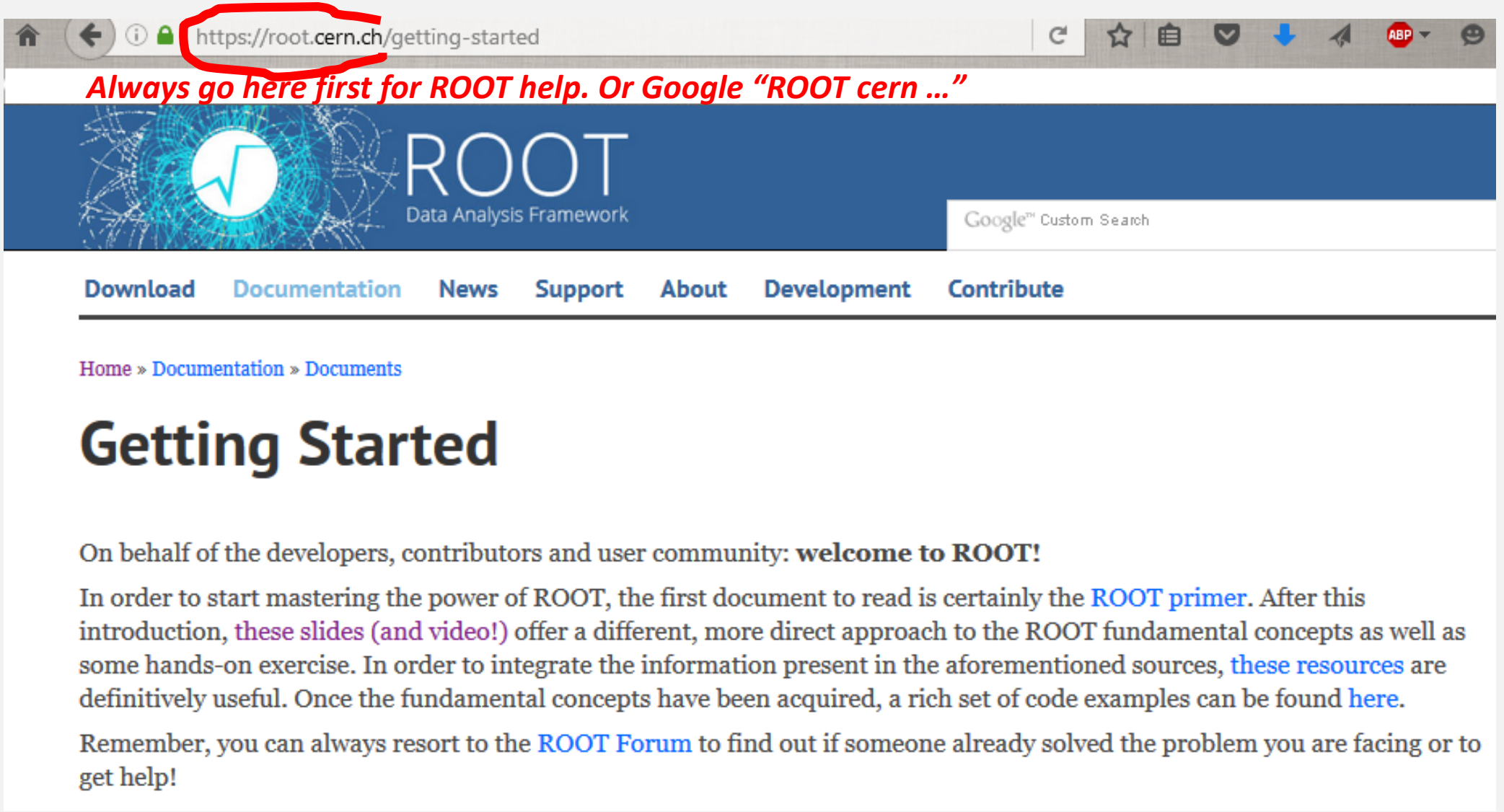
PHYS 6751: Graduate Nuclear & Particle Lab

- Basic Data Processing

Data processing needs & options

- In a nutshell, low energy nuclear physics data analysis comprises of:
  - Counting what you measured
  - Calibrating your measurement technique
  - Making cuts on and corrections to your data
    - E.g. calibrations, background subtractions
  - Fitting what you measured
    - *Often inferior to directly evaluating the data in a statistical way, e.g. finding peak centroids, peak widths, and background levels
- Analysis tools
  - Excel …could use it for our data, since it's simple, but please don't!
  - Gnuplot …good for simple, quick plotting & fitting
  - ROOT …the standard for nuclear physics analyses, highly flexible, C++ or Python, many built-in tools (including graphical tools), graphical interface available. Please use this! (will go over basics next lecture)

ROOT has a robust user community & extensive documentation



ROOT

- Powerful data analysis developed for particle physics analyses at CERN (née PAW)
- Numerous (documented) scientific & graphical tools based in C++
- Run as interpreted scripts or compiled codes
  - This simplifies things, but allows bad habits which occasionally bite you (memory leaks)

- Launch ROOT on the primary Edwards Lab machine (via PuTTY or from your built-in terminal):
  - `ssh -Y edwards.phy.ohiou.edu`
  - To start ROOT: `root -l` (the '-l' option prevents the annoying "splash screen" from displaying)
  - To exit ROOT: `.q`
  - Execute a script (a.k.a. macro) in ROOT: `.x myMacro.C`

ROOT example

- Let's analyze a simple data set by writing a ROOT macro*
- 1st read-in some data
  - Download γ-spectra from Paul King's website: http://inpp.ohiou.edu/~king/phys3702/tutorial/
  - Read-in the four available germanium spectra (^{22}Na, ^{60}Co, ^{137}Cs, ^{152}Eu)
    - In your favorite text editor, create a new file and enter it, e.g. `vi ProcessGeCalibrationData.C`
    - Write a script, something like:

```
void ProcessGeCalibrationData(){
  const Int_t Nchan=8192; //# of ADC channels
  Double_t Chan=0;
  Double_t Na22dat[Nchan];
  …

}
```

WARNING:
Copying & pasting from these slides into your text editor likely won't work (Special characters and hidden returns will be a problem).

