

ROOT 講習会 第2回 資料

Keita Mizukoshi (YMAP, JAXA)
Mail: mizukoshi.keita@jaxa.jp

- 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。

第2回 まえおき

- 今日はいよいよROOTを使っていきます。
 - 主に分布,(1次元)ヒストグラムの話です
 - 実際にコードを書いていきます.
 - コードのお作法とかにも多少うるさくなる予定です.
- 準備
 - ROOTはインストールされていますか?
 - 好きなエディタはインストールされていますか?
 - 今日用いるデータはダウンロードできていますか?
 - root_lectureとつくったディレクトリにcdで入って,
`> git clone https://github.com/ymapteam/root_lecture.git`
 - このコマンドが動かなければURLからZIPでダウンロードできます.

コマンドラインからROOTを起動

- コマンドラインでroot
- 右のスプラッシュが表示され,rootのプロンプトが起動
- Shellではなく,rootに命令できる
- root[なんか数字]の後に
C++の文を入力すると1行ずつ実行する.



```
% root
-----
| Welcome to ROOT 6.16/00          https://root.cern |
| (c) 1995-2018, The ROOT Team   |
| Built for macosx64 on Jan 23 2019, 09:06:13 |
| From tags/v6-16-00@v6-16-00    |
| Try '.help', '.demo', '.license', '.credits', '.quit'/.q' |
|-----|
root [0] cout << "Hello, ROOT" << endl;
Hello, ROOT
root [1] 1+1
(int) 2
root [2]
```

空気を読んで文末のセミコロン ; が
なくても動く

マクロ

- ・何度もROOTを立ち上げてコマンドを打ち込むのは面倒だ
- ・ファイルにコマンドを記述して,ROOTに実行させることができる
- ・実行コマンドを一行ずつファイルに記述して,{}で囲む
- ・コマンドラインで実行
 - >root first_macro.C
- ・マクロを書いて実行→一部修正,追記→実行 を繰り返して解析を進めていく
- ・C++はmain関数から処理がスタートするが,{}で括るROOTマクロは単に上から実行される

```
first_macro.C
1 {
2
3     cout << "Hello, World!" << endl;
4
5     int val = 1+1;
6     cout << "val = " << val << endl;
7
8 }
```

解析(於:実験)の目指すところ

- 実験をしてデータを集め, 物理的な値を推測する
 - 真の分布 ← 世界はどのように構成されているのか (=物理のテーマ)
 - 明日の降水量は?
 - 放射性物質の含有量は?
 - 実験をするとデータが集まるが, 真の分布の全ての事象(母集団)をすべて集めることは(多くの場合)不可能
 - せいぜい有限の場所の現在の気象情報
 - 崩壊して出てくるガンマ線の検出数
- データから, 真の分布を推定し, 統計学的検定を行う
 - 今回はまずデータの分布を表現すること (=ヒストグラム) を目指す
 - Fitting, 推定, 検定は次回以降のテーマ

