

# ROOT講習会第5回 TTree / データ構造

東大宇宙線研 M2 杉本布達

2025/06/25

# 自己紹介

名前：杉本 布達（Discord上では fsugim です）

所属：東大宇宙線研 Tibet AS $\gamma$  / ALPACA グループ

研究内容：太陽活動と宇宙線量の関連など

ボリビアの実験サイトにて



# おさらい

これまで、以下のようなことを学んだと思います

- マクロファイルの実行
  - プロット (TGraph) / ヒストグラム (TH1D etc.) を作る
  - 関数 (TF1)でのフィッティング
- … なにかがたりない？ データを保存したい！

# データ保存について

例えばこんなときに処理後のデータを保存したい：

- ・元データが動画などで容量を食いすぎる（データ圧縮）
- ・特定の処理をしたあと（フィット後の結果など）

というモチベーションがあります。

テキストでもいいですが、テキストよりも早い方法があります。

# TTree

テキスト読み込みより早いのが、ROOTのTTreeです。  
ざっくり使用方法を見てみましょう。

- TTreeでは、データは表のように格納されています。
  - それぞれの列では、データの型が決まっています。
  - 実際のプログラム中では、intならI, doubleならDなどと表記します。

Event number (int)	天頂角 (double)	方位角 (double)	粒子数 (int)	ミューオン数 (int)
0	1.025...	1.786...	13	0
1	0.034...	0.872...	45	6
2	0.591...	5.839...	95	15
3	0.130...	0.435...	27	2
...	...	...	...	...

# TTree

ということで、手を動かしていきましょう！  
まず、rootファイルをダウンロードします。

```
[fsugim@icrhome04 root_lecture]$ root -l sim_shower_events.root
root [0]
Attaching file sim_shower_events.root as _file0...
(TFile *) 0x2794100
root [1] █
```

上のようなコマンドでデータを読んでみます。

表示される (TFile \*) … が 0x0 でなければOKです。

TTree 次に、データ一覧を確認してみましょう。

.ls で表示される”shower”  
が今回のTTreeの名前です

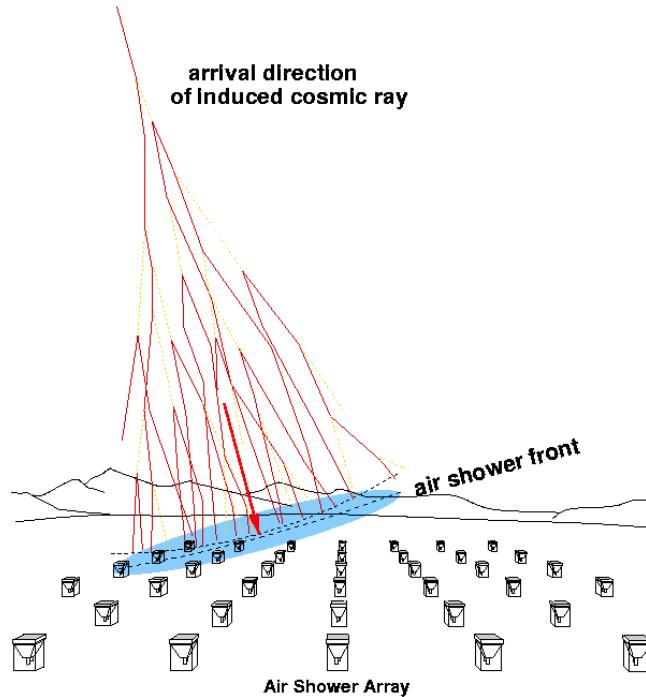
shower->Print()で、中に  
入っているデータを確認  
することができます。

```
[fsugim@icrheme04 root_lecture]$ root -l sim_shower_events.root
root [0]
Attaching file sim_shower_events.root as _file0...
(TFile *) 0x304e4a0
root [1] .ls
TFile**      sim_shower_events.root
TFile*       sim_shower_events.root
  KEY: TTree   shower;1      Simulated Shower Events
root [2] shower->Print();
*****
*Tree   :shower   : Simulated Shower Events
*Entries : 1000000 : Total =      16050632 bytes  File  Size =    9423852 *
*   :          : Tree compression factor =   1.70
*****
*Br   0 :zenith   : zenith/F
*Entries : 1000000 : Total  Size=    4012268 bytes  File Size =   3584961 *
*Baskets :     126 : Basket Size=    32000 bytes  Compression=   1.12 *
*.....
*Br   1 :azimuth   : azimuth/F
*Entries : 1000000 : Total  Size=    4012398 bytes  File Size =   3584852 *
*Baskets :     126 : Basket Size=    32000 bytes  Compression=   1.12 *
*.....
*Br   2 :num_particle : num_particle/I
*Entries : 1000000 : Total  Size=    4013048 bytes  File Size =   1324527 *
*Baskets :     126 : Basket Size=    32000 bytes  Compression=   3.03 *
*.....
*Br   3 :num_muon   : num_muon/I
*Entries : 1000000 : Total  Size=    4012528 bytes  File Size =   924345 *
*Baskets :     126 : Basket Size=    32000 bytes  Compression=   4.34 *
*.....
root [3] ■
```

