YMAP
Youth Meeting for AstroParticle

ROOT講習会 第3回
TF1 fitting TGraph

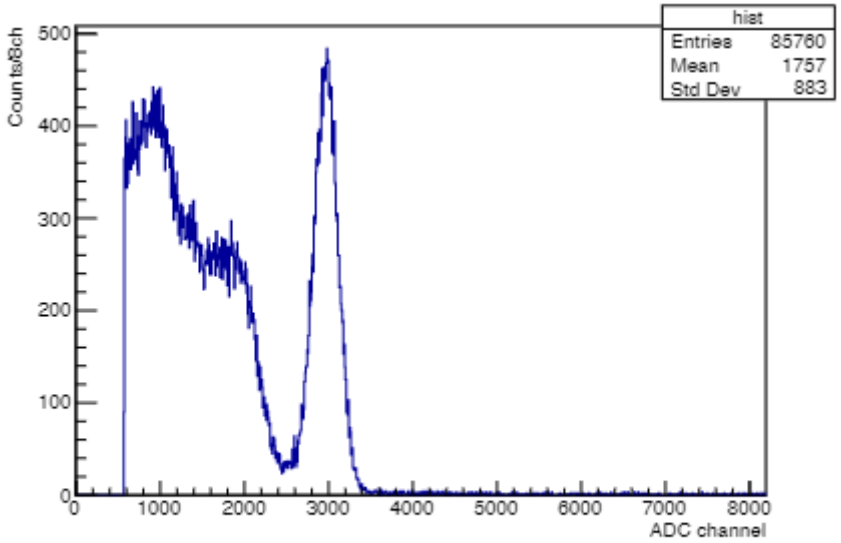
中村輝石（東北大）
kiseki@epx.phys.tohoku.ac.jp

自己紹介

- 中村輝石
- 東北大学 素粒子実験（加速器）グループ
- 実験：AXEL、T2K、SK、HK、など
- 博士：NEWAGE実験で取得

第2回を思い出す

- 第2回のまとめ
  - 物理結果を理解するためには、分布を表す図「ヒストグラム」で可視化
  - root でヒストグラム（TH1D）を作る
  - マクロのテクニック
  - はぐれメタルの寿命
  - シンチレータで取得したガンマ線の信号サイズ（ADC）の分布



Rebin(8)したヒストグラム

第3回でやること

- 関数「TF1」を作れるようになる
  - 中高で習うような関数をイメージしてもらえばOK（プログラムの関数ではない）
  - y=ax+b、y=sqrt(x)、y=a*sin(x)、y=a*exp(-b*x)、y=a*exp(-((x-b)/c)^2)　など

- ヒストグラムを関数であてはめる（fitting）
  - ヒストグラムがどのようなモデル（関数）に合うのかを確かめられる
  - fitting結果が合ってるのかどうかの判断も

- グラフ「TGraph」を作れるようになる
  - (x,y)の点列の描画のこと
  - 結果を見やすくできる

TF1：最小構成要素

- 新しく function1.C を作る
  - 変数は x と決まっている
- 実行する
- 直線が引けましたか？

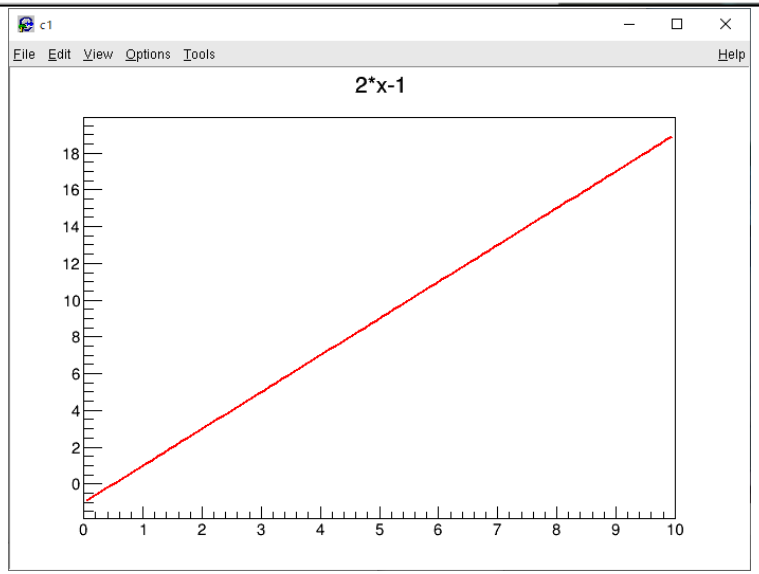
```
//function1.C
void function1(){
  TF1 *func1 = new TF1("chokusen","2*x-1",0,10);
  func1->Draw("L");
}
```

関数名

関数の定義

定義域(0~10)

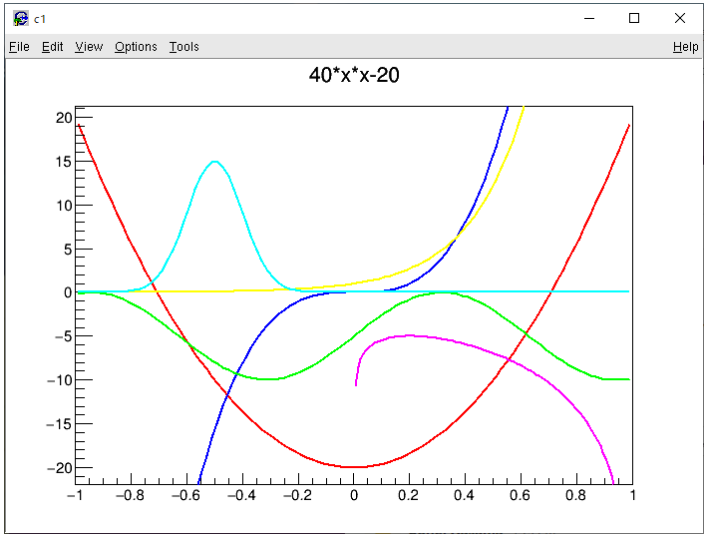
```
kiseki@dell-xps13:~/ymap2022$ root -l function1.C
root [0]
Processing function1.C...
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
root [1]
```



TF1:いろんな関数を使ってみる

- function2.C を作って実行
- TMath とは
  - rootの名前空間で、単純な数値計算を楽にできる関数がたくさん用意されている
  - https://root.cern.ch/root/html524/TMath.html