例: 試験の成績

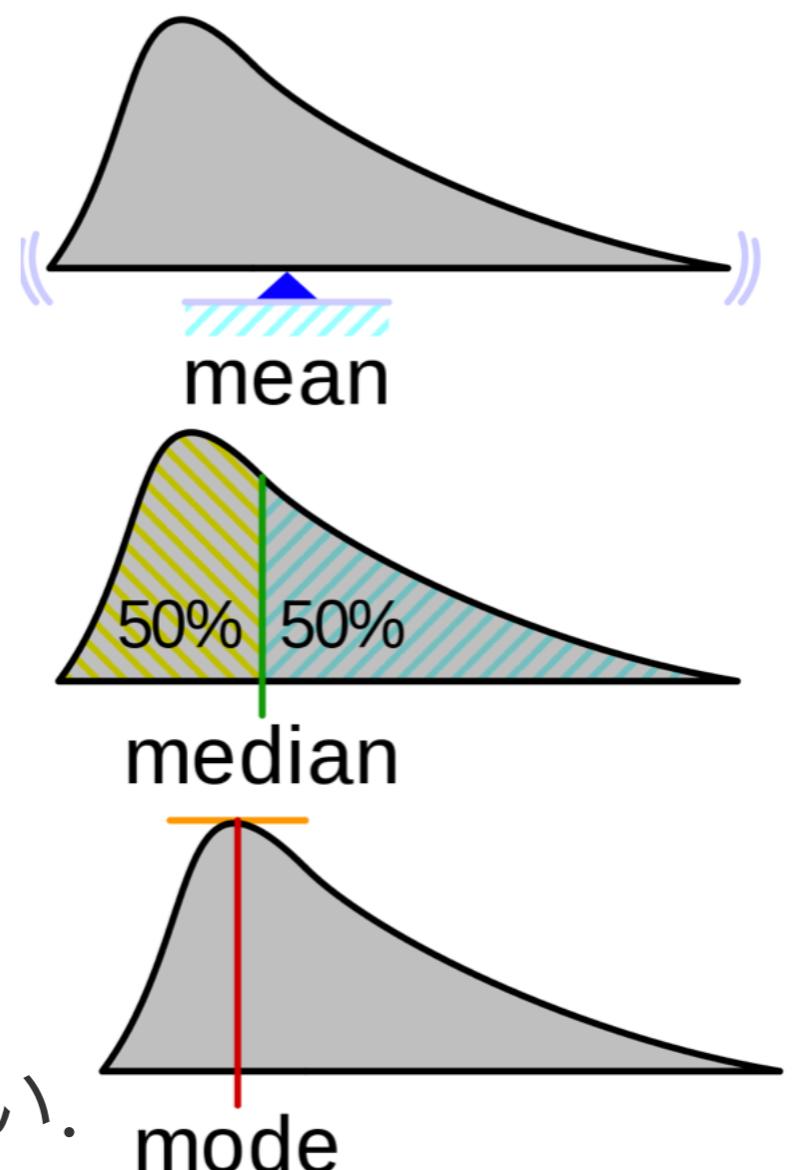
- 10点満点のテストを10000人でおこなった.
- 自分は4点だった.
- 自分の成績はどのくらいなのか?
- 無作為に30人に成績を聞いてみた.

人名	A君	B君	C君	D君	E君	F君	G君	...	略
点数	5	6	2	3	4	5	7		

- これらひとつひとつの数値に注目しても
全体の実態はわからない

平均値, 中央値, 最頻値

- 代表的な値を用いて分布を評価することを考える.
- 典型的な値はいくつ? (例: 普通の人は何点とるの?)
- 平均値 (Mean , Average) $\frac{1}{n} \sum_{i=1}^n x_i$
 - よく代表値として用いられる.
- 中央値 (Median)
 - データを昇順にした際に中心の値
- 最頻値 (Mode)
 - 最もよくデータに現れる値
- これらの値だけでは, 分布をよく表せない.



標準偏差, 四分位点

- 代表値だけでなく, 分布の広がりについても評価したい.
- 標準偏差 σ (Standard deviation) *

 - 平均 \bar{x} を用いて, $\sigma^2 = \frac{1}{n} \sum_i (x_i - \bar{x})^2$ (ROOTの定義)
 - 変数 x と同じ次元で広がりを表すことができる.

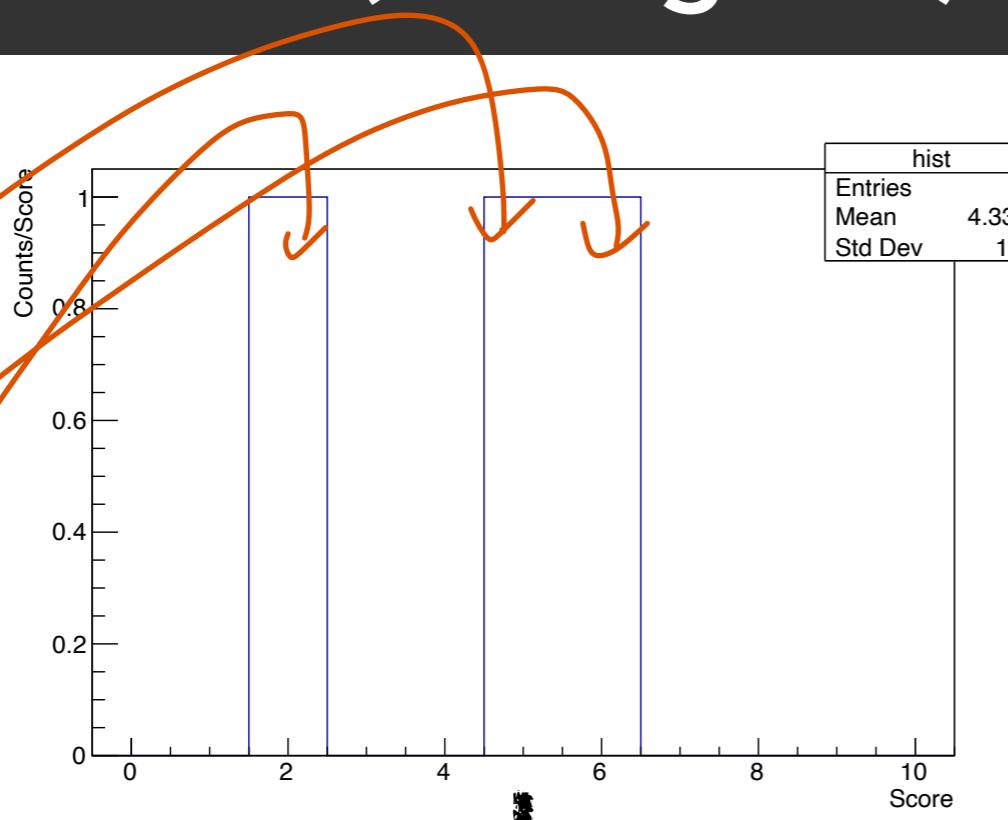
- 四分位範囲 (interquartile range, IQR)
 - 第1四分位数と第3四分位数の差
- 結局のところ図示しないと何が起こっているのか理解するのは難しい ← 解析の際は必ず図示する.