# ちょっとだけ研究グループの紹介

データの内容について説明するために、ちょっと脇道へそれます

Tibet AS $\gamma$  / ALPACA グループは空気シャワーを観測しています



空気シャワーは、大気中に入射した粒子が空気中の粒子と反応することで、二次粒子の力スケードを作る現象です。

二次粒子を地上で観測することで、元の粒子の到来方向がわかる、という仕組みです。

# 演習 - データの確認

ということで、rootファイル中のデータの列はそれぞれこんな感じになっているはずです

列名	データ型	意味
Zenith	Double	空気シャワーの天頂角
Azimuth	Double	空気シャワーの方位角
Num_particle	Int	観測された粒子数
Num_muon	Int	観測されたミューオン数

今回は(自分が書いた)シミュレーションなので、天頂角・方位角を一様確率で振っています。まずその確認をしましょう。

# 演習 – TTreeのデータの読み方

TTreeの各列のことをBranchと呼びます。枝だからですね。

TTree名->Draw("Branch名")で、その列のデータのヒストグラムを描くことができます。

例えば、方位角 ("zenith") のヒストグラムを描いてみます。

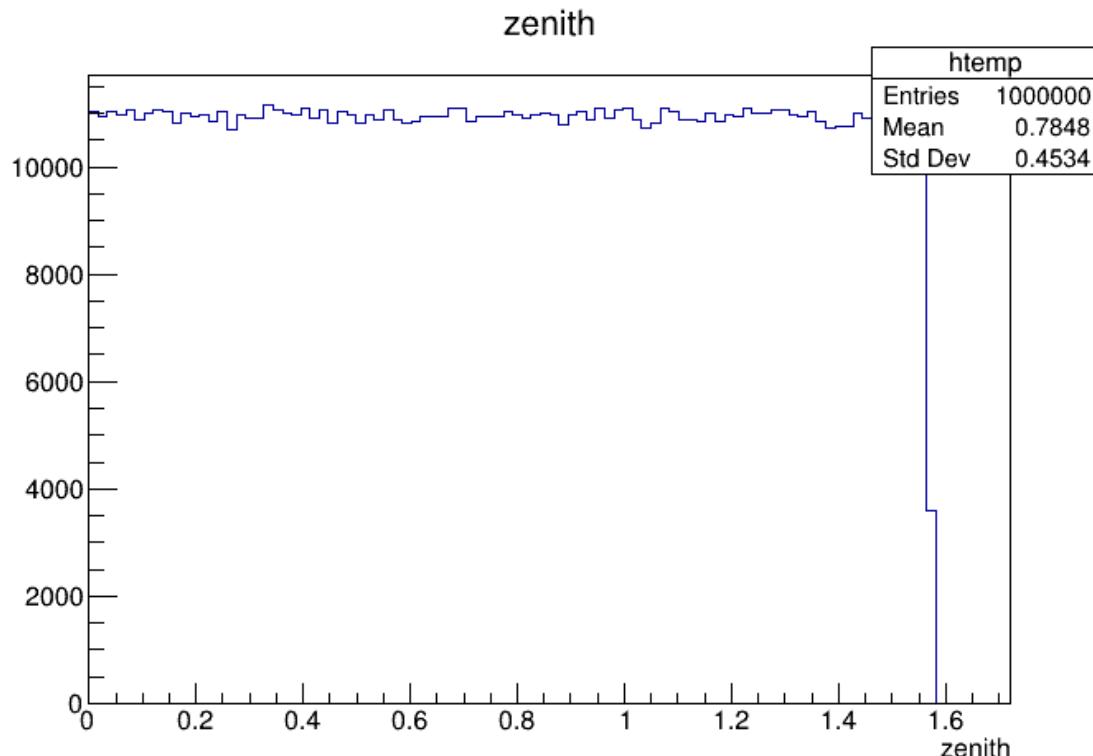
```
root [1] shower->Draw("zenith");
Info in <TCanvas::MakeDefCanvas>: created default TCanvas with name c1
```

このようなコマンドを打つと、画面が現れると思います。

# 演習 – TTreeのデータの読み方

例えば、天頂角 (“zenith”) のヒストグラムを描いてみます。

```
root [1] shower->Draw("zenith");
Info in <TCanvas::MakeDefCanvas>: created default TCanvas with name c1
```



