

# Introduction to Analysis with ROOT

SAINTS 2021 - J.Smallcombe

## Introduction

The following exercises will take you through a basic introduction to the CERN ROOT library & program. ROOT is based on the C++ programming language. While an in depth knowledge of C++ is not required to begin learning ROOT, C++ guides and resources will aid in understanding the syntax etc used (<https://www.cplusplus.com/>)

“ROOT is an object-oriented framework aimed at solving the data analysis challenges of high-energy physics.” - This means it is useful box of tricks with many functions and definitions of special variable types defined specifically with the analysis of event-based experiments utilising radiation detector.

## Resources

ROOT is very well documented and you are encouraged to make use of the following resources

<https://root.cern/manual/basics/> - Detailed step-by-step manual

<https://root.cern.ch/doc/master/classTH1.html> - Detailed Class Documentation. The contain definitions of class functions with clearly worded explanations and examples.

<https://root.cern.ch/howtos> - Introductory Guides and Tutorials

[https://root.cern/doc/master/group\\_\\_Tutorials.html](https://root.cern/doc/master/group__Tutorials.html) - Well Commented Example Codes

<https://root-forum.cern.ch/> - Active forum moderated by ROOT lead programmers

## Starting ROOT, the Interpreter

From the command line, to start a new root session simply enter “`root`”:

```
user@pc:~$ root
```

You will be presented with another command prompt that looks like `root [0]` you are now running an interactive ROOT session. The ROOT’s interactive “interpreter” environment is a powerful live coding environment in which you can directly input C++ commands (e.g.. declare variables and use them), load and compile pre-written code and interact with graphical systems.

To exit a ROOT session enter the command `.q` (note the dot/period in the command)

```
root [0] .q
```

You can enter a series of commands by saving them as a .C script file, which can either be executed by passing the file as a command line argument when the starting ROOT or by using the `.x` command within ROOT

```
user@pc:~$ root ScriptFile.C
root [0] .x ScriptFile.C
```

## Histogram Class

A [TH1](#) is the basic one-dimensional histogram class in ROOT. To create a histogram we must use one of the child classes of the TH1 (TH1D/TH1F/TH1S etc). Each uses a specific data type to store the array of histogram bin contents. In general it is fine to use the double-type **TH1D**.

<https://root.cern.ch/doc/master/classTH1.html#creating-histograms>

To create a histogram we will use the class constructor of the form:

**TH1D("name", "title", Nb , Min , Max )**

"name" - a text value entered in quotes (string-type) used in ROOT's memory table

"title" - string-type defining the display title of the histogram

Nb – Integer-type specifying the number of bins

Min – double-type, the lower edge of the histogram range

Max – double-type, the upper edge of the histogram range

```
root [1] TH1D VariableName("HistName", "HistTitle", 50 , 0.0 , 10.0 )
```

**Try creating a histogram in ROOT now.** To add entries to the histogram use the [TH1::Fill](#) method

```
root [2] VariableName.Fill(5.3)
```

```
root [3] VariableName.Fill(3)
```

```
root [4] VariableName.Fill(2.4)
```

To view the histogram use the [TH1::Draw](#) method

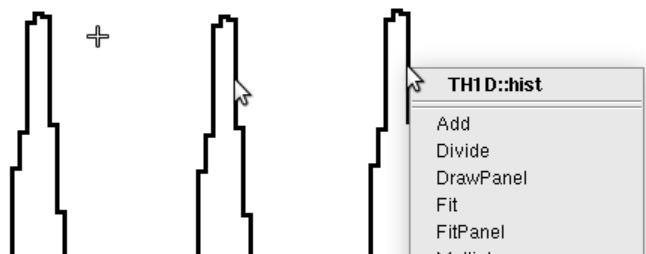
```
root [5] VariableName.Draw()
```

A full list of histogram commands with details can be found at:

<https://root.cern.ch/doc/master/classTH1.html#pub-methods>

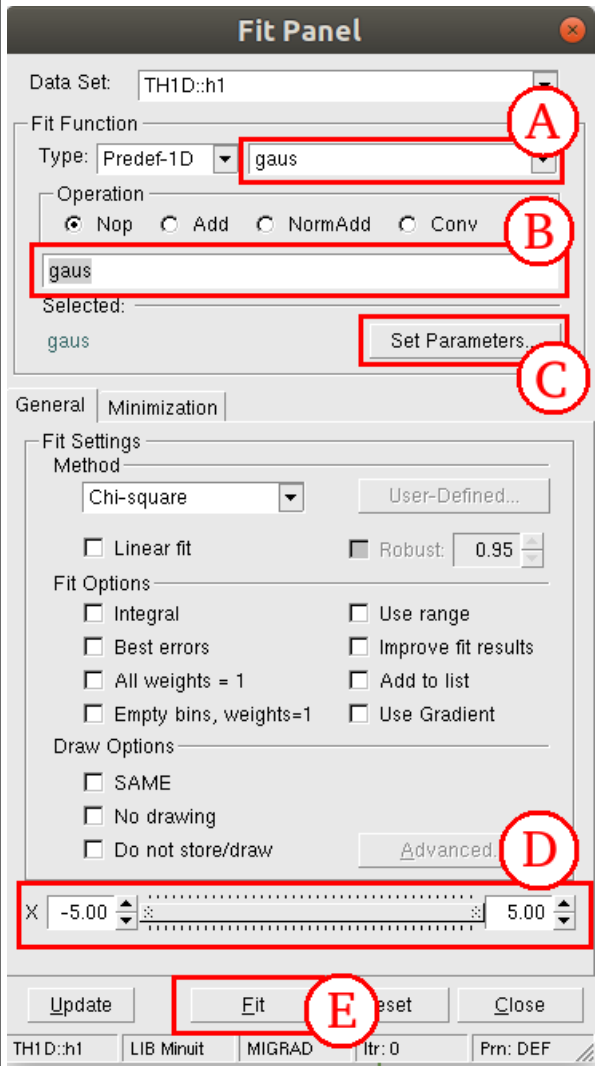
## Fit Panel

**Run the script TestScript.C to prepare and draw an example histogram.** To start fitting the histogram using the the Fit Panel graphical user interface (GUI). Move the mouse cursor over the line of the displayed histogram. The cursor should change from a cross-hair to a normal pointer. Right click the histogram line. At the top of the menu the type and name of the object selected is indicated. Select the option FitPanel and a new window will open.



The Fit Panel window contains a lot of functions, but for now focus only on the elements highlighted below.

Using the fit panel, try to fit a predefined ROOT function to the example histogram and then try to fit a user-defined straight line function to only part of the histogram, before continuing.



