ROOT

ROOT とは？

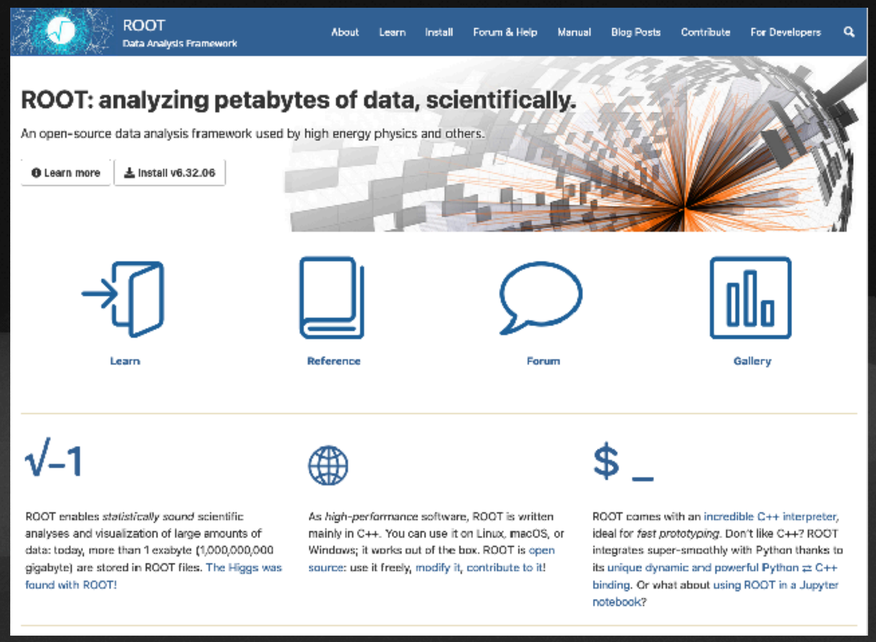
CERN が開発したデータ解析フレームワーク（≒ツールやツールを作るためのツール群）。高エネルギー物理分野でよく用いられるが、宇宙分野でも使われているようだ。

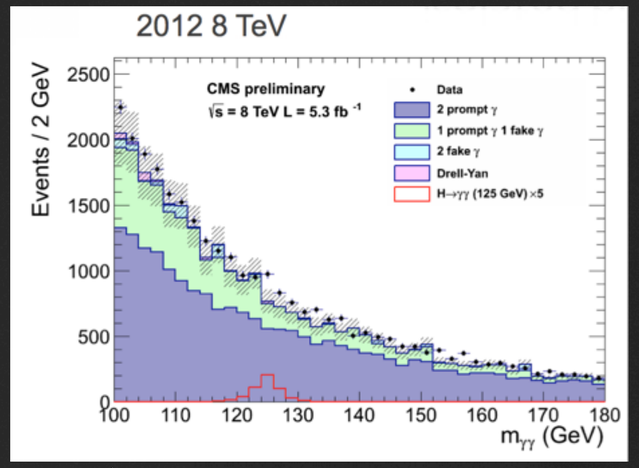
ROOT を用いて

- グラフ・ヒストグラム・関数の作成と描画
- グラフやヒストグラムに対するフィッティング
- TTree を使ったクロス集計（カットを掛けたデータの集計）
- 4 元運動量の計算
- 等など

をよく行う。

基本的に C/C++ でプログラムを記述する。多くのツールは C++ のクラスとして実装されている。





ヒッグス粒子の発見も ROOT を使った解析によるもの

自分で勉強するためのテキスト

日本語の資料はさほど多くない：

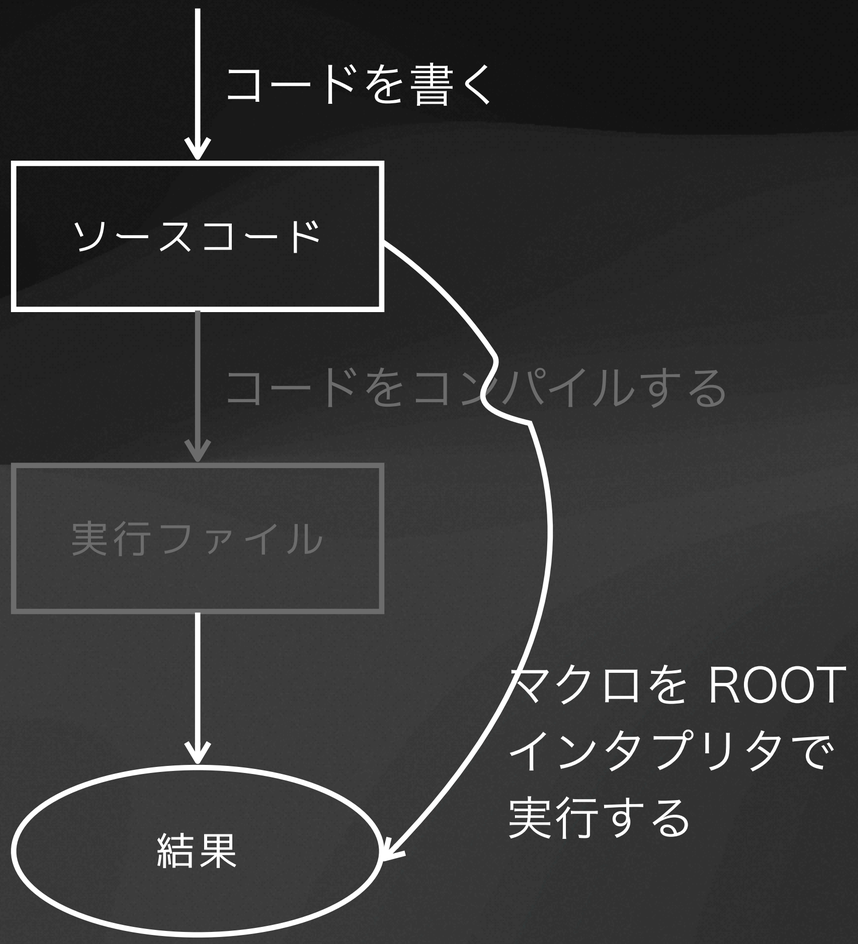
- 猿でもできる ROOT：内容が古いが全 28 ページしかないので簡単に終わる。
- ねこだんしゃくの ROOT 指南書：これも古いが、読みやすい。
- 高エネルギー宇宙物理学のための ROOT 入門：C/C++ の基礎から ROOT までを扱う。分野は違うが問題なし。
- 糠塚の資料はこちら（ちゃんと書いていないが）

英語だと

- ROOT beginners' guide:
- 公式マニュアル：読み物ではない
- チュートリアル：サンプルを実行しながら勉強できるので便利
- API ドキュメント（特にクラスインデックス）：辞書的に使う