このような形になります。

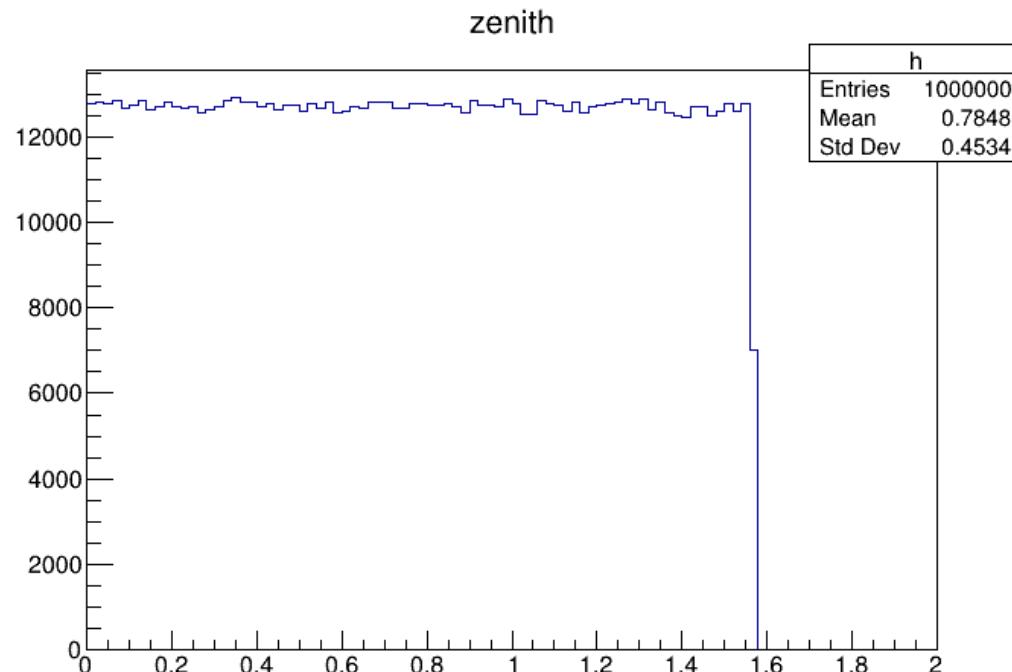
横軸はradなので、 $0 - \pi / 2$ まで等方に振れていそうです。

ただ、Legendにデータが隠れてしまっています。

# 演習 – TTreeのデータの読み方

Branch名の後に>>をつけて、ビン数、最小値、最大値を指定できます。ビン数を100, 最小値を0, 最大値を2にしてみましょう。

```
root [9] shower->Draw("zenith>>h(100,0,2)");
Info in <TCanvas::MakeDefCanvas>: created default TCanvas with name c1
```



このような形になります。

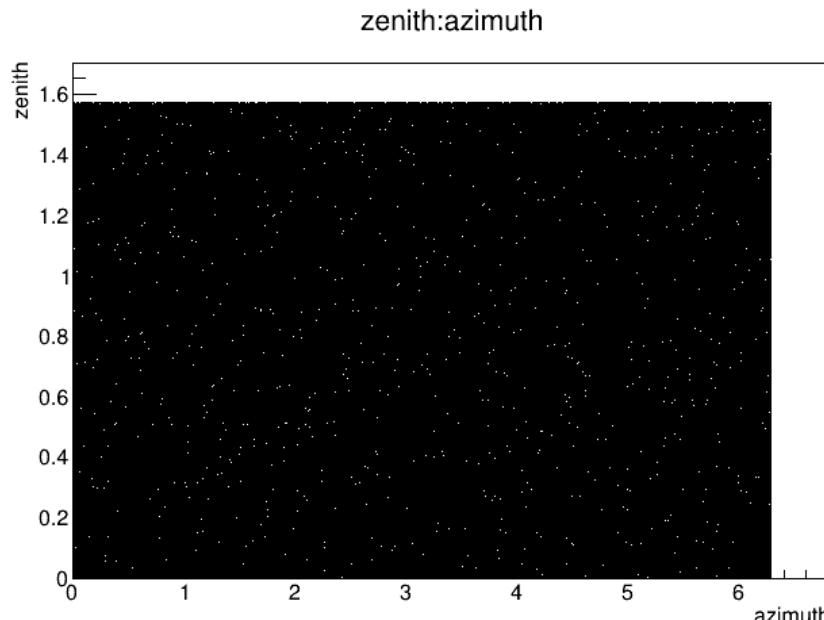
今度は、ほぼ同じくらいデータが入っているのが見えました。

# 演習 - TTreeのデータの読み方

同様に、2次元のヒストグラムも描いてみましょう。

今度は、`shower->Draw("zenith:azimuth")`のようにbranchをコロンを挟んで二つ指定します。

```
root [5] shower->Draw("zenith:azimuth");
Info in <TCanvas::MakeDefCanvas>: created default TCanvas with name c1
```