*母集団の標準偏差を考えるか, 標本を考えるかで形が少し違う

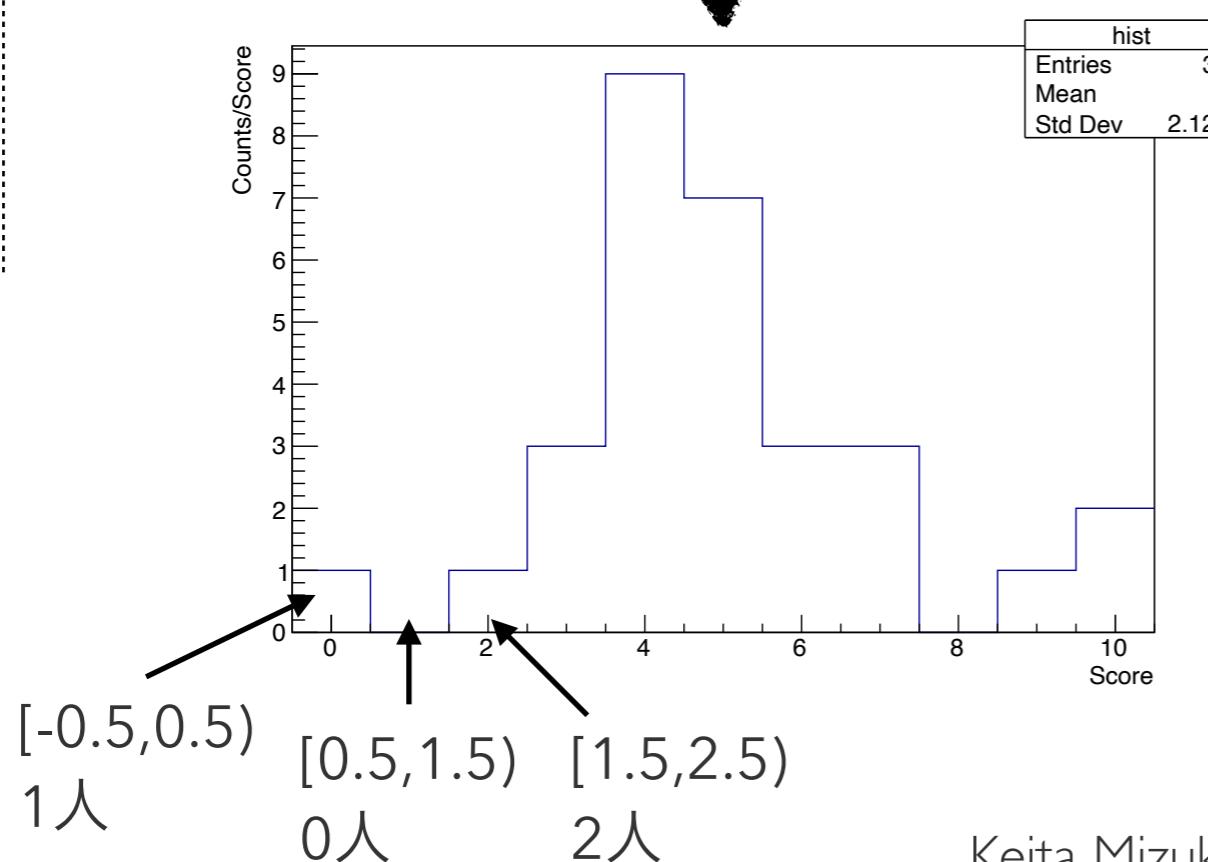
全体的に用語が混乱しているので注意

ヒストグラム (Histogram)

