Users Guide to the Physics Analysis Workstation

Introduction:

PAW was developed by the CERN laboratory in Geneva, Switzerland for use in particle physics experiments. It is a general-purpose program to fit arbitrary functions to your data via the method of Chi-square minimization (or maximum likelihood), and offers a variety of plotting options. It's main advantage over a program like Microsoft Excel is that it can fit nonlinear functions, and it returns the errors on the fitted parameters. Also, it is distributed free of charge.

You can generally abbreviate commands to the smallest unambiguous set of letters (i.e. vector/create → v/cre).

Documentation and Help:

PAW does much more than is listed in this quick tutorial. If you need advanced fitting or plotting options, consult the online tutorials and full the PAW manual, which can be found from the following web site:

```
http://wwwinfo.cern.ch/asd/paw/
```

For help with specific commands, you can type "help" while in the program and navigate through the menus.

Starting PAW:

Try one of the following to start PAW from a PC:

1. Double-click on the PAW icon
2. Open up an MS-DOS window and type "paw". You may first want to change directory to your working area where you keep all your data files.
3. Go to "Start", select "Run…", and type "c:\paw\pawNT.exe"

When the program asks for the workstation type, just hit Return. You exit PAW by typing "exit".

Data Entry:

vector/create

To enter a set of data by hand, try something like the following:

```
vec/cre x(5)  R 10.0 20.0 30.0 40.0 50.0
vec/cre y(5)  R 1.1  1.2  1.3  1.4  1.5
vec/cre dy(5) R 0.1  0.1  0.1  0.1  0.1
```

In this example, each array is of dimension 5 and contains real numbers (you can also use type "I" for integers.

vector/read

To read data from an ASCII file created by another program, try:

```
vec/read u junk.dat        (one column of data in the file)
vec/read u,v junk.dat      (two columns of data in the file)
etc.
```

Note that if a vector has not been created, it will be created for you with a size given by the length of the file.

Vector Plotting

vector/print

You can print out to the screen the contents of a vector: `vec/print x`

vector/plot

To plot the values of a vector as a histogram: `vec/plot x`
To plot how one vector depends on another: `vec/plot y%x`

graphics/hplot/errors

You can plot data with error bars drawn, but you need vectors for `x, y` and their associated error bars `dx, dy` (which might be zero). You also have to provide the number of points and the symbol type, and superimpose the plotting on an existing graph. For example:

```
null 0 1000 0 100    (draw graph with x from 0 to 1000, and y from 0 to 100)
gr/hp/err x y dx dy 3 24 0.28 S
```

The four numbers correspond to 3 data points, symbol type 24 (see below), and symbol size 0.28.
Some symbol types: 20 (solid circle), 21 (solid square), 22 (solid triangle),
24 (empty circle), 25 (empty square), 26 (empty triangle

Parameter Estimation: vector/fit

You can fit a variety of functions to your vector data using the method of Chi-square minimization (or maximum likelihood) to estimate parameters in the theory you are testing. It is assumed that your measured quantity is `y`, with uncertainty `dy`, and that it depends on variable `x`.

Polynomials

Zeroth order polynomial (constant): `vec/fit x y dy p0`
First order polynomial: `vec/fit x y dy p1`
Second order polynomial: `vec/fit x y dy p2`
etc.

Notice that a plot of the fitted line is shown through your data, and that the resulting reduced Chi-square is listed in the text window. Moreover, the errors on the fitted parameters are reported. These uncertainties are determined by allowing the Chi-square to increase by 1. They only make sense if your reported Chi-square is reasonable for the number of degrees of freedom.

Exponential, Gaussian:

In addition to “Pn” for polynomials, you can also fit an exponential or Gaussian:

```
    E : to fit Func=exp(par(1)+par(2)*x)
    G : to fit Func=par(1)*exp(-0.5*((x-par(2))/par(3))**2)
```

For example:

```
     vec/fit x y dy E
```

User Defined Function:

You can also fit a function of your own design. You need to write your function in the Fortran programming language. The details can be found by typing “help histogram/fit”. Fitting combinations of functions is described later under histogram fitting.

