

Histograms, n -tuples, etc.

I. CERNLIB and its successors

II. Histograms in general

III. Histograms in FORTRAN/C++/PAW

IV. n -tuples

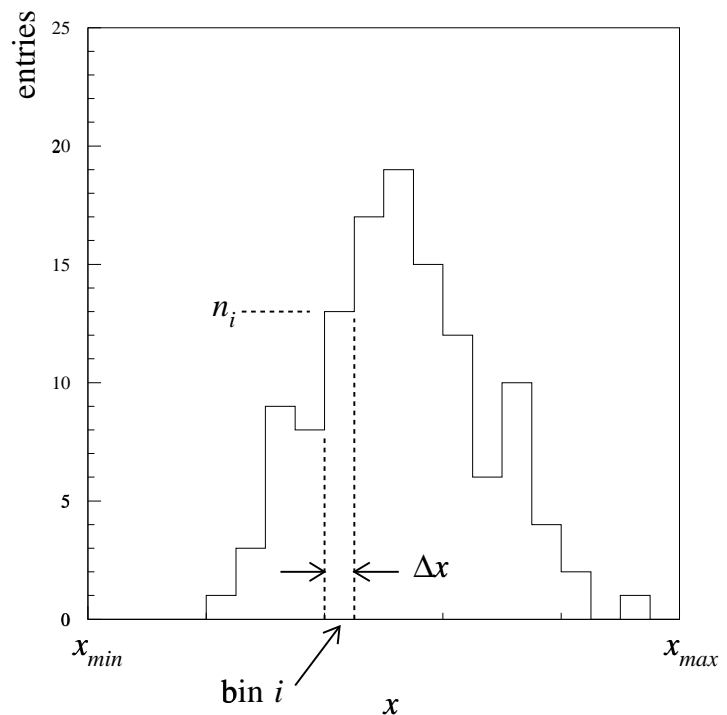
The CERN program library (CERNLIB)

- Web site: <http://wwwinfo.cern.ch/asd/index.html>
- CERNLIB contains FORTRAN subprograms for:
 - numerical analysis
 - Monte Carlo
 - data handling (histograms, search, sort, ...)
 - event generators, detector simulation, particle kinematics, ...
- Anaphe (LHC++): replace CERNLIB with OO (e.g. C++) libraries only partially complete, rapid state of flux
- Provisional solution for C++:
 - call CERNLIB routines from C++ using `cfortran.h`, see <http://wwwinfo.cern.ch/asd/cgi/listpawfaqs.pl/21>
- To use, include at end of link command:
 - `'cernlib'` or if more libraries needed, try e.g.
 - `'cernlib graflib mathlib kernlib packlib'`
- Some other libraries (see links on course web site):
 - Statlib (Carnegie Mellon)
 - Netlib (University of Tennessee)
 - Numerical Algorithms Group (NAG)
 - Numerical Recipes (Press et al.)
 - Datenanalyse (Brandt)

Histograms

- Consider a data sample $\vec{x} = (x_1, \dots, x_m)$ (m can be large)

→ summarize information
as a histogram
(N bins)



- Generic computer implementation:

I. Define bins, e.g. N bins of width Δx from x_{min} to x_{max} .

II. Declare variables to hold n_1, \dots, n_N , initialize all n_i to 0.

III. Loop over x_1, \dots, x_m ; if x value in bin i , $n_i \rightarrow n_i + 1$.

- In practice, not trivial \Rightarrow use packages HBOOK, HTL, ...

Histograms with HBOOK

- The HBOOK package (from CERN): user-callable FORTRAN subroutines for creating/manipulating:

histograms (1-dimensional)
scatter-plots (2-dimensional)
 n -tuples

- The basic steps to get a histogram:

I. 'Book' histogram: define bins, allocate memory for n_1, \dots, n_N .

```
call HBOOK1(17, 'x values', 100, xmin, xmax, 0.)
```

id number title number of bins

II. 'Fill' the histogram:

```
do i = 1, m
  call HF1(17, x(i), 1.)
end do
```

'weight' (usually 1.0)

- Steps also needed to set up output file and store results
(see example on next page)

```
program TEST_HBOOK
```

```
c Glen Cowan  
c 5 October, 1999  
c Test program for using HBOOK
```

```
implicit NONE
```

```
c Needed for HBOOK routines
```

```
integer hsize  
parameter (hsize = 100000)  
integer hmemor (hsize)  
common /pawc/ hmemor
```

```
c Local variables
```

```
character*80 outfile  
integer i, icycle, istat, num_values  
integer seed / 12345 /  
real x
```

```
c Initialize HBOOK, open histogram file, book histograms.
```

```
call HLIMIT (hsize)  
outfile = 'test_hbook.his'  
call HROPEN (20, 'histog', outfile, 'N', 1024, istat)  
call HBOOK1 (17, 'x values', 100, 0., 10., 0.)
```

```
c Get x values and enter into histogram.
```

```
write (*, *) 'enter number of x values to generate'  
read (*, *) num_values  
do i = 1, num_values  
  x = RANDOM (seed) ! returns a random number  
  call HF1 (17, x, 1.)  
end do
```

```
c Store histogram and close.
```

```
call HROUT (0, icycle, ' ')  
call HREND ('histog')
```

```
stop  
END
```

- For details see

<http://wwwinfo.cern.ch/asdcgi/listcernlibfaqs.pl/10>

and example `test_radom.cc` on SDA web site.

