# ROOT チュートリアル

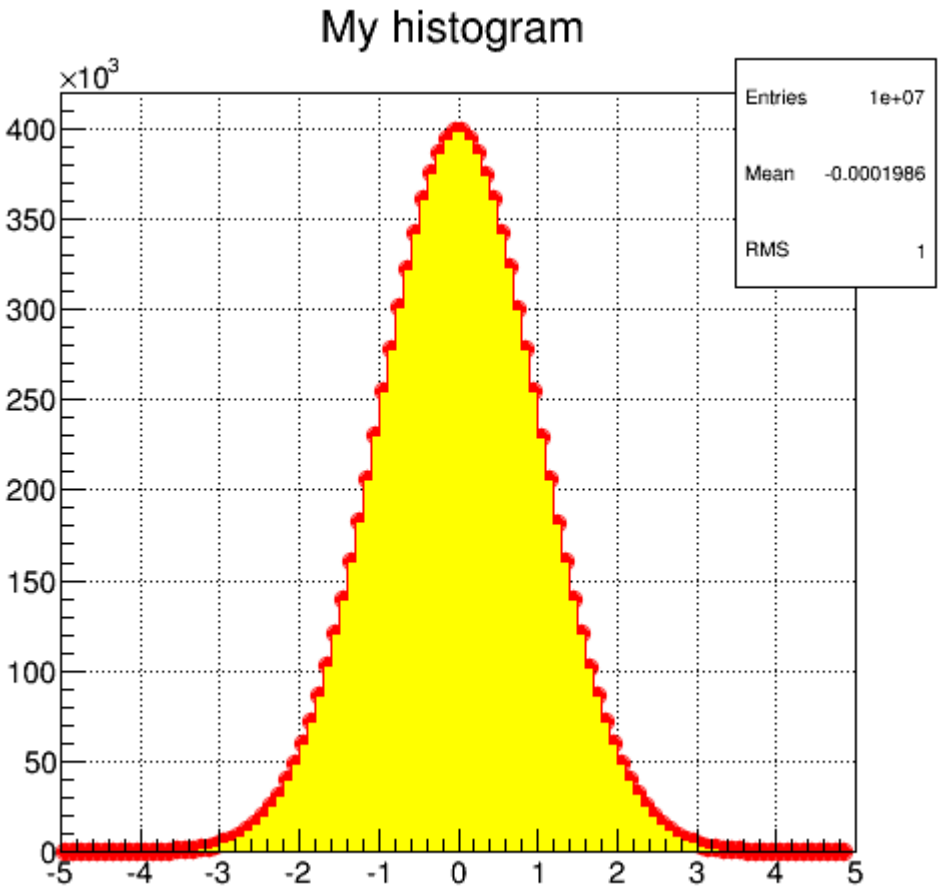
2014年度

河野能知

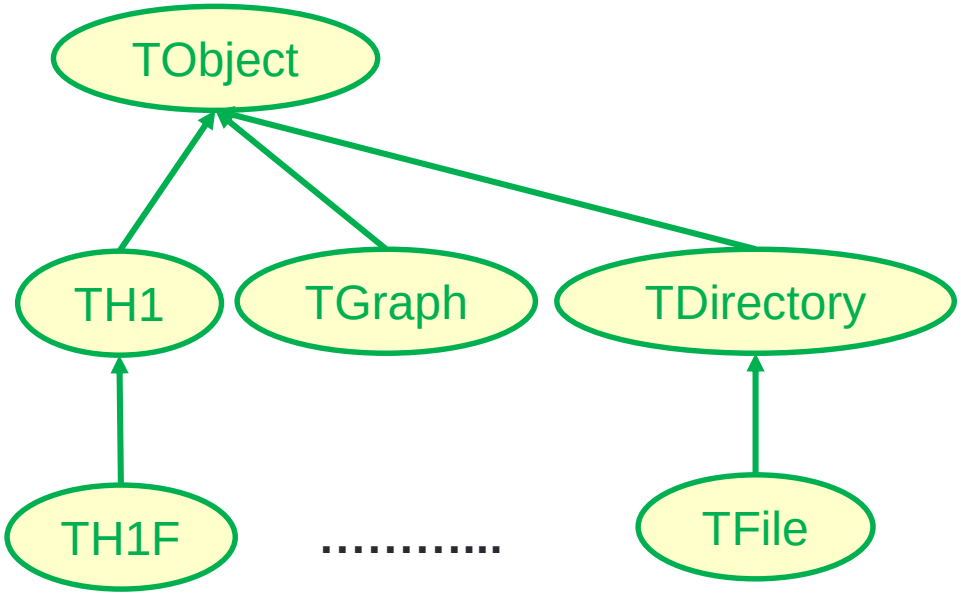
# ROOTとは

- ROOT
  - 素粒子や原子核実験分野のデータ解析の標準的ソフトウェア
  - http://root.cern.ch
- 主な機能
  - ヒストグラム、ツリー、2次元グラフ、フィット
  - 大量データの保存、解析に対応（数TBまではOK）、
  - 用意されていない機能は自分でコードを書くことで自由に拡張可能
  - 統計ツール
  - 3D表示、GUIツール

# Tutorial 1: ヒストグラム



# ROOTオブジェクトについて




```
TH1F* h = new TH1F(”hist1”,
                   “Some title”,
                   10, 0.0, 1.0);
```

- 区間[0,1]を10個のビンに分けたヒストグラム
- h: ヒストグラム型のオブジェクトを指すC++変数名
- “hist1”: ROOT内部で使用するオブジェクトの識別名
- “Some title”: タイトル。なくても良い

- ROOTのクラスは全てTObjectクラスから派生する
  - 全てのクラスに共通の操作はTObjectで定義
  - http://root.cern.ch/root/html534/ClassIndex.html
  - 全てのオブジェクトは識別名（name）を持っている
- ファイル入出力（TFileクラスを使用）

