ROOT Data Analysis, Part 2

Alberto Garfagnini

Università di Padova

27 Febbraio 2019, aula P140

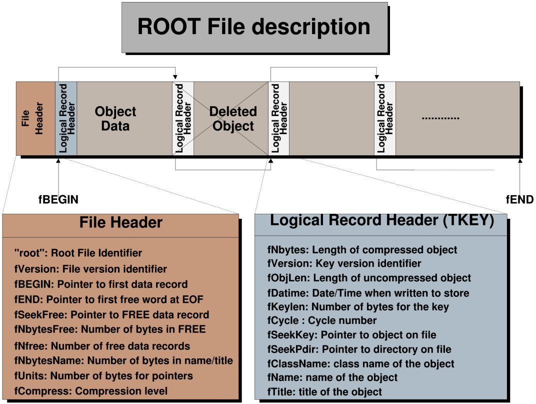
Opening a ROOT file

▷ first option: while launching ROOT, specify it as a parameter

```
$ root -l misura6_singole_int200.root
root [0]
Attaching file misura6_singole_int200.root as _file0...
```

▷ second option: open the file from inside ROOT

```
$ root -l
root [0] TFile *f = new TFi
```



- A ROOT file is a suite of consecutive data records (TKey instances) with a well defined format
- https://root.cern.ch/doc/master/classTFile.html

Listing a ROOT file content

```
root [1] .ls
TFile**         misura6_singole_int200.root
 TFile*         misura6_singole_int200.root
  KEY: TH1F     hq0;1   qlong ch0
  KEY: TH1F     hq1;1   qlong ch1
  KEY: TH1F     hq2;1   qlong ch2
  KEY: TH1F     hq3;1   qlong ch3
  KEY: TH2F     hpsd0;1 psd vs. qlong ch0
  KEY: TH2F     hpsd1;1 psd vs. qlong ch1
  KEY: TH2F     hpsd2;1 psd vs. qlong ch2
  KEY: TH2F     hpsd3;1 psd vs. qlong ch3
  KEY: TNtuple  nt;1
```

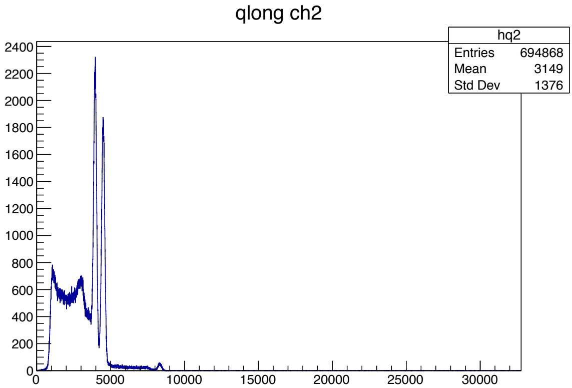
The file contains :

- four 1-D histograms, TH1F class, with a float per channel: {`hq0, hq1, hq2, hq3`}
- four 2-D histograms, TH2F class, with a float per channel: {`hpsd0, hpsd1, hpsd2, hpsd3`}
- and on TNtuple class object, with name `nt`

Plotting one histogram

- Histograms can be handled from the terminal:

```
root [1] hq0->Draw()  // Plot the histogram

root [2] c1->Print("hq0_new.pdf")
Info in <TCanvas::Print>: pdf file hq0_new.pdf has been created

root [3] .q
```



The ROOT TNtuple data type

- The TNtuple class is a simplified version of the ROOT TTree, containing a simple list of variables (in float or double type)
- it's a powerful analysis tool which allows to create single or double variable plots is a quick and simple way
- moreover, it is possible to study the correlations between two variables (A versus B) with additional selections on the other variables (multi-parametric analysis)
- it is therefore ideal for a multi-parametric analysis from the ROOT command line

TNtuple frequently used functions

- Get the number of events inthe ntuple:

- `nt->GetEntries();`

- List all the ntuple variables

- `nt->Print();`

- Plot `var1` for all events in the ntuple

- `nt->Draw("var1");`

- Scatter plot of `var1` versus `var2`

- `nt->Draw("var1:var2");`

- plot `var1`, conditioned at the restricted range for `var2`

- `nt->Draw("var1","var2>0 && var2<1");`

- plot `var2` only for `var2` $>$ 0, over the existing histogram and only for the first 1000 events

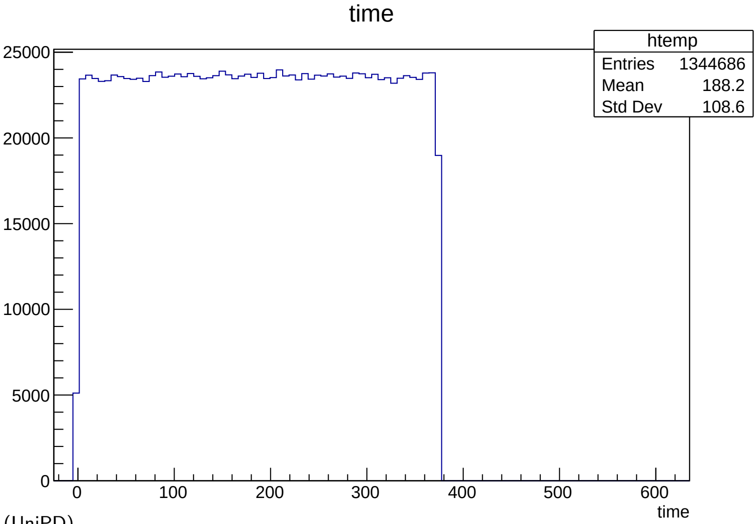
- `nt->Draw("var2","var2>0","same",1000);`

Inspecting the TNtuple content

```
root [1] nt->Print()
******************************************************************************
*Tree    :nt        :                                                        *
*Entries :  1344686 : Total =        37759030 bytes  File  Size =   20861849 *
*        :          : Tree compression factor =   1.81                       *
******************************************************************************
*Br    0 :ev        : Float_t                                                *
*Entries :  1344686 : Total  Size=    5393727 bytes  File Size  =    1929397 *
*Baskets :      169 : Basket Size=      32000 bytes  Compression=   2.79     *
*............................................................................*
*Br    1 :ch        : Float_t                                                *
*Entries :  1344686 : Total  Size=    5393727 bytes  File Size  =     118285 *
*Baskets :      169 : Basket Size=      32000 bytes  Compression=  45.57     *
*............................................................................*
*Br    2 :time      : Float_t                                                *
*Entries :  1344686 : Total  Size=    5394073 bytes  File Size  =    2967563 *
*Baskets :      169 : Basket Size=      32000 bytes  Compression=   1.82     *
*............................................................................*
*Br    3 :rawtime   : Float_t                                                *
*Entries :  1344686 : Total  Size=    5394592 bytes  File Size  =    2733942 *
*Baskets :      169 : Basket Size=      32000 bytes  Compression=   1.97     *
*............................................................................*
*Br    4 :qlong     : Float_t                                                *
*Entries :  1344686 : Total  Size=    5394246 bytes  File Size  =    4535416 *
*Baskets :      169 : Basket Size=      32000 bytes  Compression=   1.19     *
*............................................................................*
*Br    5 :qshort    : Float_t                                                *
*Entries :  1344686 : Total  Size=    5394419 bytes  File Size  =    4136129 *
*Baskets :      169 : Basket Size=      32000 bytes  Compression=   1.30     *
*............................................................................*
*Br    6 :psd       : Float_t                                                *
*Entries :  1344686 : Total  Size=    5393900 bytes  File Size  =    4430619 *
*Baskets :      169 : Basket Size=      32000 bytes  Compression=   1.22     *
*............................................................................*
```

