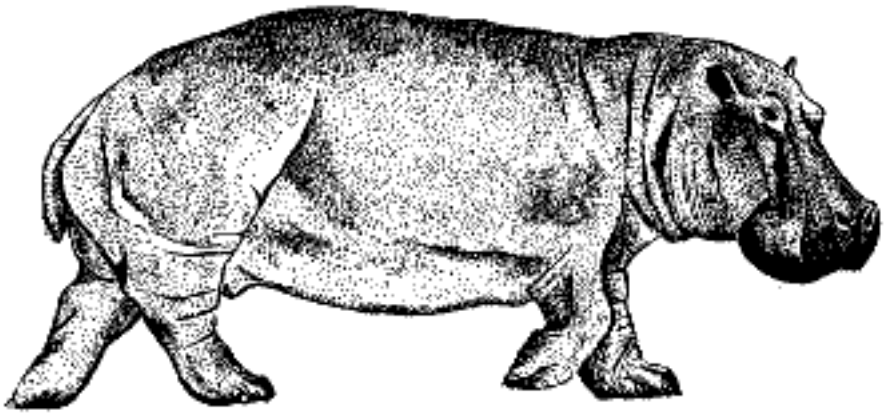
Hippoplotamus

Users Guide
(Release 1.1)

Mike F. Gravina,
Paul F. Kunz
Paul Rensing

Stanford Linear Accelerator Center
Stanford University
Stanford CA 94309



Disclaimer Notice

The items furnished herewith were developed under the sponsorship of the U.S. Government. Neither the U.S., nor the U.S. D.O.E., nor the Leland Stanford Junior University, nor their employees, makes any war- ranty, express or implied, or assumes any liability or responsibility for accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use will not in- fringe privately-owned rights. Mention of any product, its manufacturer, or suppliers shall not, nor is it intended to, imply approval, dis- approval, or fitness for any particular use. The U.S. and the Univer- sity at all times retain the right to use and disseminate the furnished items for any purpose whatsoever. Notice 91 02 01

Copyright 1992
by
The Board of Trustees of the
Leland Stanford Junior University.
All rights reserved.

Work supported by the U.S. Department of Energy under contract DE-AC03-76SF00515.

1. Introduction

The *Hippoplotamus* package, or *hippo* for short, is a library of data display and histogram functions based on n-tuples. A n-tuple is basically a table of floating point numbers with a fixed number of columns and some indeterminate, perhaps large, number of rows. The entries in the n-tuple, e.g. the rows, could be as simple as a set of x-y points. Such a n-tuple would have only two columns or be said to have a dimension of 2. Or it might be four dimensions containing x, y, error on x, and error on y. *Hippo* can also select one or two columns and produce a histogram of 1 or 2 dimensions. In addition, *hippo* can use additional columns to apply cuts on which rows will be used for 1 or 2 dimensional histograms. Finally, *hippo* can over plot arbitrary functions on its displays.

Hippo is logically divided into two parts: the n-tuple package and the display package. The philosophy of the *hippo* design that users will create n-tuples from their own programs, and use an interactive application to view them. Thus the n-tuple part of the *hippo* package is designed to make the creation of the n-tuple as easy as possible. The functions in this part of *hippo* were designed for use by an end-user. On the other hand, the display part of the *hippo* package was designed for someone who implements an interactive application to view and manipulate the displays. It is the basic premise of *hippo* that although the user may have a good idea what data to collect into an n-tuple, he has a much poorer idea of the attributes of their display. Thus, users will use an interactive application to change these attributes. Thus *hippo* is designed to postpone fixing the attributes of displays and histograms until they need to be presented on the users terminal or workstation.

In comparison to other display and histograms packages, *hippo* has the following interesting properties…

- It is written in ANSI C and intended to be used in C programs. As such it will make use of features in C such as casting parameters in function calls, use of pointers and structures, dynamic memory allocation, enumerated data types, etc. However, a FORTRAN binding for the n-tuple part of the package is supported.

- Network support is built into *hippo* by storing files to disk in a form compatible with the industrial standard XDR format[1]. Thus n-tuple files can be generated on one computer and transparently used on another. In a future release, one will be able to have a server on one machine providing n-tuple data to n-tuple viewing application on another.

- *Hippo* can be used with one program generating a n-tuple file and another program or application to view them. In this mode, it is kind of a client/server relationship. However, *hippo* can be easily incorporated into any program that generates the n-tuple data where the client and server are within the same program.

