

ROOT講習会第5回資料

Tree

Keita Mizukoshi (JAXA ISAS)

- 本資料の著作権、文責は著者に帰属し、所属機関を代表したり、機関の意見を表明するものではありません。
- 学校、研究機関の教育、研究目的であれば自由に使用することができます。報告なく改変、再配布可能です。
ただし、明記されている引用先の著作権に配慮してください。
- ただし使用者は誤りや誤字を報告する義務があります。



これまでのROOT講習会

- ROOTの起動, マクロの作成, 実行
- ヒストグラム (TH1D, TH2D)
- グラフ (TGraph)
- 関数 (TF1)
- フィッティング
- → ROOTの世界にデータを突っ込んで色々やる方法はわかった.
- では, どうやってデータを入れる?
- → `while(ifs >> buffer){...}` という手もあるが非効率
- TTreeを使えば簡単に大規模なデータを扱える

Treeとは

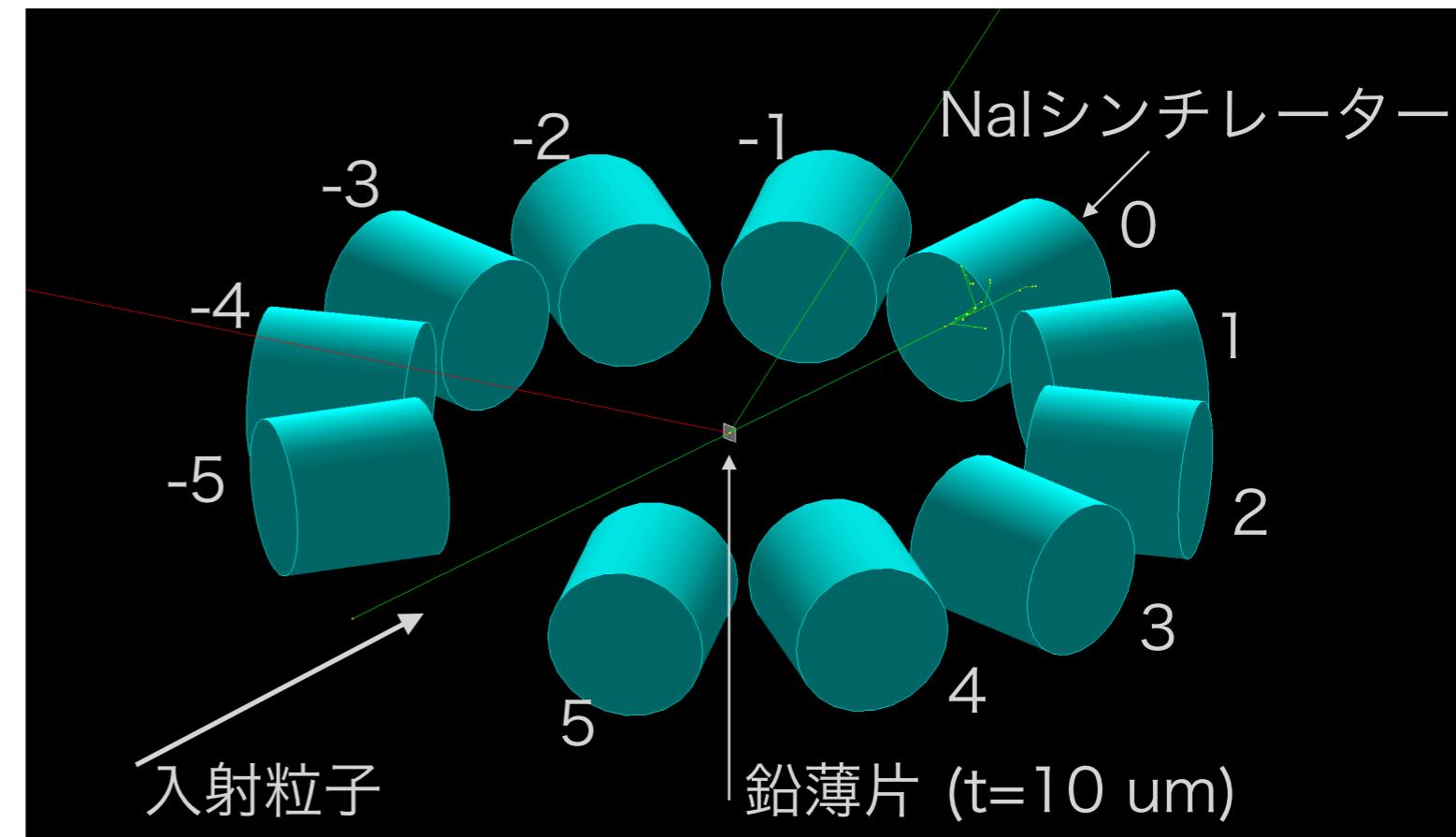
- Tree : ROOTで扱いやすい, データを保存する形式.
- まずはスプレッドシートをイメージする
 - 1行が1イベントに対応 ※ここではトリガーがかかった時のデータの集合

| イベント番号 | nHit (int型) | totalEnergyMeV (double型) | cut1 (bool型) | ... |
|--------|-------------|--------------------------|--------------|-----|
| 0 | 6 | 10.3 | FALSE | |
| 1 | 4 | 2.1 | TRUE | |
| 2 | 8 | 14.3 | FALSE | |
| 3 | 2 | 5.6 | TRUE | |

- 膨大なデータを扱う (ほぼ唯一の) データ形式
 - Excelだと最大1,048,576 行, ROOTは多分限界がない
- 事実上の素粒子実験業界の標準的データ形式
 - この形式でデータを共有することがほとんど
- 組み込み型だけでなくクラスを入れることもできる

演習データ

- 鉛薄片の周囲に距離50cm (中心--中心) 離しNaIシンチレーター (h10cm, ϕ 10cmの円筒形)を, 30度ごとに11個配置
- 図のように番号を振って検出器を区別する
- 各ファイル10,000発粒子入射