*I'm not saying my code is the best way to do things. It's probably not, so please feel free to do things another way.

Write a script to read the data, something like:

```
void ProcessGeCalibrationData(){
  const Int_t Nchan=8192; //# of ADC channels
  Int_t Chan=0;
  Double_t Na22dat[Nchan];
  …
  ifstream fNa22dat;
  fNa22dat.open("germaniumdet_na22.txt");
  …
  for(int i=0;i<Nchan;i++){
    fNa22dat>>Chan>>Na22dat[Chan];
    …
    //check we're reading what we think we are:
    cout<<Chan<<" "<<Na22dat[Chan]<<endl;
    …
  }

}
```

*I'm not saying my code is the best way to do things. It's probably not, so please feel free to do things another way.

…now fill histograms with the data, like:

```
void ProcessGeCalibrationData(){
  [Code on previous slide]
  TH1F *hNa22raw=new TH1F("hNa22raw","hNa22raw",8192,0,8191);
  …
  for(int i=0;i<Nchan;i++){
    hNa22raw->Fill(i,Na22dat[i]);
    …
  }
}
```

 # bins *lower bin* *upper bin*

Tip:
You can save run-time by filling the histogram when you're assigning data to the array (on the previous slide). I do it separately here for clarity.

…& draw them, like:

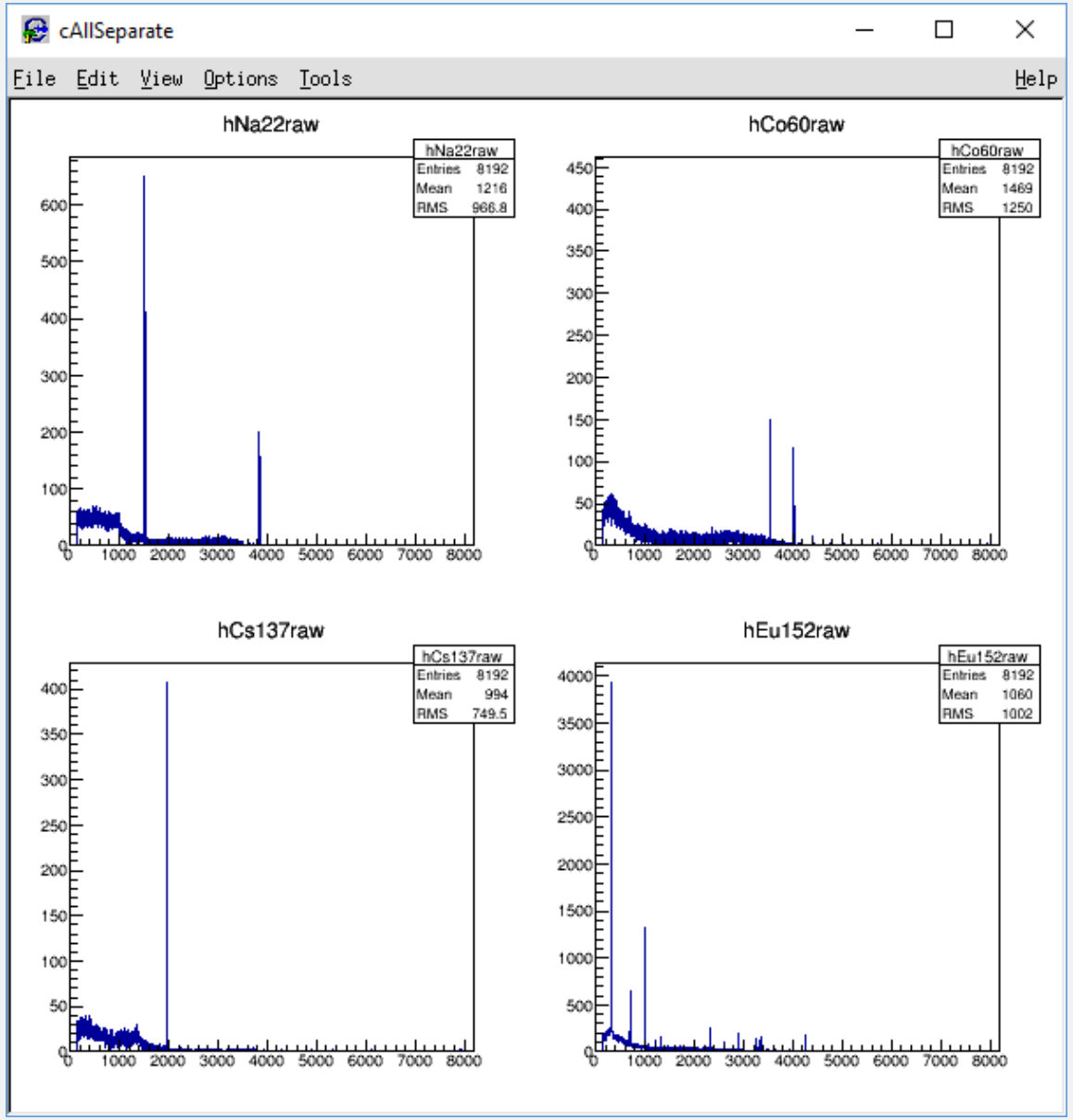
```
void ProcessGeCalibrationData(){
  [Previous code]
  TCanvas *cAllSeparate=new TCanvas("cAllSeparate","cAllSeparate",10,10,600,600);
  cAllSeparate->Divide(2,2); splits the canvas into 2-by-2
  cAllSeparate->cd(1); make the first "pad" of the canvas (upper-left) the target for drawing
  hNa22raw->Draw();
  cAllSeparate->cd(2); make the second "pad" of the canvas (upper-left) the target for drawing
  hCo60raw->Draw();
  …
}
```

x,y coordinates of top-corner of canvas on screen *width & height of canvas on screen*

Tip:
Make sure you have x11 tunneling on for your connection and that you have a x11-forwarding program, such as Xming, running. Otherwise your plots won't display on your monitor.