がある。

ROOT マクロ開発・実行の手順



プログラミング言語は

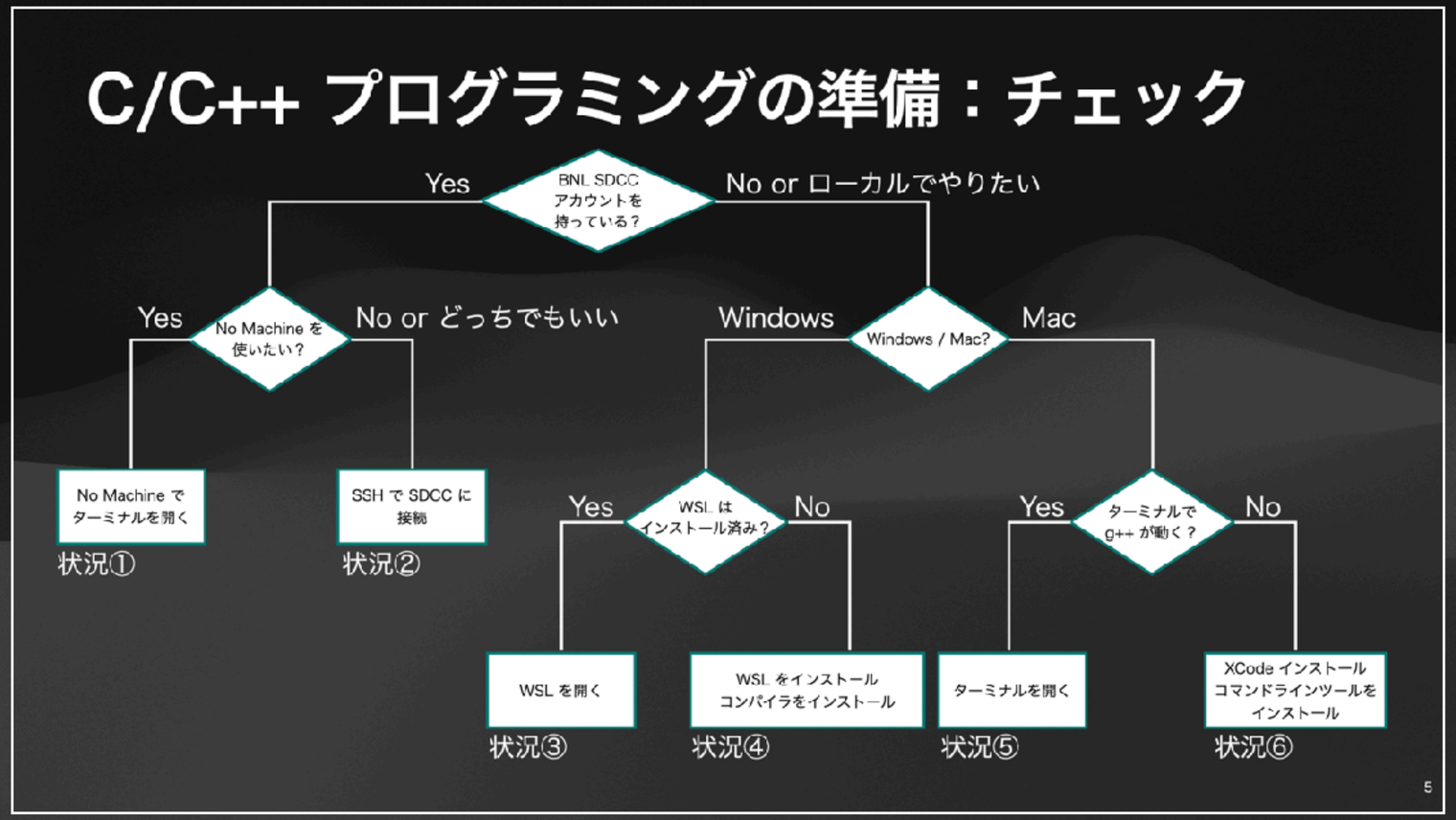
- コンパイルが必要
  - C/C++, …
- コンパイルは不要
  - Python, Javascript, …

と分類できる

ROOT マクロは C/C++ で書くが、インタープリタを使って直接マクロを実行することができる

（ROOT インタープリタでソースコードをコンパイルすることもできて、色々とややこしい）

ROOT のインストール



状況 ③〜⑥ の人は ROOT を自分でインストールする必要があります。

ROOT のインストール

1. ROOT のダウンロード
   1. 自分にあったバージョンを見つける
      (糠塚は .pkg より tar.gz をおすすめします)
   2. リンクをクリックしてダウンロードする
2. ROOT をインストールする場所を決める
   ・よくわからなかったら、
   mkdir $HOME/ROOT
   でホームディレクトリ下に ROOT ディレクトリを作り、
   mv [ダウンロードした tar.gz] $HOME/ROOT/
   で $HOME/ROOT/ 下に移動する
3. cd で $HOME/ROOT に移動し、[ダウンロードした tar.gz] を展開する
   tar fvxz [ダウンロードした tar.gz]
4. $HOEM/ROOT/bin に移動し、
   source thisroot.sh
   を実行する
5. ターミナルで root コマンドを実行し、動くことを確認する

Binary distributions

Instead of manually downloading this binary, please explore first whether your package manager already provides this version. This way, you will automatically keep up-to-date with the latest stable versions with no manual maintenance on your side.

Platform	Files	Size
Almalinux 8.10	root_v6.32.06.Linux-almalinux8.10-x86_64-gcc8.5.tar.gz	281M
Almalinux 9.4	root_v6.32.06.Linux-almalinux9.4-x86_64-gcc11.4.tar.gz	297M
Fedora 39	root_v6.32.06.Linux-fedora39-x86_64-gcc13.3.tar.gz	284M
Ubuntu 20.04	root_v6.32.06.Linux-ubuntu20.04-x86_64-gcc9.4.tar.gz	288M
Ubuntu 22.04	root_v6.32.06.Linux-ubuntu22.04-x86_64-gcc11.4.tar.gz	286M
Ubuntu 24.04	root_v6.32.06.Linux-ubuntu24.04-x86_64-gcc13.2.tar.gz	285M
macOS 13.7 arm64 Xcode 15	root_v6.32.06.macos-13.7-arm64-clang150.pkg	415M
macOS 13.7 arm64 Xcode 15	root_v6.32.06.macos-13.7-arm64-clang150.tar.gz	269M
macOS 14.7 x86_64 Xcode 16	root_v6.32.06.macos-14.7-x86_64-clang160.pkg	446M
macOS 14.7 x86_64 Xcode 16	root_v6.32.06.macos-14.7-x86_64-clang160.tar.gz	292M
macOS 15.0 arm64 Xcode 16	root_v6.32.06.macos-15.0-arm64-clang160.pkg	435M
macOS 15.0 arm64 Xcode 16	root_v6.32.06.macos-15.0-arm64-clang160.tar.gz	283M
Windows Visual Studio 2022 32-bit x86	root_v6.32.06.win32.vc17.exe	117M
Windows Visual Studio 2022 32-bit x86	root_v6.32.06.win32.vc17.zip	160M
Windows Visual Studio 2022 64-bit x64	root_v6.32.06.win64.vc17.exe	122M
Windows Visual Studio 2022 64-bit x64	root_v6.32.06.win64.vc17.zip	167M