This document is organized first as a users guide and then as a reference manual. The first sections deal with creation of n-tuple data and files. It should be all that a typical user needs to know. The next sections describe the display package for the display programs and applications.

2. N-tuple Creation

A *hippo* n-tuple can be created in one of two basic methods. The first is to incorporate the *hippo* n-tuple package in a program. Basic functions are provided to initialize, accumulate, and save the n-tuple into a file. Other functions allow one to give the n-tuple a title and to label the columns. Program binding for the C, C++, Objective-C, and FORTRAN languages are supported. The second method is to create a plain text file representing the n-tuple data, title, and column labels and to use the *hippo* text-to-binary conversion utility.

2.1 N-tuple creation with C, C++, or Objective-C programs

The basic steps in creation of an n-tuple file are initialization, accumulation and saving the file. The n-tuple is initialized with the function call shown below.

```
my_tuple = h_new( ndim );
```

The function `h_new` takes one integer parameter: `ndim` which is the number of variables, i.e. the number of columns, per entry in the n-tuple. It returns a pointer to a C structure that contains all the parameters of the n-tuple needed for further work on the n-tuple. There is no need for the user to know about the internal of this structure. *hippo* functions are provided to access any of the information in the structure an advanced user may need to have. The *hippo* package declares a typedef for this pointer. Thus the user should include data type declaration like the one shown below. He

```
ntuple   my_tuple;
```

can thus use an n-tuple as a data variable in his code. The n-tuple variable will be used as a parameter in all subsequent *hippo* functions dealing with the n-tuple.

To collect data for a simple x-y plot, the dimension of at least 2 could be used, while for an x-y plot with errors on both x and y, a dimension of at least 4 could be used. For eventual generations of a 1D histogram, `ndim` could be as small as 1, while for weighted 1D histogram `ndim` is 2, one variable for the value (`x`) and the other for the weight (`w`). For a typical 2D histogram generation `ndim` is 2, while for a weighted 2D histogram `ndim` is 3 (`x`, `y`, and `w`). However, any collection of n variables may be collected into the n-tuple.

Attributes normally associated with histograms, such as number of bins, low edge of first bin, bin width, etc. are not defined when the n-tuple is initialized. In

hippo package, they are considered display attributes and thus defining them is deferred until a projection of the n-tuple is displayed.

Note the *hippo* package does not maintain state. `h_new` allocates memory space for the new n-tuple structure and returns it to the caller. It will not remember that it did that. The user or a higher level package will be responsible for maintaining the list of n-tuples in memory.

To accumulate data into an n-tuple, the user can invoke the *hippo* function `h_fill` as shown below. The first parameter is the n-tuple data variable that was

```
irc = h_fill( my_tuple, x, y, ... );
```

previously declared and initialized. The subsequent parameters to the function call are the data to be accumulated into the n-tuple entry. Their number *must* be equal to the dimension of the n-tuple defined by the `h_new` function call and be of type float.

The function `h_fill` returns the integer return code 0 on success or -1 on failure. The only known failure mode is exhaustion of virtual memory space. Thus, the limit of the number of entries of an n-tuple is determined only by virtual memory space available.

Sometimes it is more convenient organize the data to be accumulated as a floating point array. An alternate *hippo* function accepts such an array as its second parameter. The fragment of code show below illustrates its use. The size of the array

```
float x[10];
...
irc = h_arrayFill( mytuple, x );
```

must be at least the dimension of the n-tuple.

Once the n-tuple is collected in memory, it can be written to disk with the `h_write` function. It allows for one or more n-tuples to be written to a file in a single call. It also can write any displays attached to the n-tuple that the user may have defined. The prototype for the `h_write` function is shown below. The first argument is

```
int h_write( const char *filename,
             display dlist[],
             ntuple ntlist[] );
```

a string with the name of the file. The remaining two arguments are `NULL` terminated arrays of displays and n-tuples respectively. Passing a `NULL` is the same as an empty list thus a simple example of generating an n-tuple without displays might look as

```
#include "hippo.h"
main()
{
   ntuple ntlist[2];
   float x; int rc;

   ntlist[0] = h_new(1);
   for ( x = 0; x < 1000.; x++ ) {
      rc = h_fill( ntlist[0], x );
   }
   ntlist[1] = NULL;
   h_write("test", NULL, ntlist);
}
```

Figure 1. Basic n-tuple creation steps.

shown in Figure 1. In this example, an n-tuple with 1000 entries containing floating point number from 0. to 999. is created and saved to a file named test.