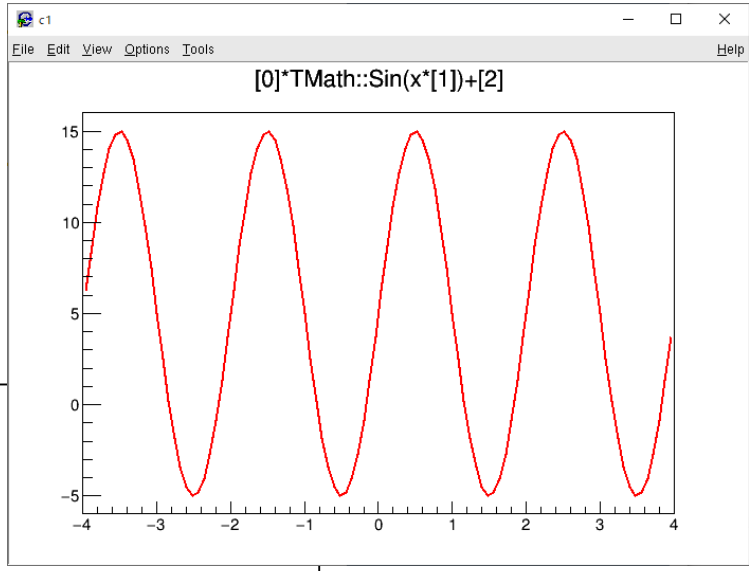
```
//function2.C
void function2(){

  TF1 *func1 = new TF1("niji","40*x*x-20",-1,1);
  TF1 *func2 = new TF1("sanji","TMath::Power(5*x,3)",-1,1);
  TF1 *func3 = new TF1("sin","5*TMath::Sin(5*x)-5",-1,1);
  TF1 *func4 = new TF1("exp","TMath::Exp(5*x)",-1,1);
  TF1 *func5 = new TF1("log","8*TMath::Log(1-x)+2*TMath::Log(x)",0,1);
  TF1 *func6 = new TF1("gaus","15*TMath::Gaus(x,-0.5,0.1)",-1,1);

  func2->SetLineColor(4);//ao
  func3->SetLineColor(3);//midori
  func4->SetLineColor(5);//ki
  func5->SetLineColor(6);//momo
  func6->SetLineColor(7);//mizu

  func1->Draw();
  func2->Draw("same");
  func3->Draw("same");
  func4->Draw("same");
  func5->Draw("same");
  func6->Draw("same");

}
```



TF1：パラメータを使ってみる

- function3.C を作って実行
- amp、freq、offsetをいじってみよう

- p0 * sin(p1*x) + p2




```
//function3.C
void function3(){

  Double_t amp=10;
  Double_t freq=0.5;
  Double_t offset=5;

  TF1 *func1 = new TF1("sin","[0]*TMath::Sin(x*[1])+[2]",-4,4);

  func1->SetParameters(amp, freq*2*TMath::Pi(), offset);
  func1->Draw("L");

}
```

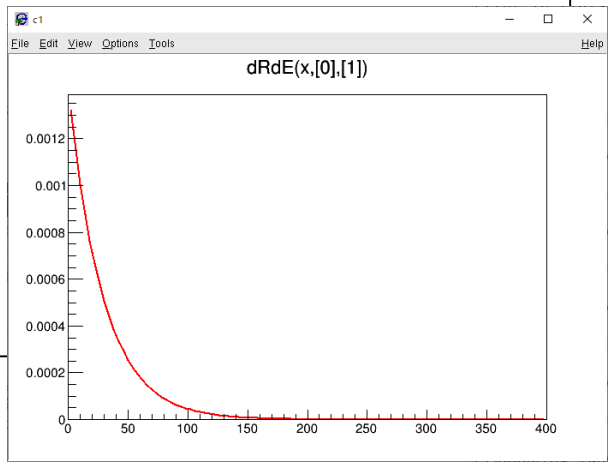
パラメータは大括弧と数字で書く

SetParameters で変数に順番に値を入れる

TF1：c++の関数を流用する

- function4.C
- 関数を別に用意しておいて、横軸にするものをxに、他をパラメータにして定義したりもできる

```
//function4.C

Double_t dRdE(double E_R, double mass_DM, double A){
  double v_0 = 230;
  double rho_DM = 0.4;
  double sigma_0 = 1.0;
  double mass_p = 0.932;
  double R_0 = 2.0/TMath::Sqrt(TMath::Pi()) * 6.02e26/A *rho_DM/mass_DM *sigma_0*1e-36 *v_0*1e5 *60*60*24;
  double v_c=3.0e5; // km/s
  double E_0 = 0.5*mass_DM*v_0*v_0/v_c/v_c *1000000; // GeV-->keV
  double r = 4.0*mass_DM*A*mass_p/TMath::Power(mass_DM+A*mass_p, 2);
  return R_0/E_0/r*TMath::Exp(-E_R/E_0/r);
}

void function4(){

  TF1 *func1 = new TF1("energy spectrum","dRdE(x,[0],[1])",0,400);
  func1->SetParameters(100, 131);
  func1->Draw();

}
```



TF1：余裕な人向け

- function5.C
- 数値計算とかはTF1で可視化すると良い
- 余裕があればparametersをいじってみてください

左上図
暗黒物質に弾性散乱される原子核のエネルギースペクトル。標的はF,Ge,Xeの三種類。このときは、運動量移行はが十分に小さいという仮定をおいたもの。また、地球の速度を0、銀河の脱出速度を∞と置いたシンプルな状況（まじめにやるときはちゃんと考慮する）。

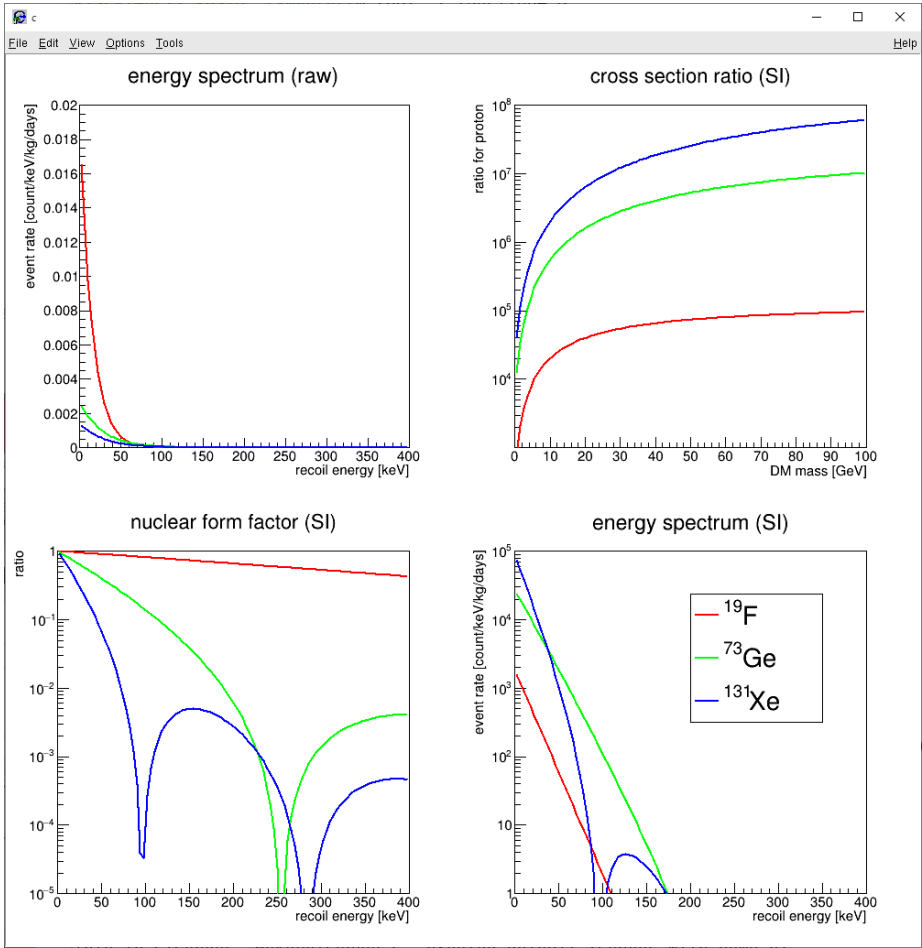
右上図
暗黒物質と原子核の散乱断面積について、暗黒物質の質量依存性、標的依存性を陽子のときとの比として図示。この図ではSpin Independent (SI) な反応を仮定（マクロ内でSpin Dependent も選べるようにした）。SI反応はコヒーレント反応とも呼ばれ、質量数の二乗に比例するため、キセノンのような重い原子核で大きくなっている。一方でSDは原子核のスピンに依存するため、軽い原子核が有利になる場合もある。