- このシミュレーションデータを取得データだと思って解析してみる.
- データファイルをダウンロードする.

```
> ls data
electron_1_0MeV_10000_0.root  gamma_1_5MeV_10000_1.root  gamma_1_5MeV_10000_4.root  proton_1_0MeV_10000_0.root
gamma_1_0MeV_10000_0.root    gamma_1_5MeV_10000_2.root  gamma_2_0MeV_10000_0.root
gamma_1_5MeV_10000_0.root    gamma_1_5MeV_10000_3.root  muon_1_0MeV_10000_0.root
```

<入射粒子>_<エネルギー>_<イベント数>_<ラン番号>.root

とりあえず手を動かしてみる

```
> root -l gamma_1_5MeV_10000_0.root
root [0]
Attaching file gamma_1_5MeV_10000_0.root as _file0...
(TFile *) 0x10564adb0
root [1] .ls
TFile**      gamma_1_5MeV_10000_0.root
TFile*       gamma_1_5MeV_10000_0.root
  KEY: TTree tree;1  tree
root [2] tree->Print()
*****
*Tree   :tree      : tree
*Entries : 10000 : Total =          929607 bytes  File  Size =    152764
*           : Tree compression factor =   6.11
*****
*Br   0 :nHit      : nHit/I
*Entries : 10000 : Total  Size=     40623 bytes  File Size =     1066 *
*Baskets :      2 : Basket  Size=    32000 bytes  Compression= 37.66   *
*.....
*Br   1 :total_adc : total_adc/D
*Entries : 10000 : Total  Size=     80737 bytes  File Size =    28367 *
*Baskets :      3 : Basket  Size=    32000 bytes  Compression=  2.83   *
*.....
*Br   2 :tdc       : vector<double>
*Entries : 10000 : Total  Size=   222207 bytes  File Size =   21465 *
*Baskets :      9 : Basket  Size=    32000 bytes  Compression= 10.33   *
*.....
*Br   3 :adc       : vector<double>
*Entries : 10000 : Total  Size=   222207 bytes  File Size =   59964 *
*Baskets :      9 : Basket  Size=    32000 bytes  Compression=  3.70   *
*.....
*Br   4 :detector_id : vector<int>
*Entries : 10000 : Total  Size=   181627 bytes  File Size =   20666 *
*Baskets :      7 : Basket  Size=    32000 bytes  Compression=  8.76   *
*.....
*Br   5 :particle_id : vector<int>
*Entries : 10000 : Total  Size=   181627 bytes  File Size =   20057 *
*Baskets :      7 : Basket  Size=    32000 bytes  Compression=  9.03   *
*.....
```

ROOTの引数に.rootファイルを渡すと読み込んでくれる

.ls でrootファイルに入っているものを調べると, "tree" という名前の TTreeが入っていることがわかる

tree->Print(); でTreeの列名 (branch) を確認することができる

演習データの内容

| Branch名 | 型 | 説明 |
|-------------|------------------|----------------------------------------------------------|
| nHit | Int_t | 検出器に粒子がヒットした回数 |
| total_adc | Double_t | 前検出器で粒子が落としたエネルギーに対応するADC値 |
| tdc | vector<Double_t> | i番目のデータが入ってきた時間のTDC値 |
| adc | vector<Double_t> | i番目のデータで粒子が落としたエネルギーに対応するADC値 |
| detector_id | vector<Int_t> | i番目のデータを検出した検出器ID |
| particle_id | vector<Int_t> | i番目のデータで検出器に入射した粒子に対応する番号 ※実はGeant4の粒子コードそのまま。ガンマ線は22 |

- tree->Print(); で出てきた情報と一致しているはず。
- 実験で誰かが作ったデータにも、このような簡単な説明がくっついているはず。(ついてなければ聞くか、ソースコード発掘)
- vector<double> は長さを変えられる配列のようなもの
- シミュレーションで作っているので実データではありえないものも入っていますが、そこはご愛嬌ということで

値を覗いてみる その1

```

root [x] tree->Scan()
*****
*   Row   * Instance * nHit.nHit * total_adc *      tdc *      adc * detector_ * particle_ *
*****
*       0 *       0 *       1 *     12609 *       0 *     12609 *       0 *      22 *
*       1 *       0 *       1 *      8400 *       0 *     8400 *       0 *      22 *
*       2 *       0 *       1 *      7303 *       0 *     7303 *       0 *      22 *
*       3 *       0 *       1 *     12787 *       0 *    12787 *       0 *      22 *
*       4 *       0 *       1 *     12831 *       0 *    12831 *       0 *      22 *
*       5 *       0 *       1 *     12482 *       0 *    12482 *       0 *      22 *
*       6 *       0 *       1 *     11865 *       0 *    11865 *       0 *      22 *
*       7 *       0 *       1 *     12894 *       0 *    12894 *       0 *      22 *
*       8 *       0 *       1 *     12668 *       0 *    12668 *       0 *      22 *
*       9 *       0 *       1 *     12350 *       0 *    12350 *       0 *      22 *
*      10 *       0 *       1 *     12744 *       0 *    12744 *       0 *      22 *
*      11 *       0 *       1 *     12380 *       0 *    12380 *       0 *      22 *
*      12 *       0 *       1 *     12902 *       0 *    12902 *       0 *      22 *
*      13 *       0 *       1 *     12748 *       0 *    12748 *       0 *      22 *
*      14 *       0 *       1 *       0 *       0 *       0 *       0 *      22 *
*      15 *       0 *       1 *     8302 *       0 *    8302 *       0 *      22 *
*      16 *       0 *       1 *     12937 *       0 *   12937 *       0 *      22 *
*      17 *       0 *       1 *     12470 *       0 *   12470 *       0 *      22 *
*      18 *       0 *       1 *       0 *       0 *       0 *       0 *      22 *
*      19 *       0 *       1 *     12868 *       0 *   12868 *       0 *      22 *
*      20 *       0 *       1 *     12668 *       0 *   12668 *       0 *      22 *
*      21 *       0 *       1 *     12691 *       0 *   12691 *       0 *      22 *
*      22 *       0 *       1 *     9022 *       0 *   9022 *       0 *      22 *
*      23 *       0 *       1 *       0 *       0 *       0 *       0 *      22 *
*      24 *       0 *       1 *     12982 *       0 *   12982 *       0 *      22 *

```

値を覗いてみる その2

```
root [x] tree->Show(33)
=====> EVENT:33
nHit          = 1
total_adc    = 12904
tdc           = (vector<double>*)0x600002085ae0
adc            = (vector<double>*)0x60000208fa60
detector_id   = (vector<int>*)0x60000208fb0
particle_id   = (vector<int>*)0x60000208eca0
```

- こちらは特定のイベントの中身を覗く方法
- テキストデータを最初に示したときに， まずは中身を確認するという話をした。
- TTreeに対しては， Print()や，Scan()， Show()が中身の確認に相当する
- 全部0とかカラじゃないよね??