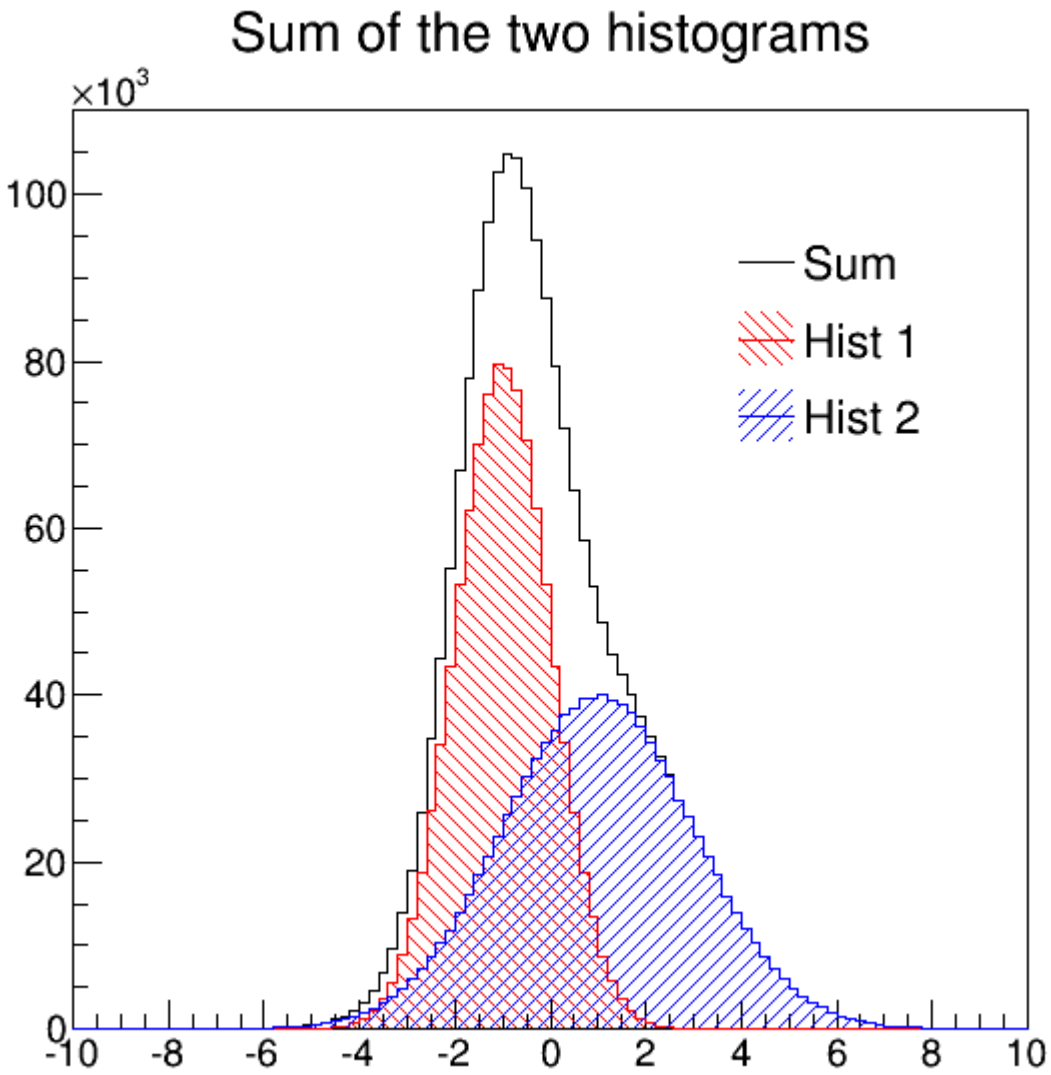
# オブジェクトのファイルへの読み書き

- histExample2.Cでオブジェクトの書き出し
  - TObject::Write()
  - 事前にTFileを開いておく必要あり
- histExample3.Cでファイルからの読み込み
  - TFile::Get(“識別名”)

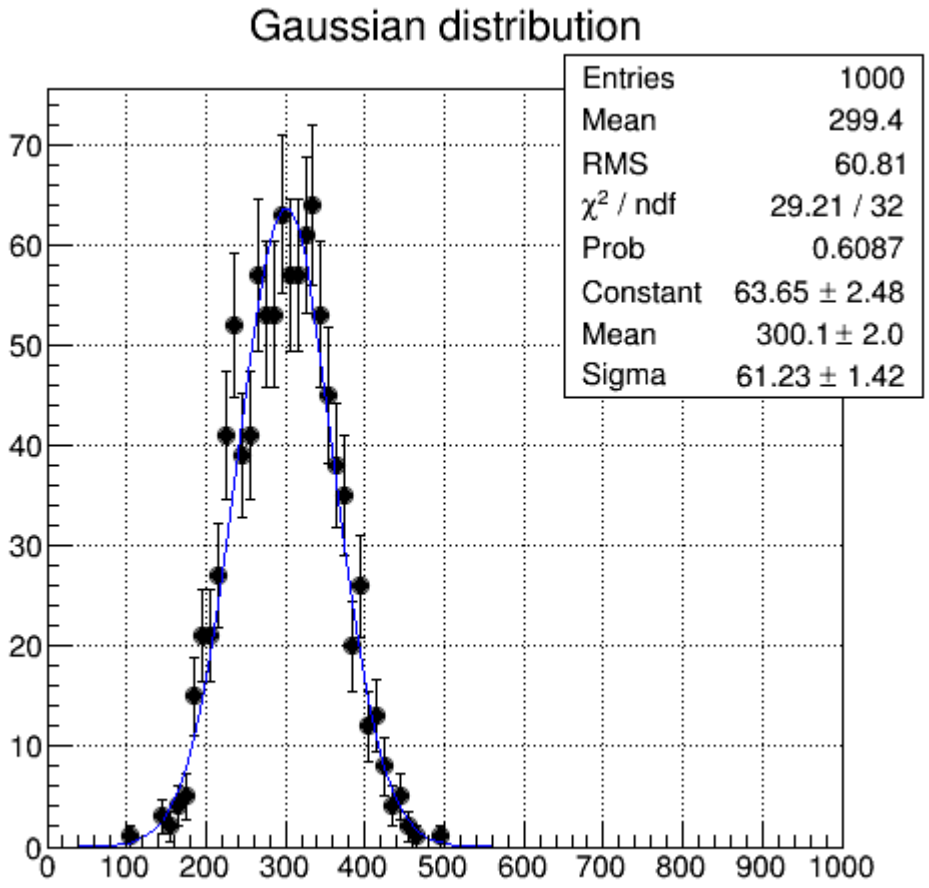
```
TFile* f = TFile::Open(“a.root”, “RECREATE”);
TH1F* h = new TH1F(“hist1”, “”, 10, 0.0, 1.0);
h->Write(); // ヒストグラムをファイルに書き込む
f->Write(); // ファイルをディスクに保存
f->Close(); // ファイルを閉じる。この時、ヒストグラムも解放される
…….

TFile* f = TFile::Open(“a.root”, “READ”); // ファイルを開く
TObject* obj = f->Get(“hist1”);                    // 保存した時と同じ識別名を指定。この
                                                   時オブジェクトの型はまだわからない
TH1F* h2 = dynamic_cast<TH1F*>(obj); // TObject* → TH1F*へ型変換
```