*I'm not saying my code is the best way to do things. It's probably not, so please feel free to do things another way.

Bask in the glory of your plot



Get the Mean & Standard Deviation of the important peaks, like:

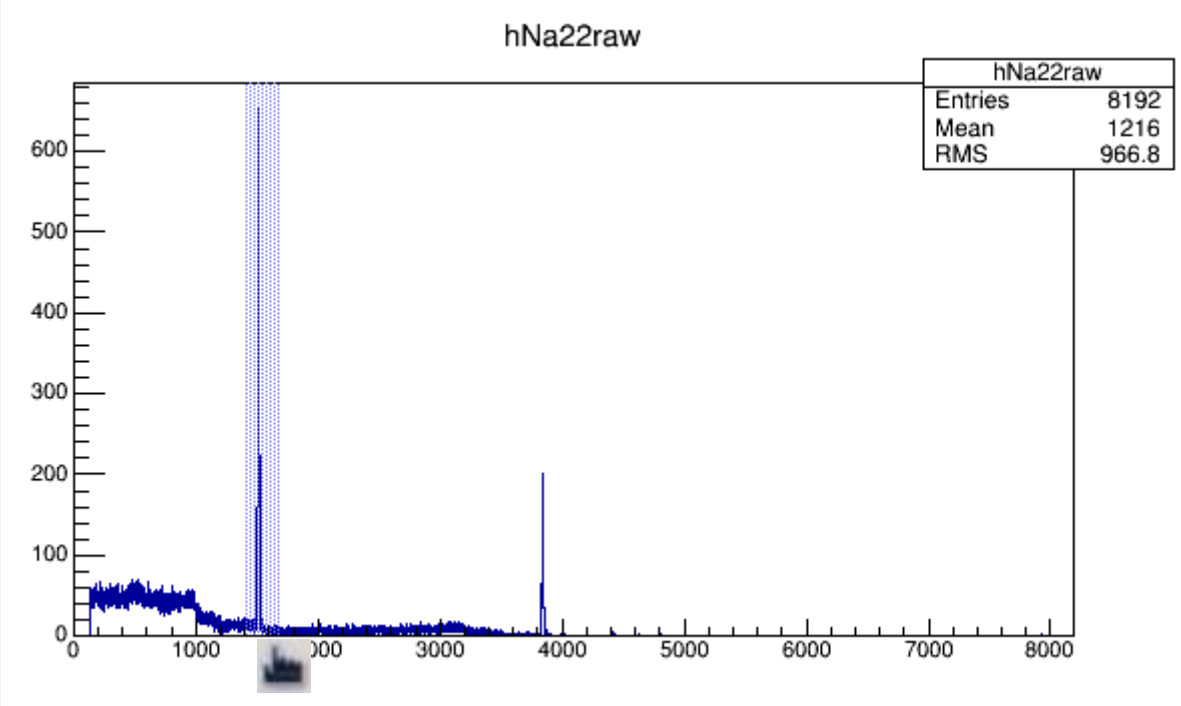
```
void ProcessGeCalibrationData(){
  [Previous code]
  const Int_t Na22_LowerBound1=1490;
  const Int_t Na22_UpperBound1=1540;
  hNa22raw->GetXaxis()->SetRange(Na22_LowerBound1,Na22_UpperBound1);
  Double_t Na22_counts1=hNa22raw->Integral(Na22_LowerBound1,Na22_UpperBound1);
  Double_t Na22_mean1=hNa22raw->GetMean();
  Double_t Na22_sig1=hNa22raw->GetRMS();
  Double_t Na22_Dmean1=Na22_sig1/sqrt(Na22_counts1);
  …
}
```

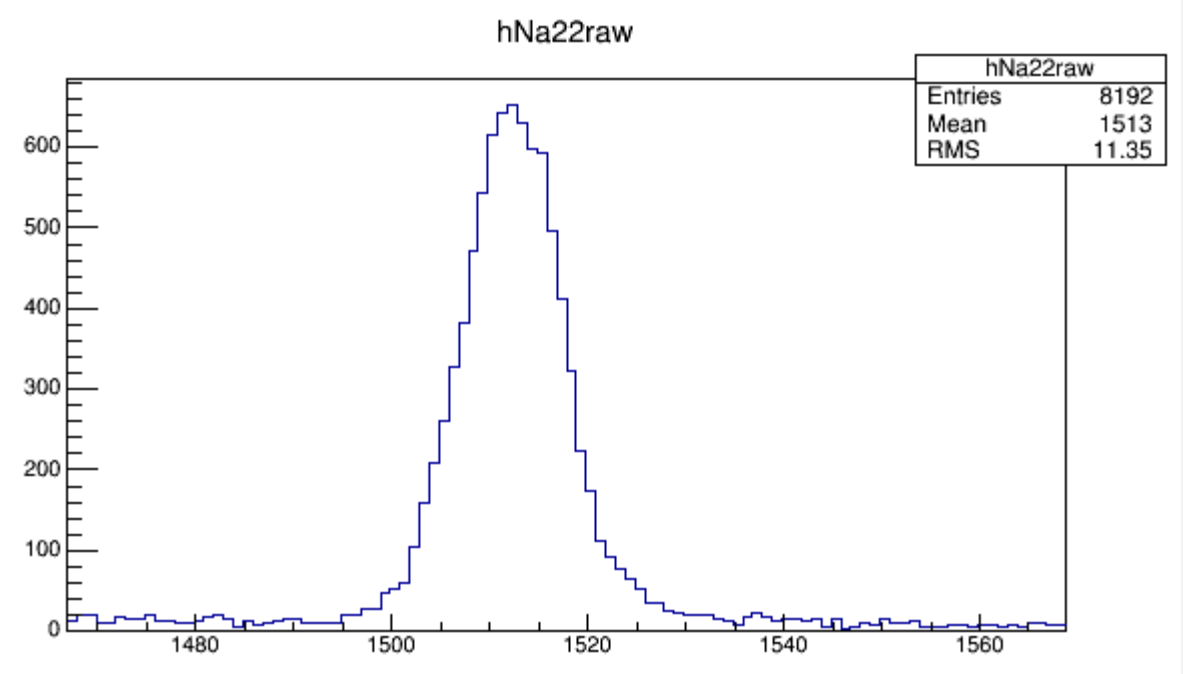
determine the ranges for your peak by looking at the spectrum or the text file

Do this for 2 sodium peaks, 2 cobalt peaks, and 1 cesium peak

*I'm not saying my code is the best way to do things. It's probably not, so please feel free to do things another way.