このような形になります。

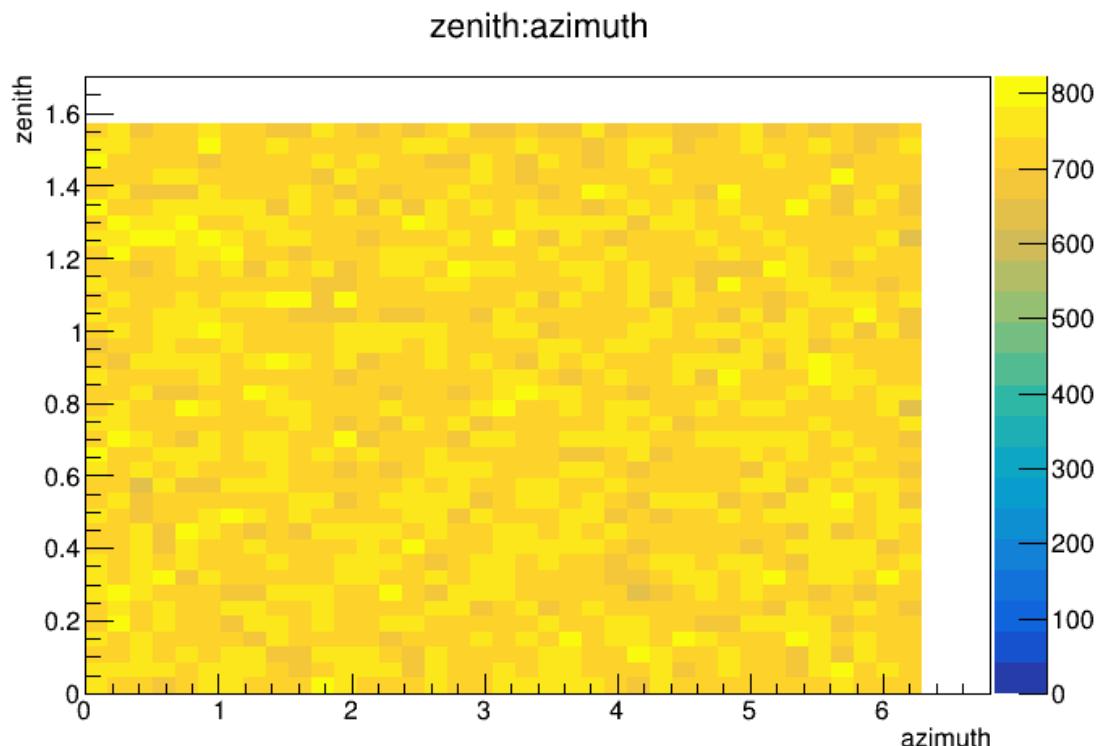
真っ黒ですね…！  
もう少し見やすくしてみます。

# 演習 - TTreeのデータの読み方

Drawという関数の第三引数で、描画形式を指定しましょう。

shower->Draw("zenith:azimuth", "", "COLZ")と書き換えます。

```
root [6] shower->Draw("zenith:azimuth", "", "COLZ");
Info in <TCanvas::MakeDefCanvas>: created default TCanvas with name c1
```



すると、今度はбинで区切られたデータが出力されました。

天頂角、方位角ともにほぼ一様にデータが振られていそうです。

# 演習 – TTreeのデータの読み方

さて、天頂角と方位角はほぼ一様に振れていることが分かったと思います。

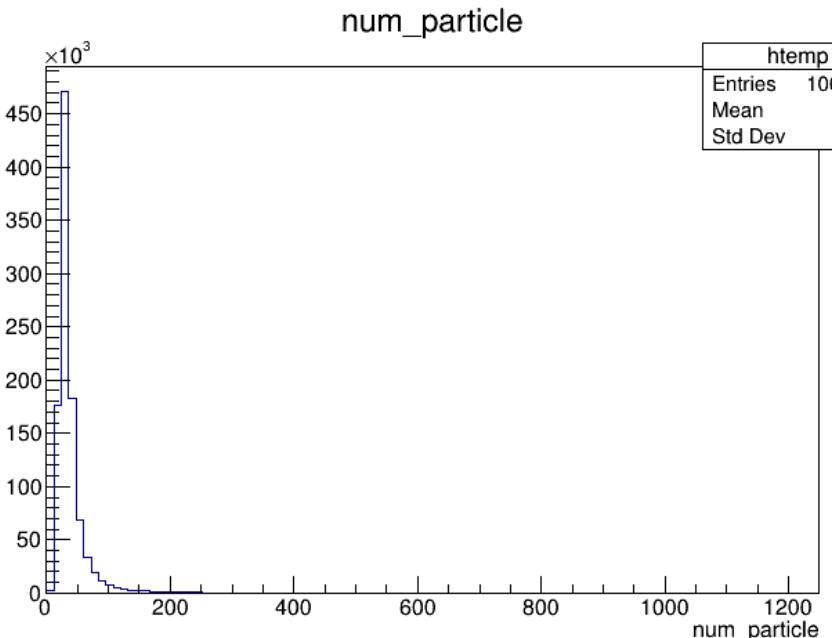
そこで、粒子数・ミューオン数の一次元分布を見てみましょう。

粒子数のBranch名は”num\_particle”、ミューオン数のBranch名は”num\_muon”でした。先ほどの例を参考にしつつ、実際に手を動かしてみてください。

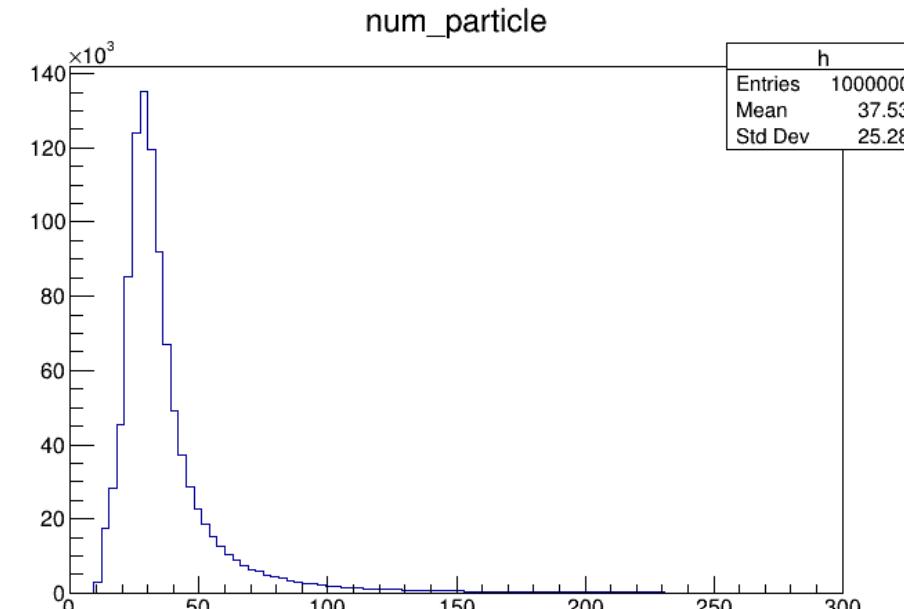
# 演習 – TTreeのデータの読み方

おさらいですが、TTree名->Draw("Branch名")でヒストグラムを描画できるのでした。

```
root [15] shower->Draw("num_particle");
Info in <TCanvas::MakeDefCanvas>: created default TCanvas with name c1
```