左下図
運動量移行があるときに、断面積が小さくなる効果（フォームファクター）について、反跳エネルギー依存性として図示。重い原子核ほどフォームファクターの影響が大きく、実効的な断面積は小さくなってしまう。低エネルギー（低運動量移行）なら影響はマシ。ギャップになっている部分は、原子核内の核子の分布をベッセル関数で置いているため（球殻や一様球をフーリエ変換）。

右下図
反跳スペクトル、断面積、フォームファクターを考慮したエネルギースペクトル。40keV以下ではキセノン標的が一番イベント数が期待できることが分かる。SDだと有利な原子核はどうなるか？

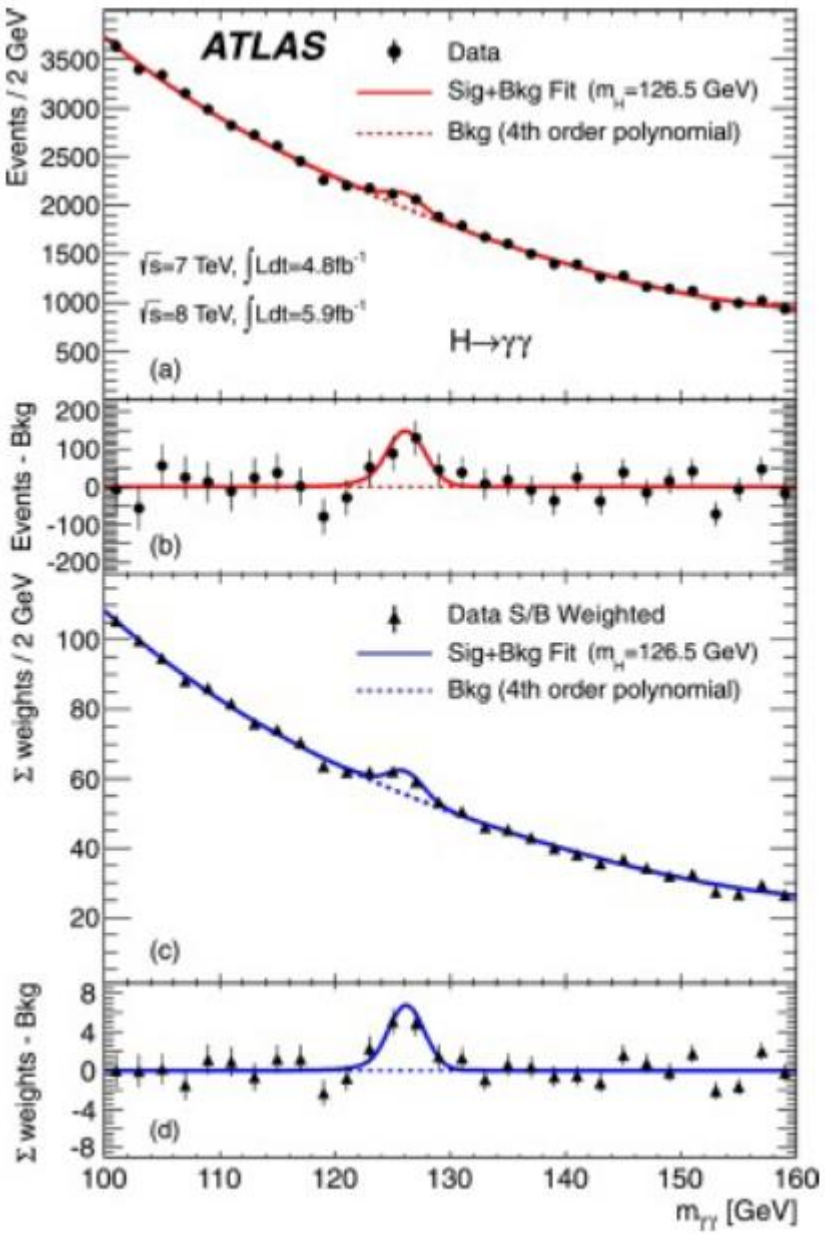
参考文献：J.D.Lewin and P.F.Smith, Astroparticle Physics 6 (1996) 87-112

```
  else if(mode==1) return TMath::Power(sin(qr)/qr, 2) *TMath::Exp(-q*q*s
  else             return -1;
}


void function5(){

  // parameters
  Double_t mass_DM = 100;//GeV/c^2
  Double_t A_array[3] = {19, 73, 131};// 19F, 73Ge, 131Xe
  Double_t Ja_array[3] = { 0.647*1.0, 0.065*0.078, 0.055*0.212 };// J-fa
  int mode = 0;// SI:0, SD:1

  // canvas
  int i_color[3] = {2,3,4};
  TCanvas *c = new TCanvas("c","c",1000,1000);
  c->Divide(2,2);

  c->cd(1)->SetMargin(0.15,0.1,0.1,0.1);
  c->cd(1)->DrawFrame(0,0,400,0.02,"energy spectrum (raw);recoil energy
  TF1 *func1[3];
  for(int i=0; i<3; i++){
    func1[i]= new TF1(Form("f1_%d",i),"dRdE(x,[0],[1])",0,400);
    func1[i]->SetParameters(mass_DM, A_array[i]);
    func1[i]->SetLineColor(i_color[i]);
    func1[i]->Draw("same");
  }
```



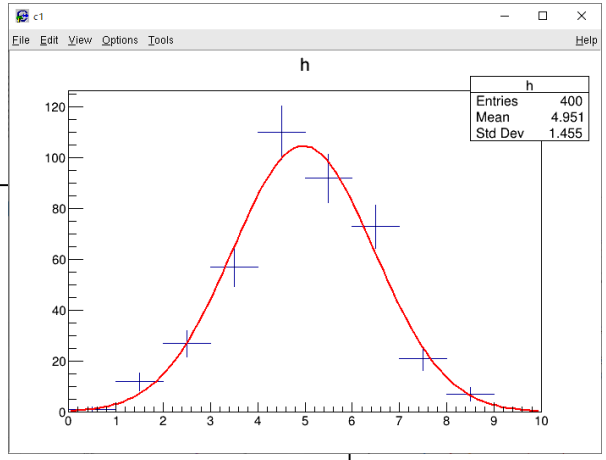
フィッティング

- なぜフィッティングをするのか？
  - 測定データ（ヒストグラム）を説明できるモデル（関数とパラメータ）を探し当てたい
  - 関数形を知りたい場合もあるし、パラメータが知りたい場合もある
  - フィッティング結果がどれくらい合ってるかも分かる



root でフィッティング（手続き）

- fitting1.Cを作ってrootで実行




```
// fitting1.C

void fitting1(){

  // prepare histogram
  int bin_num=10;
  TH1D *h = new TH1D("h", "h;", bin_num, 0, 10);
  for(int i=0; i<400; i++) h->Fill(gRandom->Gaus(5,1.4));
  h->Draw("E");

  // fitting
  TF1 *f = new TF1("f","[0]*TMath::Exp(-(x-[1])*(x-[1])/[2]/[2])",0,10);
  f->SetParameters(10,5,2);
  h->Fit(f);

}
```