…zooming-in is a quick (non-automated) way to check your result





The standard deviation ("RMS" in ROOT) is highly range-dependent here because?

Create a graph of channel # vs energy, like:

```
void ProcessGeCalibrationData(){
  [Previous code]
  Double_t Na22_E1=511.00;
  Double_t Na22_E2=1274.54;
  TGraphErrors *gSourceCal=new TGraphErrors(1);
  gSourceCal->SetPoint(0,Na22_mean1,Na22_E1);
  gSourceCal->SetPoint(1,Na22_mean2,Na22_E2);
  gSourceCal->SetPointError(0,Na22_Dmean1,0.);
  gSourceCal->SetPointError(1,Na22_Dmean2,0.);
  gSourceCal->SetPoint(2,Co60_mean1,Co60_E1);
  …
  TCanvas *cCalFn=new TCanvas("cCalFn","cCalFn",500,10,400,300);
  cCalFn->cd();
  gSourceCal->SetMarkerStyle(7);
  gSourceCal->Draw("APE");
}
```

From Gordon Gilmore's Practical Gamma-ray Spectrometry Appendix B:

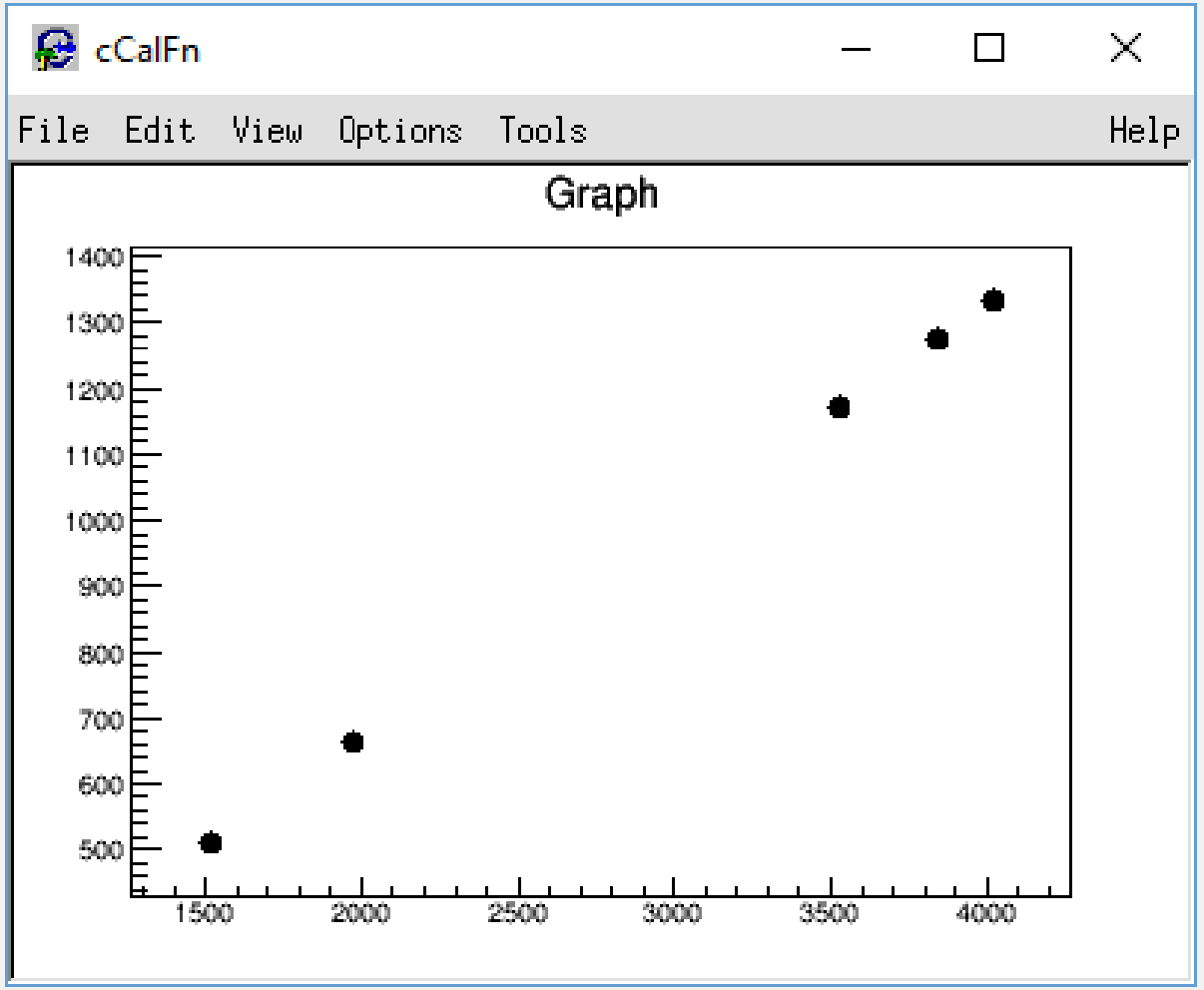
^{22}Na: 511.00 keV, 1274.54 keV
^{60}Co: 1173.23 keV, 1332.49 keV
^{137}Cs: 661.66 keV
^{152}Eu: You'll obtain these using the calibration

**uncertainties are on the ~eV level, so don't bother including them*

The "A" option is needed for the first graph you draw on a canvas pad. The "P" option means draw points. The "E" option means draw error bars.
…many more options exist

*I'm not saying my code is the best way to do things. It's probably not, so please feel free to do things another way.