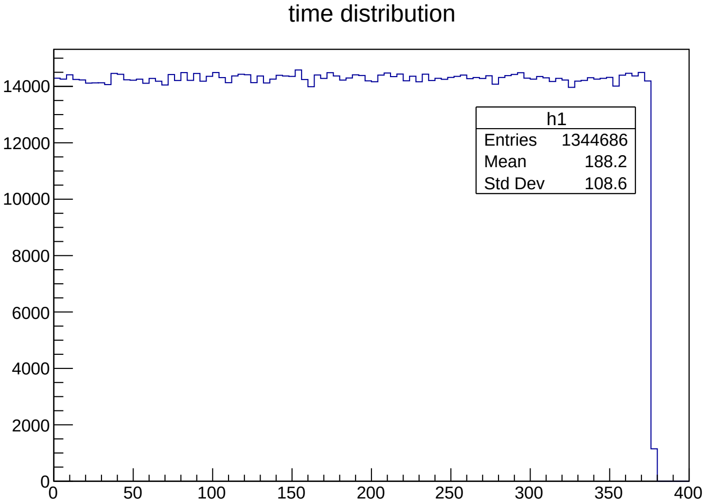
Plotting the event time

```
root [6] nt->Draw("time")
```



Filling an histogram with the event time

```
root [1] TH1D *h1 = new TH1D("h1",
                    "time_distribution", 100, 0, 400)
root [2] nt->Draw("time>>h1")
```

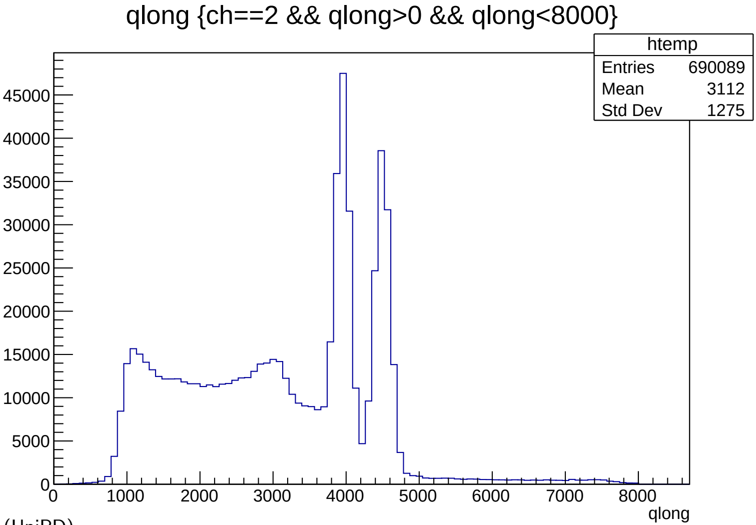


Histogram frequently used functions

- fill an histogram with a weight
- `Fill(value, weight);`
- insert manually an histogram bin content
- `SetBinContent(bin, content);`
- get the histogram bin content
- `GetBinContent(bin)j`
- sum up the histogram bin contentes form `binx1` to `binx2`
- `Integral(binx1, binx2);`
- perform a fit with a function or formula
- `Fit(func/formula,"opt");`
- clone the histogram production a copy with a new name
- `Clone("newname");`
- perform operations on an histogram
- `Add(...)/Divide(...)/Scale(...)`
- change, compatting, the bin size
- `Rebin(n)`
- reset the histogram bin content
- `Clear()/Reset()`
- return a pointer to the histogram axis object
- `GetXaxis()/GetYaxis()`