※ 細かいことは個別に対応します

20 分

やってみよう１：１ 次関数を書く

ソース：function_1.cc

```
 1 int function_1d()
 2 {
 3
 4   TF1* f = new TF1( "name", "[0] + [1] * x", 0, 10 );
 5   f->SetParameter( 0, 5 );
 6   f->SetParameter( 1, 0.5);
 7
 8   f->Draw();
 9
10   return 0;
11 }
```

オブジェクト名前

関数型: $p_0 + p_1 x$

x の 最小値　最大値

←1 次元関数クラス TF1 のオブジェクト作成

← パラメータ p_0 の設定

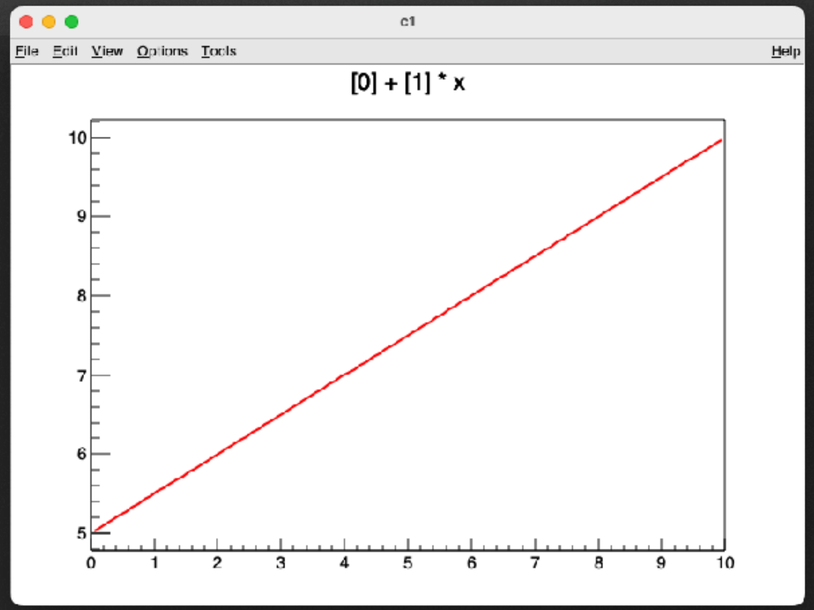
← パラメータ p_1 の設定

パラメータの値

パラメータ番号

ターミナルから実行する

```
[nukazuka 21:19:56 ROOT_sample] $ root function_1d.cc
root [0]
Processing function_1d.cc...
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
(int) 0
root [1] .q ← 終了は .q
```



やってみよう１：１ 次関数を書く

ファイルは用意しないので、
自分でコードを書いて
実行してみましょう

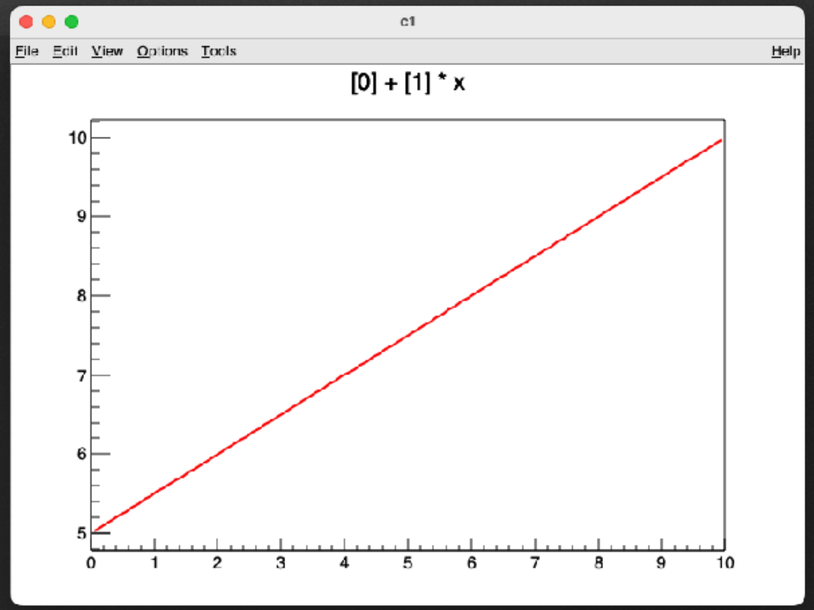
ソース：function_1.cc

```
 1 int function_1d()
 2 {
 3
 4   TF1* f = new TF1( "name", "[0] + [1] * x", 0, 10 );
 5   f->SetParameter( 0, 5 );
 6   f->SetParameter( 1, 0.5);
 7
 8   f->Draw();
 9
10   return 0;
11 }
```

オブジェクト名前
関数型: $p_0 + p_1 x$
x の 最小値
最大値
←1 次元関数クラス TF1 のオブジェクト作成
← パラメータ p_0 の設定
← パラメータ p_1 の設定
パラメータの値
パラメータ番号

ターミナルから実行する

```
[nukazuka 21:19:56 ROOT_sample] $ root function_1d.cc
root [0]
Processing function_1d.cc...
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
(int) 0
root [1] .q ← 終了は .q
```



やってみよう１：１ 次関数を書く + α（紹介だけ）

ソース：function_1.cc

```
1 int function_1d()
2 {
3
4   TF1* f = new TF1( "name", "[0] + [1] * x", 0, 10 );
```

関数型の指定方法はいろいろある

1. 直接書く（例のやりかた）
2. ROOT 内で定義されている関数を使う

例

```
TF1* f_gaus = new TF1( “name”, “gaus”, -10, 10 );
f_gaus->SetParameter( 0, 1 ); // p0, 高さ
f_gaus->SetParameter( 1, 0 ); // p1, 平均
f_gaus->SetParameter( 2, 1 ); // p2, 標準偏差
```

$$f(x) = p_0 \exp\left\{\frac{-1}{2}\left(\frac{x-p_1}{p_2}\right)^2\right\}$$

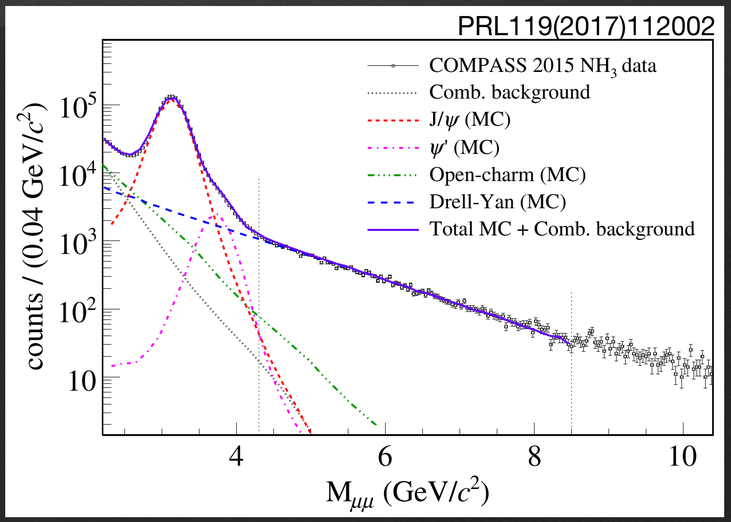
3. TMath クラスに定義されている関数を使う（tanh, arcsin など）
4. C や C++ でいう関数を使う（≒ なんでもできる）