A ) Dropdown menu listing predefined functions in ROOT's mathematical library

B) Formula window. This window defines the function of x to be fit. Input may consist of standard c++ expressions, using the variable x and numbers in square brackets for free parameters e.g.

$$\text{sqrt}(2)*\exp(-x/[0])$$

$$[0] + x*[1] + \text{pow}(x,2)*[2]$$

Or predefined ROOT functions e.g.

gaus  
expo  
pol3

Or a combination

$$[0] + x*[1] + \text{gaus}(2) + \text{expo}(5)$$

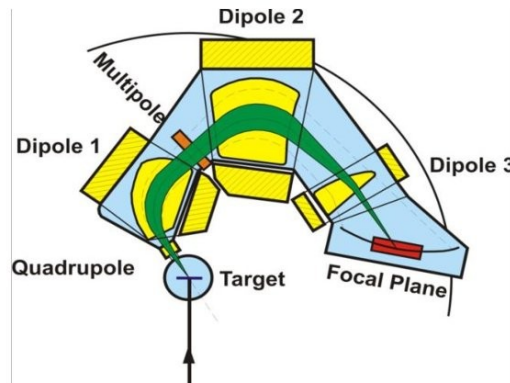
(Note: When not in isolation, ROOT functions must be given a number indicating where in the list of parameters it's own parameters start.)

C) Open another window for setting initial parameter values and constraints

D) Set the range over which to fit the data. Corresponding lines are displayed in the histogram window

E) Perform the fit. The result is drawn in the histogram window and details are printed to the terminal.

## Exercise 1 – Fitting a (p,p') spectrum



In this exercise we will study a histogram containing data from the Munich Q3D spectrometer. In this experiment target nuclei were excited by bombardment of a proton beam from the (now defunct) Munich tandem accelerator. By precisely measuring the energy of the inelastically scattered protons on the spectrometer focal plane, the corresponding excitation energy of the target nuclei can be calculated.

**Run the ROOT script LoadTH1Script.C** to load and draw a histogram of excitation energy.

### Fit Panel Fitting

**Using the Fit Panel**, fit the first peak of the spectrum as a Gaussian peak (Normal distribution), either by using the predefined functions or by entering the formula manually. (Be sure to set the range to select only the peak of interest for the fit.)

**Question 1) What is the approximate energy of the nuclear state corresponding to the first peak? (What is the peaks centroid?)**

The target in this data is a mass 28 isotope. Comparing your measured state energy to published data, determine which nucleus you are looking at. Use your preferred data base or [nndc https://www.nndc.bnl.gov/nudat2/indx\\_adopted.jsp](https://www.nndc.bnl.gov/nudat2/indx_adopted.jsp).

Fit the energy of a few more peaks to be sure.

**Question 2) What is the target nucleus?**

The width of the peaks observed is a product of both the resolution of the detector system and of the state width itself. Assume the resolution of the detector dominates the peak you have fit.

**Question 3) What is the resolution of spectrometer (at this energy) in %**

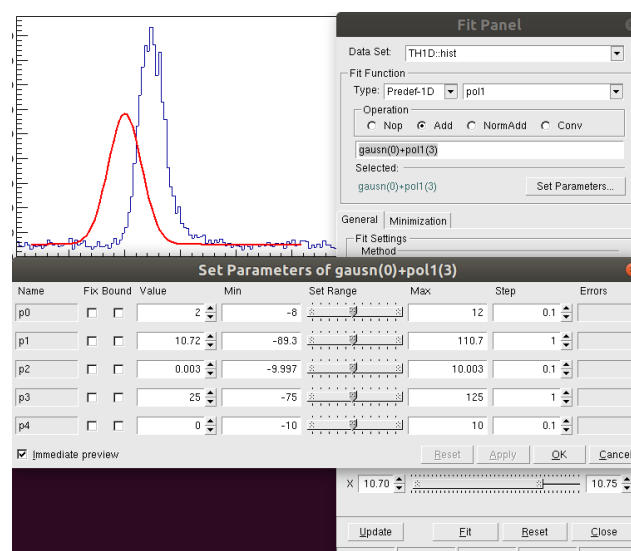
 %

Peaks are said to be resolvable when they are separated by their FWHM.

**Question 4) What is the closest in energy two states can be, in order to be resolvable with this equipment?**

The area of each peak is proportional to cross-section for population of the nuclear states. In order to determine the relative cross-sections for different states one must determine the number of counts in the corresponding peaks. However, the spectroscopic peaks of interest sit on a continuum background of events from elastic and multiply-scattered protons which cannot be identified on an event-by-event basis. In order to determine the peak's area we must take account of the background as we wish only to measure the peak counts, and determine an appropriate statistical uncertainty.

**Using the Fit Panel,** fit the peaks of the spectrum as a **Gaussian + a linear background** , either by using the predefined functions or by entering the formula manually (Ensure the range is set wide enough to fit the background). You will need to make use of the “Set Parameters” window in Fit Panel to set initial parameter values (and constraints) in order to achieve a successful fit.



**Question 5) What is the area (including uncertainty) of a Gaussian fit to the smallest peak in the spectrum?**

**Note:** The area of a Gaussian can be calculated from the sigma and height parameters, however this necessitates correct propagations of parameter uncertainties and covariances. By fitting with a **normalised-Gaussian (gaussn)** an area parameter is fit directly allowing direct extraction of errors from the fit result.

**Second Note:** If your results seems unphysical, remember the Y-axis of a histogram is always in units of “Counts per Bin” and not “Counts per X-value”. So you must divide by the bin width to correct the units of your area to counts.

## Histogram Integration

In order to call the following functions of the histogram object, you can either add additional lines **LoadTH1Script.C** script or type commands directly into the interpreter prompt of a ROOT session after running the script.

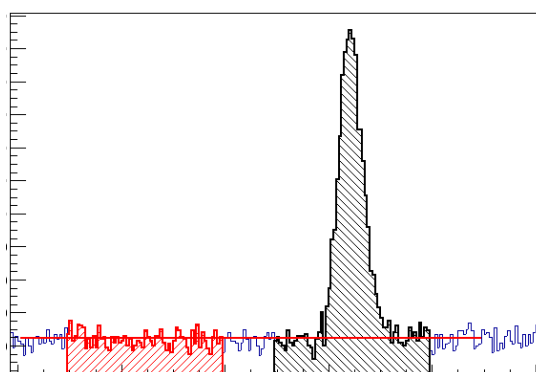
In the previous section, you may have noticed that the peaks in our spectrum are not perfectly Gaussian and the fits we performed were approximate. In order to extract a more accurate peak area one should either using an improved fit function or a counting method which is insensitive to the precise peak shape.

One possible method is to simply integrate the counts of the histogram and determine the background contribution. The integral of the bins of histogram can be extracted with the [TH1::Integral](#) function.

```
root [6] double ReturnedNumber = HistogramPointer->Integral(BinStart,BinFin);
```

As with many TH1 functions, [TH1::Integral](#) requires input of bin number and not X-value/coordinate. One can use the function [TH1::FindBin](#) to convert from value of interest (i.e. Energy) to bin number.

```
root [7] int ReturnedBinNumber = HistogramPointer->FindBin(XValue);
```



If we assuming that the background is constant, the peak area may be taken as difference between the integral across the peak and an integral across the same number of bins where there is no peak.