TTreeからヒストグラム

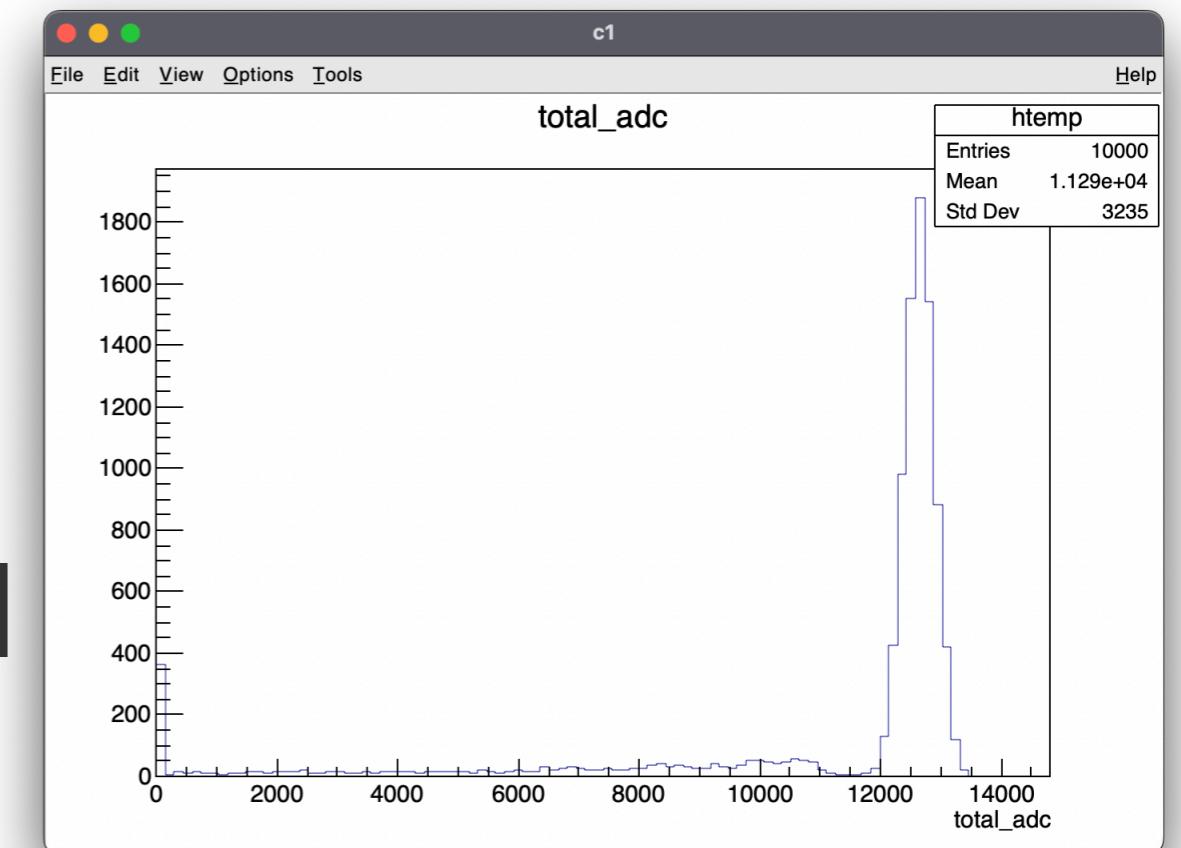
- `tree->Draw("<ブランチ名>");`
とするだけ。
- 今までの苦労がなんだったのかと思えるくらい簡単
- ビン幅, 最小値, 最大値も勝手に決めてくれる

```
root [x] tree->Draw("total_adc>>h(100,0,20000);")
```

- このようにブランチ名の後に`>>`をつけてビン数, 最大値, 最小値の指定も可能
- 2次元ヒストグラムを描く時は:
で2つのブランチを挟んでかく

```
root [x] tree->Draw("total_adc:nHit");
```

```
root [x] tree->Draw("total_adc");
```