# Tutorial 1： ヒストグラムの操作

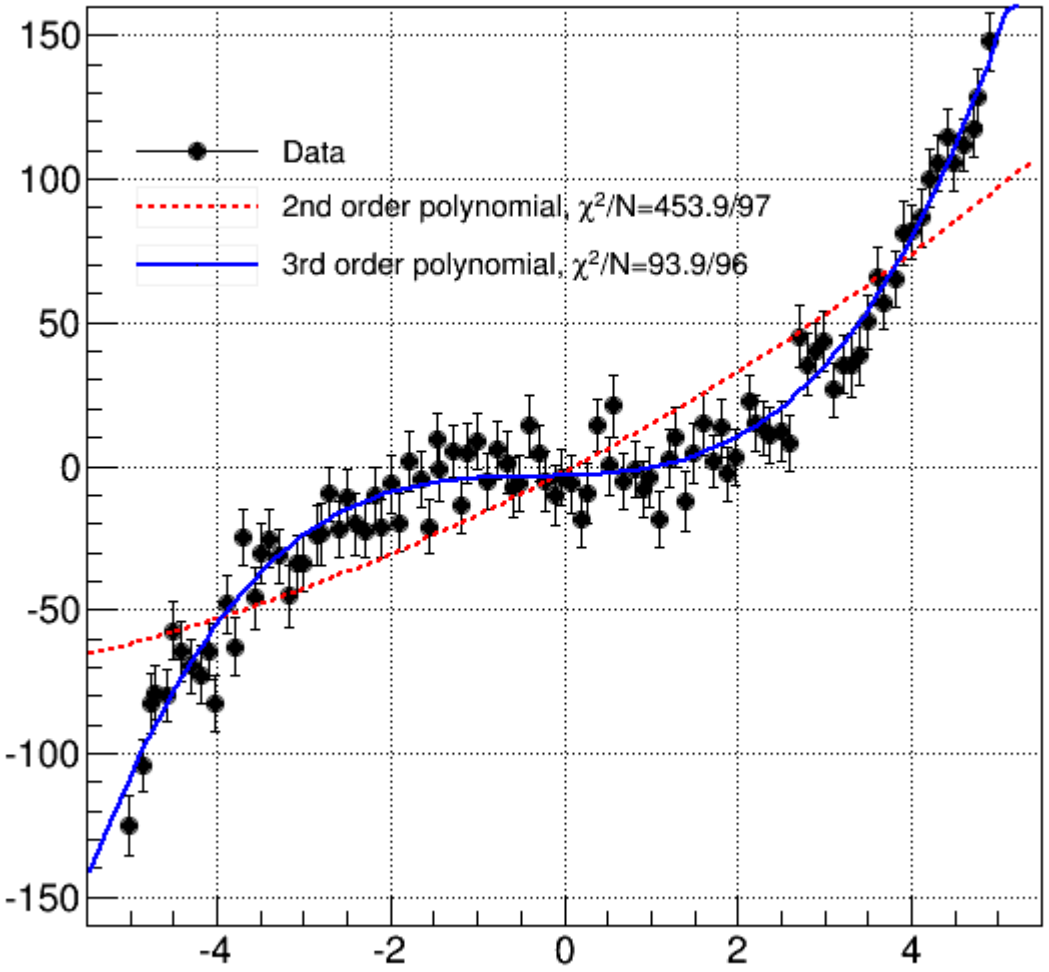


# Tutorial 2： データのフィット




- fitExample1.C
- ヒストグラムをフィットして、結果を表示

# Tutorial 2： データのフィット




- fitExample2.C
- 2次元データをフィット（TGraph）
- 2, 3次関数によるフィット
- $\chi^2/N_{DOF}$

# TTree

- 素粒子実験で収集するデータの特徴として、データが事象単位になっていることが挙げられる
- 量子力学の過程 → (微分)散乱断面積 $\frac{d\sigma}{dEd\Omega}(E,\theta,\varphi)$
  - 散乱実験により、終状態に粒子$(E,\theta,\varphi)$ が生成する過程
  - 理論的には多次元の関数(分布)
- 理論的に計算されるのは分布であるが、実験では多数の事象が観測される
  - 具体的な事象は特定の$(E,\theta,\varphi)$を持つ
  - 多数の事象を集めることで、分布が得られる

このような表で表される →

| | $E$ | $\theta$ | $\varphi$ | ..... |