単純に描いた

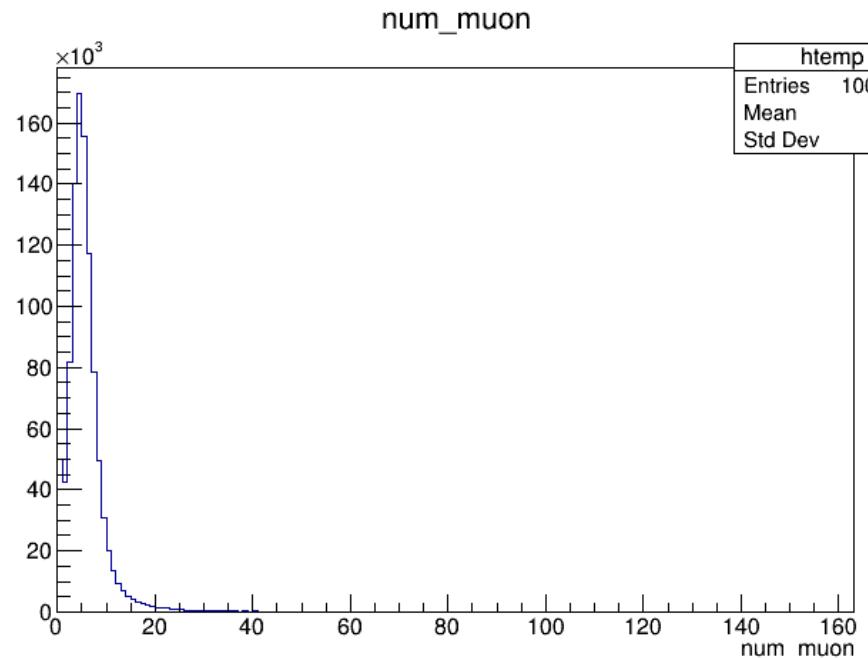


描画範囲を狭めた

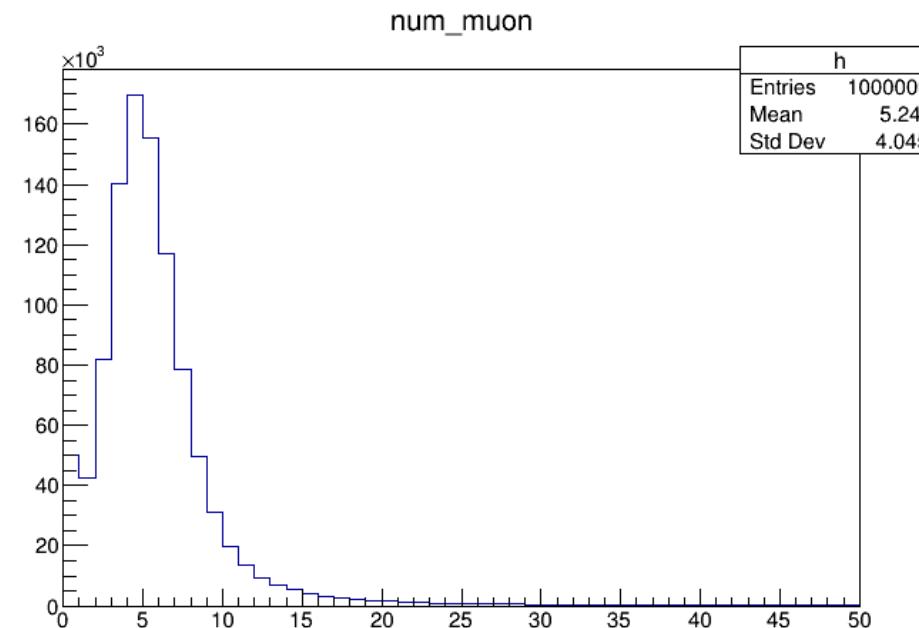
# 演習 - TTreeのデータの読み方

おさらいですが、TTree名->Draw("Branch名")でヒストグラムを描画できるのでした。

```
root [21] shower->Draw("num_muon");
Info in <TCanvas::MakeDefCanvas>: created default TCanvas with name c1
```