Remember the statistical uncertainty on a count  $N$  may be taken as  $\sqrt{N}$  and for addition/subtraction uncertainties are combined in quadrature.

**Question 6) Using the integration method, what is the area (including uncertainty) of the smallest peak in the spectrum?**

While the tails of the peak are technically infinite, one may approximation to a few sigma, depending on [statistical precision](#).

**Question 7) An integration range of how many sigma would yield an error equivalent to the statistical error determined in question 6? [Useful Link](#)**

  $\sigma$ 

**Note:** If you wish to calculate exactly make use of the TMath library function [TMath:ErfInverse\(\)](#)

## Histogram Background Subtraction

Alternatively to the above method we may try to calculate the background of histogram and remove it. This is useful when the background is not (approximately) uniform. One method to determine a background is by iterative smoothing. The **TSpectrum** class in ROOT is a useful class for estimating continuum background of spectral histograms.

<https://root.cern.ch/doc/master/classTSpectrum.html>

Create a TSpectrum object (use the default constructor, which requires no arguments)

```
root [8] TSpectrum MyVariable;
```

Use the TSpectrum function [TSpectrum::Background](#) which creates a new histogram (and returns a pointer to it) based on a provided histogram which is iteratively smoothed a choose number of times.

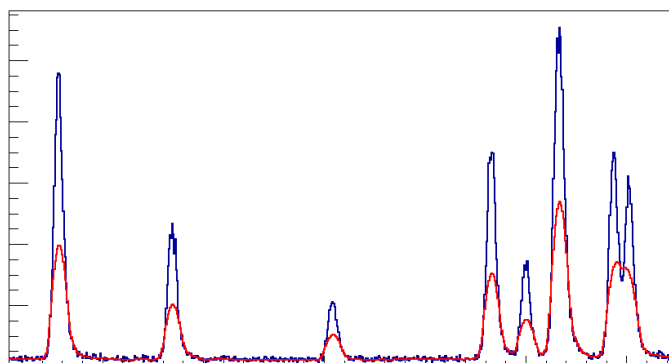
```
root [9] TH1* PointerReturned = MyVariable.Background(HistogramPointer, NumberOfIterations);
```

**Note:** We can use the ampersand operator **&** to provide a pointer when we have a local variable **&LocalHistogram**.

Draw the returned histogram using the option "same" in order to draw a new histogram over a currently drawn histogram for easier comparison.

```
root [10] HistogramPointer->Draw();  
root [11] OtherHistPointer->Draw("same");
```

**Note:** To help distinguish histograms drawn together use [TH1::SetLineColor](#) e.g. **MyHist.SetLineColor(EGColor::kRed);**



**Question 8) What value for Number Of Iterations produces a smooth background under the small peak measured in Question 5?**

Now you have a histogram representing the background of the spectrum you can subtract it using the [TH1::Add](#) function with a **-1** scaling factor.

```
root [12] HistogramPointer->Add(HistogramToAddPointer, ScalingFactor);
```

Integrate the peak area again, but using the [TH1::IntegralAndError](#) which returns the statistical error on the integrated bins as well. (A double-type variable must be created to store the error value and passed to function)

```
root [13] double ErrorDouble;
```

```
root [14] double Integral = HistogramPointer->IntegralAndError(BinStart,BinFin,ErrorDouble);
```

**Question 9) How does your answer using this method compare to Question 6?**

With IntegralAndError ROOT has calculated the statistical uncertainty of each bin being integrated with the  $\sqrt{N}$  rule and summed them correctly. But, following the subtraction the statistical error on the bins **should be larger** than  $\sqrt{N}$ . We have to tell ROOT to calculate (and store) the bin errors **before** we Add/Subtract histograms. (This is not done by default as it doubles the size of the histogram in memory and is frequently not required.)

The function [TH1::Sumw2](#) must be called on **either** of the histograms **before** the subtraction.

```
root [15] HistogramPointer->Sumw2();
```

```
root [16] HistogramPointer->Add(HistogramToAddPointer, ScalingFactor);
```

Restart ROOT and run the **LoadTH1Script.C** script again to re-load the original histogram.

**Note:** The [TH1::Sumw2](#) function instructs root to “Create structure to store sum of squares of weights.” i.e. create an array and store the bin errors. If either histogram used in the [TH1::Add](#) function had a stored array of errors, then errors will be summed for the resultant histogram bin-by-bin.

**Question 10) Following the correct use of Sumw2 and background subtraction, how does your peak area and statistical uncertainty compare to Question 6?**

**Note:** We have only determined statistical uncertainties. All methods of dealing with background and extracting spectroscopic peak areas have their own systematic uncertainties associated with the assumptions and choices made e.g. Peak function to fit, range to integrate etc. One should always attempt to quantify the systematic uncertainty of your chosen method, at least to determine if it is significant compared to statistical uncertainty (which often dominates).



## Exercise 1 - Extra Problems

**Question 11) What is the ratio of the two peaks at approx 10.95 MeV**

A function which is a convolution of an exponential and a Gaussian provides a better description of the true peak shape ([Exponentially modified Gaussian distribution](#)). Fit such a function to the first peak in the spectrum.

**Question 12) What value do you determine for the exponential tail parameter tau?**

**Question 13) Based on your answer to questions 3. What is the approximate sensitivity limit to nuclear lifetimes of the equipment?**

## Exercise 2 – TTrees and Automated Fitting

A TTree object is clever list. It is a data storage object which is highly adaptable. You can think of it as big table, defined by a series of named column (called a **branch**) each containing specific type of data and each row (**entry**) consists of a complete set of values for each.

TTree Branches can contain simple variables, arrays, complex objects like TH1s, TF1 etc and even user defined classes.

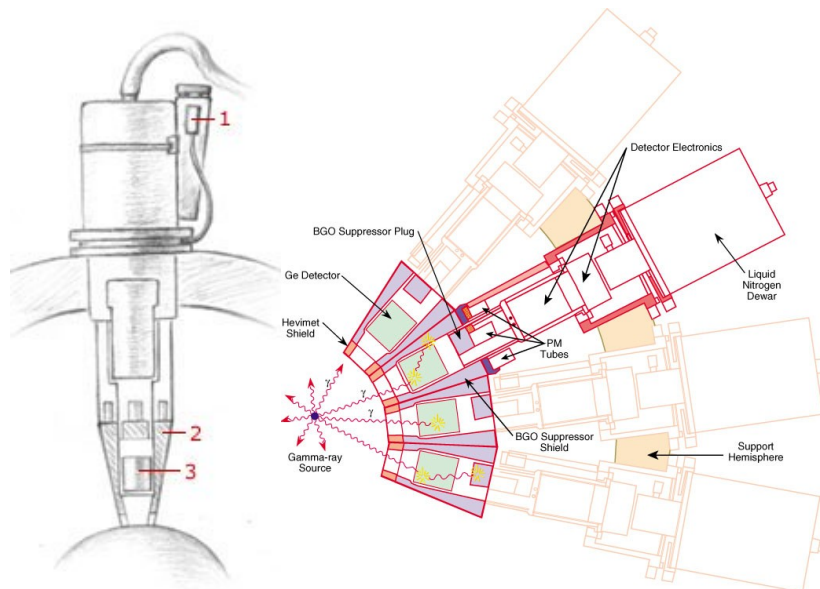
Branches can be split to allow direct access to member variables if the branch-type is a complex object. These sub-branches are called **Leafs**. Branches of a basic data type have one leaf, which is the branch.