Before writing an n-tuple to a disk file, however, the user may wish to give the n-tuple a title and to label the individual columns. The prototypes for the three *hippo* provided for this purpose are shown below. Each takes as its first parameter the n-tu-

```
int h_setNtTitle( ntuple nt, const char *title );
int h_setNtLabel( ntuple nt, int dim, const char *label );
int h_setAllNtLabels( ntuple nt, const char *label[] );
```

ple data variable. The title of the n-tuple is set by the `h_setNtTitle` function. The `h_setNtLabel` sets the label of the column given in the second parameter, while the `h_setAllNtLabels` function sets the labels of all the columns in one call. These functions can be called any time after the n-tuple has been initialized with the `h_new` function call.

If the user has a n-tuple display program based on *hippo*, then this is all that the user really needs to know about *hippo* function package. The rest of this document is `hippo` reference manual for users that want to handle their own displays, for people who want to write a *hippo* display program, or for users need more sophisticated use of the *hippo* package.

2.2 N-tuple creation with FORTRAN program

A limited set of FORTRAN callable routines exist so that n-tuple creation can be done within FORTRAN programs. An example of FORTRAN code using *hippo* is

given in Figure 2. For each *hippo* C function that has a FORTRAN binding, the corresponding FORTRAN function has the same name with "h_" replaced with "ip".

The arguments used by the FORTRAN function are also the same, or have the same meaning as the corresponding C function. However, where the C function argument calls for data of type `ntuple`, an `integer` data type is used for FORTRAN. Thus, `nt = ipnew(ndim)` returns a new n-tuple with `ndim` columns and `irc = ipsetNtTitle(nt, 'A Title')` sets its title. Also, since by default FORTRAN indices start at 1 instead of 0 as in C, the ntuple column variable, `ndim`, should be equal to 1 for the first column. Since standard FORTRAN doesn't support variable length arguments lists, only the `iparrayFill()` is supported but not `ipfill()`. Finally, trailing blank characters will be removed from any character variables. In the current release, creation of displays with FORTRAN is not supported, so the corresponding argument in the call to `ipwrite()` must be 0. Note also that the n-tuple argument in `ipwrite()` is an array of integers which dimensioned at least one large

```
      Real*4 x(10)
      Integer ntlist(4)
      Integer nt
      Integer irc
C
      nt = ipnew(4)! generate the tuple
C
     irc = ipsetNtTitle( nt, 'First Tuple Title' )
C
      irc = ipsetNtLabel( nt, 1, 'First Column' )
      irc = ipsetNtLabel( nt, 2, 'Second Column' )
      irc = ipsetNtLabel( nt, 3, 'Third Column' )
      irc = ipsetNtLabel( nt, 4, 'Fourth Column' )
C
      Do 10 i = 1, 4
         x(i) = i
   10 Continue
C
      Do 20 i = 1,4
         irc = iparrayFill( nt, x )
   20 Continue
C
      ntlist(1) = nt
      ntlist(2) = 0
      irc = ipwrite( 'test.histo', 0, ntlist )
      end
```

Figure 2. Example of Hippo with FORTRAN.

then the number of n-tuples to be written. The value 0 is used to terminate this array in place of the NULL used in the C function.

2.3 Plain text to binary conversion

By default, *hippo* generates machine independent binary files when the *hippo* package is incorporated in the user's C or FORTRAN program. An alternate method of generating such files is to first generate a n-tuple in plain text (ASCII) format and use a conversion utility to convert that format to binary. This utility is provided as a line mode command on all platforms supported by *hippo*.

The full details of the conversion program, `text2nt`, is shown below in UNIX syntax. The input and output file names can be specified by switches (`-i` or `-f`, and

```
text2nt [-a hippo_file] [-v] [-n number]
        [-f text_file] [-i text_file]
        [-o hippo_file] [text_file [hippo_file]]
```

`-o`, respectively) or by positional arguments. Thus, the following two commands have the same result…

```
text2nt -i mydata.dat -o mydata.hippo
text2nt mydata.dat mydata.hippo
```

If the input or output file is not specified, then `text2nt` reads from `stdin` or writes to `stdout` respectively. Optionally, conversion can append its output to an existing file. The `-a` switch is used for this purpose and it is followed by the existing filename. When this switch is used, the `-n` may be used to direct the output to a particular n-tuple of the existing file. The number following this switch is used as an index (starting at 0) of the n-tuple to be appended. Finally, the `-v` switch is the verbose flag that causes `ntuple` to print various messages of progress to `stderr`.