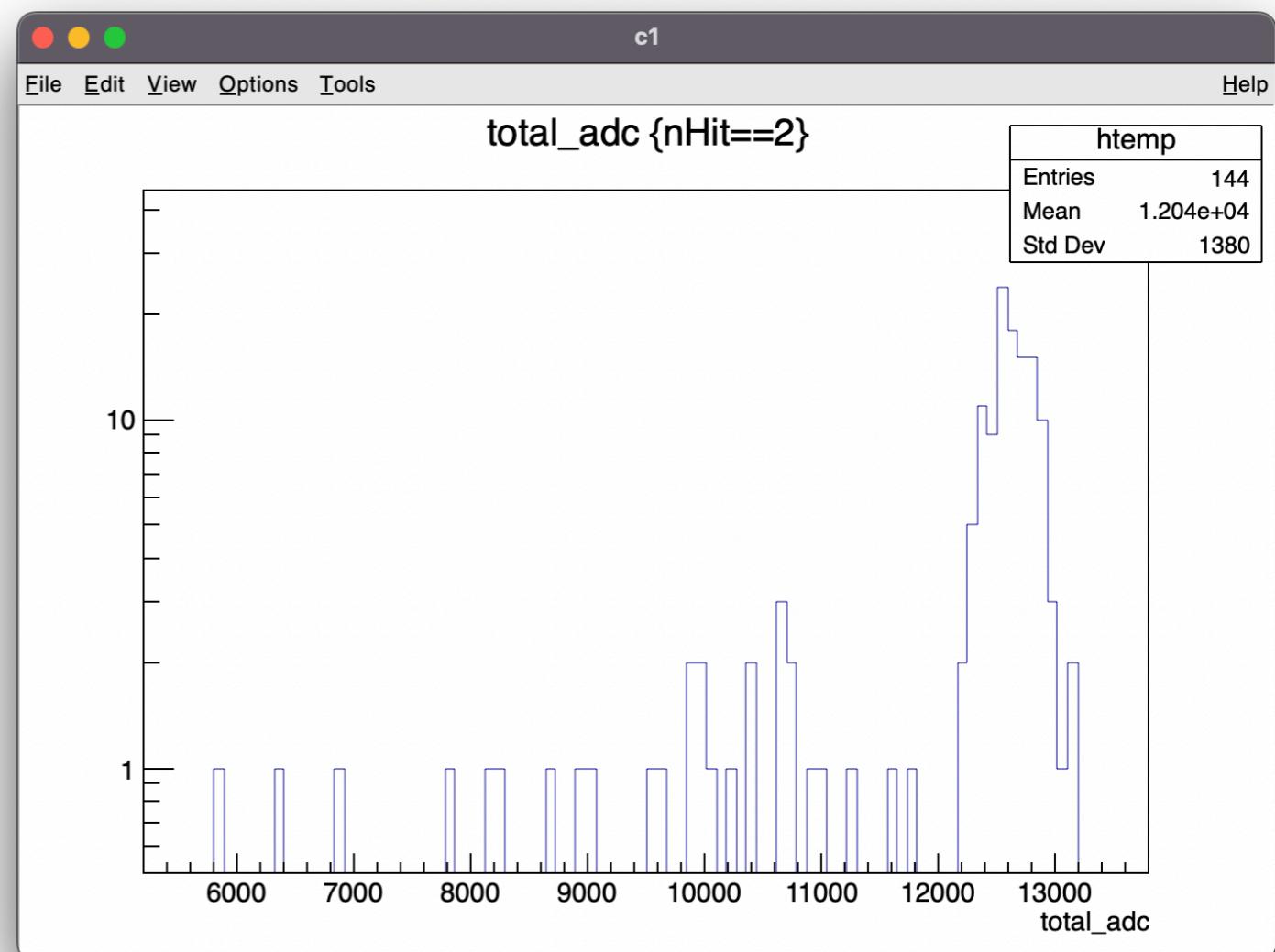
この場合, 軸はブランチ名に
勝手になる

カットをかける

- 実験データの中から、興味のある事象に注力したい。
 - 特に、稀事象探索実験では大量のゴミデータが存在
 - 条件をかけて、該当するデータだけを抜き出して解析する。
- この操作が高速で可能なツールは現状ROOT一択
 - ←ROOTを使う理由の一つ

```
root [x] tree->Draw("total_adc", "nHit==2")
```

`tree->Draw()`の第2引数はカット条件
必要に応じて `&&` や `||` で細かく条件指定できる
ここでは、検出器全体でHit数2の時の、`total_adc`のヒストグラムを描画している。



マクロでやってみる

デフォルト引数でファイル名を指定 (第2回参照)

```
int read_tree(TString filename=".../data/gamma_1_5MeV_10000_0.root"){

    auto file = TFile::Open(filename);
    auto tree = dynamic_cast<TTree*>(file->Get("tree"));

    Int_t nHit;
    Double_t total_adc;
    vector<double> *tdc = 0;
    vector<double> *adc = 0;
    vector<int> *detector_id = 0;
    vector<int> *particle_id = 0;

    tree->SetBranchAddress("nHit", &nHit);
    tree->SetBranchAddress("total_adc", &total_adc);
    tree->SetBranchAddress("tdc", &tdc);
    tree->SetBranchAddress("adc", &adc);
    tree->SetBranchAddress("detector_id", &detector_id);
    tree->SetBranchAddress("particle_id", &particle_id);

    ULong64_t n_entries = tree->GetEntries();
}
```

次ページに続く。

ファイルと、その中に入っている"tree"という名前のtreeを開く。

treeの中身を呼び出すのに使う定義など。
1番上の塊は、ソースコード内でデータに
アクセスするときに使う変数を定義
2番目の塊は、Treeの中のデータと変数を
紐付けするために必要

最後にtreeの中に入っているエントリ数
(イベント数) を調べておく

マクロでやってみる

続き

```

constexpr int nbin = 200;
constexpr double min_bin = 0;
constexpr double max_bin = 20000;
auto hist = new TH1D("hist", "hist", nbin, max_bin, min_bin);
hist->SetTitle(";ADC counts;Counts/100 ADC");

for(ULong64_t i_entry=0; i_entry<n_entries; ++i_entry){
    tree->GetEntry(i_entry);

    hist->Fill(total_adc);
}
hist->Draw();

return 0;
}

```

いつものノリでヒストグラムを準備。
constexprはコンパイル時に値を
 決めちゃえる時に使う**const** (雑な説明)

この**for**文は各エントリ毎にloopされる。
 tree->GetEntry(i)すると, iエントリ目のデータが, 前ページ
 で紐付けされた変数に格納される。
 この**for**文の中では, iエントリ目の total_adcやnHitが入って
 いるので, hist->Fill()などをやればよい

- お疲れ様でした...!

auto

```
auto file = TFile::Open(filename);
auto tree = dynamic_cast<TTree*>(file->Get("tree"));
```

- ・前述のマクロで使ったautoはコンパイラに型を考えろという意味
- ・auto value = 1; とかくと, 後ろが整数の1だから, valueの型を1と推測する.
- ・TFile::Openの戻り値の型はTFile* しかないので, 推測してくれる
- ・dynamic_cast<...> は...の型に変換せよ, ということ.
 - ・dynamic_cast<TTree*>(file->Get("tree")); とかくと, file->Get("tree") の結果をTTree*の型に変換しているので, それが初期化の値として与えられているtreeの型は必ずTTree*と推測できる
 - ・別に真面目に型を描いても良い.