One can loop over a TTree one entry at a time or quickly access and plot data from individual leaves or even functions of branch objects for a range of entries.

<https://root.cern.ch/doc/master/classTTree.html>

<https://root.cern.ch/root/html/doc/guides/users-guide/Trees.html>

In this exercise we will be looking at raw data from a set of Compton-suppressed High Purity Germanium (HPGe) gamma-ray detectors. The file **TTreeOne.root** contains data from a standard  $^{60}\text{Co}$  calibration source saved as a TTree. Each entry of the TTree corresponds to data from the digitizer of one of the detector which recorded a signal (of magnitude greater than some predefined threshold value).



### TTree and TFile Inspection

Let us start by opening the .root file and inspecting the TTree. From the command line run ROOT giving the file as an argument to open it with as a TFile object. ROOT should confirm it has opened the file. Use the command `.pwd` in the ROOT interpreter to confirm the interpreter focus is "in" the file. Then use the command `.ls` to list the contents of the file.

```
user@pc:~$ root TTreeOne.root
Attaching file TTreeOne.root as _file0...
root [0] .pwd
root [1] .ls
```

You should see a “KEY” of type TTree followed by a name for this object. Keys tell us objects which are stored in the .root file on the disk, rather than “live” objects loaded in memory.

[**Note:** Key’s show a namecycle (name;cycle) where the integer cycle value separates multiple objects written to disk with the same name. When there is only one cycle this part can be ignored and simply use the name alone.]

While the focus of the interpreter is a particular file/directory the “names” of the objects/keys in that directory can be used as pointers. Use the name-pointer to call the [TTree::GetEntries](#) function.

```
root [2] TTreeKeyName->GetEntries();
```

If you now call the `.ls` command again you will see there is now an TTree OBJ (object) with an associated memory address (in hexadecimal), as the key has loaded the saved TTree from the disk into memory so that the object can be used.

[**Remember**, in general there is a difference between a ROOT “name”, which is a character string ROOT can interpret as a pointer to an object, and the identifier we use in code for an object/variable we declare i.e. `double MyDouble`]

To see overview information about the TTree branches and leaves use the function [TTree::Print](#)

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

**Question 1) How many branches does the tree have? Which is the largest and which the most compressed?**

You can look through the entries of the tree individually with [TTree::Show](#) function and you can look through multiple entries with [TTree::Scan](#), which can also be used to show only selected leaves etc.

```
root [4] TTreePointer->Show(EntryNumber)
```

```
root [5] TTreePointer->Scan()
```

```
root [6] TTreePointer->Scan("LeafName")
```

```
root [7] TtreePointer->Scan("LeafName:OtherLeafName")
```

**Question 2) How many of the first 50 tree entries have ID value 2 and a positive ShieldCharge?**

**Question 3) What is the TimeStamp of entry 512?**

## TTree Drawing

If we want to look at the data as a whole we probably want to parse it into histograms, for this we can use the `TTree::Draw` command.

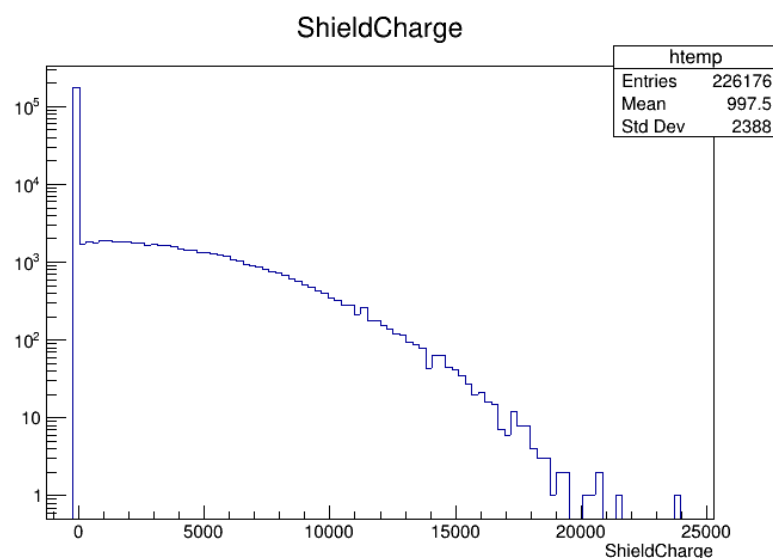
We start by giving the command one input, a string-type (enter with quotes), which specifies the value to histogram. **The values can be a leaf, a function of a branch, or simple maths with leaf values.**

**Note:** We can give multiple inputs separated by colons to look at correlations in 2 & 3 dimensional plots e.g. "Value1:Value2", but will only look at 1D for now.

For now we omit the other function inputs accepting the default values, this will parse all entries and fill a histogram with one entry for each entry of the tree.

Try drawing each of the tree leaves, ROOT will automatically decide an appropriate range and binning of the histogram, but these won't necessarily be optimum.

```
root [8] TTreePointer->Draw("LeafName");
```



The data stored in the branch TimeStamp is in units of **ns**.

**Question 4) What was the event rate of recorded data?**

To manually set the binning of the drawn histogram you can add `>>(Nb , Min , Max)` to the value selection string, where the inputs have the same meaning as in the TH1 constructor.

```
root [9] TTreePointer->Draw("LeafName>>(Nb , Min , Max)");
```

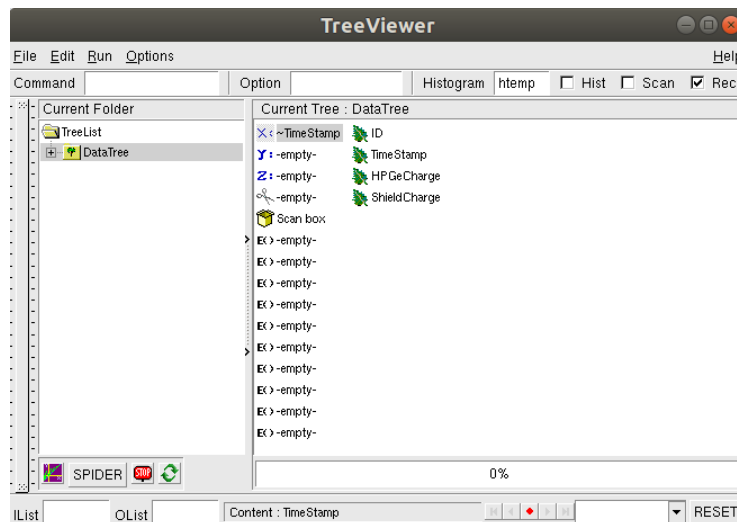
**Question 5) What binning and range produces a decent gamma-ray spectrum?**