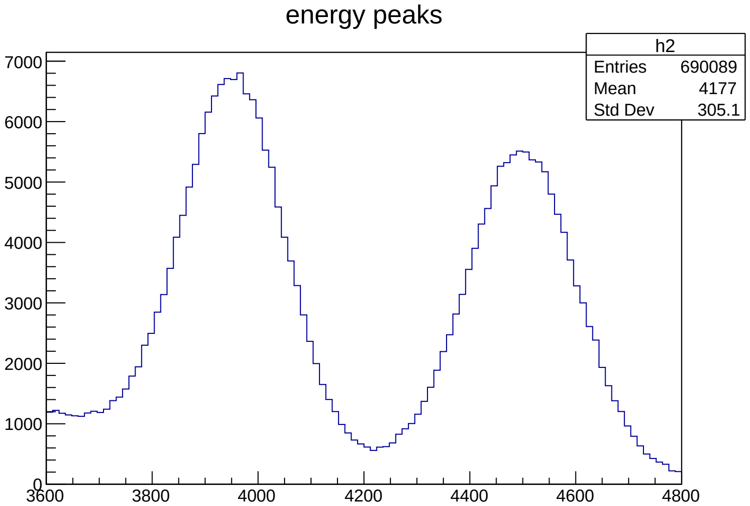
Plotting the energy spectrum

```
root [14] TH1D *h2 = new TH1D("h2", "energy peaks", 100, 3600, 4800)
root [15] nt->Draw("qlong>>h2", "ch==2 && qlong>0 && qlong<8000")
```



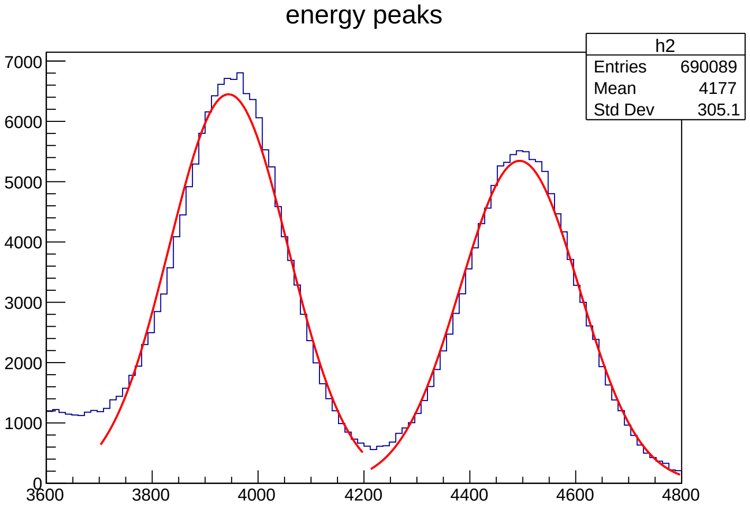
Plotting a zoomed energy spectrum

```
root [14] TH1D *h2 = new TH1D("h2", "energy peaks", 100, 3600, 4800)

root [15] nt->Draw("qlong>>h2", "ch==2 && qlong>0 && qlong<8000")
```



Fitting two gaussian peaks

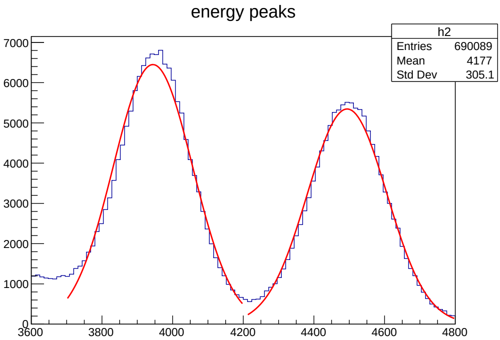
```
h2->Draw()

TF1 * m1 = new TF1("m1","gaus",3700,4200)
h2->Fit(m1,"R")
TF1 * m2 = new TF1("m2","gaus",4210,4800)
h2->Fit(m2,"R+")
```



Extracting Fit parameters

- specific methods can be invoked to extract the χ^2, number of degree of freedom and function parameters
  - `GetChisquare()`
  - `GetParameter(parameter_index)`
  - `GetParError(parameter_index)`

```
root [7] m1->GetChisquare()
(Double_t) 1119.94
root [8] m1->GetParameter(0)
(Double_t) 6449.56
root [9] m1->GetParameter(1)
(Double_t) 3944.21
root [10] m1->GetParError(1)
(Double_t) 0.320995

m2->GetChisquare()
(Double_t) 657.712
// The Gaussian mean
m2->GetParameter(1)
(Double_t) 4494.04
m2->GetParError(1)
(Double_t) 0.333256
// The Gaussian sigma
m2->GetParameter(2)
(Double_t) 112.426
m2->GetParError(2)
(Double_t) 0.288038
```