SetBranchAddressが面倒

- rootのコンソールを立ち上げ,
tree->MakeClass()などとしてや
ると, <treeの名前>.Cと<treeの名
前>.hが作られる.
- .h のファイルの方には, 必要な定
義の部分が含まれているので, これ
らをコピペしてやればOK
- ポインタの定義 (*がついているや
つ)は = 0; をつけて初期化する
- fChainという名前で作られるの
で, 自分のtreeの名前に合わせる
- TBranchの定義は別にいらない
が, SetBranchAddressの第3引数
を消すのが面倒なのでコピペ推奨

tree.h

```
///////////
// This class has been automatically generated on
-- 省略 --
Int_t          nHit;
Double_t        total_adc;
vector<double> *tdc;
vector<double> *adc;
vector<int>     *detector_id;
vector<int>     *particle_id;

// List of branches
TBranch        *b_nHit;    //!
TBranch        *b_total_adc; //!
TBranch        *b_tdc;      //!
TBranch        *b_adc;      //!
TBranch        *b_detector_id; //!
TBranch        *b_particle_id; //!

-- 省略 --
fChain->SetBranchAddress("nHit", &nHit,
                           &b_nHit);
fChain->SetBranchAddress("total_adc",
                           &total_adc, &b_total_adc);
fChain->SetBranchAddress("tdc", &tdc, &b_tdc);
fChain->SetBranchAddress("adc", &adc, &b_adc);
fChain->SetBranchAddress("detector_id",
                           &detector_id, &b_detector_id);
fChain->SetBranchAddress("particle_id",
                           &particle_id, &b_particle_id);

-- 省略 --
```

高度な扱い方

ここ以外の部分はread_tree.Cと共通

```

constexpr int target_detector_id = 1;
for(ULong64_t i_entry=0; i_entry<n_entries; ++i_entry){
    tree->GetEntry(i_entry);

    if(nHit == 0) continue;

    for(auto i=0;i<detector_id->size();++i){
        if(detector_id->at(i) == target_detector_id){
            hist->Fill(adc->at(i));
        }
    }
    hist->Draw();
}

```

ゴミイベントの条件を決めて、さっさと
continueしてしまえば、それ以降の処理
がスキップされて、次のloopに入る

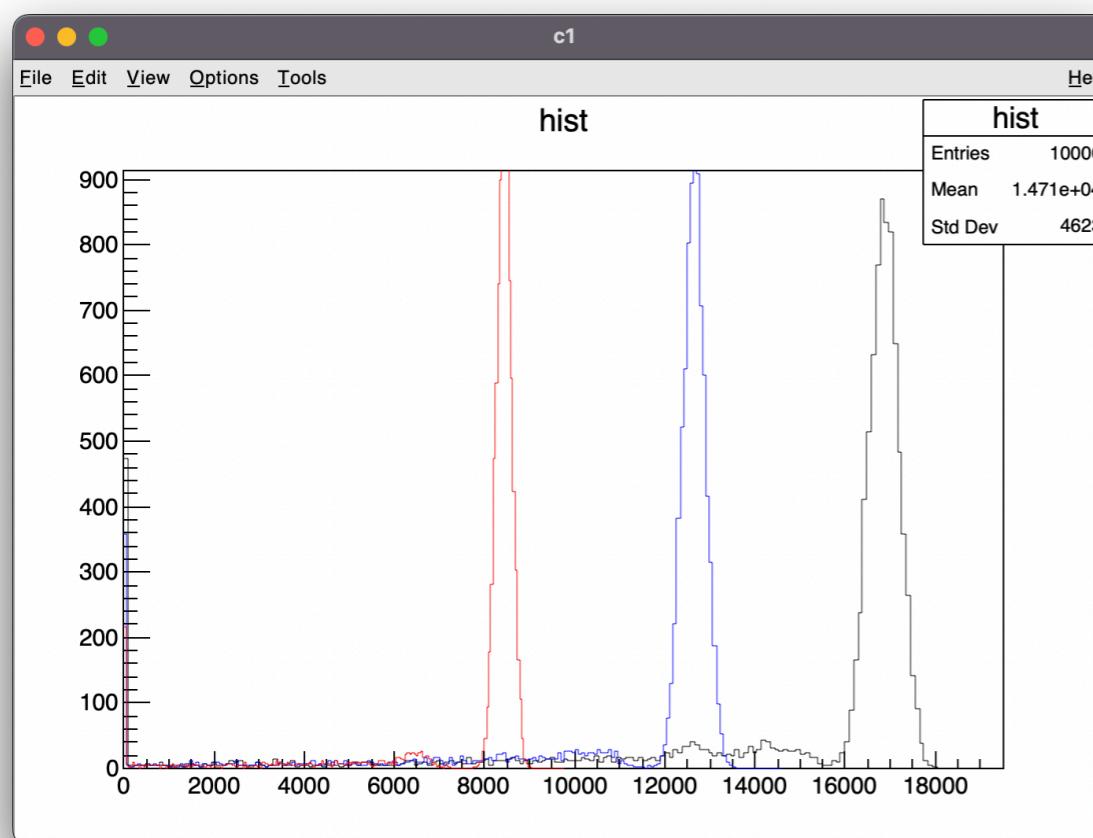
このfor文でvectorの全要素を調べる。
vectorなので、中身のアクセスは [] では
なく->at()で行うのが良い

このif文を突破してきたイベントだけFillすることで、
他のゴミイベントを落とすことができる。

- tree->Draw()だけでできることができる。
 - 他の条件ファイルを参照したり、1式ではかけないような面倒な計算を実行したりすること
 - ここでは、vector(動的配列)の中で、特定のdetectorに入ったエネルギーを取得する例を示す

tree->Draw()をマクロで

- マクロ内で、TH1Dを作つて,tree->Draw()から入れ込むことができる
- マクロ内で複数のファイルを開いて重ね書きしてみる



```

TH1D* get_histogram(TString filename=
                     "../data/gamma_1_0MeV_10000_0.root"){
    auto file = TFile::Open(filename);
    auto tree = dynamic_cast<TTree*>(file->Get("tree"));

    constexpr int nbin = 200;
    constexpr double min_bin = 0;
    constexpr double max_bin = 20000;
    TH1D* hist = new TH1D("hist", "hist", nbin, max_bin,
min_bin);

    tree->Draw("total_adc>>hist");

    return hist;
}

int make_histogram_from_tree(){

    auto hist1 = get_histogram(
        "../data/gamma_1_0MeV_10000_0.root");
    auto hist2 = get_histogram(
        "../data/gamma_1_5MeV_10000_0.root");
    auto hist3 = get_histogram(
        "../data/gamma_2_0MeV_10000_0.root");

    hist3->SetLineColor(kBlack);
    hist3->Draw();
    hist2->SetLineColor(kBlue);
    hist2->Draw("same");
    hist1->SetLineColor(kRed);
    hist1->Draw("same");

    return 0;
}