List of predefined functions

The list of available predefined functions which can be used as shortcuts is the following:

1. One Dimensional functions:
   - gaus is a substitute for [Constant]*exp(-0.5*((x-[Mean])/[Sigma])*((x-[Mean])/[Sigma]))
   - landau is a substitute for [Constant]*TMath::Landau (x,[MPV],[Sigma],false)
   - expo is a substitute for exp([Constant]+[Slope]*x)
   - crystalball is substitute for [Constant]*ROOT::Math::crystalball_function (x,[Alpha],[N],[Sigma],[Mean])
   - breitwigner is a substitute for [p0]*ROOT::Math::breitwigner_pdf (x,[p2],[p1])
   - pol0,1,2,...N is a substitute for a polynomial of degree N : ([p0]+[p1]*x+[p2]*pow(x,2)+....[pN]*pow(x,N)
   - cheb0,1,2,...N is a substitute for a Chebyshev polynomial of degree N: ROOT::Math::Chebyshev10(x,[p0],[p1],[p2],...[pN]). Note the maximum N allowed here is 10.
2. Two Dimensional functions:
   - xygaus is a substitute for [Constant]*exp(-0.5*pow(((x-[MeanX])/[SigmaX]),2)- 0.5*pow(((y-[MeanY])/[SigmaY]),2)), a 2d Gaussian without correlation
   - bigaus is a substitute for [Constant]*ROOT::Math::bigaussian_pdf (x,y,[SigmaX],[SigmaY],[Rho],[MeanX],[MeanY]), a 2d gaussian including a correlation parameter.
3. Three Dimensional functions:
   - xyzgaus is for a 3d Gaussians without correlations:
     [Constant]*exp(-0.5*pow(((x-[MeanX])/[SigmaX]),2)- 0.5*pow(((y-[MeanY])/[SignaY]),2)- 0.5*pow(((z-[MeanZ])/[SigmaZ]),2))

ROOT 内で定義されている関数の一覧（リンク）



ヒストグラムをフィッティング関数に組み込むこともできる（テンプレートフィト）

10 分

やってみよう２：グラフを書く

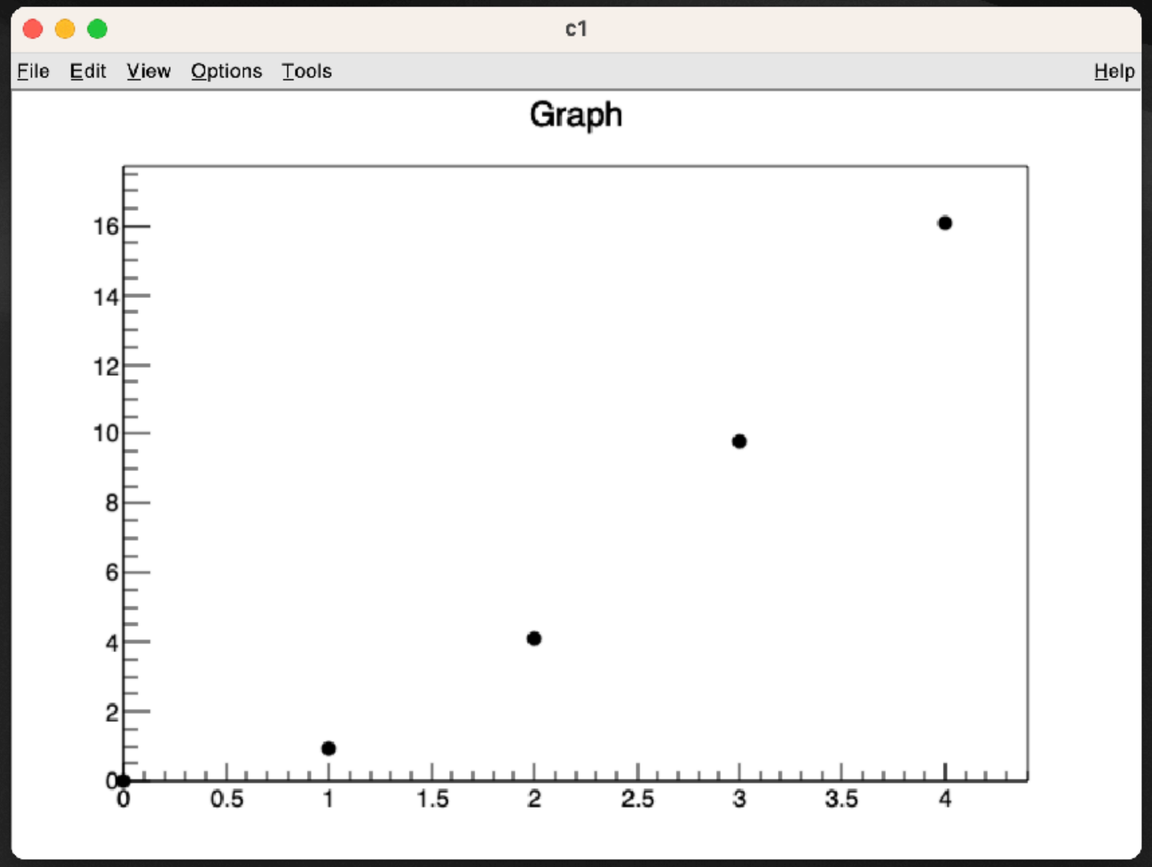
ファイルは用意しないので、
自分でコードを書いて
実行してみましょう

ソース：graph.cc

```
1  int graph()
2  {
3
4    const int kData_num = 5; // 点の数
5    double x[kData_num] = { 0.0, 1.0, 2.0, 3.0, 4.0 }; // データ点の x 座標
6    double y[kData_num] = { 0.0, 0.95, 4.1, 9.8, 16.1 }; // データ点の y 座標
7
8    TGraph* g = new TGraph( kData_num, x, y ); // グラフオブジェクト作成
9    g->SetMarkerStyle( 20 ); // 点のスタイルを変更
10   g->Draw( "AP" ); // グラフを描く，A: 描画範囲を自動で決定，P: 点を描画
11
12   return 0;
13 }
```

ターミナルから実行する

```
[genki 01:43:34 root_tutorial] $ root graph.cc
root [0]
Processing graph.cc...
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
(int) 0
root [1] .q
```