If we want access to the histogram, in order to issue further commands, you place a "name" before the binning brackets i.e. `>>HistName(Nb , Min , Max)`, this will assign the histogram's "name" in the

ROOT memory system, which will act as a pointer in the ROOT interpreter environment or via special ROOT functions. More generally we can use only the “name” (no binning info) of a histogram that has already been created. The latter is more appropriate for compilable scripts and codes. **An example is given in DrawTTreeScript.C**

```
root [10] TTreePointer->Draw("LeafName>>HistName(Nb , Min , Max)");
root [11] HistName->SetLineColor(2);
root [12] TH1D OtherHist("NamedHist","Title", Nb , Min , Max);
root [13] TTreePointer->Draw("LeafName>>NamedHist");
root [14] OtherHist.Rebin(2);
```

**Note:** TTree Drawing can also be achieved via the [TTreeView](#) GUI class. We will not cover this here, but it can be accessed via `TTreePointer->StartViewer()`.



To select a subset of the data you can provide a second argument to the draw function. The selection (or cut) can be given as a string-type input using the same selection of leaf names, basic maths and functions as for first input, but combined with C++ logical operators e.g.

<	Less Than
>=	Greater or equal to
==	Equal
	Or
&&	And

```
root [15] TTree->Draw("Leaf","OtherLeaf>12");
root [16] TTree->Draw("Leaf","OtherLeaf>=10&&OtherLeaf<=20");
root [17] TTree->Draw("Leaf","Branch.Function<OtherBranch.Member");
```

**(Remember** In TTreeOne.root the branches are basic data types not complex objects, so they have no member variables, only a single leaf which is equivalent to the branch.)

**Note:** Selections may also be inputs as [TCut](#) objects and more complex “graphical” cuts on two variables (such as selecting a locus on a particle identification spectrum) can be input as a [TCutG](#) object. We will not cover this here.

**Question 6) How many events with an ID value of 1 have a ShieldCharge value less than 0?**

The third input is a drawing options string-type input to which we can pass the same arguments we give to `TH1::Draw` function, such as the `"same"` option which overlays the histogram to be drawn over any previously draw (in the currently selected drawing pad).

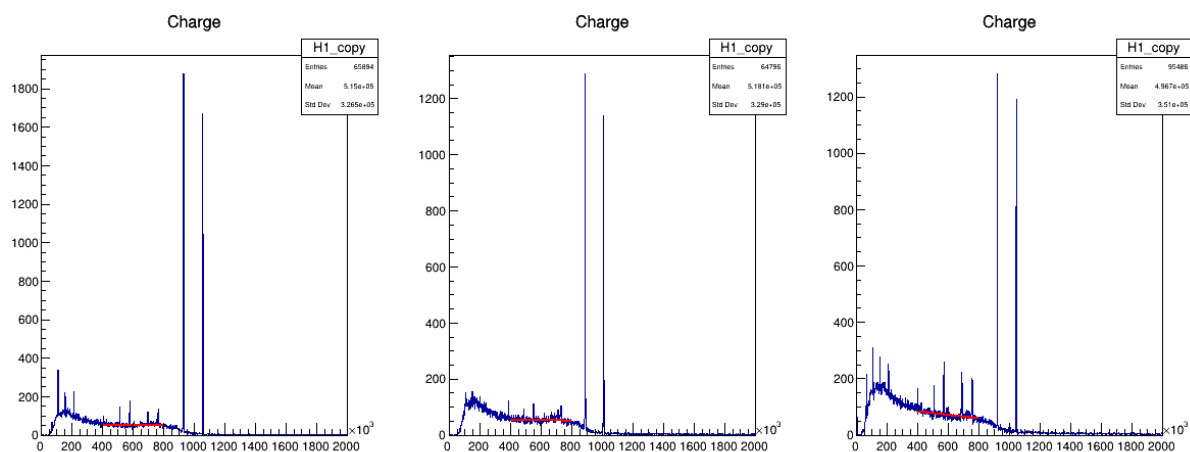
Draw a HPGeCharge histogram (based on your answer to Question 5) and then overlay a HPGeCharge for events with ShieldCharge greater than 0

**Question 7) What do you deduce about ShieldCharge?**

### Scripted Fitting

For a detector calibration we might have >1000 similar spectra (one for each channel) which we need to fit. It is clearly impractical to do this manually with the FitPanel, so we must automate/script the process.

Run the script **ScriptedFitting.C** (in a fresh ROOT session). This will produce 3 different histograms from TTreeOne.root, perform a fit to each and display the results.



Look at the lines of code in **ScriptedFitting.C** and read the comments. There are a few new concepts: The use of the `TCanvas` class to control the graphical drawing environment, the use of the `stringstream` class to parse numbers to text and the use of the `TF1` class for fitting in code.

The `TF1` class is used to hold the 1 dimensional function formula and parameters for functions used in fitting. (The FitPanel uses the TF1 class behind the scenes.)

<https://root.cern.ch/doc/master/classTF1.html#F2>

To create a TF1 we have used the class constructor of the form:

**TF1("name", "formula", Min , Max )**

"name" - string-type used in ROOT's memory table

"formula" - string-type defining function – As FitPanel input; may consist of standard c++ expressions, using the variable x and numbers in square brackets for free parameters or predefined ROOT functions or a combination.

Min – double-type, the lower edge of the fit range

Max – double-type, the upper edge of the fit range

```
root [18] TF1 MyFunction("FunctionOne", "[0]*exp(-x/[1])", 0.0 , 10.0 );
```

(There are a number of other ways pass a formula to TF1, including c++ functions and classes ,which are useful for more complex functions. These can be found on the [TF1 class page](#).)

To fit the TF1 to data we call the fit function of the object holding the data, in this case a histogram, [TH1::Fit](#) .

**TH1::Fit(TF1Pointer,"options")**

The fit functions takes input of a pointer to a TF1 and an options string-type. This option string is a series of single characters that control the fitting process; it may be omitted to use the defaults.

Some key option characters:

R	Use the current range of the TF1, NOT of the histogram.
Q	Quiet - Do not output information to terminal
+	Add a copy of TF1 to the histogram in addition to any that have previous been added. By default only the latest is stored.
0	Do not draw the histogram and fit.