Histogram Creation and Filling:

histogram/create/1dhisto

To book a one-dimensional histogram, you have to give it a numerical label, title, number of bins, and a range. For example:

```
     1d 10 ‘Energy (keV)’ 50 0.0 1000.0
```

histogram/create/2dhisto

You can also book a two-dimensional histogram, where you list the number of bins and range for the x-axis first, followed by the y-axis:

```
     2d 20 ‘Time vs. Energy’ 50 0.0 1000.0 50 0.0 5.0e-9
```

vector/hfill

If you have an array of data stored in a vector that you would like to make a histogram out of, use the following:

```
     vec/hfill y 10
```

The first argument is the vector name, the second is the numerical label of the booked histogram. This will analyze the data and fill the histogram for you.

histogram/put_vector/contents

Alternatively, your data may already be in a histogram format. For example, you may have dumped the contents of a multi-channel analyzer to a file containing a single column of values representing the contents of each channel. After reading this file into a vector, you can create and fill a histogram with the same number of bins as the multi-channel analyzer. The steps would be something like:

```
     vec/read x muon1.dat
     1d 99 ‘Time (us)’ 10000 0. 312.5
     hi/put_vec/cont 99 x
```

Notice that you can conveniently change the units of the x axis to something useful, like microseconds or keV rather than just counts. It is only important that the number of bins be set correctly.

Histogram Plotting:

histogram/plot

To plot a histogram, use:

```
     hi/pl 10
```

where "10" is the numerical label of the histogram. You can superimpose one histogram over another by adding the option "S" after the label.

You can switch the line type between solid, dashed, and dotted by:

```
     set dmod 1       (solid line)
     set dmod 2       (dashed line)
     set dmod 3       (dotted line)
```

You also set fill style of the histogram, and the color. See the manual for more details.

Histogram Fitting:

The same functions available to `vector/fit` are also available to `histogram/fit`. For example:

```
     hi/fit 10 E       (exponential)
     hi/fit 10 G       (Gaussian)
```

The contents of each bin of the histogram are assumed to obey Poisson statistics, where the variance is equal to the mean. Thus, the uncertainty of the contents of each bin is taken to be the square root of the number of bin entries. This uncertainty is used for the Chi-square minimization procedure. Be aware that this method breaks down when the bin has only one or zero entries. PAW will ignore bins with zero entries. Generally, one should use the method of maximum likelihood for low statistics.

You can also fit combinations of functions to a histogram. For example, to fit an exponential on top of a flat background, try the following:

```
     hi/fit 10 E+P0 ! 3 par
```

To fit a Gaussian signal on top of a sloping background, try:

```
     hi/fit 10 G+P1 ! 5 par
```

The exclamation point after the combination of functions tells PAW to do a Chi-Square minimization. If you type 'L' instead, it will perform a maximum likelihood fit (good for low statistics bins). The next argument is the number of parameters to fit (depends on the function) and the last parameter is the parameter array. You may have to first initialize this array (using "vector/create par …" for example) with reasonable starting values to get the fit to converge.

You also can fit multiple Gaussians (G+G), multiple exponentials (E+E), or even products of functions (p2*G). You can fit your own user-defined function as well. The details can be found by typing "help histogram/fit".

To fit over a range of bins in your histogram, rather than the full range, you have to specify the range of bin numbers. For example, if your histogram has 100 bins, and you want to fit the central half of the histogram, use something like the following:

```
     hi/fit 10(25:75) G
```

Miscellaneous Plotting Commands:

For logarithmic scales: `opt liny` and `opt linx` for the y and x axes.
For linear scales: `opt liny` and `opt linx` for the y and x axes.
For gridlines: `opt grid` For no grid: `opt ngri`

For multiple plots on the same page use the "zone" command, where you specify how many columns and rows. For example,

```
     zone 2 3
```

gives you two columns by 3 rows for a total of 6 plots on one page. To return to just 1 per page:

```
     zone 1 1
```

Working with Ntuples

Ntuples are basically binary files used to store data. The file format was invented at CERN years ago, and is known as Zebra. Ntuples are generally stored in random access Zebra files. Data produced from Geant3 are normally stored in sequential Zebra files. A user generally stores a subset of all possible experimental data useful for analysis in an Ntuple, an approach sometimes referred to as creating a Data Summary Tape (DST). Several Ntuples and histograms can be stored in one Zebra file. It should be noted that two types of Ntuples can be made: "row-wise" and "column-wise". The differences will be made clear below.