10 分

やってみよう３：グラフにフィッティング

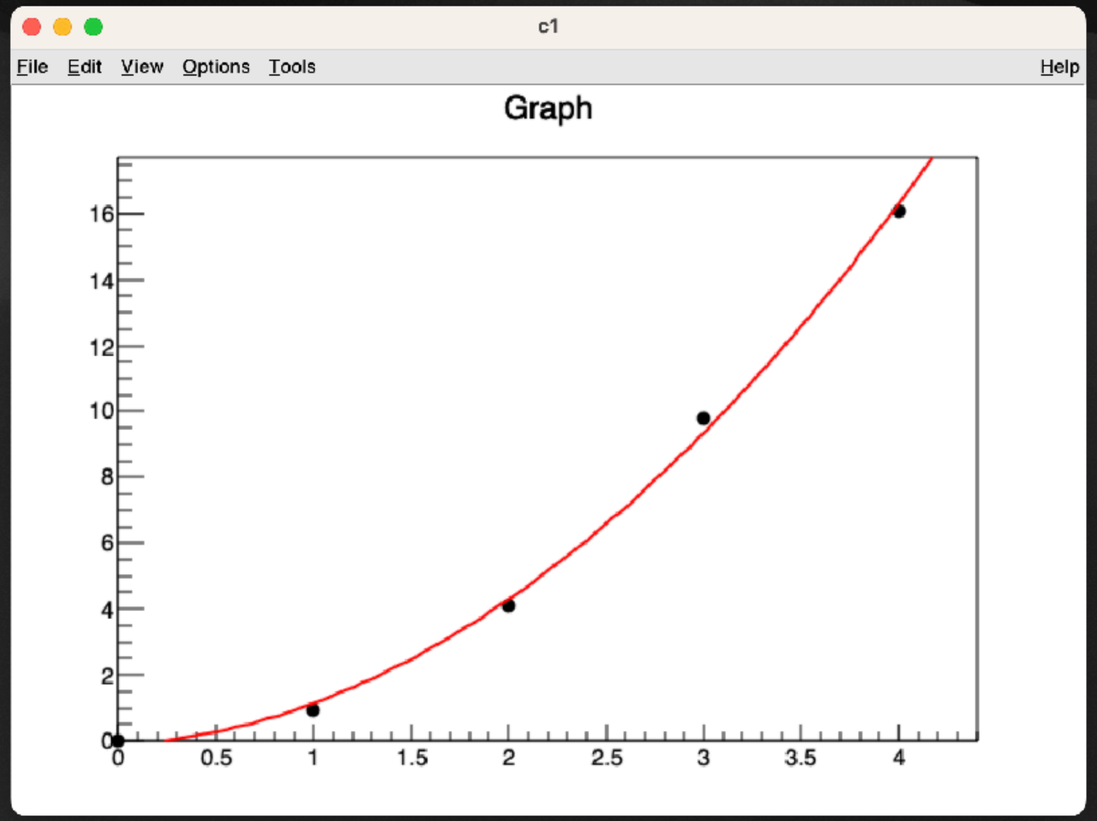
ファイルは用意しないので、
自分でコードを書いて
実行してみましょう

ソース：graph.cc

```
1  int graph()
2  {
3
4    const int kData_num = 5; // 点の数
5    double x[kData_num] = { 0.0, 1.0, 2.0, 3.0, 4.0 }; // データ点の x 座標
6    double y[kData_num] = { 0.0, 0.95, 4.1, 9.8, 16.1 }; // データ点の y 座標
7
8    TGraph* g = new TGraph( kData_num, x, y ); // グラフオブジェクト作成
9    g->SetMarkerStyle( 20 ); // 点のスタイルを変更
10
11   TF1* f = new TF1( "name", "pol2", 0, 20 ); // 2 次関数のオブジェクト作成
12   g->Fit( f ); // グラフに関数 f をフィットする
13
14   g->Draw( "AP" ); // グラフを描く，A: 描画範囲を自動で決定，P: 点を描画
15   return 0;
16 }
```

ターミナルから実行する

```
[genki 01:46:36 root_tutorial] $ root graph.cc
root [0]
Processing graph.cc...
****************************************
Minimizer is Linear / Migrad
Chi2                      =     0.331571
NDf                       =            2
p0                        =    -0.127143   +/-   0.383195
p1                        =     0.319286   +/-   0.453925
p2                        =     0.946429   +/-   0.10882
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
(int) 0
root [1] .q
```



10 分

やってみよう４：ヒストグラム

ソース：histogram.cc

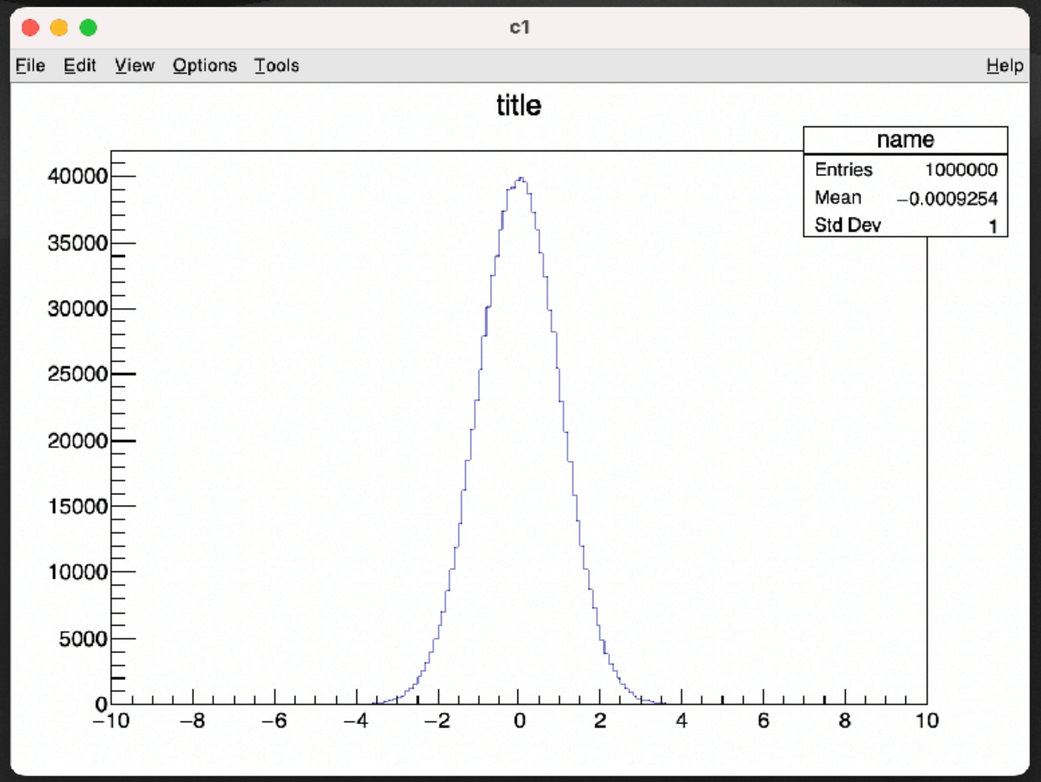
```
1  int histogram()
2  {
3    TRandom3 rand(0); // 乱数クラスのオブジェクト作成
4
5    // 1 次元ヒストグラムのオブジェクト作成
6    TH1D* hist = new TH1D( "name", "title", 200, -10, 10 );
7
8
9
10
11   // for ループで乱数をヒストグラムに詰める
12   for( int i=0; i<1e6; i++ )
13     {
14       double value = rand.Gaus(); // 平均0, 標準偏差 1 のガウス分布に従う乱数を作成
15       hist->Fill( value ); // 乱数をヒストグラムに詰める
16
17     }
18
19   hist->Draw(); // ヒストグラムを描画する
20   return 0;
21 }
```