Bask in the glory of your plot



Fit the channel # vs energy graph to obtain an energy calibration, like:

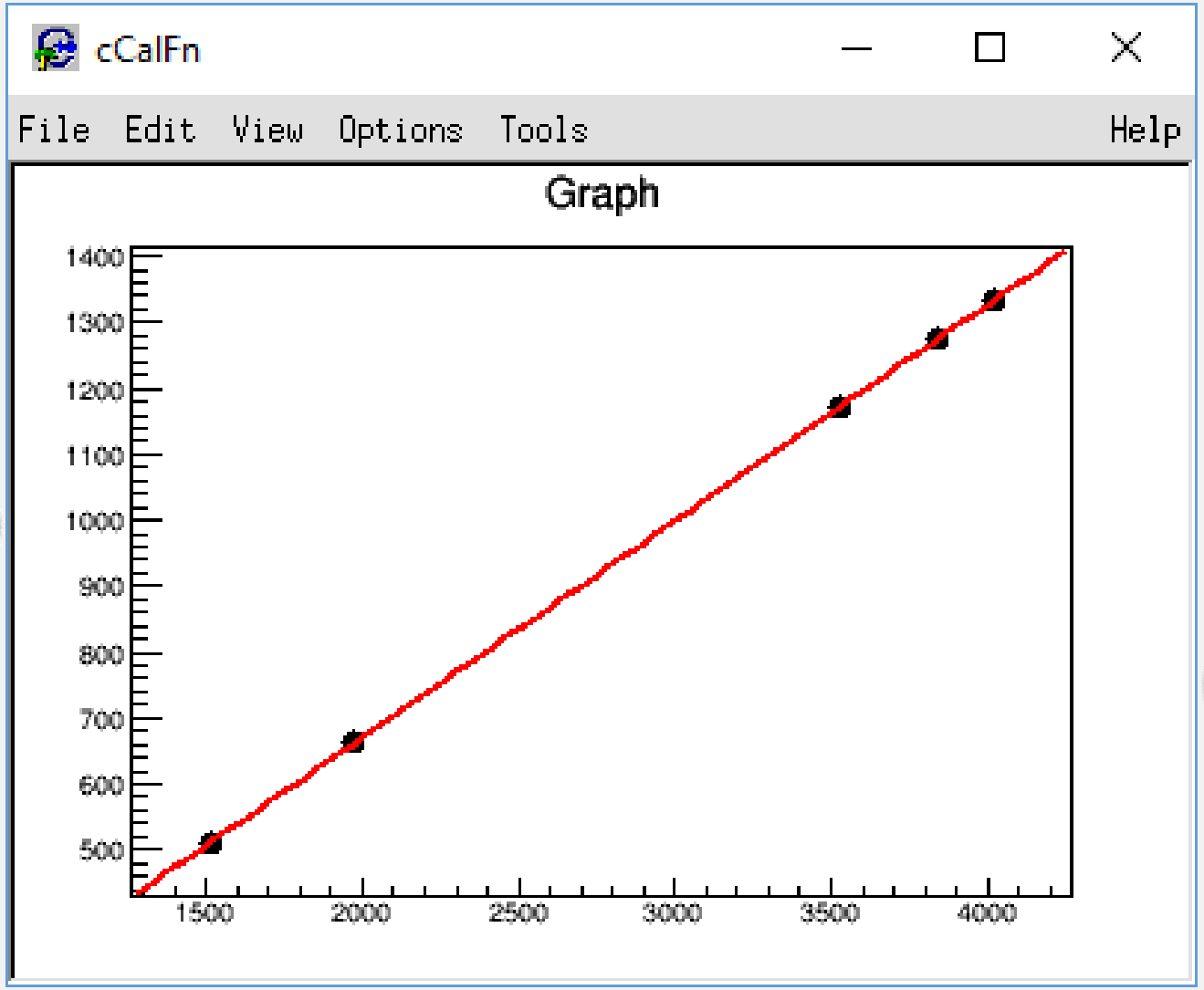
```
void ProcessGeCalibrationData(){
  [Previous code]
  TF1 *fnQuad=new TF1("fnQuad","[0]+[1]*x+[2]*x*x");
  fnQuad->SetParameter(0,10);
  fnQuad->SetParameter(1,0.5);
  fnQuad->SetParameter(2,0.0001);

  gSourceCal->Fit("fnQuad");
  Double_t yint=fnQuad->GetParameter(0);
  Double_t slope=fnQuad->GetParameter(1);
  Double_t curv=fnQuad->GetParameter(2);
  Double_t Dyint=fnQuad->GetParError(0);
  Double_t Dslope=fnQuad->GetParError(1);
  Double_t Dcurv=fnQuad->GetParError(2);
  cout<<" "<<endl;
  cout<<yint<<" "<<slope<<" "<<curv<<endl;
  cout<<Dyint<<" "<<Dslope<<" "<<Dcurv<<endl;
}
```

Simple functions like this are built-in …but that's boring

guesses for parameters made from looking at the graph

*I'm not saying my code is the best way to do things. It's probably not, so please feel free to do things another way.