それっぽい初期値を入れておく

hをfでフィッティング

```
kiseki@dell-xps13:~/ymap2022$ root -l fitting1.C
root [0]
Processing fitting1.C...
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
 FCN=6.90395 FROM MIGRAD    STATUS=CONVERGED     130 CALLS         131 TOTAL
                     EDM=4.99631e-08    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  p0           1.04743e+02   6.67767e+00   7.25344e-03   4.85724e-05
   2  p1           4.96672e+00   7.71534e-02   1.05719e-04   2.01273e-03
   3  p2           2.12485e+00   8.55045e-02   9.27754e-05   1.29459e-03
root [1]
```

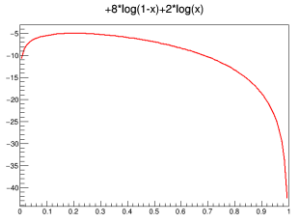
フィット結果は標準出力に

世の中にはざっくり2種類のフィッティングがある

- カイ二乗フィット（最小二乗法ともいう）
  - データ点とモデルの線の差（カイ二乗値）を最小にする
  - rootのデフォルトのフィッティングはこの方法になる

$$\chi^2 = \sum_{k=1}^{n} \frac{\left(O_k - E_k\right)^2}{E_k}$$

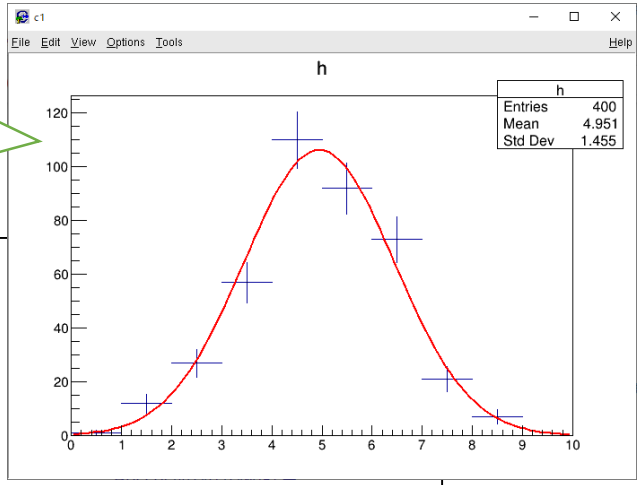
- Likelihoodフィット（最尤推定法ともいう）
  - モデルが実データを予測する確率（尤度）を最大にする
  - 例）
    - さいころが1が出る確率（xとおく）を求めたくて、10回振ったら、2,5,4,3,1,6,4,1,3,5 だった
    - 上記実験が起きる確率（尤度）は、L = (1-x) (1-x) (1-x) (1-x) x (1-x) (1-x) x (1-x) (1-x) となる
    - 上の式をLで置いて、最大化するときのxを求めたい
    - 掛け算は大変なので、log(L)で足し算にばらして計算することが多い
    - log(L)=8log(1-x)+2log(x) TF1で描くと右図のようになる
    - 最大のときのxは0.2になる（f->GetMaximumX() でとれる）



  - ヒストグラムの場合、各ビンにおいて、測定で得られたイベント数を、モデルが予想する確率を計算して掛け算していって、Likelihoodを作って最大化する

root でlikelihoodフィットをする方法

- オプションに"L"を入れる



デフォルトとほとんど同じ（でもほんのちょっと違う）

```
// fitting2.C

void fitting2(){

  // prepare histogram
  int bin_num=10;
  TH1D *h = new TH1D("h", "h;", bin_num, 0, 10);
  for(int i=0; i<400; i++) h->Fill(gRandom->Gaus(5,1.4));
  h->Draw("E");

  // fitting
  TF1 *f = new TF1("f","[0]*TMath::Exp(-(x-[1])*(x-[1])/[2]/[2])",0,10);
  f->SetParameters(10,5,2);
  h->Fit(f,"L","",0,10);

}
```

フィットの種類を決めるオプション

描画のオプション（フィット後の自動描画をやめたい→ "goff"）

フィット範囲

デフォルトの問題点

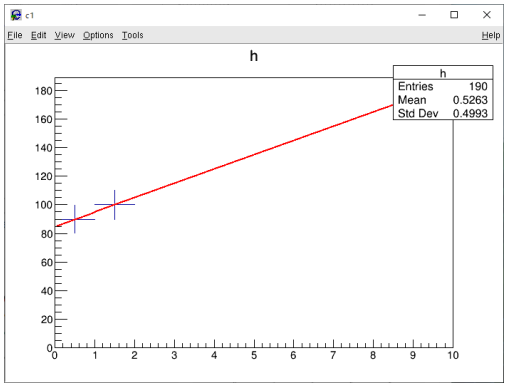
下図を作っているコードはfitting3.Cです。
興味がある方はコメントアウトを切り替えたり外したりしながら描画してみてください。

- 実はゼロイベントのビンがフィッティングに参加しない

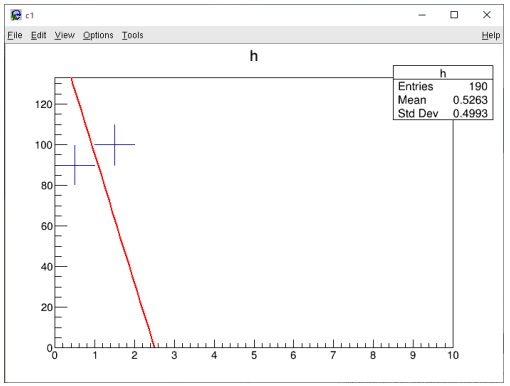
- 例：2ビンにエントリーがあるヒストグラムを直線でフィットしてみる
  - デフォルトはエントリーがあるビンの点をつなぐ直線
    - 3ビン目以降にイベントがなかったという情報が重要なら、このフィットは正しくない
  - "L"オプションは3ビン目も通るような直線
    - 関数がマイナスになっているところは、たぶん0にクリッピングされてると思われる挙動
    - likelihoodのときに関数が負になっているときはちょっと注意した方がいいかも

  - → これは極端な例だけど、どっちを使うべきか、は状況による

デフォルト



"L"