最大値
ヒストグラムの名前　タイトル　範囲の最小値
範囲の分割数（ビンの数）

ファイルは用意しないので、
自分でコードを書いて
実行してみましょう

ターミナルから実行する

```
[genki 12:38:58 root_tutorial] $ root histogram.cc
root [0]
Processing histogram.cc...
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
(int) 0
root [1] .q
```



10 分

やってみよう５：ヒストグラムにフィッティング

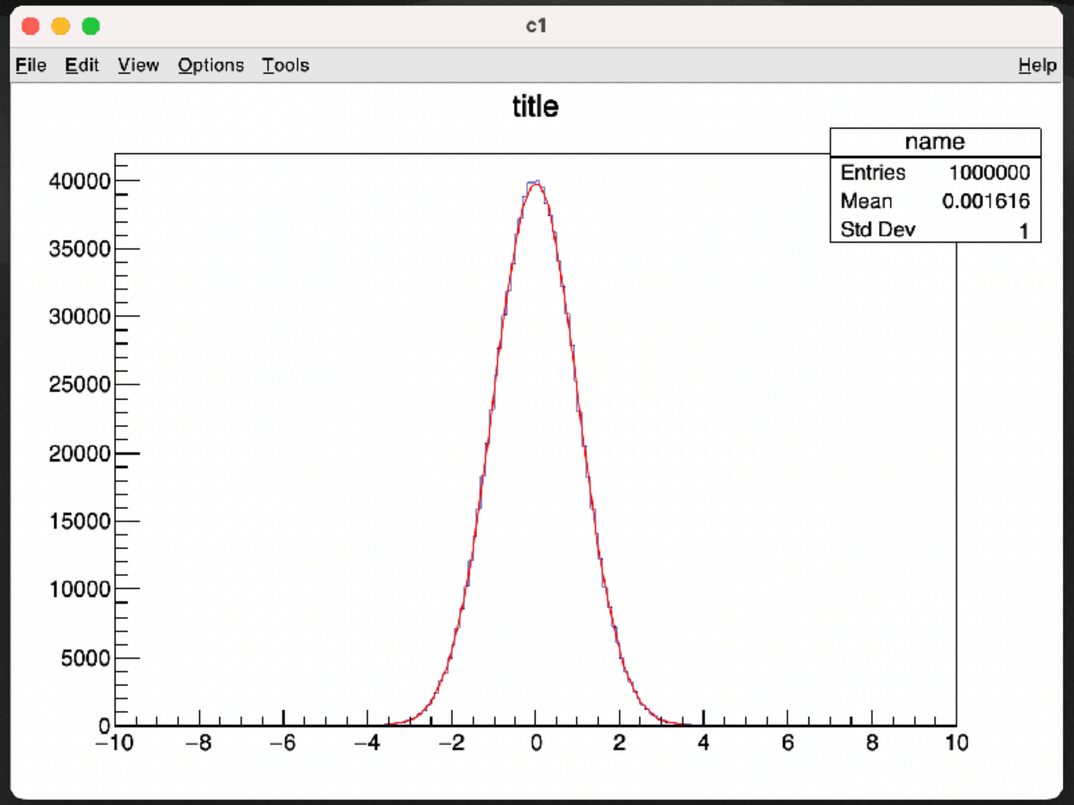
ファイルは用意しないので、
自分でコードを書いて
実行してみましょう

ソース：histogram.cc

```
1  int histogram()
2  {
3    TRandom3 rand(0); // 乱数クラスのオブジェクト作成
4
5    // 1 次元ヒストグラムのオブジェクト作成
6    TH1D* hist = new TH1D( "name", "title", 200, -10, 10 );
7
8    // for ループで乱数をヒストグラムに詰める
9    for( int i=0; i<1e6; i++ )
10     {
11       double value = rand.Gaus(); // 平均0, 標準偏差 1 のガウス分布に従う乱数を作成
12       hist->Fill( value ); // 乱数をヒストグラムに詰める
13
14     }
15
16   TF1* f = new TF1( "function", "gaus", -10, 10 ); // ガウス関数作成
17   hist->Fit( f ); // ヒストグラムにフィッティング
18
19   hist->Draw(); // ヒストグラムを描画する
20   return 0;
21 }
```

ターミナルから実行する

```
[genki 12:44:20 root_tutorial] $ root histogram.cc
root [0]
Processing histogram.cc...
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
****************************************
Minimizer is Minuit2 / Migrad
Chi2                      =      83.0303
NDf                       =           91
Edm                       =  2.97329e-10
NCalls                    =           56
Constant                  =      39869.1   +/-   48.7908
Mean                      =   0.00159563   +/-   0.00100064
Sigma                     =      1.00055   +/-   0.000705808     (limited)
(int) 0
root [1] .q
```