The format of the plain text file is very simple. They consist of "lines" of no more than 256 characters that are delimited by the new line character (`'\n'` in C) or a by "`;`". Within a line there can be either numeric fields or string fields. A numeric field starts with a number ("`0-9`", "`-`", or "`+`") are delimited by one or more of "`,`", `<space>`, or `<tab>`. Any letters contained in a numeric fields are ignored and the remaining digits are read as numbers. String fields are used for the n-tuple title and column labels. They are delimited by single quote (`'`) or double quote (`"`) characters. An open quote must be matched by an identical close quote (e.g. `"matches"`, `'matches'`), but either of the two types can be used in a file.

The title and label fields can be specified anywhere in the file, but the title must be before the labels. The first “line” which does not start with a numeric field and which contains string field (typically, the first line of the file) is used as the title; everything else on that “line” is ignored. The labels come from the second “line” that contains starts with a string field and all labels must be on one “line”.

The dimension of the n-tuple is determined from the first line which starts from a numeric field. This line will be scanned until the first field that does not read as a number. The number of valid numeric fields sets the number of columns of the n-tuple. All subsequent lines must have that many (or more) numeric fields; each line corresponding to one row of the n-tuple. Lines which don't contain enough numeric fields, or which don't contain string fields (once the title and labels have been determined) are ignored.

An example of a plain text file readable by the `ntuple` program that contains two rows of three columns is shown below. Note that one can freely mix either style of

```
"This is the title line"
"column0" 'column1' "column2" extra junk ignorned
4.5 32.5 68.3
4.7,29.8, 58.7 extra junk ignored
this line is ignored
-2.18, 66. another line ignroed
4.9, 27.4 55.7 this line used
```

quoted strings and that letters that are not quoted are ignored.

2.4 HBOOK4 to Hippo conversion.

The utility program bh2hippo will extract n-tuples from file created by the HBOOK4 package[2] and save the n-tuples in *hippo* file format. The program is run as a command with 1 or two arguments as follows…

```
hb2hippo hbook_file [hippo_file]
```

The first argument is the name of the HBOOK4 file. The second argument is optional and is used as the name of the *hippo* file. If not given, then the *hippo* file will be named `out.hippo`. In the current release, any histograms in the HBOOK4 file are ignored.

3. History

The prototype for *hippo* was a package called HandyC written by Benoit Mours while he was at SLAC. It was intended to support the Reason project. HandyC was written in C, and it demonstrated the power of the C programming language for such packages. It followed the calling sequence of SLAC's HandyPak[3] but added a few new features. Features from the DESY package GEP[4], and the CERN package HBOOK[2], which have been incorporated into *hippo*. The authors of *hippo* are indebted to work done Benoit Mours and the authors of these other packages.

HandyC was not only a good test bed to try out new ideas, but also to measure the performance and/or response time the user would feel for these features. In the summer of 1990, Jonas Karlsson, then working as a summer student, suggested that one should really start over again with a better base in order to implement yet further new ideas. William Shipley (another summer student) and Gary Word contributed to the foundation of *hippo* soon thereafter. Jonas Karlsson did the original implementation. The current authors started working on *hippo* in earnest early in 1991. There also contributions from Tom Pavel and others in the current package.

4. References

[1] XDR External Data Representation Standard, RFC1014, Sun Microsystems, Inc., USC-ISI (see also man pages on UNIX systems).

[2] R. Brun, D. Lienart, HBOOK USER GUIDE: CERN COMPUTER CENTER PROGRAM LIBRARY LONG WRITEUP:VERSION 4, CERN-Y250, Oct 1987. 108pp.

[3] A. Boyarski, HANDYPAK: A HISTOGRAM AND DISPLAY PACKAGE (RELEASE 6.5), SLAC-0234-Rev-2, Sep 1988. 132pp. Revised version.

[4] E. Bassler, THE GRAPHICAL EDITOR PROGRAM: GEP, COMPUT. PHYS. COMMUN. 45 (1987) 201-205.