**Note:** We can use the ampersand operator **&** to provide a pointer input to the fit function when we have a local TF1 variable `MyHistogram.Fit(&MyFunction,"RQ")`

After calling the fit function the output of running the fit minimiser is displayed in the terminal (identical to FitPanel) and the final parameter values and associated errors are stored in the provided TF1 (overwriting the initial parameters). By default the histogram will be drawn, along with the fit result, and a copy of the TF1 is saved to the histogram object.

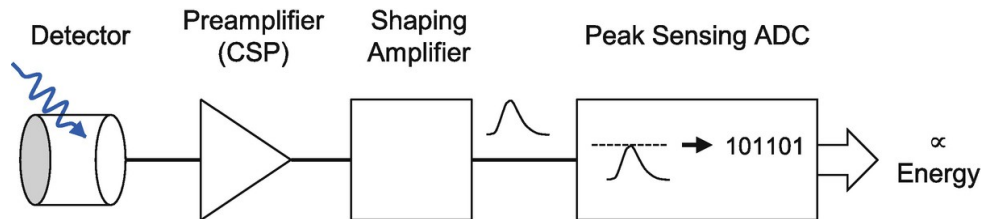
```
root [19] MyHistogram.Fit(&MyFunction,"R")
FCN=1221.78 FROM MIGRAD      STATUS=CONVERGED      37 CALLS      38 TOTAL
          EDM=5.79061e-20    STRATEGY= 1      ERROR MATRIX ACCURATE
EXT  PARAMETER
NO.   NAME      VALUE          ERROR        STEP          FIRST
  1   p0        4.11238e+01    1.58436e+00    4.95962e-03    4.58450e-10
  2   p1        -1.21971e-05    2.57611e-06    1.39764e-07    3.98746e-04
```

As discussed in the first session, it is important to provide reasonable initial guesses for parameters. This can be done for all parameters at once with or individually with [TF1::SetParameters](#) and [TF1::SetParameter](#)

```
root [20] MyFunction.SetParameter(0,0.12);
```

We will see that one can use TH1 functions to interrogate the histogram in order to get initial approximate values for parameters of our fit function.

The histograms show the gamma-ray spectra for a standard  $^{60}\text{Co}$  source. These are given in uncalibrated charge, directly from the experiment's digitizer. As all detectors and electronics are slightly different, the exact charge recorded in each channel for the same energy particle is slightly different. By fitting the peak centroids and comparing to known energy we can determine the calibration of each channel.



Using online resources such as <https://www.nndc.bnl.gov/> find the expected gamma-ray emissions associated with the decay of  $^{60}\text{Co}$ .

**Question 8) What are the energies of the two most intense gamma-rays from a  $^{60}\text{Co}$  source?**

	keV
	keV

Using the histogram functions [TH1::GetMaximumBin](#) and [TH1::GetBinCenter](#) modify **ScriptedFitting.C** to **determine the X-value of the largest peak** in each spectra.

**Question 9) What is the approximate gain for each channel?**

**Hint: All channel gains  $\ll 1$  keV/channel.**

	keV/channel
	keV/channel
	keV/channel

This gain is pretty good, but has a maximum accuracy limited by the bin width of the histogram, not good enough for precision spectroscopy. We also need a second point to calculate any offset of the calibration (0 measured charge rarely exactly corresponds to 0 true energy due to imperfections in the hardware) and would need more than two points to correct for any non-linearity in the energy/charge calibration.

Modify **ScriptedFitting.C** in order to **fit a Gaussian + linear background to the largest 2 peaks** for each detector ID. Use your answers to questions 8 and 9 to set an appropriate fit range around the target peak for each of the 6 fits using [TF1::SetRange](#). You will need to set reasonable initial values for the parameters of the TF1 before fitting.

**See below for further fitting hints**



**Question 10) Using the results of your 2 peak fits, what is offset for each channel?**

**Hint: All channel offsets < -1 keV. See below for further hints.**

	keV
	keV
	keV

**Hint:** The following functions may be useful for estimating initial parameters: [TH1::FindBin](#), [TH1::GetBinContent](#), [TH1::GetBinWidth](#), [TH1::SetAxisRange](#), [TF1::GetParameter](#). See below for further guidance, if desired.

#### Question 10 Parameter Estimation Hints

Bin # of peak	<a href="#">TH1::GetMaximumBin</a> returns the bin with the most counts within the <b>current drawing range</b> of the histogram. To change this range we can use the function <a href="#">TH1::SetAxisRange</a> . By setting the range roughly around the target peak (so that it is the local maximum) we get the peak's bin number, rather than the absolute tallest bin.
Gaussian constant/height	Once you know the bin number of the peak, the peak height can be estimated from the bin height by calling <a href="#">TH1::GetBinContent</a> .
Gaussian mean/centroid	Use the bin number of the peak with <a href="#">TH1::GetBinCenter</a> .
Gaussian sigma/width	The peak is only a few bins wide, so the bin width is a good estimate for sigma, use <a href="#">TH1::GetBinWidth</a> . (As all bins have the same width, simply <code>GetBinWidth(1)</code> )
Polynomial P0	The content of any bin in the peak-less region between target peaks is a good estimate for the background constant. Use <a href="#">TH1::FindBin</a> to convert from X-coordinate to bin number and then <a href="#">TH1::GetBinContent</a>
Polynomial P1	The background is flat enough that a good estimate for P0 should be sufficient for a true minimum to be found. Enter 0 for P1.

**Note :** For simplicity we have used TH1 functions directly, however for bin/axis functions it is generally recommended to first use [TH1::GetXaxis](#) which returns a pointer to a TAxis class member of the histogram, which stores and controls axis data, and functions are then called of this TAxis object. This more general method is applicable for 2 and 3 dimensional TH2s and TH3s histograms, in which there are multiple axes, and many functions must be accessed via a specified TAxis.

```
root [21] LocalHistogram.GetXaxis()->GetBinCenter(10);
root [22] HistogramPointer->GetXaxis()->SetRangeUser(100.0,150.0);
root [23] TAxis* ax = LocalHistogram.GetXaxis();
root [24] ax->SetRange(ax->FindBin(100.0), 60);
root [25] ax->GetBinWidth(1);
```

**Note :** As well as the above method with using SetRange and FindMaximum bin, one can also use the [TSpectrum](#) class to identify spectroscopic peaks of a histogram. An example is given here: [https://root.cern/doc/master/peaks\\_8C.html](https://root.cern/doc/master/peaks_8C.html)

### Extra Problems

Using your 2 point calibration from questions 10 and the [TH1::Draw](#) command, draw a calibrated energy spectrum for each detector ID and sum them together with [TH1::Add](#).

**Question 11) Given the activity of the  $^{60}\text{Co}$  source is 5.4 KBq. What is the photo-peak efficiency of the system at 1332 keV? Remember from Questions 4 that TimeStamp is in units of ns.**

 %

**Hint:** Photo-peak efficiency is given by  $\text{Eff}_i = (N_i / I_i)$ , where  $N_i$  is the area of a specific known peak in the source spectrum and  $I_i$  is the expected intensity from the known nuclear decay of the source.