|---|---|---|---|---|
| 事象1 | | | | |
| 事象2 | | | | |
| 事象3 | | | | |
| ..... | | | | |

これを実装したのが、ROOTのTTreeと呼ばれるクラス

昔はN-tupleと呼ばれていた。
tupleというのは、(a, b, c, …)のように変数の組みのこと

# 素粒子反応データ(シミュレーション)

LHCで行われている実験のシミュレーションを使用する
陽子と陽子を重心系エネルギー14 TeVで衝突させる

$$p + p \rightarrow X$$

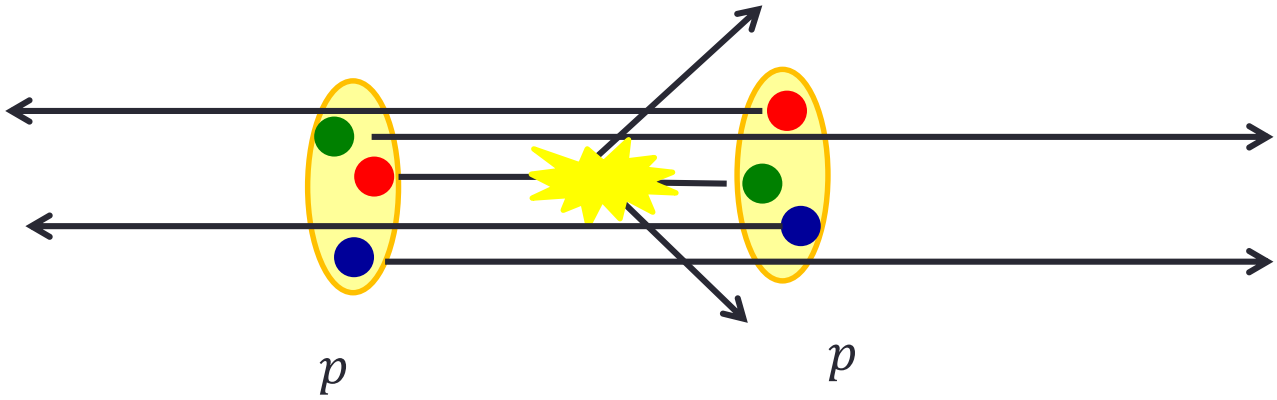
- どのような終状態Xが出てくるかは、場の理論で記述される
- 保存則(エネルギー、運動量、電荷等)が満たされている限り、無数の終状態が存在する。粒子数は保存則しない
- 場の理論で予言できるのは、特定の終状態を生成する断面積

- 実験的には、多数の事象データを取得して、
  - どのような散乱過程($p + p \rightarrow X$)かを判定
  - 目的の終状態$X$が生成した事象のみを集める
  - 分布を作り、理論と比較する