```

>> でTH1Dにつけた名前を呼ぶとそこに格納される

Treeを作る

- C++の世界にデータを入れてしまえば、自分でTTreeを作ることができる。

```
// make_tree_ascii.C
int make_tree_ascii(TString filename=
                     "../data/data1.txt"){

    auto file_out =
        new TFile("data1.root", "RECREATE");
    auto tree = new TTree("tree", "Title");

    Int_t n_turn;
    tree->Branch("n_turn", &n_turn);

    int buffer;
    ifstream ifs(filename);
    while(ifs >> buffer){
        n_turn = buffer;
        tree->Fill();
    }

    tree->Write();
    file_out->Close();

    return 0;
}
```

お馴染みのデフォルト引数。

この関数を後々使い回すことを初めから想定

まずはファイルを作成する。

.rootで終わるファイル名をつけておくのが無難
"RECREATE" を指定すると、ファイルを上書き
treeの第1引数はtreeの名前

treeに入れる値を一時的に格納する変数と、
ブランチ (スプレッドシートでいう列名) をこ
こに必要な数だけ列挙する

ここでは、ファイルから値を読み込んでいるが、
どんなやり方でもOK。
値を入れたら tree->Fill() しておく

終わったらtree->Write(), file_out->Close()を
読んで、お片付けしておく。

TreeからTreeを作る

- 物理解析では、膨大な量の生データから、興味のある事象に注目する→解析過程で扱うイベント数が減り、より抽象度の高い変数を作る。
- これまでにやったことの組み合わせで作ることができる。

ここで、新しいtreeに入れるための新しい変数を作る。

cutに引っかかった事象は新しいTreeには入らない

読み込むTreeの変数から、新しい変数を計算して、

新しいtreeに入る。

ここでは、生のADC値に何かの値を掛け算している

将来的にはエネルギーの値に較正することを想定

```
// tree_to_tree.C
int tree_to_tree(TString filename="../data/
gamma_1_5MeV_10000_0.root"){

    auto file = TFile::Open(filename);
    auto tree = dynamic_cast<TTree*>(file->Get("tree"));

    -- 省略 -- (p.11と同じ)

    ULong64_t n_entries = tree->GetEntries();

    auto new_file = new TFile("new_file.root",
"RECREATE");
    auto new_tree = new TTree("new_tree", "Tree title");

    Double_t calibrated_adc;
    new_tree->Branch("calibrated_adc", &calibrated_adc);

    constexpr double calibration_factor = 1.0; // dummy
    //

    for(ULong64_t i_entry=0; i_entry<n_entries; +
+i_entry){
        tree->GetEntry(i_entry);

        if(nHit!=1) continue;

        calibrated_adc = total_adc * calibration_factor;
        new_tree->Fill();
    }
    new_tree->Write();
    new_file->Close();

    return 0;
}
```

TChain

- 複数のTreeを1つのTreeとして扱いたい。
 - 実験をしていて取得データが多くなってくると、ファイルのサイズが膨大になってしまい→たくさんのファイルに分けて保存する。

```
// tchain.C
int tchain(){
    auto tree = new TChain("tree", "title");
    tree->Add("../data/gamma_1_5MeV_10000_0.root");
    tree->Add("../data/gamma_1_5MeV_10000_1.root");
    tree->Add("../data/gamma_1_5MeV_10000_2.root");
    tree->Add("../data/gamma_1_5MeV_10000_3.root");
    tree->Add("../data/gamma_1_5MeV_10000_4.root");
    // tree->Add("../data/gamma_1_5MeV_10000_*.root");

    cout << tree->GetEntries() << endl;

    return 0;
}
```