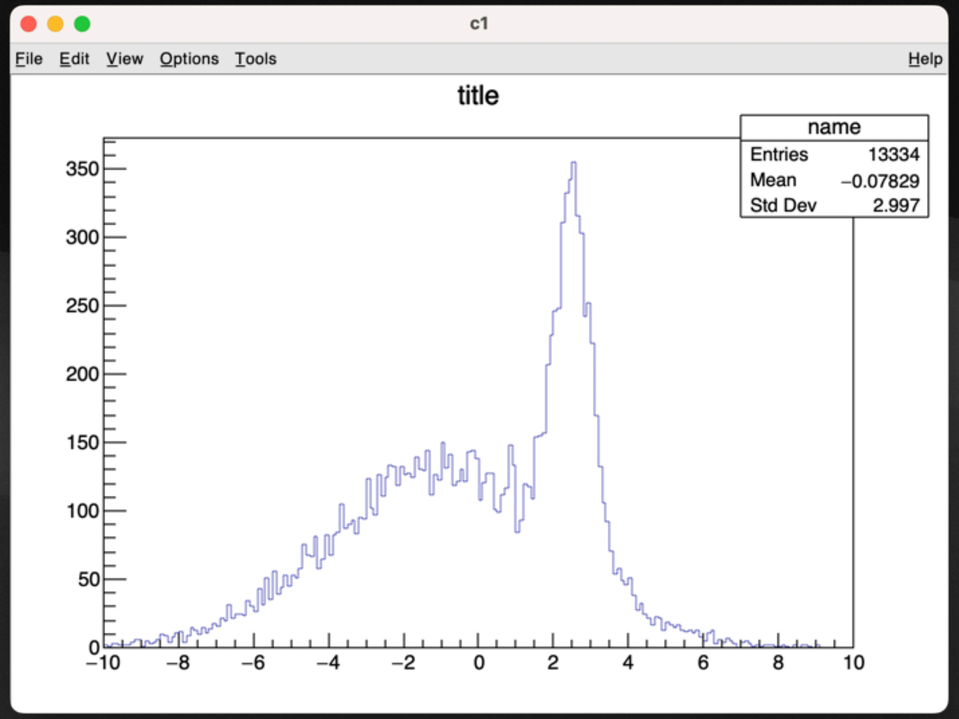


10 分

やってみよう５：ヒストグラムにフィッティング②

ソース：histogram.cc

```
int histogram()
{
  TRandom3 rand(0); // 乱数クラスのオブジェクト作成

  // 1 次元ヒストグラムのオブジェクト作成
  TH1D* hist = new TH1D( "name", "title", 200, -10, 10 );

  // for ループで乱数をヒストグラムに詰める
  for( int i=0; i<1e4; i++ )
    {
      double value = rand.Gaus( -1, 3); // 平均0, 標準偏差 3 のガウス分布に従う乱数を作成
      hist->Fill( value ); // 乱数をヒストグラムに詰める

      // i を 3 で割ったときのあまりが 0 のときだけ追加で別のガウス分布の乱数を詰める
      if( i%3 == 0 )
        hist->Fill( rand.Gaus( 2.5, 0.5 ) );

    }

  TF1* f = new TF1( "function", "gaus(0) + gaus(3)", -10, 10 ); // ガウス関数作成

  /* // 最初はパラメータの初期値設定なし，後で初期値設定をしてみよう
  f->SetParameter( 0, 1e5 ); //   1 つ目のガウス関数の高さ
  f->SetParameter( 1, 0.0 ); // 1 つ目のガウス関数の平均
  f->SetParameter( 2, 1.0 ); //   1 つ目のガウス関数の標準偏差
  f->SetParameter( 3, 1e5 ); //   1 つ目のガウス関数の高さ
  f->SetParameter( 4, 1.0 ); // 1 つ目のガウス関数の平均
  f->SetParameter( 5, 1.0 ); //   1 つ目のガウス関数の標準偏差
  */
  hist->Fit( f ); // ヒストグラムにフィッティング

  hist->Draw(); // ヒストグラムを描画する
  return 0;
}
```



2 種類のガウス分布の和にフィットしてみよう

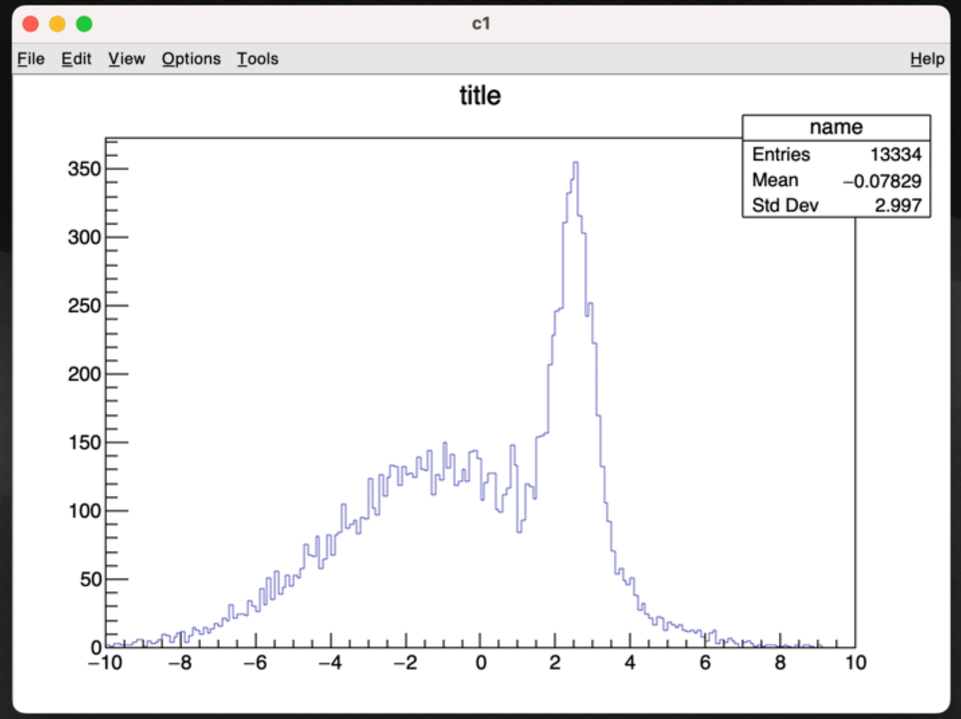
パラメータの初期値設定が大事になることもある
コメントを解除して初期値を設定してみよう

10 分

やってみよう５：ヒストグラムにフィッティング②

ソース：histogram.cc

```
1  int histogram()
2  {
3    TRandom3 rand(0); // 乱数クラスのオブジェクト作成
4
5    // 1 次元ヒストグラムのオブジェクト作成
6    TH1D* hist = new TH1D( "name", "title", 200, -10, 10 );
7
8    // for ループで乱数をヒストグラムに詰める
9    for( int i=0; i<1e4; i++ )
10     {
11       double value = rand.Gaus( -1, 3); // 平均0, 標準偏差 3 のガウス分布に従う乱数を作成
12       hist->Fill( value ); // 乱数をヒストグラムに詰める
13
14       // i を 3 で割ったときのあまりが 0 のときだけ追加で別のガウス分布の乱数を詰める
15       if( i%3 == 0 )
16         hist->Fill( rand.Gaus( 2.5, 0.5 ) );
17
18     }
19
20   TF1* f = new TF1( "function", "gaus(0) + gaus(3)", -10, 10 ); // ガウス関数作成
21
22   /* // 最初はパラメータの初期値設定なし，後で初期値設定をしてみよう
23   f->SetParameter( 0, 1e5 ); //   1 つ目のガウス関数の高さ
24   f->SetParameter( 1, 0.0 ); // 1 つ目のガウス関数の平均
25   f->SetParameter( 2, 1.0 ); //   1 つ目のガウス関数の標準偏差
26   f->SetParameter( 3, 1e5 ); //   1 つ目のガウス関数の高さ
27   f->SetParameter( 4, 1.0 ); // 1 つ目のガウス関数の平均
28   f->SetParameter( 5, 1.0 ); //   1 つ目のガウス関数の標準偏差
29   */
30   hist->Fit( f ); // ヒストグラムにフィッティング
31
32   hist->Draw(); // ヒストグラムを描画する
33   return 0;
34 }
```