Bask in the glory of your plot



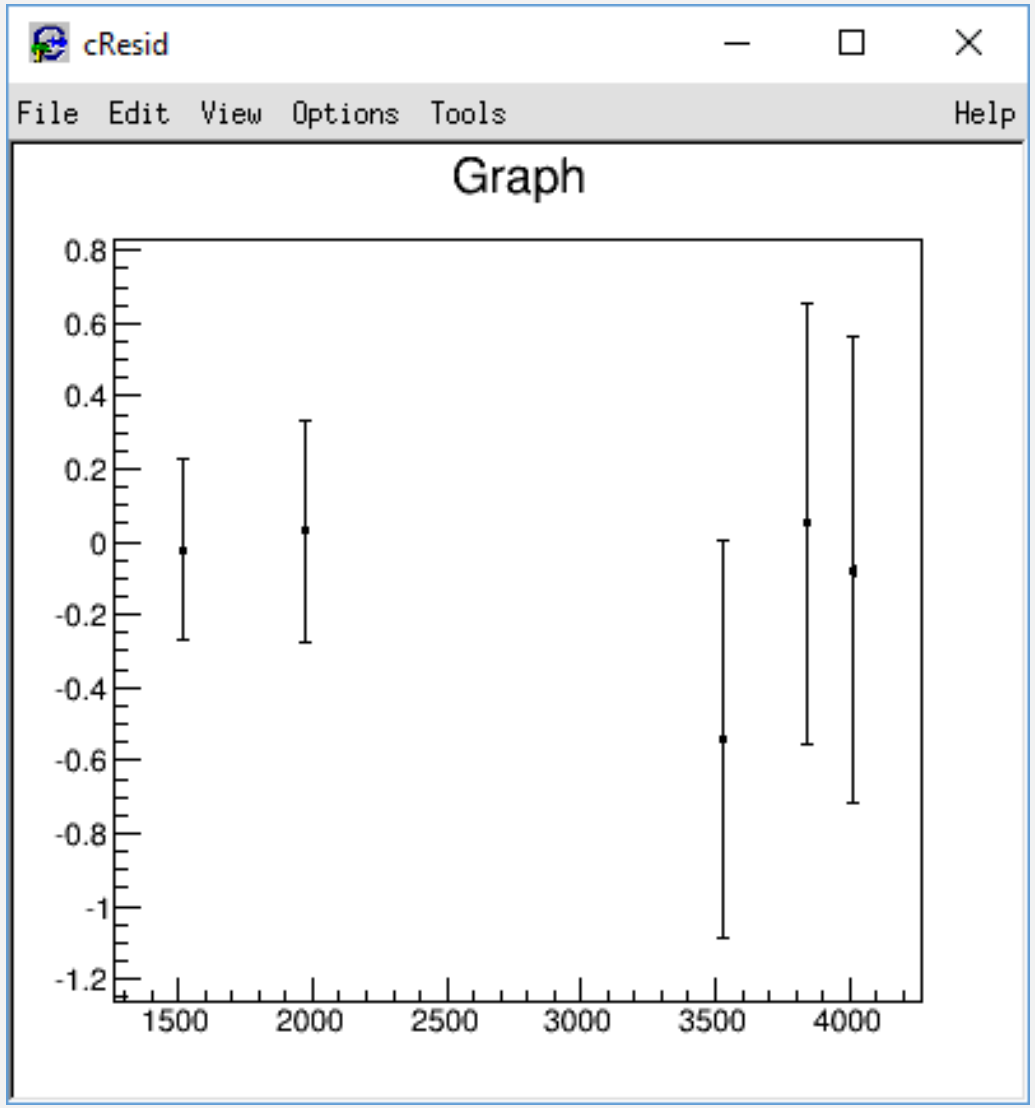
Create & Plot fit residuals, like:

```
Double_t ReturnYerror(Double_t x, Double_t y, Double_t dy, Double_t m, Double_t dm,
Double_t c, Double_t dc){
  //f(x)=a+b*x+c*x^2
  //-->df|x= SQRT{[(df/da)|x*(da)]^2 + [(df/db)|x*(db)]^2 + [(df/dc)|x*(dc)]^2}
  Double_t YError=sqrt(pow(1.*dy,2.)+pow(x*dm,2.)+pow(x*x*dc,2.));
  return(YError);
}

void ProcessGeCalibrationData(){
  [Previous code]
  TGraphErrors *gFitResid=new TGraphErrors(1);
  gFitResid->SetPoint(0,Na22_mean1,fnQuad->Eval(Na22_mean1)-Na22_E1);

  gFitResid->SetPointError(0,Na22_Dmean1,ReturnYerror(Na22_mean1,yint,Dyint,slope,
                           Dslope,curv,Dcurv));
  …
  TCanvas *cResid=new TCanvas("cResid","cResid",10,500,400,400);
  cResid->cd();
  gFitResid->SetMarkerStyle(7);
  gFitResid->Draw("APE");
}
```

Uncertainty propagation from Chapter 3 of John Taylor's Error Analysis

*I'm not saying my code is the best way to do things. It's probably not, so please feel free to do things another way.