単純に描いた



描画範囲を狭めた

# 演習 – TTreeのデータの読み方

粒子数・ミューオン数の一次元分布を見てみましたが、何かの特徴に気づきましたか？

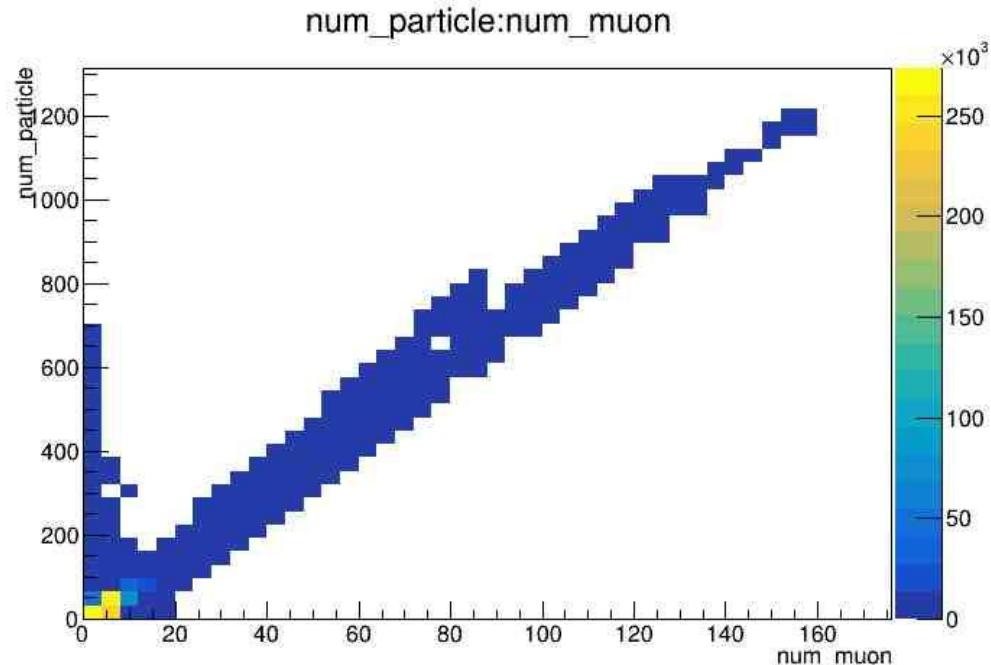
なかなか難しいそうですね。。。二次元分布にして特徴を見てみましょう。

二次元分布を描くコマンドは、`Draw("branch1:branch2", "", "COLZ")`という形式でした。

# 演習 – TTreeのデータの読み方

二次元分布を描くコマンドは、Draw("branch1:branch2", "", "COLZ")という形式なでした。

```
root [1] shower->Draw("num_particle:num_muon", "", "COLZ");
Info in <TCanvas::MakeDefCanvas>: created default TCanvas with name c1
```



なんと、データに明確な傾向が見えました！

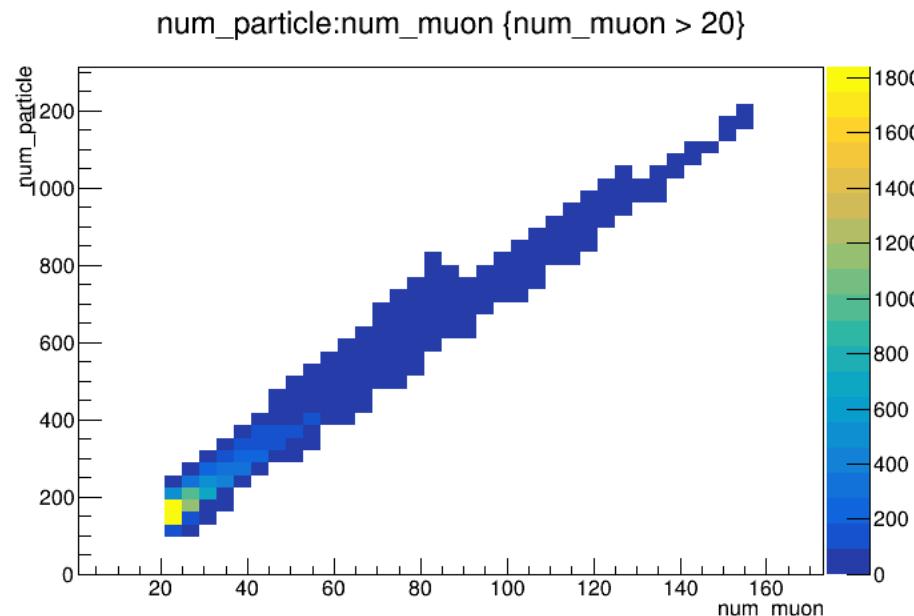
粒子数とミューオン数が比例して見えるデータと、そうでないものが混ざっているように見えます。

# 演習 – TTreeのデータの読み方

とりあえずカットをかけてみましょう。

Drawの第二引数にカット条件を書くことができます。今回は、  
 $\text{num\_muon} > 20$ をかけてみます。

```
root [3] shower->Draw("num_particle:num_muon", "num_muon > 20", "COLZ");
Info in <TCanvas::MakeDefCanvas>: created default TCanvas with name c1
```



ちゃんとミューオン数が20以下のデータはカットされているように見えます。

また、この範囲ではミューオン数と粒子数が比例しているように見えます。

# 演習 – TTreeのデータの読み方

ということで、4つのBranchのデータを取り扱ってきました。  
さらに、同じことをマクロでもやってみましょう。

```
1 void read_root(TString filename="data/sim_shower_events.root") {  
2  
3     auto file = TFile::Open(filename);  
4     auto tree = dynamic_cast<TTree*>(file->Get("shower"));  
5  
6     double zenith, azimuth;  
7     int num_particle, num_muon;  
8  
9     tree->SetBranchAddress("zenith", &zenith);  
10    tree->SetBranchAddress("azimuth", &azimuth);  
11    tree->SetBranchAddress("num_particle", &num_particle);  
12    tree->SetBranchAddress("num_muon", &num_muon);
```