デフォルトの問題点

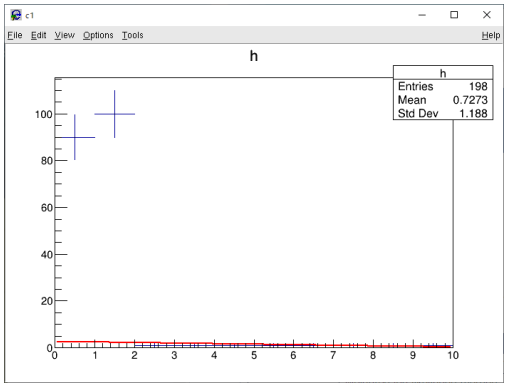
下図を作っているコードはfitting3.Cです。
興味がある方はコメントアウトを切り替えたり外したりしながら描画してみてください。

- 実はゼロイベントのビンがフィッティングに参加しない

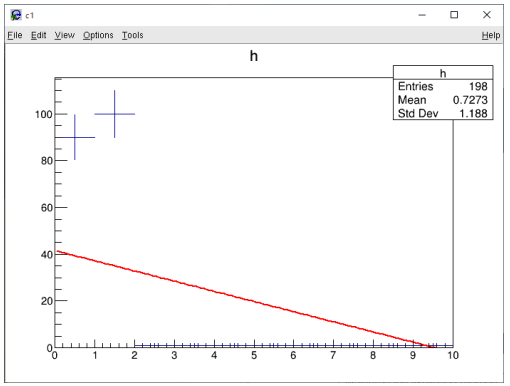
- 例：前ページの3~10ビンに1eventだけ追加したとき
  - デフォルトはほぼ下に張り付く直線
  - "L"オプションはなんとなく一番ずれるところが同じくらいになるような直線

  - → これもどっちを使うべきか、は状況による
  - ただし、イベント数を表すヒストグラムをフィットする場合、"L"の方が正しいと思う
    - rootの公式ページにもそう書いてある

デフォルト



"L"



デフォルトの問題点

https://root.cern.ch/doc/master/classTH1.html#a63eb028df86bc86c8e20c989eb23fb2a

- 本家

Chi-square Fits

By default a chi-square (least-square) fit is performed on the histogram. The so-called modified least-square method is used where the residual for each bin is computed using as error the observed value (the bin error) returned by `TH1::GetBinError`

$$Chi2 = \sum_i \left(\frac{y(i) - f(x(i)|p)}{e(i)} \right)^2$$

where `y(i)` is the bin content for each bin `i`, `x(i)` is the bin center and `e(i)` is the bin error (`sqrt(y(i)` for an un-weighted histogram). Bins with zero errors are excluded from the fit. See also later the note on the treatment of empty bins. When using option "I" the residual is computed not using the function value at the bin center, `f(x(i)|p)`, but the integral of the function in the bin, Integral{ f(x|p)dx }, divided by the bin volume. When using option P (Pearson chi2), the expected error computed as `e(i) = sqrt(f(x(i)|p))` is used. In this case empty bins are considered in the fit. Both chi-square methods should not be used when the bin content represent counts, especially in case of low bin statistics, because they could return a biased result.

Likelihood Fits

When using option "L" a likelihood fit is used instead of the default chi-square fit. The likelihood is built assuming a Poisson probability density function for each bin. The negative log-likelihood to be minimized is

$$NLL = -\sum_i \log \mathrm{P}(y(i)|f(x(i)|p))$$

where `P(y|f)` is the Poisson distribution of observing a count `y(i)` in the bin when the expected count is `f(x(i)|p)`. The exact likelihood used is the Poisson likelihood described in this paper: S. Baker and R. D. Cousins, "Clarification of the use of chi-square and likelihood functions in fits to histograms," Nucl. Instrum. Meth. 221 (1984) 437.

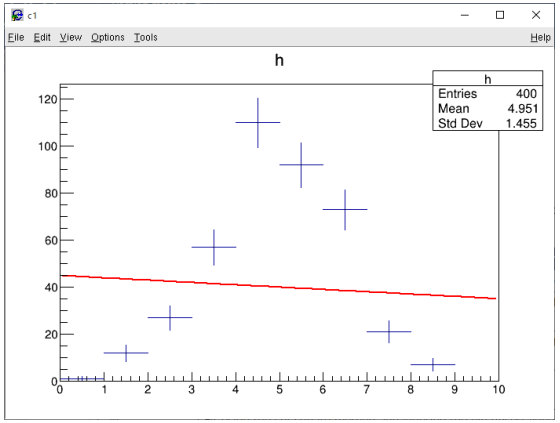
$$NLL = \sum_i \left(f(x(i)|p) + y(i) \log(y(i)/f(x(i)|p)) - y(i) \right)$$

By using this formulation, `2*NLL` can be interpreted as the chi-square resulting from the fit.

This method should be always used when the bin content represents counts (i.e. errors are sqrt(N)). The likelihood method has the advantage of treating correctly bins with low statistics. In case of high statistics/bin the distribution of the bin content becomes a normal distribution and the likelihood and the chi2 fit give the same result.

The likelihood method, although a bit slower, it is therefore the recommended method, when the histogram represent counts (Poisson statistics), where the chi-square methods may give incorrect results, especially in case of low statistics. In case of a weighted histogram, it is possible to perform also a likelihood fit by using the option "WL". Note a weighted histogram is a histogram which has been filled with weights and it has the information on the sum of the weight square for each bin (**TH1::Sumw2()** has been called). The bin error for a weighted histogram is the square root of the sum of the weight square.

フィットの妥当性の確認



- 例えば
  - ガウスっぽいヒストグラムを直線でフィットしてみる
  - 当然合ってない
  - どれくらいずれているのか、どういう指標で判断しよう？

- カイ二乗値
  - 各点と線の距離の二乗和
  - これが小さいほど合ってるし、大きいほどずれている

- どれくらい小さければ妥当？
  - カイ二乗値を自由度で割った値が1くらいならOK（詳しくは統計の教科書）
    - 自由度とは、ビン数-フィットパラメータ数、のこと
    - 上の例だと、10（ビン数）-2（傾きと切片で2パラメータ）=8になる
  - 極端に0に近いと、合い過ぎて不自然なので、誤差の評価などを見直したりする
  - 検定の話は消化不良になりそうなのでスキップ（統計の教科書を読む）
    - キーワード：信頼区間、カイ二乗検定、Confidence Level（C.L.）

フィットの妥当性の確認

- コード内で情報を取ってくる方法
  - TF1のメソッドに、GetChisquare()、GetNDF() というのがある
  - （ついでに）GetParameter(i)、GetParError(i) で関数のパラメータや誤差を取得できる

```
// fitting4.C

void fitting4(){

  // prepare histogram
  int bin_num=10;
  TH1D *h = new TH1D("h", "h;", bin_num, 0, 10);
  for(int i=0; i<400; i++) h->Fill(gRandom->Gaus(5,1.4));
  h->Draw("E");

  // fitting
  TF1 *f = new TF1("f","[0]*x+[1]",0,10);
  h->Fit(f,"L","");

  // show fitted infomation
  cout << endl;
  for(int i=0; i<f->GetNpar(); i++)
    cout << "p["<<i<<"] = " << f->GetParameter(i) << " +- " << f->GetParError(i) << endl;
  cout << endl;
  cout << "chi2 = " << f->GetChisquare() << endl;
  cout << "chi2/dof = " << f->GetChisquare()/f->GetNDF() << endl;

}
```