**Question 12) Assuming identical cylindrical detectors 20 cm from the source, what is their minimum possible radius (Given your answer to questions 11)**

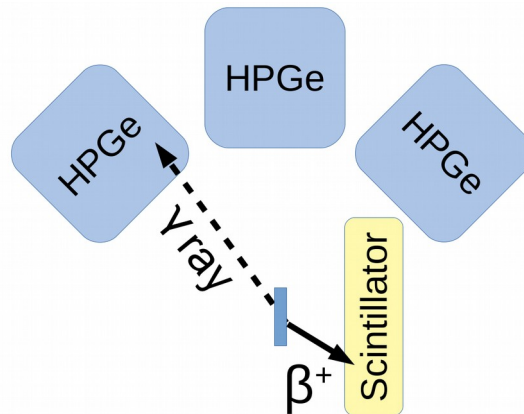
 cm

## (Extra) Exercise 3 – TTree Sorting and Lifetimes Fitting

In this exercise we will use the data stored in **TTreeTwo.root**. Open the file with a fresh ROOT session and use what you learnt in exercise 2 to probe the branches the TTree **DataTree**.

```
user@pc:~$ root TTreeTwo.root
```

This TTree contains (simplified) simulated data for a typical decay-station setup for the study of the decay of radioactive isotopes. A Scintillator detector records emitted beta particles and HPGe detectors record gamma-rays emitted from excited states of the nucleus populated in the decay.



The entries of the TTree consist of one event from each detector type which have been roughly correlated as occurring within a 0.4 us window of each other. By looking more precisely at the time difference between the beta-particle and gamma-ray we can study the lifetimes of populated states. **Draw a histogram of the time difference.**

For this exercise you can either use the [TTree::Draw](#) command you learnt previously or edit the sorting script **SortTTreeScript.C**, this script uses [TTree::SetBranchStatus](#) to connect variables into which the data from branches will be loaded and then [TTree::GetEntry](#) to sequentially load the data for each event.

**Note:** The [TTreeReader](#) class may alternatively be used to sequentially iterate event data, but will not be covered here.

**Question 1) What are the 3 primary features of the time difference spectrum?**

As each gamma ray may be from a different state with a different lifetime it is desirable to look at the time spectrum for specific gamma-ray energies. For large data sets (which may be spread over many files totalling several TB), sorting the entire tree each time you wish to adjust your gamma-ray selection (gate) is inadvisable. Instead it is useful to produce higher dimensional histograms which can be quickly re-gated.

**Note:** It is generally advisable to finely bin your histograms when sorting (though not so fine memory becomes a problem). If your spectra are too sparse after sorting, you can always reduce the number of bins using [TH1::Rebin](#), but if you require more bins the information has been lost and you must re-sort.

Create, fill and draw a [TH2](#) of time difference vs gamma ray energy.

Use the class constructor of the form:

`TH2F("name", "title", NbX, MinX, MaxX, NbY, MinY, MaxY )`

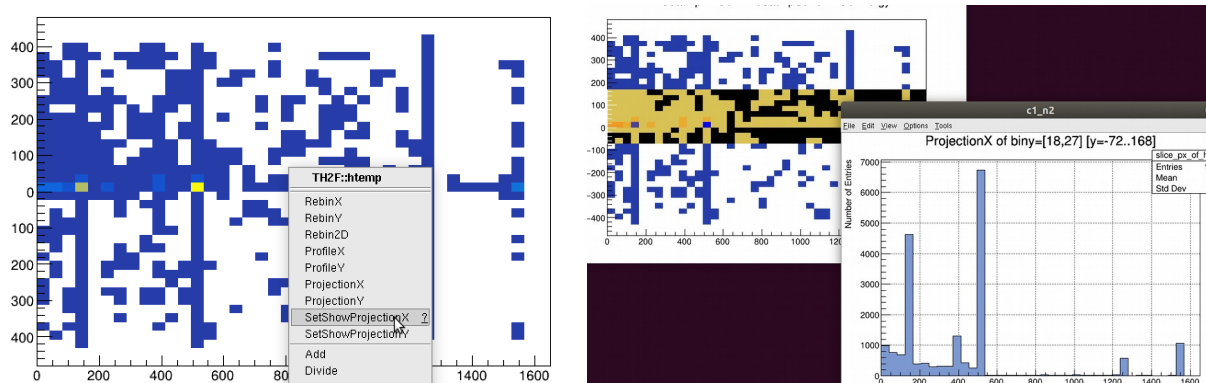
When drawing a TH2 it is advisable to use the option "col" to produce the plot in contoured colour (3D drawing options are also available).

```
root [1] TH2F HistVariable("HistName", "HistTitle", 50, 0.0, 10.0, 100, 0.0, 1000.0)
root [2] HistVariable.Fill(3,6);
root [3] HistVariable.Draw("col");
```

**Note:** The [TH2::Fill](#) function requires 2 inputs and X and a Y value.

To look at data of one axis with respect to a selection of the other axis we can use the TH2 functions: [TH2::ProjectionX](#), [TH2::ProjectionY](#), [TH2::SetShowProjectionX](#) and [TH2::SetShowProjectionY](#)

Use [TH2::SetShowProjectionX](#) from the right-click after drawing the histogram, this allows you to look at different “slices” of the matrix in the GUI by moving the mouse.



[TH2::ProjectionX](#) creates a new TH1 from the given selection and returns a pointer to it. The arguments to the projection functions are in bin number so one must first access the relevant axis member (X or Y) to determine bin number from coordinate.

```
root [4] int BinFromX = HistVariable.GetAxis()->FindBin(Xvalue);
root [5] TH1* ReturnedPointer = HistVariable.ProjectionY("RetHistName", BinFromX, BinToX);
```

Look at the time difference spectra for each gamma ray.

**Question 2) Two of the gamma rays indicate a significant lifetime. Which two?**

When the lifetime of a state is significantly shorter the intrinsic time resolution of the detectors the time peak appears to be perfectly symmetrical. The width of this peak (resolution of the system) represents a reasonable limit on the lifetimes to which we may be sensitive.

**Question 3) What is the intrinsic time resolution of this experimental setup?**

 ns

For the lower energy gamma ray identified in question 2, fit the exponential tail of the timing distribution to determine the lifetime of the corresponding state. You should include the random coincidence background (constant/po10 background) in your fit for maximum precision.

**Question 4) What is the extracted lifetime for the shorter lived state (with error)?**

*ns*

Try fitting with the option “L” which specifies to use binned likelihood fitting rather than the chi-squared default. At very low counts the Pearson’s chi squared test becomes unreliable and binned maximum likelihood method is preferable for an accurate result.

**Question 5) What is the lifetime for the longer lived state (with error)?**

*ns*

As you have no data beyond the exponential tail to suitably characterise uniform background, you may assume the background is the same on both sides of the peak and make use of [TF1::FixParameter](#).