ファイルの中の、統合したいtreeの名前を使う

統合したいファイル名を列举

* を使ってまとめて指定することも可能。

この後は普通のtreeと同じやり方ができる。
変数を用意して、
SetBranchAddressすれば良い

- コマンドラインで複数のファイルに含まれる同種のTreeを統合するコマンド、haddというものもある。

進んだ話題 - 自習のヒント

- そうは言ってもやっぱりTreeを読むだけで書くこと多すぎ...
- pyroot
 - pythonから使えるROOT
 - SetBranchAddressも変数の定義もしなくて良い
 - Treeを読むこと以外で余計なことをすると遅くなってしまう
- RDataFrame
 - ROOTの比較的新しめの機能
 - 機能はすごいが、新しい概念を覚える必要があり、やや難しい

```

root [0] ROOT::RDataFrame df("tree", "../data/
gamma_1_5MeV_10000_0.root")
root [1] df.Describe()
(ROOT::RDF::RDFDescription) Dataframe from TChain
tree in file ../data/gamma_1_5MeV_10000_0.root

Property           Value
-----
Columns in total   6
Columns from defines 0
Event loops run    0
Processing slots   1

Column             Type
Origin
-----
adc               ROOT::VecOps::RVec<double>
Dataset
detector_id       ROOT::VecOps::RVec<int>
Dataset
nHit              Int_t
Dataset
particle_id       ROOT::VecOps::RVec<int>
Dataset
tdc               ROOT::VecOps::RVec<double>
Dataset
total_adc         Double_t

root [2] auto h1 =
df.Filter("nHit>0").Histo1D("total_adc");
root [3] h1->Draw();
Info in <TCanvas::MakeDefCanvas>: created
default TCanvas with name c1

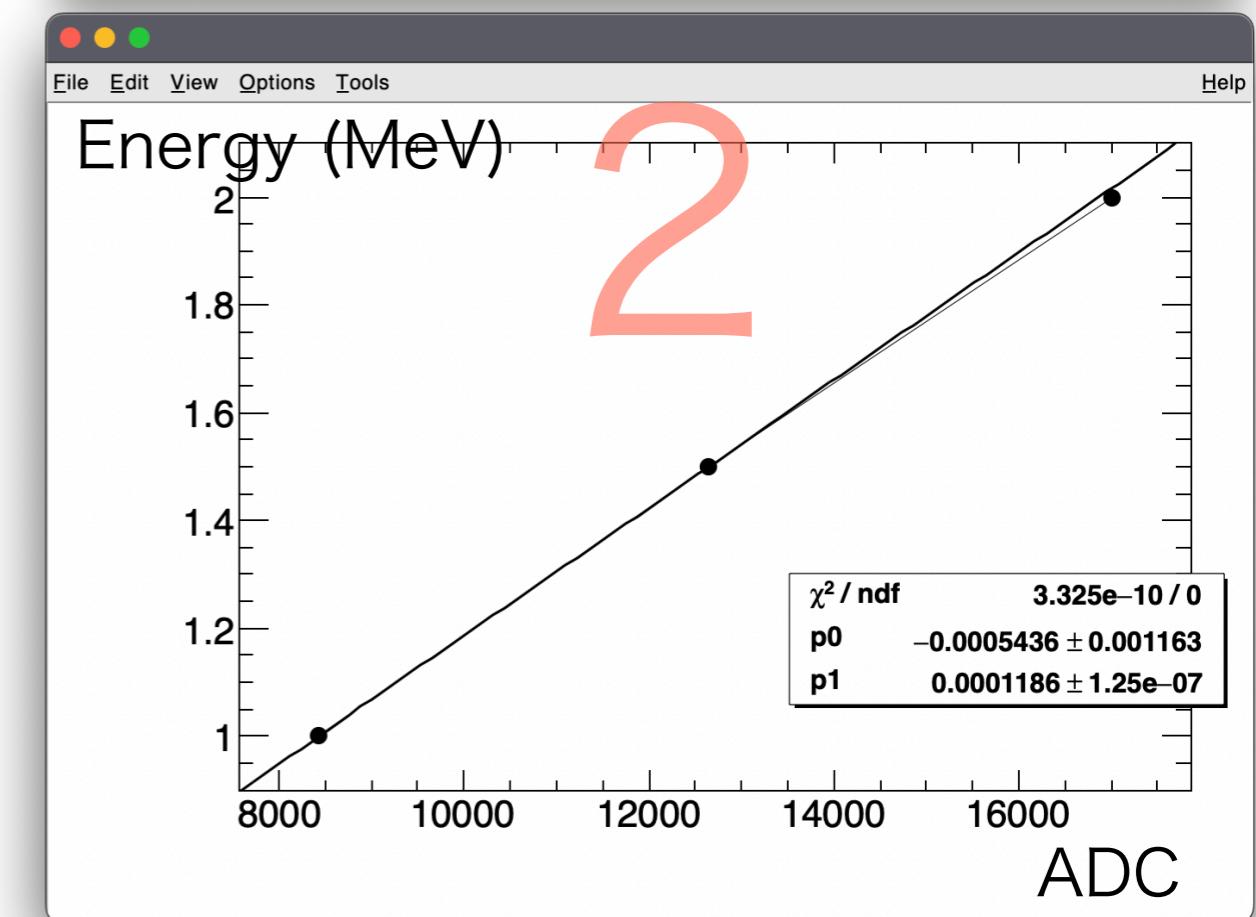
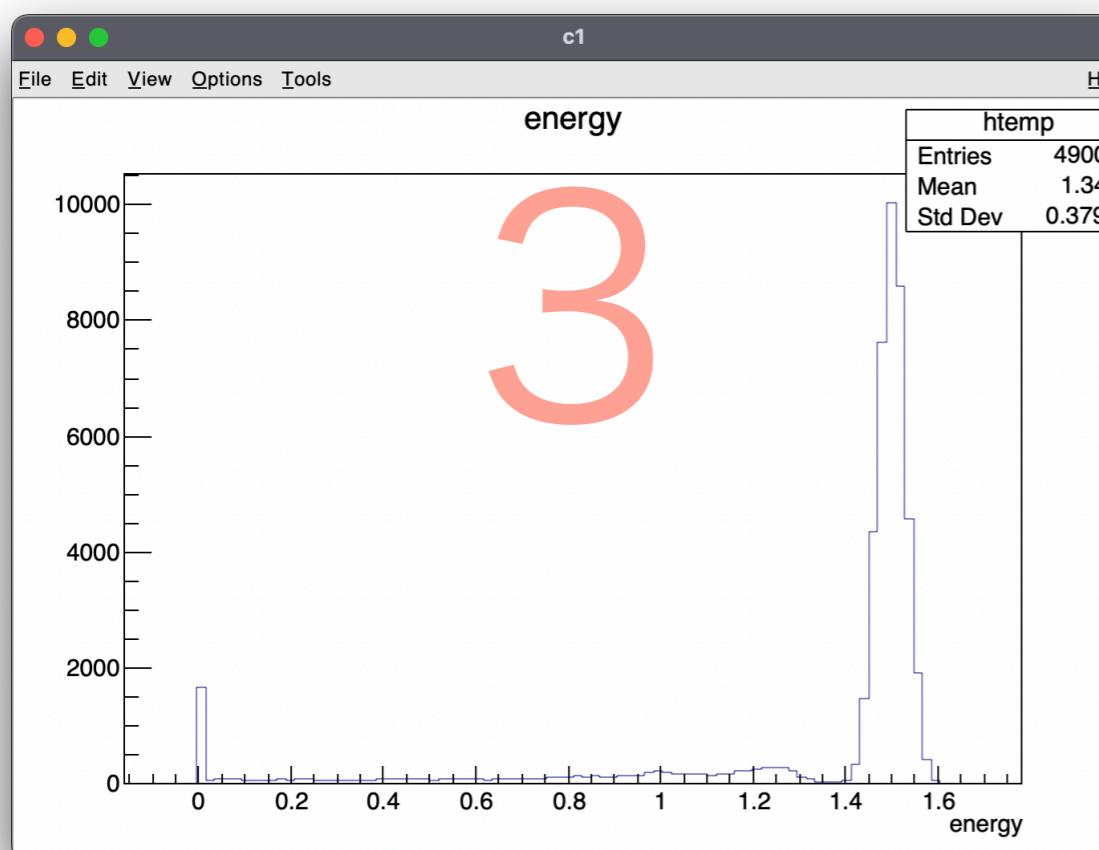
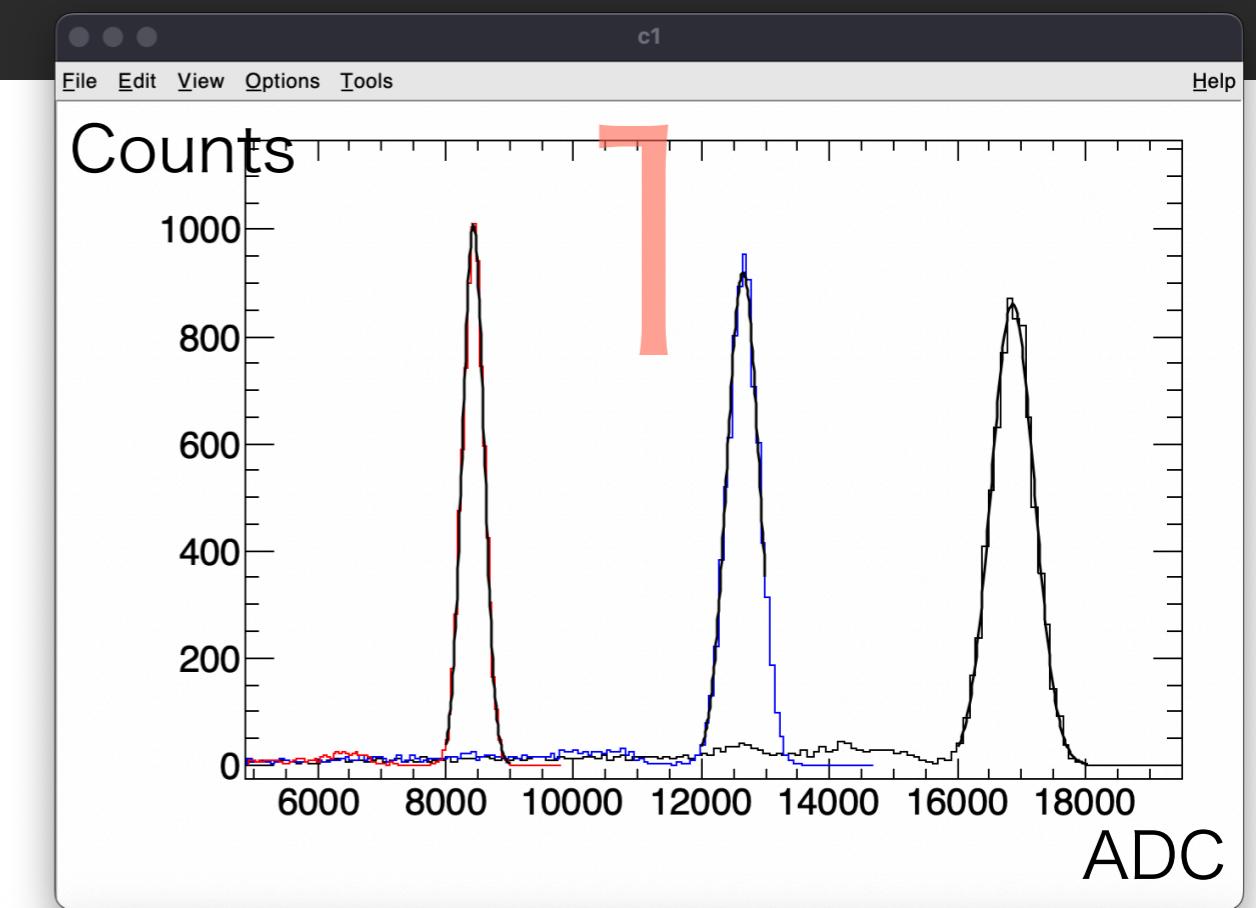
```