パラメータ数

パラメータ値

パラメータ誤差

カイ二乗値

自由度

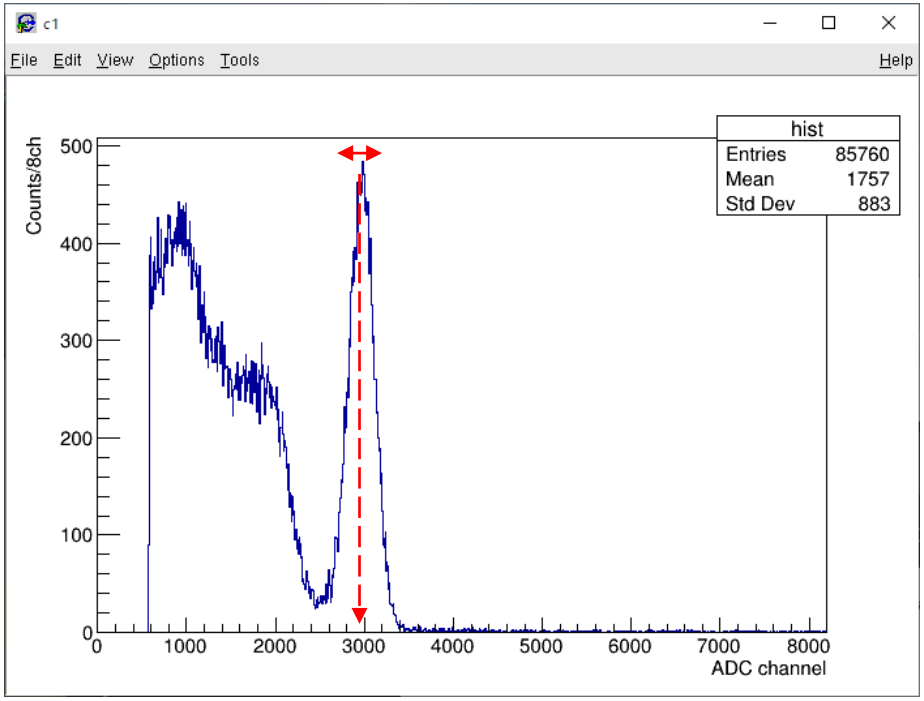
```
kiseki@dell-xps13:~/ymap2022$ root -l fitting4.C
root [0]
Processing fitting4.C...
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
 FCN=197.255 FROM MIGRAD    STATUS=CONVERGED     121 CALLS         122 TOTAL
                     EDM=2.29279e-08    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  p0           -9.87697e-01   1.32694e+00   3.68250e-03  -3.31902e-04
   2  p1            4.49382e+01   6.99239e+00   1.94052e-02  -7.23916e-05
                               ERR DEF= 0.5

p[0] = -0.987697 +- 1.32694
p[1] = 44.9382 +- 6.99239

chi2 = 3474.98
chi2/dof = 434.372
root [1]
```

chi2/dofが1よりめちゃくちゃ大きい
→フィッティングした関数は合ってないなあ

実験データ解析ぽいことをしてみよう

- ヒストグラムを解釈するために、フィッティングをすることがよくあります

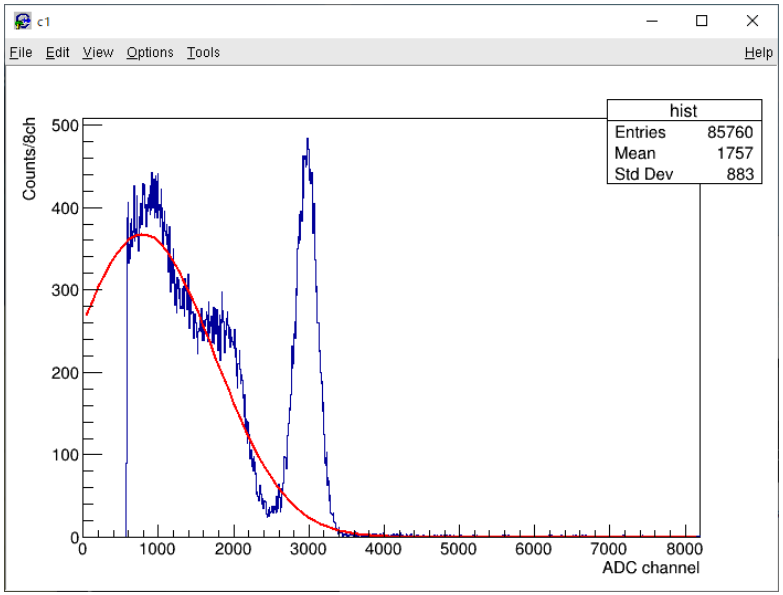
- 第2回で作ったヒストグラムの解釈
  - ピーク部分をガウス関数でフィット
  - mean → エネルギーキャリブレーション
  - sigma → エネルギー分解能
  - scintillator_exp/macro_examples/fit_hist1.C を動かしてみよう



実験データ解析ぽいことをしてみよう

- scintillator_exp/macro_examples/fit_hist1.C を実行してみよう
- 思ってたのと違いますね
- fit_hist1_v2.Cに行く前に、どうすればいいか少し考えてみよう

```
// fit_hist1.C

void fit_hist1(){

  // Define Range and Num of bin of histogram
  const int MCACh = 8192;
  const double HistMin = 0.;
  const double HistMax = 8192.;

  TH1D* hist = new TH1D("hist", "hist", MCACh, HistMin-0.5, HistMax-0.5);

  double BufferValue;
  ifstream ifs("../data/data1.txt");
  while(ifs >> BufferValue){
    hist->Fill(BufferValue);
  }

  const int rebin_factor=8;
  hist->Rebin(rebin_factor);
  hist->SetTitle(Form(";ADC channel;Counts/%dch",rebin_factor));
  hist->Draw();

  // fitting
  TF1 *func = new TF1("func", "gaus(0)", HistMin, HistMax);
  hist->Fit(func);

}
```



https://root.cern.ch/doc/master/classTF1.html

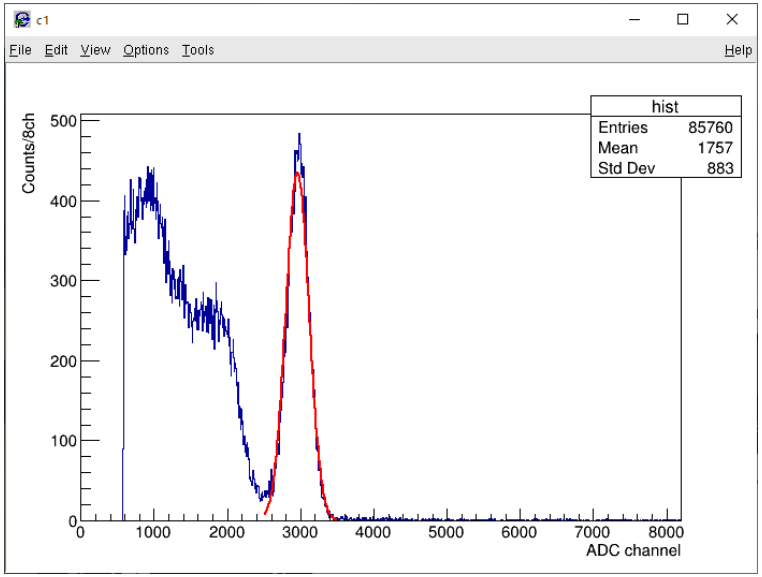
gaus(0) is a substitute for [0]*exp(-0.5*((x-[1])/[2])**2) and (0) means start numbering parameters at 0.