- Need C++ header files in `/cern/pro/include/cfortran`
- To call a FORTRAN CERNLIB routine from C++,

(1) Include the necessary header files:

```
#include "cfortran/cfortran.h"
#include "cfortran/hbook.h"
```

(2) Call routine (name in capitals) with same parameters:

```
main(){
    :
    HBOOK (1, "test histogram", 100, 0., 1., 0.);
    for (int i = 1; i<1000; i++){
        float x = random(seed);
        HF1 (1, x, 1.);
    }
    :
}
```

- Special flags needed to compile/link (see documentation on web)

```
#include <stdlib.h>
#include <iostream>
#include "cfortran/cfortran.h"
#include "cfortran/hbook.h"
#include "random.h" // prototype for random function used below

// define array used for hbook memory
#define PAWC_SIZE 50000
float pawc_[PAWC_SIZE];

int main(){

    // Initialize HBOOK and open histogram file
    HLIMIT(PAWC_SIZE);
    int lun = 20;
    int istat = 0;
    int lrec = 1024;
    char* outfile = "test_hbook.his";
    HROPEN (lun, "histog", outfile, "N", lrec, istat);
    if (istat != 0){
        std::cout << "HROPEN error, istat = " << istat << std::endl;
        exit(istat);
    }

    HBOOK1 (1, "uniform", 100, 0., 1., 0.);

    int num_values;
    cout << "Enter number of values to generate" << endl;
    cin >> num_values;
    int seed = 12345;
    for (int i = 0; i<num_values; i++){
        float x = random(seed);
        HF1(17, x, 1.);
    }

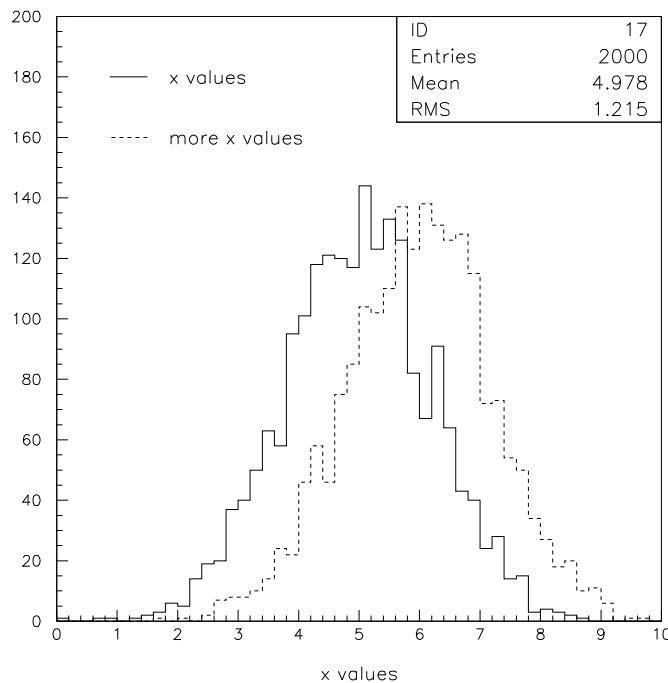
    int icycle=0;
    HROUT (0, icycle, " ");
    HREND ("histog");

    return 0;
}
```

Looking at the histograms with PAW

- Running the sample program creates the file `test_hbook.his`.
To view/manipulate the histograms with PAW,

```
h/file 1 test_hbook.his ← read in file
h/list                  ← show list of histograms
===> Directory :
17 (1) x values
23 (1) more x values
h/pl 17                ← plot histogram 17
h/pl 23 s              ← put 23 on same plot
```



- See documentation for details on commands like:

```
opt stat, set dmod, h/set max, key, ...
```


Two-dimensional histograms (scatter plots)

- Bins are now cells in 2-d plane. HBOOK routines similar to 1-d:

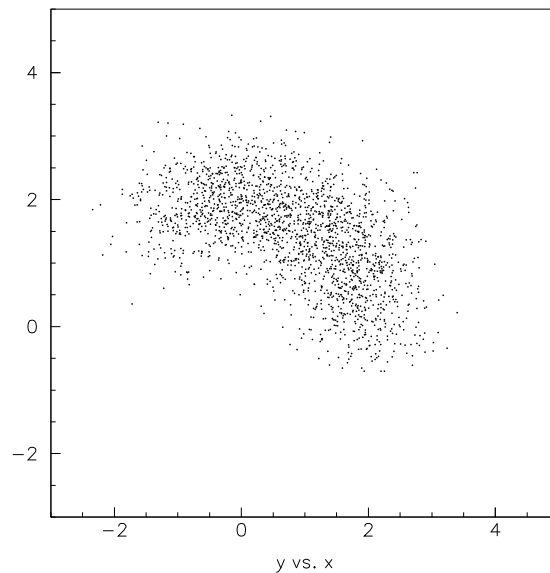
I. To book:

```
      id number      title
      ↓             ↓
call HBOOK2 (37, 'y vs. x', nx, xmin, xmax,
& ny, ymin, ymax, 0.)
      └──────────┘
      same stuff for y
      ↑
      number of bins in x
```

II. To fill:

```
call HF2 (37, x, y, 1.)
```

- Viewing with PAW same as in 1-d case:



N.B. Exact x , y values not recorded, only numbers of entries in each bin ($N_x \times N_y$ values stored).

Access to information

- In FORTRAN (see HBOOK manual for details)

Open file and read in the histograms:

```
call HROPEN (30, ' ', 'myfile.his', ' ', lrec, istat)
call HRIN (0, icycle, 0)
```

Access contents of histograms, errors, etc.

```
call HNOENT (id, num_entries)      ← number of entries
call HUNPAK (id, contents, ' ', 0) ← unpack into array
call HGIVE (id, title, nx, xmin, xmax, ny,
& ymin, ymax, nwt, loc)          ← get booking info
```

- In PAW (see PAW manual or online help)

Read histograms into memory, use variables and system functions, best used in macros ('kumac' files).

```
h/file 1 myfile.his
hrin 0      ← read histograms into memory
id = 17     ← define variable id, use brackets to evaluate.
nx = $HINFO([id], 'XBINS') ← system function HINFO
vec/create myvector([nx]) R ← vector to hold histogram
vec/print myvector      ← show vector contents
mess 'events =' $HINFO([id], 'EVENTS') ← # entries
```

n -tuples

- An n -tuple is a matrix of numbers, e.g. m instances of an n -dimensional vector:

$$m \text{ 'events' (rows)} \left\{ \begin{array}{l} (x_1, x_2, \dots, x_n)_1 \\ (x_1, x_2, \dots, x_n)_2 \\ (x_1, x_2, \dots, x_n)_3 \\ \vdots \\ (x_1, x_2, \dots, x_n)_m \end{array} \right.$$

n columns, i.e. n variables for each event

- Example: for 3-body decays of a certain type of particle, record:

$$p_{x_1}, p_{y_1}, p_{z_1}, p_{x_2}, p_{y_2}, p_{z_2}, p_{x_3}, p_{y_3}, p_{z_3}$$

- Data volume = number of events \times number of columns
- Compare to histogram with n dimensions: data volume = N_{bins}^n

\Rightarrow for large enough n (usually 2 or more), n -tuple wins

- Use n -tuple to store event properties for use in further analysis, e.g. to make histogram of invariant mass of two of the particles

$$m_{12} = [(E_1 + E_2)^2 - (\vec{p}_1 + \vec{p}_2)^2]^{1/2}$$

after making a cut on m_{23} .

n -tuples with HBOOK

- HBOOK provides two types of n -tuples:

(1) Row wise:

each event (row) stored sequentially;

events have fixed length;

only real (floating point) variables allowed;

to book: **HBOOKN**

to fill: **HFN**

to read: **HGN, HGMPAR, HGNF** (or use PAW)

(2) Column wise:

columns stored sequentially

→ faster access if only a few columns out of many needed

events can contain mixed variable types;

events can contain data structures of variable length;

to book: **HBNT, HBNAME**

to fill: **HFNT, HFNTB**

to read: **HGNT, HGNTB, HGNTV, HGNTF** (or use PAW)

- See examples on course web site, PAW/HBOOK documentation.

n -tuples with PAW

- The usual procedure:

create n -tuple in analysis program, then analyze it with PAW

- Suppose we've created an n -tuple with 3 columns: x , y , z .

Read in with `h/file` as before; suppose n -tuple id is 17.

The basic commands:

```
ntuple/plot 17.x    ← plot histogram of  $x$ 
nt/pl 17.y%x        ← make scatter plot of  $y$  vs.  $x$ 
nt/print 17         ← show variable names, min/max values
nt/scan 17          ← show entire contents of  $n$ -tuple
nt/pl 17.x 2*y-z<1.5 ← histogram  $x$  after cut on  $y$ ,  $z$ 
nt/cut $1 2*y-z<1.5 ← define cut number 1
nt/pl 1.x $1        ← histogram of  $x$  after cut 1
```

- To set histogram properties (e.g. id=20, 100 bins from 0 to 50):

```
ldhisto 20 'histo of x' 100 0. 50.
nt/proj 20 17.x ← project contents into histogram
h/pl 20        ← display histogram as usual
```

- More advanced features possible (loops, masks, ...) but at some point it's better to use a high level program (FORTRAN, C++).