Opening the file: histogram/file

You open an ntuple file within PAW by specifying a name (or logical unit number) and the file to open:

```
    hi/fil 70 csc_track1.hbook
    hi/fil 71 csc_track2.hbook
```

To view any subdirectories within a file, type:

```
    ldir
```

To change to one of the subdirectories, try something like:

```
    cd mydirectory
```

To set the default directory to the first file given in the example above, type:

```
    cd //lun70
```

If you have several files with identical format ntuples, you can chain them together:

```
    nt/chain lun72 csc_track1.hbook csc_track2.hbook
```

Then type:

```
    cd //lun72
```

If you would rather merge all the files into one file:

```
    nt/hmerge csc_merge.hbook csc_track1.hbook csc_track2.hbook
```

ntuple/list, ntuple/print, and ntuple/scan

To see what ntuples exist in the current file and subdirectory, type:

```
    nt/list
```

You generally get a list of i.d. numbers labeling all ntuples and histograms stored in the file. To see what variables are attached to any particular ntuple, type:

```
    nt/print 10
```

The number "10" should be replaced with whatever numerical i.d. labels your ntuple. If you want to print to the screen some of the variables in your ntuple, use the format:

```
    nt/sc 10 ! ! ! ! 2 x y
```

Here we have specified to list two variables called "x" and "y". You can apply some selection as well, replacing the first "!" with your expression:

```
    nt/sc 10 y>20 ! ! ! 2 x y
```

ntuple/plot

To plot one of the variables in the ntuple, use:

```
      nt/pl 10.x
```

where “x” is the variable you want to see plotted from ntuple 10. PAW will create a histogram automatically for you (with i.d. 1000000), and set the range and binning as well. If you want to make a 2-D scatter plot of “y” vs. “x”, the syntax is:

```
      nt/pl 10.y%x
```

You can also plot any Fortran-style arithmetic expression of the variables:

```
      nt/pl 10.sqrt(x**2+y**2)
```

If you want to fill a histogram you booked already (assume it has id 90), rather than have PAW choose automatically, use:

```
      nt/pl 10.x ! –90
```

If you want to apply selection cuts before filling your histogram, replace the “!” with the Fortran-style selection:

```
      nt/pl 10.x y>10.and.abs(z)<20 -90
```

Indexed variables

As mentioned at the beginning of this section, there are two types of Ntuples. Column-wise ntuples allow you to store and access arrays of data rather than creating a variable name for each array element as in a row-wise ntuple. If you want to access an element of an array, you need to know what the indexing variable is:

```
      nt/pl 10.ptvalue(1) ntrack>0 –80
```

Here “ntrack” is the indexing variable. We have to require the existence of at least one track in order to make the plot, otherwise we would get an access violation. The above will plot the first value in the ptvalue array for all events. If you want to plot all array entries for all events, you could just do:

```
      nt/pl 10.ptvalue
```

without the parentheses.

Fortran selection functions (COMIS functions):

PAW allows you to execute Fortran code in place of the selection cuts and plotting variables. These “COMIS” functions are interpreted by PAW rather than compiled like normal Fortran. You have access to all the utility functions stored in the CERNLIB libraries as well. If you want to be able to access variables within your ntuple, you need to define the Fortran common blocks for them. The way to do that is to execute the following command:

```
      nt/uwfunc 10 junk.f
```

This command will create a file called `junk.f` that has all the variables in ntuple 10 defined. Notice that the created Fortran function must have the same name as the file. You can now edit this file and put whatever selection, loops, and analysis that you want. To use it as a selection function, try:

```
      nt/pl 10.x junk.f>0
```

To plot the output of your function, type:

```
      nt/pl 10.junk.f
```

If your function takes one or more arguments, you define them in your Fortran file and call it from PAW as in:

```
     nt/pl 10.x dphi.f(1,2)>10
```

If your Fortran file is an analysis program that books and fills several histograms, you may not want any particular output to go to the screen. Rather, the results are stored into histograms by the analysis program. You can then use the `ntuple/loop` command:

```
     nt/loop 10 myprogram.f
```

Macros

In addition to Fortran files, PAW has a scripting language that allows you to execute interactive commands listed in a file. For example, after your PAW session, all the commands you typed are stored in the file `last.kumac.` If you would like to execute these commands in a later session, you can edit the file (and rename it so that it won't be overridden!) and type:

```
     exec junk.kumac
```

or whatever your file is called. Usually you can omit the "exec" command as well and just type "junk".

Printing

The simplest way to capture a plot for printing or insertion into a document may be to capture the screen or window and paste in the picture. However, two useful macros have been provided to you to save your plots or print them. They are called `saveit.kumac` and `plotit.kumac`, and should be defined in your current PAW path (c:\paw on a PC, or ~/paw on Unix). If you are running PAW under Unix, you can type:

```
     plotit
```

and the current display should be sent to the default Postscript printer. Keep in mind that the default printer *must* be a Postscript printer. If not, you will get lots of unintelligible text. The current script does not work on a PC yet. If you only want to save the output to a postscript file, type:

```
     saveit
```

The output will be stored in a file called `plot.ps`.