMAX (SIGMA), **249**
VMEM (NTUPLE/VMEM), **405**
VMIN, 241
VMIN (SIGMA), **249**
VMS, 15, 326
VMULTIPLY (VECTOR/OPERATIONS/VMULTIPLY), **366**
VSCALE (VECTOR/OPERATIONS/VSCALE), **366**
VSIZ
SET parameter, 297
VSUBTRACT (VECTOR/OPERATIONS/VSUBTRACT), **367**
VSUM, 241
VSUM (SIGMA), **249**

WAIT (KUIP/WAIT), **333**
WAVE (NTUPLE/WAVE), **401**
weight, 261
weighting factor, 263
WHILE (MACRO/SYNTAX/Looping/WHILE), **359**
WHILE, 199
WORKSTATION (GRAPHICS/WORKSTATION), **411**
workstation, 3, 15
type, 17
workstation type, 285
WRITE (VECTOR/WRITE), **363**

X axis
colour, 297
tick marks length, 297
X margin
left, 297
right, 297
X space between windows, 297
X windows, 11, 15
X11, 15, 18
XCOL
SET parameter, 297
XLAB
SET parameter, 297
XMGL
SET parameter, 297
XMGR
SET parameter, 297
XSIZ
SET parameter, 297
XTIC
SET parameter, 297
XVAL
SET parameter, 297
XWID
SET parameter, 297
XWIN
SET parameter, 297

Y axis
colour, 297
tick marks length, 297
Y margin

low, 297
up, 297
Y position of
global title, 297
histogram title, 297
page number, 297
Y space between windows, 297
YCOL
SET parameter, 297
YGTI
SET parameter, 297
YHTI
SET parameter, 297
YLAB
SET parameter, 297
YMGL
SET parameter, 297
YMGU
SET parameter, 297
YNPG
SET parameter, 297
YSIZ
SET parameter, 297
YTIC
SET parameter, 297
YVAL
SET parameter, 297
YWID
SET parameter, 297
YWIN
SET parameter, 297

ZEBRA, 12, 84, 252, 266, 321, 439–444
FRALFA, 26
FZ file, 26
RZ file, 26, 324
TOALFA, 26
ZFL
OPTION parameter, 295
ZFL
OPTION parameter, 295
ZFL (option), 287
ZFL1
OPTION parameter, 295
ZFL1 (option), 288
ZFTP, 324
zftp, 321
ZONE (GRAPHICS/VIEWING/ZONE), **413**
ZONE, 285
ZOOM (HISTOGRAM/ZOOM), **370**