名前	点数
A君	5
B君	6
C君	2
D君	3
E君	4
F君	7



- 各値をある区間(:= bin)毎に集計し、値を積み上げて表示する
- 値を順に入れていくイメージ
- この区間の幅: bin幅
 - 今の場合1
 - 別に1じゃなくていい
- 基本的には積分すると総数 (Sample sizeやEvent数)になる
- 物理解析で常用
 - というかほとんどヒストグラムしか使わない
- ヒストグラムを描くと分布は一目瞭然



クラス (C++)

- 数値を計算機に理解させるためにデータ型 (type, built-in type) を用いる
 - int – 整数型
 - double – 実数型
 - char – 文字型
 - bool – 論理型 (true or false)
- 現実のコードの中ではこれらの複数の数値間に関連がある
- これらの型をセットにしてユーザー (あなたや, ROOTの開発者) が定義した型として扱う = クラス (class)
- この型でつくったデータの実体をインスタンスと呼ぶ

言語にある ユーザー定義	型 クラス	変数 インスタンス	定義のしかた	インスタンス
			int value; MyClass class1;	

クラス例とメソッド

- 3次元ベクトルを扱うクラス
(ROOTではTVector3)
 - x, y, zの3方向の大きさをdouble型×3でもつ

例:3次元ベクトルだと回転やノルムの値などの計算を組み込んでおきたい。クラスを呼ぶ時はメソッドを呼ぶだけで詳細については忘れてしまいたい。

- これらの値の集合の扱い方についてもクラス内できめる = メソッドやメンバ関数
 - インスタンス.メソッド(); とかく
 - これらの値の集合の初期化の仕方もきめる = コンストラクタ
 - コンストラクタも一種のメソッド

()をつけると括弧内の値で初期化する。
初期化に必要な値は1つとは限らない。
明示的にコンストラクタを呼ぶ際は

```
TVector3 vector1 = TVector3(1.0, 0.5, 0.5);
```

クラス名 —
インスタンス名

- いきなりクラスを作ることは考えなくてよい
- まずは人の作ったものを使う

`a() << endl;`

コンストラクタ
(クラス名と同じ名前の
関数が呼ばれる
既に定義されている)

メソッド

この場合はθの値を取得する

実践

- ドラクエウォークではぐれメタルが逃げるまでのターン数
- はぐれメタルはドラクエシリーズでのボーナスモンスター
- 倒すと桁違いの経験値が得られる
- すぐに逃げてしまう
- はぐれメタルを倒すために必要な準備をするために,はぐれメタルがいつ逃げるのか,という情報が必要



スマホアプリ ドラゴンクエストウォーク画面

実験と結果

- はぐれメタルに遭遇した時,パーティ全員で'ぼうぎょ'を選択して,はぐれメタルが逃げるまでのターン数を記録.
- 1セット20回試して,5セットデータ収集した.
→ヒストグラムにして図示する(もちろんROOTで)

試行	1	2	3	4	5	6	7	...
ターン	2	1	2	3	4	0	0	

解析の準備

- ROOTでマクロをつくりながら解析していく。
- ターミナルでまず解析用の場所をつくる
- git clone ... コマンドでデータをダウンロードしてくる
- ROOT2021というディレクトリが直下にできるはず
- まずはroot_lecture/macros/mzks/hagure_expにcdで移動
- ここに`data`ディレクトリがあり, data1.txtからdata5.txtまでデータがある
- `data`ディレクトリがある場所に`macros`ディレクトリをつくってマクロ置き場にする

```
> cd  
> mkdir root_tutorial/  
> cd root_tutorial/  
> git clone https://github.com/ymapteam/root_lecture.git  
> cd root_lecture/macros/mzks/hagure_exp  
> ls data  
data1.txt data2.txt data3.txt data4.txt data5.txt  
> mkdir macros  
> mkdir figures  
> cd macros
```