実験データ解析ぽいことをしてみよう

- フィッティングの範囲を指定すればOKですね
- （今回はイベント数のヒストグラムなので、"L"オプションも入れた）

- いい感じ

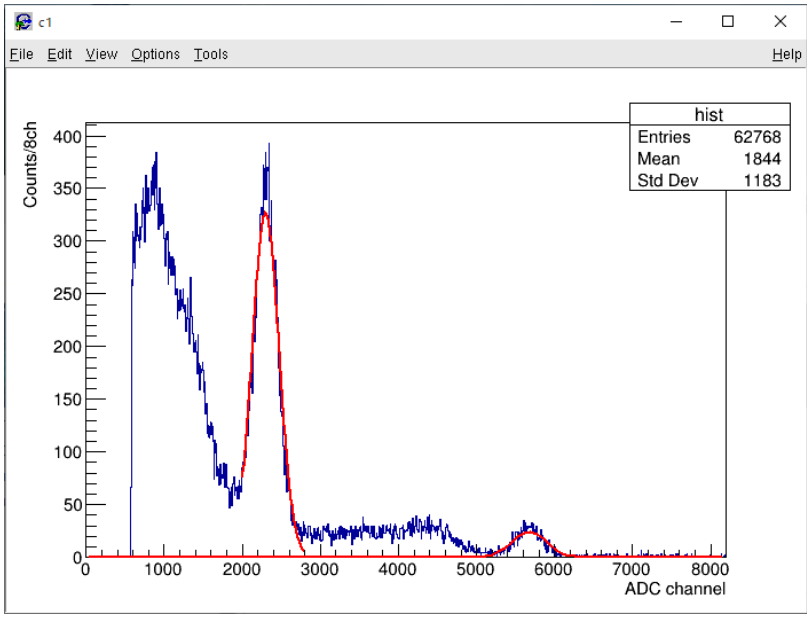
```
// fit_hist1_v2.C

void fit_hist1_v2(){

  // Define Range and Num of bin of histogram
  const int MCACh = 8192;
  const double HistMin = 0.;
  const double HistMax = 8192.;

  TH1D* hist = new TH1D("hist", "hist", MCACh, HistMin-0.5, HistMax-0.5);

  double BufferValue;
  ifstream ifs("../data/data1.txt");
  while(ifs >> BufferValue){
    hist->Fill(BufferValue);
  }

  const int rebin_factor=8;
  hist->Rebin(rebin_factor);
  hist->SetTitle(Form(";ADC channel;Counts/%dch",rebin_factor));
  hist->Draw();

  // fitting
  TF1 *func = new TF1("func", "gaus(0)", HistMin, HistMax);
  hist->Fit(func,"L","",2500,3500);
  cout << "(^o^) chi2/dof = " << func->GetChisquare()/func->GetNDF() << endl;

}
```



実験データ解析ぽいことをしてみよう

- data2 もフィッティングしてみよう
- ピークが2個ありますね

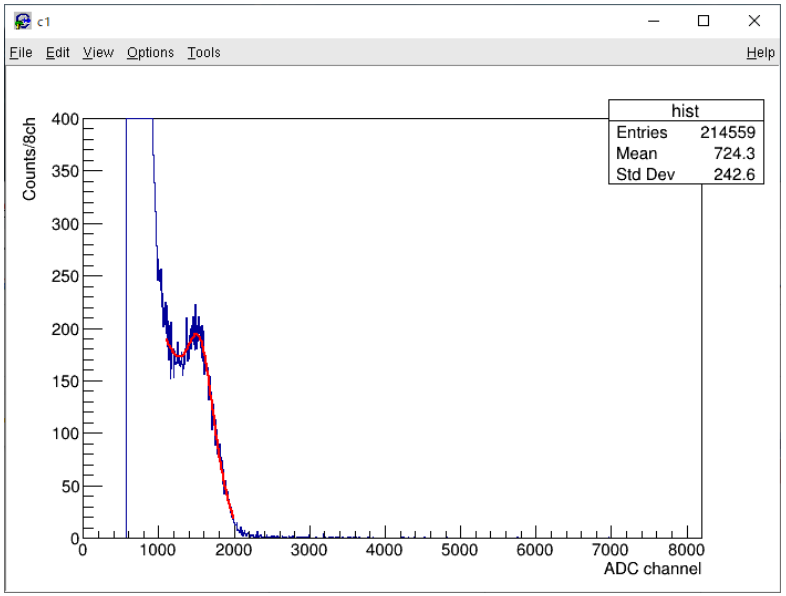
```
// fit_hist2.C

void fit_hist2(){

  // Define Range and Num of bin of histogram
  const int MCACh = 8192;
  const double HistMin = 0.;
  const double HistMax = 8192.;

  TH1D* hist = new TH1D("hist", "hist", MCACh, HistMin-0.5, HistMax-0.5);

  double BufferValue;
  ifstream ifs("../data/data2.txt");
  while(ifs >> BufferValue){
    hist->Fill(BufferValue);
  }

  const int rebin_factor=8;
  hist->Rebin(rebin_factor);
  hist->SetTitle(Form(";ADC channel;Counts/%dch",rebin_factor));
  hist->Draw();

  // fitting
  TF1 *func1 = new TF1("func1", "gaus(0)", HistMin, HistMax);
  func1->SetParameters(30, 5600, 200);
  hist->Fit(func1,"L","same",5200,6500);
  cout << "(-->) chi2/dof = " << func1->GetChisquare()/func1->GetNDF() << endl;

  TF1 *func2 = new TF1("func2", "gaus(0)", HistMin, HistMax);
  func2->SetParameters(300, 2400, 100);
  hist->Fit(func2,"L","",2000,2800);
  cout << "(<--) chi2/dof = " << func2->GetChisquare()/func2->GetNDF() << endl;
  func1->Draw("same");

}
```



実験データ解析ぽいことをしてみよう

- data3 もフィッティングしてみよう
- ピークが崖にかぶってますね

```
// fit_hist3.C

void fit_hist3(){

  // Define Range and Num of bin of histogram
  const int MCACh = 8192;
  const double HistMin = 0.;
  const double HistMax = 8192.;

  TH1D* hist = new TH1D("hist", "hist", MCACh, HistMin-0.5, HistMax-0.5);

  double BufferValue;
  ifstream ifs("../data/data3.txt");
  while(ifs >> BufferValue){
    hist->Fill(BufferValue);
  }

  const int rebin_factor=8;
  hist->Rebin(rebin_factor);
  hist->SetTitle(Form(";ADC channel;Counts/%dch",rebin_factor));
  hist->SetMaximum(400);
  hist->Draw();

  // fitting
  TF1 *func = new TF1("func", "gaus(0)+[3]+x*[4]", HistMin, HistMax);
  func->SetParameters(200, 1500, 100, 200, -0.1);
  hist->Fit(func,"L","same",1100,2000);
  cout << ":-) chi2/dof = " << func->GetChisquare()/func->GetNDF() << endl;

}
```