Get Histogram and Function Integral

- the histograms is organzied in bins, therefore

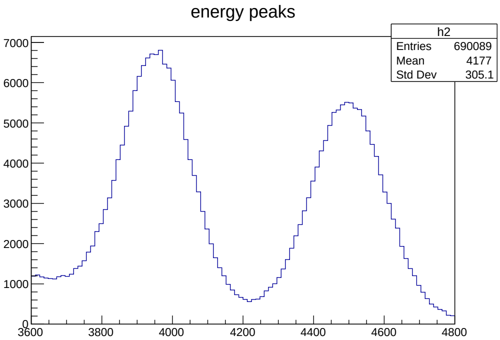
1▷ get the bin number corresponding to a specific value

```
h->GetXaxis()->FindBin("x_value")
```

2▷ use the bin number to compute the integral

```
h2->Integral(start_bin, end_bin)
```

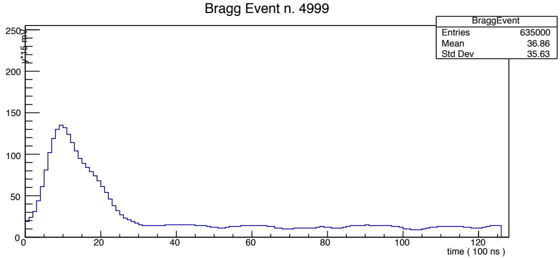
```
h2->GetXaxis()->FindBin(3700)
(Int_t) 9
h2->GetXaxis()->FindBin(4200)
(Int_t) 51
root [26] h2->Integral(9,51)
(Double_t) 149440.

h2->GetXaxis()->FindBin(4210)
(Int_t) 51
h2->GetXaxis()->FindBin(4800)
(Int_t) 101
h2->Integral(51,101)
(Double_t) 145435.
```



Bragg data analysis

- The Bragg chamber data acquisition system

```
root [0] TFile * f = new TFile("C4_Camberra2022.root")
(TFile *) 0x32d6470
root [1] .ls
TFile**          C4_Camberra2022.root
 TFile*          C4_Camberra2022.root
  KEY: TTree     bragg;1 Bragg Signals
  KEY: TH1F      BraggEvent;1    Bragg Event n. 4999

root [1] TFile * f = new TFile("C4_Camberra2022.root")

root [2] TCanvas *csig = new TCanvas("csig","Bragg Last Event", 400, 200)

root [3] BraggEvent->Draw("same")
root [4] csig->Print("bragg_last_event.pdf")
Info in <TCanvas::Print>: pdf file bragg_last_event.pdf has been created
```



Bragg ROOT file

- The TTrees has the following structure:

```
root [3] bragg->Print()

******************************************************************************
*Tree    :bragg     : Bragg Signals                                          *
*Entries :        0 : Total =         1284531 bytes  File  Size =     441061 *
*        :          : Tree compression factor =   2.91                       *
******************************************************************************
*Br    0 :signals   : s[128]/s                                               *
*Entries :     5000 : Total  Size=    1284185 bytes  File Size  =     440335 *
*Baskets :       41 : Basket Size=      32000 bytes  Compression=   2.91     *
*............................................................................*
```

- event waveforms are packed in an array with size of 128
- we need to define an array of shorts to hold the waveforms to be plotted

```
struct bragg_signal {
    short int s[128];
};
```