2 種類のガウス分布の和にフィットしてみよう

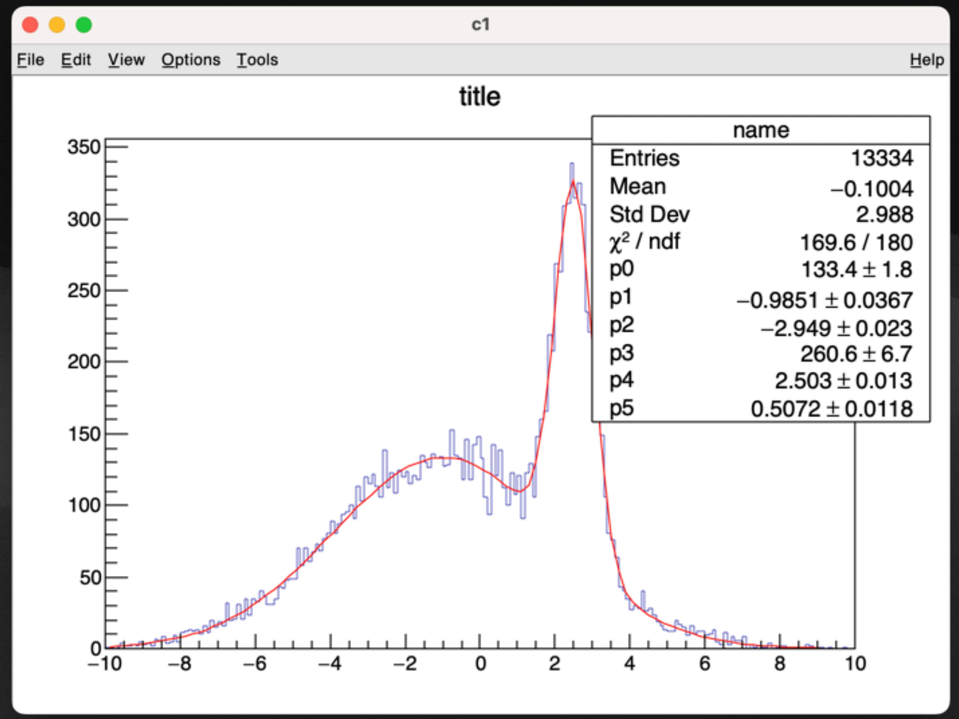
パラメータの初期値設定が大事になることもある
コメントを解除して初期値を設定してみよう

10 分

やってみよう５：ヒストグラムにフィッティング②

ソース：histogram.cc

```
int histogram()
{
  TRandom3 rand(0); // 乱数クラスのオブジェクト作成

  // 1 次元ヒストグラムのオブジェクト作成
  TH1D* hist = new TH1D( "name", "title", 200, -10, 10 );

  // for ループで乱数をヒストグラムに詰める
  for( int i=0; i<1e4; i++ )
    {
      double value = rand.Gaus( -1, 3); // 平均0, 標準偏差 3 のガウス分布に従う乱数を作成
      hist->Fill( value ); // 乱数をヒストグラムに詰める

      // i を 3 で割ったときのあまりが 0 のときだけ追加で別のガウス分布の乱数を詰める
      if( i%3 == 0 )
        hist->Fill( rand.Gaus( 2.5, 0.5 ) );

    }

  TF1* f = new TF1( "function", "gaus(0) + gaus(3)", -10, 10 ); // ガウス関数作成

  // 最初はパラメータの初期値設定なし，後で初期値設定をしてみよう
  f->SetParameter( 0, 1e5 ); //  １つ目のガウス関数の高さ
  f->SetParameter( 1, 0.0 ); // １つ目のガウス関数の平均
  f->SetParameter( 2, 1.0 ); //  １つ目のガウス関数の標準偏差
  f->SetParameter( 3, 1e5 ); //  １つ目のガウス関数の高さ
  f->SetParameter( 4, 1.0 ); // １つ目のガウス関数の平均
  f->SetParameter( 5, 1.0 ); //  １つ目のガウス関数の標準偏差
  hist->Fit( f ); // ヒストグラムにフィッティング

  gStyle->SetOptFit( true );
  hist->Draw(); // ヒストグラムを描画する
  return 0;
}
```

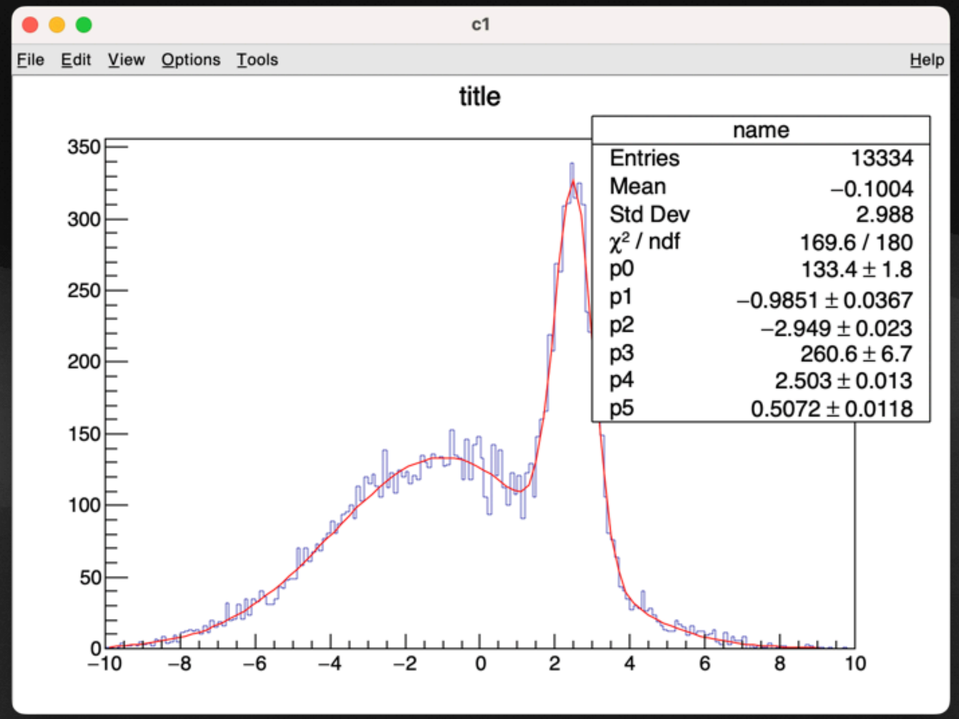


２種類のガウス分布の和にフィットしてみよう

10 分

やってみよう５：ヒストグラムにフィッティング②

```
[genki 13:02:27 root_tutorial] $ root histogram.cc
root [0]
Processing histogram.cc...
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
****************************************
Minimizer is Minuit2 / Migrad
Chi2                      =      169.604
NDf                       =          180
Edm                       =   3.0543e-08
NCalls                    =          604
p0                        =      133.415   +/-   1.78303
p1                        =    -0.985142   +/-   0.0367401
p2                        =     -2.94875   +/-   0.0231048
p3                        =      260.604   +/-   6.74348
p4                        =      2.50332   +/-   0.0127224
p5                        =     0.507167   +/-   0.0117543
(int) 0
root [1]
root [1]
root [1] .q
```



2 種類のガウス分布の和にフィットしてみよう