実験データ解析ぽいことをしてみよう

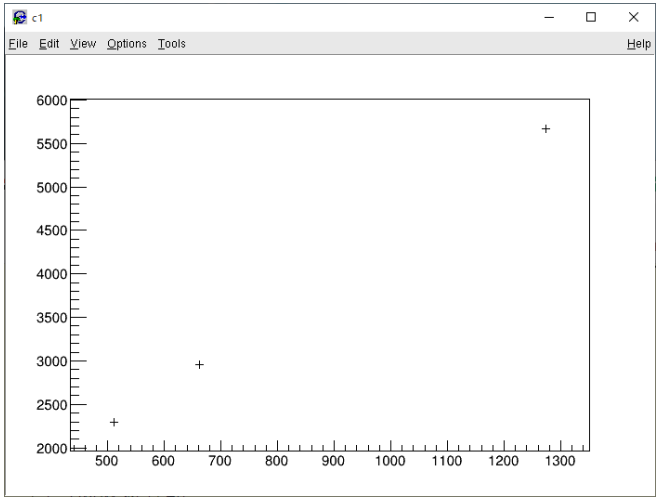
- data1~3 の mean と sigma をまとめてみよう
- データの測定条件は README.md を参照

- エネルギーが大きいほどmeanやsigmaが大きいように見える
- 数字の羅列だと、ぱっと見でそれ以上の情報を引き出すのが難しい
  - TGraph で可視化しよう

data	detector	energy	mean	sigma
data1	CsI	662 keV (137Cs)	2956.57±1.09	158.93±0.81
data2 右	CsI	1274keV (22Na)	5673.64±5.85	221.79±4.58
data2 左	CsI	511 keV (22Na)	2300.42±1.56	172.97±1.32
data3	GAGG	662keV (137Cs)	1556.57±6.36	155.98±7.88

TGraph

- TGraph で平面上の点列を描画できる
- draw_graph1.C を書いて実行してみよう
- エネルギーとADCが直線に乗ってそうですね




```
// draw_graph1.C

void draw_graph1(){

  // CsI mean data

  TGraph *graph = new TGraph();
  graph->SetPoint(0, 662,  2956.57);
  graph->SetPoint(1, 1274, 5673.64);
  graph->SetPoint(2, 511,  2300.42);

  graph->SetMarkerStyle(2); // default point is very small

  graph->Draw("AP");

}
```

TGraphを引数無しで定義

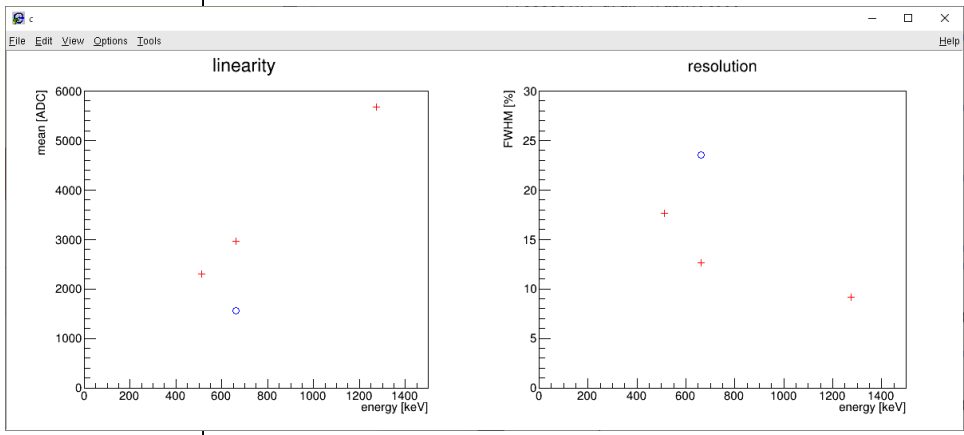
SetPoint(i,x,y) メソッドで
i番目の点として(x,y)を登録

DrawOptionに "A" を入れると
枠が自動(auto)で作られる

TGraph：もうちょいきれいに

- CsI / GAGG の線形性とエネルギー分解能のエネルギー依存性を描画

```
// draw_graph2.C

void draw_graph2(){

  double ene[4] = {662, 1274, 511, 662};
  double mean[4] = {2956.57, 5673.64, 2300.42, 1556.57};
  double sigma[4] = {158.93, 221.79, 172.97, 155.98};

  TGraph *g_CsI_mean = new TGraph();
  for(int i=0; i<3; i++) g_CsI_mean->SetPoint(i, ene[i], mean[i]);

  TGraph *g_CsI_resolution = new TGraph();
  for(int i=0; i<3; i++) g_CsI_resolution->SetPoint(i, ene[i], sigma[i]/mean[i]*100*2.35);

  TGraph *g_GAGG_mean = new TGraph();
  g_GAGG_mean->SetPoint(0, ene[3], mean[3]);

  TGraph *g_GAGG_resolution = new TGraph();
  g_GAGG_resolution->SetPoint(0, ene[3], sigma[3]/mean[3]*100*2.35);

  g_CsI_mean->SetMarkerStyle(2);
  g_CsI_mean->SetMarkerColor(2);
  g_CsI_resolution->SetMarkerStyle(2);
  g_CsI_resolution->SetMarkerColor(2);
  g_GAGG_mean->SetMarkerStyle(4);
  g_GAGG_mean->SetMarkerColor(4);
  g_GAGG_resolution->SetMarkerStyle(4);
  g_GAGG_resolution->SetMarkerColor(4);

  TCanvas *c = new TCanvas("c","c",1200,500);
  c->Divide(2,1);
  for(int i=0; i<2; i++) c->cd(1)->SetMargin(0.15,0.1,0.1,0.1);

  c->cd(1)->DrawFrame(0,0,1500,6000,"linearity; energy [keV]; mean [ADC]");
  g_CsI_mean->Draw("sameP");
  g_GAGG_mean->Draw("sameP");

  c->cd(2)->DrawFrame(0,0,1500,30,"resolution; energy [keV]; FWHM [%]");
  g_CsI_resolution->Draw("sameP");
  g_GAGG_resolution->Draw("sameP");

}
```



最初にキャンバスを書いてから
グラフを"same"オプションで重ね書き

おわり

- やったこと
  - TF1：いろんな関数を定義して描く
  - Fitting：ヒストグラムを関数であてはめる。カイ二乗フィットとLikelihoodフィットがあり、統計量のときはLikelihoodが吉
  - TGraph：二次元平面上での点列を定義して描く

- YMAP入会
  - https://www.icrr.u-tokyo.ac.jp/YMAP/join.html
  - 講師も募集中