Bask in the glory of your plot

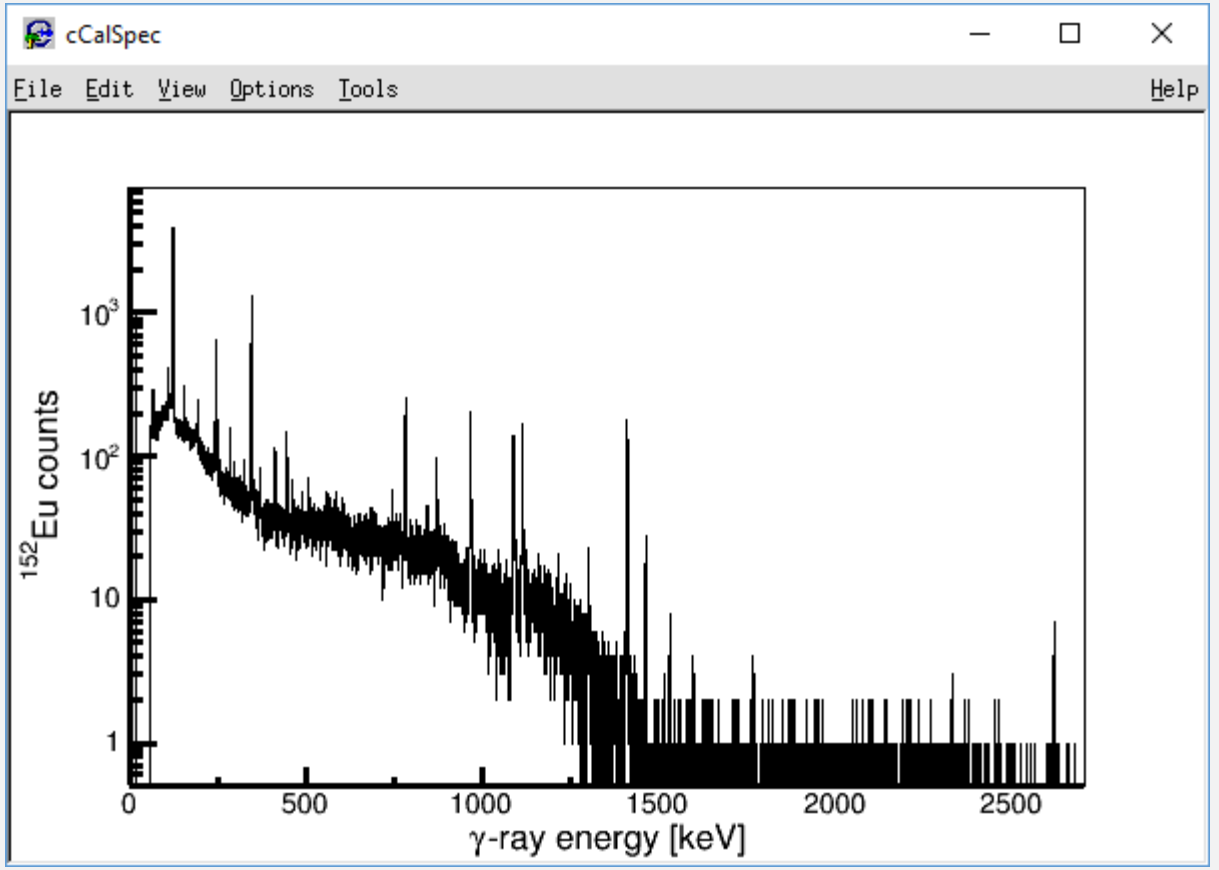


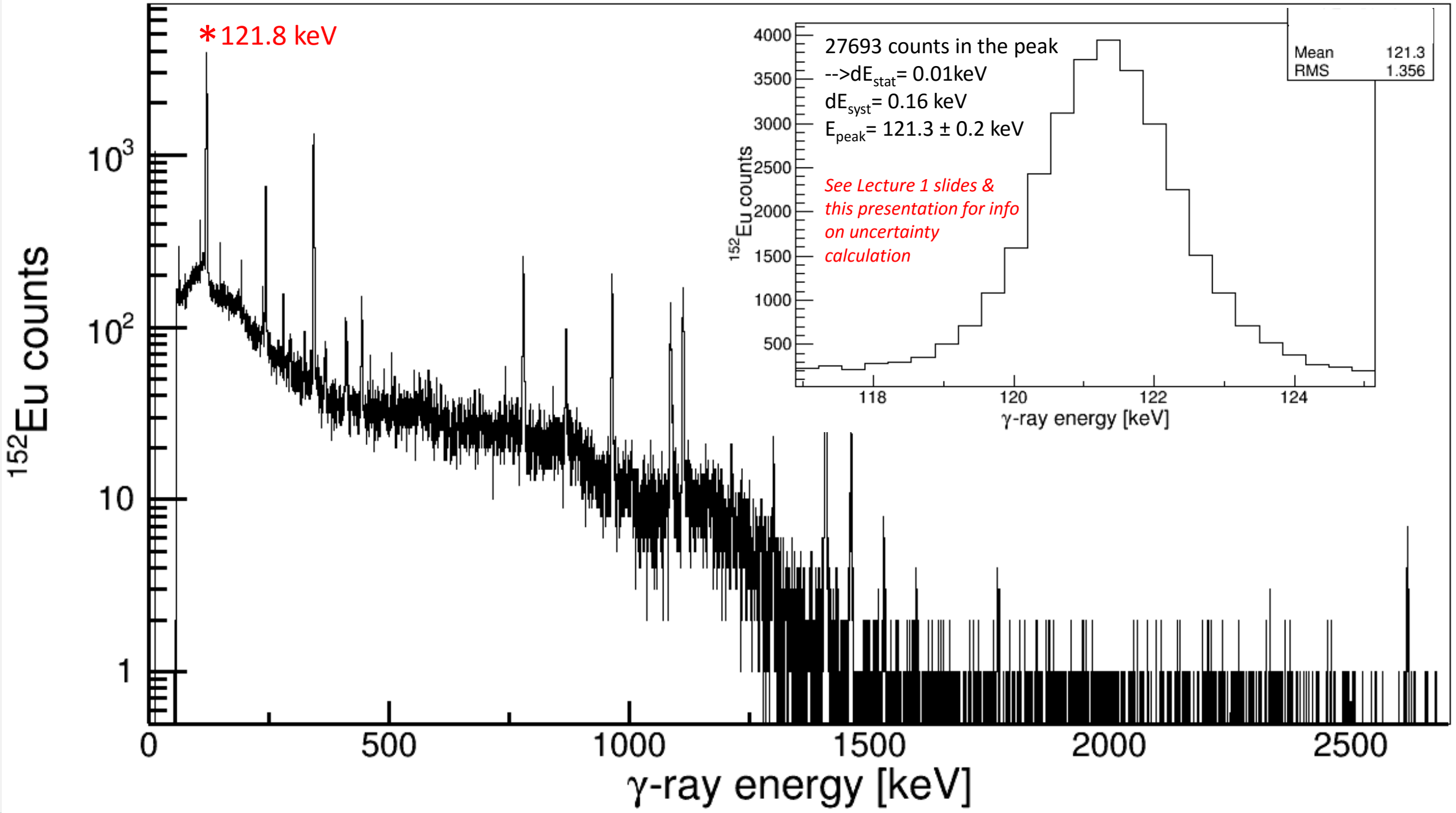
Now make a nice looking calibrated spectrum for ^{152}Eu, like:

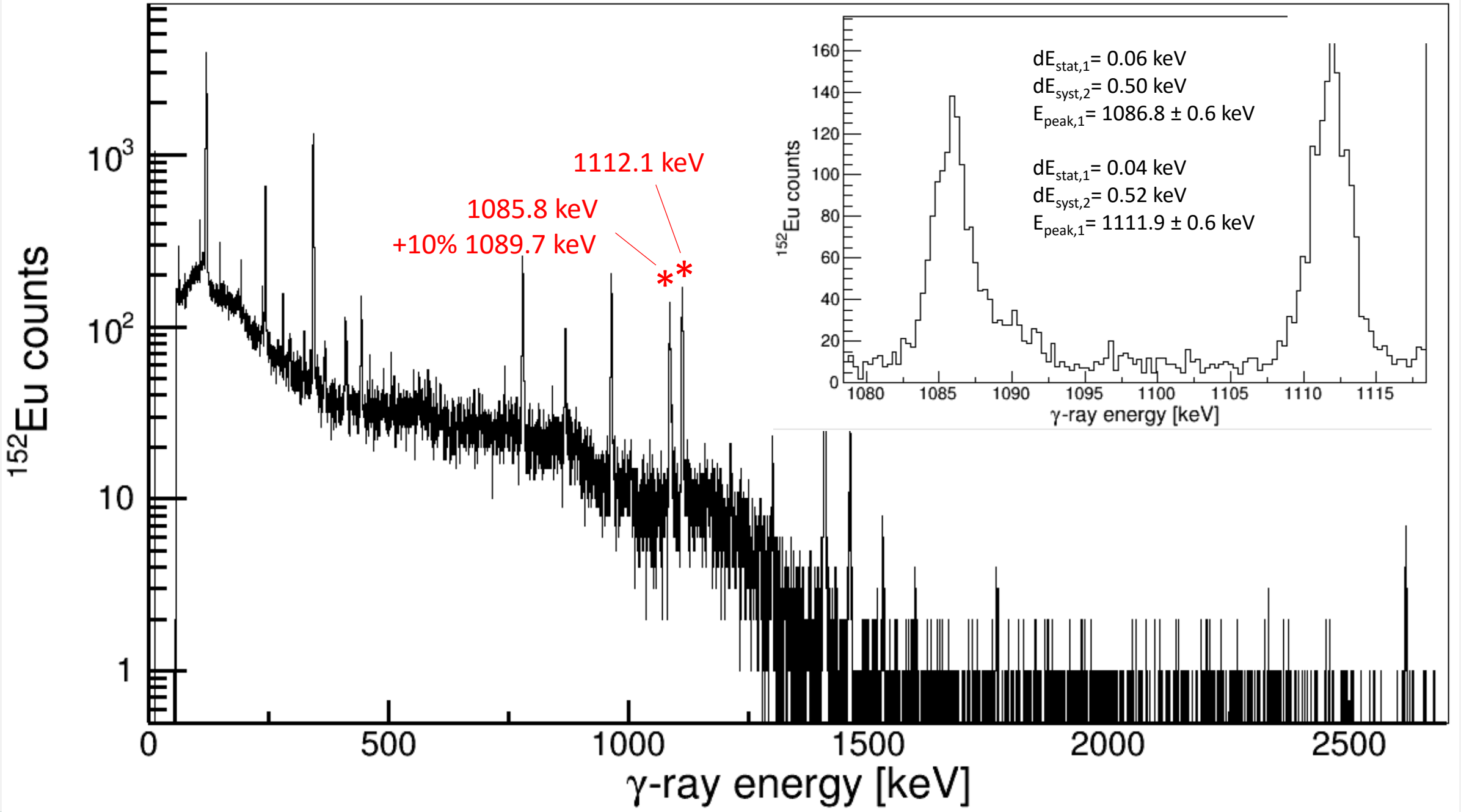
```
[Previous code]
void ProcessGeCalibrationData(){
  [Previous code]
  TH1F *hEu152cal=new TH1F("hEu152cal","hEu152cal",8192,0,2705);
  for(Int_t i=0;i<Nchan;i++){
    hEu152cal->Fill(fnQuad->Eval((double)i),Eu152dat[i]);
  }
  TCanvas *cCalSpec=new TCanvas("cCalSpec","cCalSpec",10,10,600,400);
  cCalSpec->cd();
  gROOT->SetStyle("Pub");
  hEu152cal->GetXaxis()->SetTitle("#gamma-ray energy [kev]");
  hEu152cal->GetXaxis()->SetTitleFont(42);
  hEu152cal->GetXaxis()->SetTitleSize(0.05);
  hEu152cal->GetXaxis()->SetTitleOffset(0.9);
  hEu152cal->GetXaxis()->SetLabelSize(0.04);
  hEu152cal->GetXaxis()->CenterTitle();
  hEu152cal->GetXaxis()->SetNdivisions(510);
  hEu152cal->GetYaxis()->SetTitle("^{152}Eu counts");
  …
  hEu152cal->SetLineColor(1);
  hEu152cal->SetLineStyle(1);
  hEu152cal->SetLineWidth(1);
  cCalSpec->SetLogy();
  hEu152cal->Draw();
}
```

*I'm not saying my code is the best way to do things. It's probably not, so please feel free to do things another way.

Bask in the glory of your plot




*121.8 keV
27693 counts in the peak
-->dE_stat= 0.01keV
dE_syst= 0.16 keV
E_peak= 121.3 ± 0.2 keV
See Lecture 1 slides & this presentation for info on uncertainty calculation
Mean 121.3
RMS 1.356
152Eu counts
γ-ray energy [keV]
152Eu counts
γ-ray energy [keV]


1112.1 keV
1085.8 keV
+10% 1089.7 keV
**
^{152}Eu counts
γ-ray energy [keV]
$dE_{stat,1}$= 0.06 keV
$dE_{syst,2}$= 0.50 keV
$E_{peak,1}$= 1086.8 ± 0.6 keV
$dE_{stat,1}$= 0.04 keV
$dE_{syst,2}$= 0.52 keV
$E_{peak,1}$= 1111.9 ± 0.6 keV

Assignment

- Plot 2 more ^{152}Eu peaks and compare to the accepted value, including uncertainty.
  - Hand-in the plots (publication-quality) along with the energy-comparison
- Run-plan/preparatory notes for your experiment
  - Keep in mind that these notes should serve as useful time-savers for your in the lab.
  - E.g. Order of operations, useful calculations, citations to relevant publications/book chapters