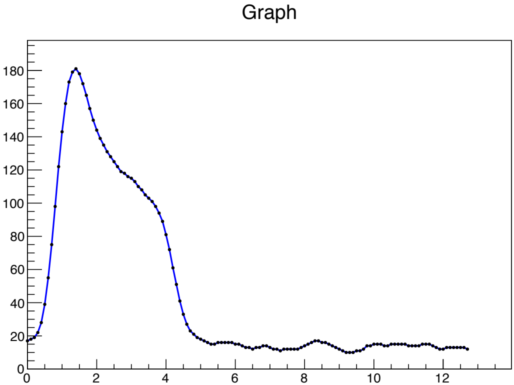
Plotting the single event waveform : 1

```
// Define a buffer to store the single waveforms
bragg_signal waveform;

TFile *fin=new TFile(filename.c_str());
if (!fin->IsOpen()) {
     std::cerr << "file not found!\n";
     return;
}

// Get a link to the TTree
TTree *tree = (TTree*) fin->Get("bragg");
if (!tree) {
     std::cerr << "Bragg TTree not found!\n";
     return;
}

TBranch * br = tree->GetBranch("signals");
if ( ! br ) {
     std::cerr << "TTree Branch signals not found!\n";
     return;
}

// Link the local buffer waveform to the branch
br->SetAddress( & waveform );
```

Plotting the single event waveform : 2

```
// Load the desired event in the data buffer
br->GetEntry( event );

// Copy the data to two arrays of float
float x[128] = {0.0};
float y[128] = {0.0};
for (auto i=0; i<128; ++i) {
     x[i] = i*0.1;
     y[i] = waveform.s[i];
}

TGraph * g =
   new TGraph(128, x, y);
g->SetMarkerStyle(7);
g->SetLineColor(4);
g->SetLineWidth(2);

g->Draw("apl");
```

From `PlotSignals.C`



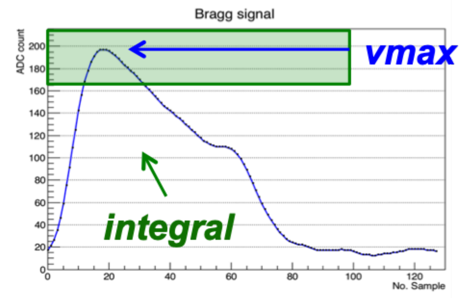
Creating the TNtuple for data analysis

```
// Open a new output file to store the TNtuple
TFile * fout = new TFile("AnaBragg.root", "RECREATE");

// Create a new TNtuple with name 'nt' and the following variables:
int i; // Event counter
float vmax; // baseline maximum value
float integral; // charge integral
float width; // signals time width
float bl;    // single event baseline
TNtuple * nt = new TNtuple("nt","","ev:vmax:integral:width:baseline");

// LOOP over the Tree Signal Events
for (int i=0; i<maxev; i++) {

    br->GetEntry(i); // Load the event variables

    // Reset variables values
    bl=0; integral=0; vmax=0; width=0;

    // Here we calculate all the values
    ...
    ...

    nt->Fill(i,vmax,integral,width,bl);
}

// Data are flushed in the file which is finally closed
fout->Write();
fout->Close();
```

The AnaBragg.C students' ROOT macro

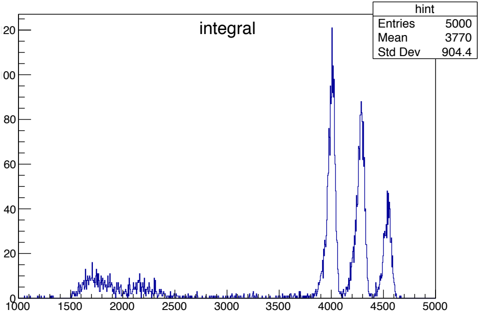
- reads in the Bragg waveforms which are registered in the Bragg tree
- the simple waveform is analyzed and the following observables are computed:
  - signal integral
  - maximum voltage
  - baseline constant value
- values are coded in a TNtuple and written to a ROOT file

```
.L ../macros/AnaBragg.C
AnaBragg("C4_Camberra2022.root")
Number of events in file : 5000
5000 events analyzed...
(int) 0
```



Plotting the Bragg TNtuple data

```
TH1F * hint = new TH1F("hint", "integral",
                        800, 1000, 5000)
nt->Draw("integral>>hint")
```




```
nt->Draw("vmax:integral",
         "integral<5000 && vmax<200")
```