# 演習 – TTree のデータの読み方

```
14 long long entries = tree->GetEntries();
15
16 const int nbin = 100;
17 const int val_max = 200;
18 const int val_min = 200;
19
20 auto hist = new TH1D("hist", "hist", nbin, val_min, val_max);
21 hist->SetTitle(";;");
22
23 for (int i = 0; i < entries; i++) {
24     tree->GetEntry(i);
25     hist->Fill(num_particle);
26 }
27 hist->Draw();
28
29 }
```

# C++の名前空間、型推論(発展)

TFile::Open は Tfile

dynamic\_cast<TTree\*>

auto は自動で型を推論してくれる

詳しくはcppreference.jpなどを参照してください。

# 演習 – TTreeのデータの読み方

では、もう一つのrootファイルと合わせて解析をしてみましょう。

rootファイルのTTreeからTH1Dにデータを入力する部分を関数として切り出します。

```

1 TH1D * read_data(TString filename="data/sim_shower_events.root") {
2
3     auto file = TFile::Open(filename);
4     auto tree = dynamic_cast<TTree*>(file->Get("shower"));
5
6     float zenith, azimuth;
7     int num_particle, num_muon;
8
9     tree->SetBranchAddress("zenith", &zenith);
10    tree->SetBranchAddress("azimuth", &azimuth);
11    tree->SetBranchAddress("num_particle", &num_particle);
12    tree->SetBranchAddress("num_muon", &num_muon);
13
14    long long entries = tree->GetEntries();
15
16    const int nbin = 100;
17    const int val_min = 0;
18    const int val_max = 200;
19
20    auto hist = new TH1D("hist", "hist", nbin, val_min, val_max);
21    hist->SetTitle(";;");
22
23    for (int i = 0; i < entries; i++) {
24        tree->GetEntry(i);
25        hist->Fill(num_particle, num_muon);
26    }
27
28    return hist;
29
30 }
```

# 演習 – TTreeのデータの読み方

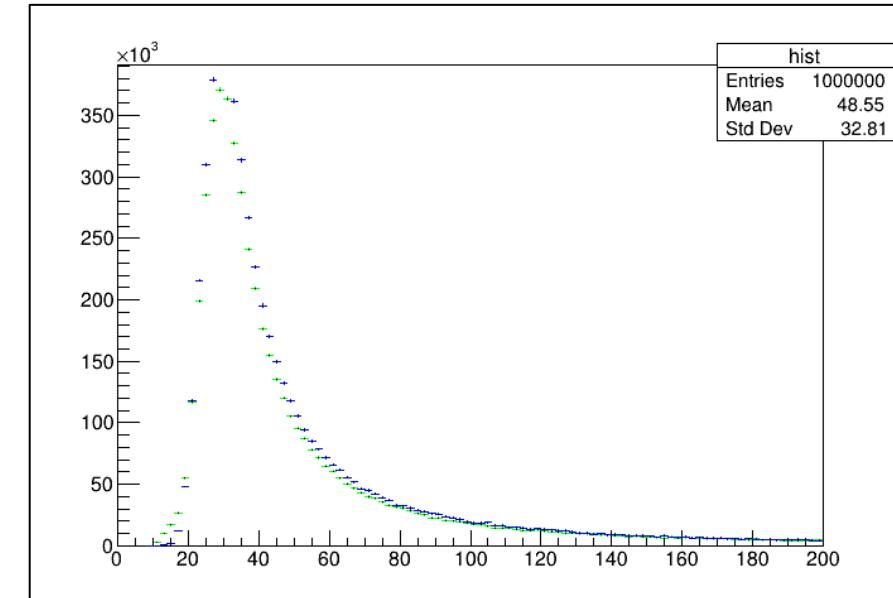
この関数を使って、複数の root ファイルからデータを取得します。

その後、TH1Dをそれぞれ Draw します。

右下の図のようなヒストグラムが描かれるはずです。

```

32 void read_root_mult() {
33
34     auto hist1 = read_data("data/sim_shower_events.root");
35     auto hist2 = read_data("data/sim_shower_events_gamma.root");
36
37     hist1->SetLineColor(kGreen);
38     hist1->Draw();
39
40     hist2->SetLineColor(kBlue);
41     hist2->Draw("SAME");
42 }
```



# 演習 - TTreeの作り方

rootファイルにデータを保存するには、新しいTTreeを作成し、tree->Fill()でデータを埋め込んでいきます。

右の例は、CSVファイルからデータを読み込むサンプルです。

データが膨大な場合、TChainでツリーを複数ファイルに分けて扱うこともできます。

```

1 void csv_to_root()
2 {
3     // 入力ファイル名
4     const char* infile = "sim_shower_events_gamma.csv";
5     const char* outfile = "sim_shower_events_gamma.root";
6
7     // 出力ROOTファイル
8     TFile* f = new TFile(outfile, "RECREATE");
9     TTree* tree = new TTree("shower", "Simulated Shower Events");
10
11    // 変数定義
12    float energy, zenith, azimuth;
13    int num_particle, num_muon, label, composition;
14
15    tree->Branch("zenith", &zenith, "zenith/F");
16    tree->Branch("azimuth", &azimuth, "azimuth/F");
17    tree->Branch("num_particle", &num_particle, "num_particle/I");
18    tree->Branch("num_muon", &num_muon, "num_muon/I");
19
20    // ファイルを開く
21    std::ifstream fin(infile);
22    #-- 4 lines: if (!fin.is_open()) { ..... }
23
24
25    std::string line;
26    // 1行目はヘッダーなので読み飛ばし
27    std::getline(fin, line);
28
29    while (std::getline(fin, line)) {
30        std::istringstream ss(line);
31        std::string val;
32        // カンマ区切りで各カラムをパース
33        std::getline(ss, val, ','); energy      = std::stof(val);
34        std::getline(ss, val, ','); zenith      = std::stof(val);
35        std::getline(ss, val, ','); azimuth     = std::stof(val);
36        std::getline(ss, val, ','); num_particle = std::stoi(val);
37        std::getline(ss, val, ','); num_muon    = std::stoi(val);
38        std::getline(ss, val, ','); label       = std::stoi(val);
39        std::getline(ss, val, ','); composition  = std::stoi(val);
40
41        tree->Fill();
42    }
43
44    tree->Write();
45    f->Close();
46    fin.close();
47
48 }
49