演習問題

- gamma_1_0MeV_10000_0.root,
gamma_1_5MeV_10000_0.root,
gamma_2_0MeV_10000_0.root, それぞれのtotal_adcの
ピークを適当な範囲でガウシアンでフィッティングしよう.
- それぞれのピークを1MeV, 1.5MeV, 2MeVとして較正しよう.
 - 横軸ADC値, 縦軸エネルギーでTGraphで3点をプロットしよう
 - 直線でFitしたら, ADC値→エネルギーにする式が得られる
- エネルギー較正した値を "calibrated_energy" と言う名前でBranchを作成して, 新しいTreeに保存しよう
- gamma_1_5MeV_10000_*.rootの5つのファイルのデータを1つに統合したTreeを作り, "calibrated_energy"のヒストグラムを書こう
- electron, protonのデータと比べてみよう (重ね書き)

演習問題ヒント

- 前回までの内容を復習する
- 1つのマクロで全てやる必要はない。
- ヒストグラムを書いて、Fitするマクロ
- Fit結果をTGraphにして、Fitするマacro
- 較正結果を詰めて、新しいTreeを作るマクロ
- Treeを統合するマクロ
- 以上のように分割してやっても良い
- 解答例はanswer1.Cやanswer2.Cにある。



まとめ

- ROOTのTreeは宇宙素粒子原子核業界で標準的なデータ方式
- 高速に解析を実施すること(特にヒストグラム)ができる
- Treeに格納されたデータの確認方法
- Treeのデータからカットをかけてヒストグラムを作成する方法
- マクロでTreeを読み出す方法
- Treeの作り方
- 今回でROOTの基本的機能については全て攫ったつもりです.
- ちなみに, 第2回の宿題 (自由研究) は進んでいますか?
- 宿題提出ついでに副賞として欲しいものも書いておいてください
- もらえるという保証はありませんが...