# 陽子-陽子散乱データの解析

$p + p \rightarrow X$

様々な終状態Xが起こり得る



# MCシミュレーション

bクォーク、反bクォーク対生成

$$p + p \rightarrow b\bar{b}$$

シミュレーションでは、さらに
- Parton Shower : QCDの高次の効果($pp \rightarrow b\bar{b} + X$)を再現するため
- Hadronization(ハドロン化) : クォークが複数集まってハドロンを構成する過程

をシミュレートして、実際に観測される終状態を再現している

したがって、最終的には、終状態に数100～数1000個のハドロンが生成される

ファイル :
/nfs/space1/tkohno/work/Pheonomenology/data/tut_bb_01.root ～tut_bb_10.root

# MCシミュレーションにおける粒子の情報

| 物理量または属性 | 変数名 | コメント |
|---|---|---|
| エネルギー | Particle_E[] | |
| 運動量 | Particle_Px, _Py, _Pz[] | |
| 横方向運動量 | Particle_PT[] | |
| 擬ラピディティー($\eta$) | Particle_Eta[] | |
| 方位角($\phi$) | Particle_Phi[] | |
| 粒子の識別番号(PDG ID) | Particle_PID | PDGが管理している番号 |
| 粒子の状態(安定/崩壊済み) | Particle_Status[] | 安定粒子の場合「=1」 |
| 崩壊元の粒子 | Particle_Mother1, 2[] | |

シミュレーションに特有の情報

# TTreeを使ってみる

TTree::Draw<変数名>", "<データ選択条件>", "<描画オプション>")
- 1次元の分布、2次元分布（相関）
- 条件を課してデータを選択
- 結果をヒストグラムやグラフに保存

TTree ～ 各行が1つの事象に対応する大きな表
- 全ての事象から、特定の変数を取り出すことを効率的に行える
- 様々な分布を簡単に作ることができる

TTree* t;

t->Draw("Particle_PT");
粒子の$p_T$を全ての粒子に対してプロットする

t->Draw("Particle_PT", "Particle_PID==13 && fabs(Particle_Eta)<3");
$|\eta| < 3$に存在する$\mu^-$粒子の$p_T$をプロットする

# もっと複雑な解析

複数の粒子の情報から、別の量を計算する
- 例えば、粒子の崩壊過程を再構成する
- TTree::MakeClass(“MCEvent”)とすると、このような解析をするためのプログラムの枠組みを自動的に生成してくれる

$X \to \mu^+\mu^-$とミューオン対に崩壊する親粒子を探す
- ミューオン対の不変質量を計算して、その分布を作る
- 親粒子の質量に対応するところに、ピークが観測される（共鳴状態）

RootTutorial4
- pl_particles.C
- pl_muons.C

# 見た目を良くする

TCanvas: 描画するためのウィンドウ
TPad: 実際に描画する領域（TCanvas自体もPad）
ウィンドウ内に複数の絵を表示させたい時は、canvas->Divide(n, m)。
その後、canvas->cd(1)等で分割された各Padへ移動して描画できる

**描画オプション**
- 点、線、塗りつぶしの色、スタイル等を指定
- 詳しくはTAttMarker, TAttLine, TAttFill参照

**全体のスタイル（gStyleを使用）**
- 描画領域の位置、マージン、軸ラベルの大きさ等

**Padに対して**
- 軸の対数・線形表示、グリッド線、マージン等

# Backup slides

# Tutorial 3： TTreeを作ってみる

- 通常は、1事象分のデータを1行に保存するが、まずは以前やったボールの運動のデータを扱う
- 時間ステップ毎にstd::vector<Ball>型のデータを各行に保存する
  - 実際には少しデータを追加

| | t | N | Ball.X[N] | Ball.Y[N] | Ball.Vx[N] | Ball.Vy[N] |
|---|---|---|---|---|---|---|
| 時刻1 | | | | | | |
| 時刻2 | | | | | | |
| 事象3 | | | | | | |
| ….. | | | | | | |

**手順**

- C++で使っていたデータを用意
- メモリ上のアドレスをTTreeに登録する。TTree::Branch()
- TTree::Fill()を呼ぶ度に、メモリ上のデータがTTreeに追加されていく。行が増える
- 複雑なデータ構造を持つデータも保存可能
- 詳しくは、http://root.cern.ch/root/html534/TTree.html参照