```

# 演習 - TTreeの作り方

rootファイルにデータを保存するには、新しいTTreeを作成し、tree->Fill()でデータを埋め込んでいきます。

rootファイルから読み込んだデータを選別し、新しいrootファイルを作ることもできます。

たとえば、ここではnum\_muonの制限をしています。

```

1 void data_to_root()
2 {
3     // 入力ファイル名
4     const char* infile = "data/sim_shower_events.root";
5     const char* outfile = "sim_shower_events_gamma.root";
6
7     // 入力ROOTファイル
8     TFile* file_in = TFile::Open(infile);
9     auto in_tree = dynamic_cast<TTree*>(file_in->Get("shower"));
10
11    // 出力ROOTファイル
12    TFile* file_out = new TFile(outfile, "RECREATE");
13    TTree* tree = new TTree("shower", "Simulated Shower Events");
14
15    // 変数定義
16    float zenith, azimuth;
17    int num_particle, num_muon;
18
19    in_tree->SetBranchAddress("zenith", &zenith);
20    in_tree->SetBranchAddress("azimuth", &azimuth);
21    in_tree->SetBranchAddress("num_particle", &num_particle);
22    in_tree->SetBranchAddress("num_muon", &num_muon);
23
24    tree->Branch("zenith", &zenith, "zenith/F");
25    tree->Branch("azimuth", &azimuth, "azimuth/F");
26    tree->Branch("num_particle", &num_particle, "num_particle/I");
27    tree->Branch("num_muon", &num_muon, "num_muon/I");
28
29
30    // ファイルを開く
31    long long entries = in_tree->GetEntries();
32
33    for (int i = 0; i < entries; i++) {
34        in_tree->GetEntry(i);
35        if (num_muon < 20) {
36            tree->Fill();
37        }
38    }
39
40    tree->Write();
41    file_out->Close();
42 }
```

# レクチャー部分終わり

- お疲れさまでした！

# 演習問題

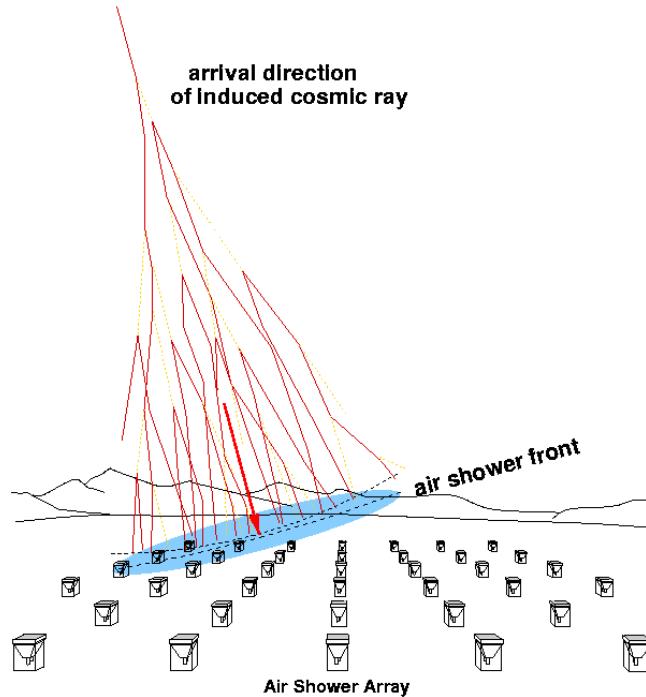
- TH1Dでのデータ読み込みをTH2Dに拡張してみましょう。  
変数の定義が変わるので、binの最小値・最大値をx, y軸分用意する必要があります。
- 2つのファイルのデータを1つのTH2Dに読み込みましょう。  
関数の引数を一つ増やしてみましょう。
- TH2Dのx軸をnum\_muon, y軸をnum\_particleに設定してデータを確認してみましょう。

# 演習問題

- ・粒子数とミューオン数が比例するデータと、しないデータをカット条件を設けて区別してみましょう。
- ・それぞれを別のrootファイルに書き出しましょう。
- ・それぞれ、天頂角分布と方位角分布をプロットしてみましょう。

# おまけ

Tibet AS $\gamma$  / ALPACA グループは空気シャワーを観測しています  
空気シャワーには、主に電磁成分が入射するものと、ハドロン成  
分が入射するものがあります。



ハドロン（陽子）などが入射するとミューオン  
を多く作り、電磁成分だとミューオンはほぼ作  
られないという性質があります。

今回のシミュレーションはこの違いを念頭にお  
いて作っていたので、ミューオン数でこれらを  
区別することができた、ということでした。