For an extra challenge, you might try to fit the whole spectrum using a convolution of an exponential and a Gaussian ([Exponentially modified Gaussian distribution](#)). It is recommended to fix the parameters for sigma and  $X_0$  based on the gamma rays with no observable lifetime .

## (Extra) Exercise 4 – Fit Evaluation & Errors

Read through **GraphScript.C**, here we make use of the class [TGraphErrors](#). The class [TGraph](#) and its child class [TGraphErrors](#) are used for working with individual data points. Data points can be set individually by the [TGraph::SetPoint](#) function or read in as a group from array objects or files.

```
root [1] TGraph GraphVariable;  
root [2] GraphVariable.SetPoint(0,Xcord1,Ycord2);  
root [3] GraphVariable.SetPoint(1,Xcord2,Ycord2);  
root [4] TGraphErrors ErrorGraph("file.txt");
```

**Note:** Unlike histograms TGraph/TGraphErrors do not have a special ROOT memory name (They do not inherit from TNamed class).

**Run GraphScript.C**, this loads gamma-ray efficiency data from a calibration source (with gamma-ray energy in keV along X and normalised detection efficiency along Y) and performs a linear fit to the data. TGraph fitting is the same as for a histogram. The fit converges, it has determined the best possible parameter values for this function, but we have not chosen the best function to fit.

**Question 1) What is the [reduced chi squared](#) of the fit?**

**Hint:** Use [TF1::GetChisquare](#) and [TF1::GetNDF](#).

There are many possible functions to describe the efficiency curve of a gamma-ray detector (the shape of which is dominated by gamma-ray attenuation coefficients) the following is an example you can use which converges easily (some functions such as those used by radware and sigma require extra effort to ensure a successful fit). **Start with N=2**

$$y = e^{\left( \sum_{n=0}^{n=N} P_n \log(x)^n \right)}$$

**Question 2) What is the [reduced chi squared](#) of the new fit?**

With a good fit to the data we can extrapolate confidently to energies between data points. Use the function [TF1::Eval](#) to calculate the value of the function at a specific point.

**Question 3) What is the efficiency ratio between 300 and 1200 keV?**

This appears to be a good fit, but it still has some uncertainty, which is given by the relation of variances and covariances of the determined parameters :

$$V(f) = \sum_j \left( \frac{\partial f}{\partial x_j} \right)^2 V(x_j) + \sum_j \sum_{k \neq j} \left( \frac{\partial f}{\partial x_j} \right) \left( \frac{\partial f}{\partial x_k} \right) \text{cov}(x_j, x_k),$$

Luckily we don't need to calculate this by hand ROOT can do it for us. While the TF1 class holds the parameters, the function formula, and some details of the fit result, it does not store the error matrix. The actual fitting minimisation is performed by a global instance of a minimiser class, by default [TMinuit](#). To get additional details regarding the fit we can access the global instance of the minimiser, immediately following performing of the fit, with the static function [TVirtualFitter::GetFitter](#) which returns a pointer to it.

```
root [5] TVirtualFitter* ReturnedPointer = TVirtualFitter::GetFitter();
```

We can then use the function [TVirtualFitter::GetConfidenceIntervals](#) to calculate the error in the fit function, to a given confidence interval, at a range of points. We specify the range of points by creating a histogram that is passed (by pointer) as an argument to the function. The value of the function at the centre of each bin is set as the bin value and each bin error is set as the function error.

```
root [6] TH1D ErrorHist("EH","Ehf",bins,min,max);
root [5] FitterPointer->GetConfidenceIntervals(&ErrorHist, ConfidenceIntergal);
```

Remember in nuclear physics we typically take a central confidence interval at the 1 sigma limit, assuming Normally distributed uncertainties.

A full ROOT example is available here: [https://root.cern.ch/doc/master/ConfidenceIntervals\\_8C.html](https://root.cern.ch/doc/master/ConfidenceIntervals_8C.html)

**Question 5) What is the % uncertainty of the efficiency curve at 1000 keV?**

**Hint: Use the function [TH1::GetBinError](#).**

 %

**Note:** The following functions can be used to draw data from your confidence interval histogram over the top of the TGraph as a pretty error band :

TH1::[SetFillColor](#)(EColor::kRed); TH1::[SetFillStyle](#)(3001); TH1::[Draw](#)("E3same");

### Extra Problems

**Question 5) What is unphysical about the fit result ?**

Using the data from **GraphDataExtended.txt** try fitting a more physical shape for the full energy range. This may be achieved by increasing N of the previous function, though this can have issues reaching a true fit minimum. Alternatively, implement the following function:

$$y = d \cdot 10^{(a \cdot \log_{10}(x) + b \cdot \log_{10}(x)^2 + c \cdot x^{-2})}$$

**Question 6) What is the efficiency (and its uncertainty) at 70 keV?**

## (Extra) Exercise 5 – Vectors and Kinematic Correction

In many in-beam experiments excited nuclei emit gamma rays (and other particles) while moving at relativistic velocities. The Lorentz boost leads to significant degradation of the energy spectra. In order to study the nucleus in its rest frame we must correct for the boost.

For a photon the relationship is given by:

$$E' = \frac{E(1 - \beta \cos \theta)}{\sqrt{1 - \beta^2}}$$

Where E is the lab frame energy, E' the energy in the nuclear frame and theta the angle between the photon and the motion of the nucleus with velocity of beta.

The file **TTreeThree.root** contains filtered data from a Coulomb excitation experiment. Each event of the contained TTree is a confirmed coincident event between the detection of a recoiling nucleus in a charged particle detector and a gamma ray in a HPGe array. For each event the energy of the particles and their positions are recorded, the latter using the [TVector3](#) class.

In a experiment with different mass nuclei, the different kinematics can allow separation of the nuclei by their position and energy.

**Note:** The beam direction is given as +Z direction with theta being the angle away from the beam.

**Question 1) How many distinct masses can you identify in the data?**

**Hint:** Use the [TVector3::Theta](#) function and look at energy vs angle.

**Question 2) What is the shape and position of the charged particle detector?**

Due to the known kinematics of the experiment, the velocity of the nuclei can be calculated more accurately from the angle than from the measured energy. This has been done for one of the nuclei in the experiment and saved as a TGraph in **TTreeThree.root** giving angle theta vs nuclei velocity beta. **SecondTTreeSort.C** shows how to read the TGraph into memory from the file.

Using the direction of the particle and [TGraph::Eval](#) determine the velocity of the nuclei in each event. Use this to correct the gamma ray energy to the nuclear frame and produce a clear gamma-ray spectrum. You should make use of the function [TVector3::Angle](#) or the class [TLorentzVector](#) and its function [TLorentzVector::Boost](#)

**Question 3) Which of the particles does the TGraph give the correct velocity for?**

**Question 4) What is the nucleus?**