青帯は講習会参加者が手を動かすパート

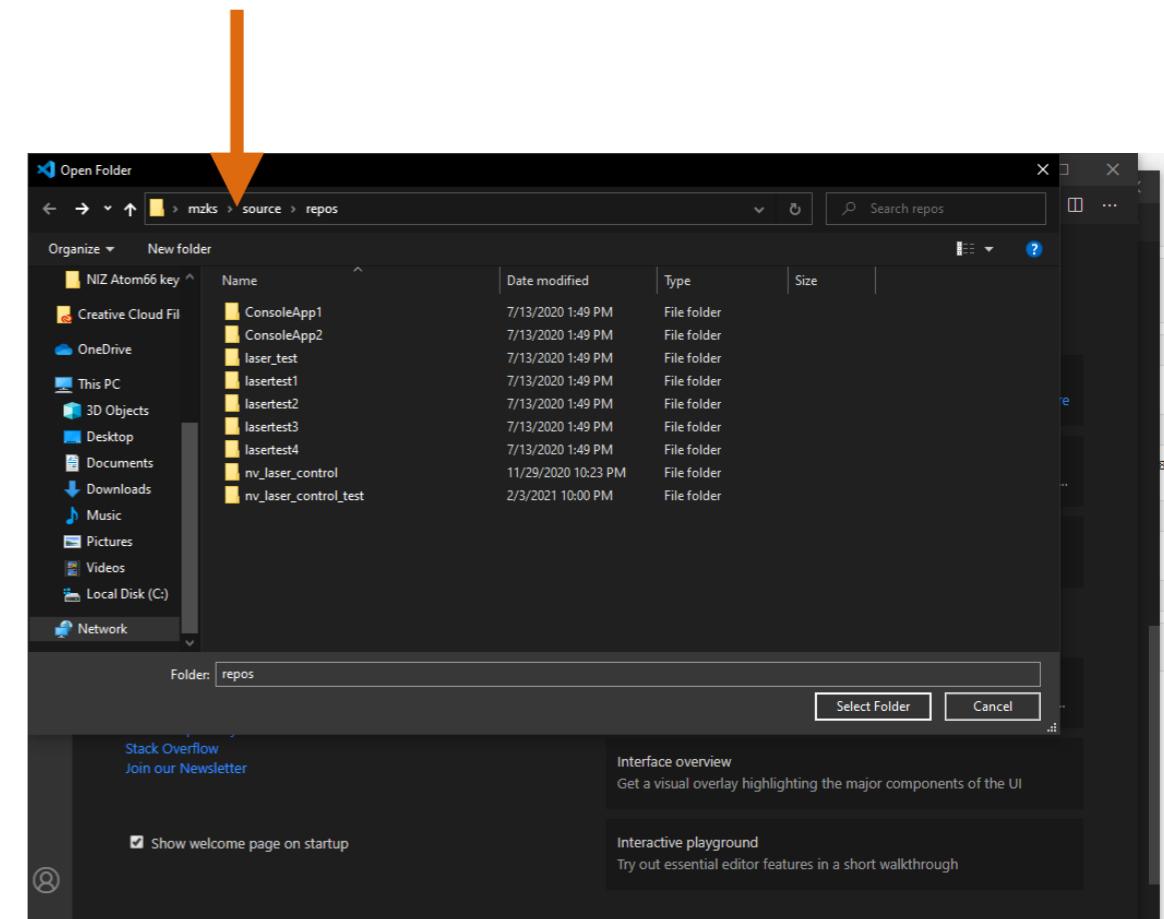
講習会でつかうマクロ例はmacro_examplesディレクトリに置いてあるので詰まつたら参照

現段階では分からなくてもOK
単にダウンロードコマンドとしてつかう

いろいろ混ざるとわかりづらいので
ディレクトリでデータ,マクロ,図,...とわける

Windowsの人向け ヒント

- WSLの世界からWindowsのファイルにアクセスする
 - ホームディレクトリは
/mnt/c/Users/ユーザ名/
- Windowsの世界からWSLのファイルへ
 - ファイラーの上の窓に \\wsl\$
 - Atom, VSCodeなどのエディタからマクロを編集する
- VSCode を設定していれば、ターミナル上で`code <file_name>`で開くことができる
- Vim, Emacsを使えばWSLだけで完結する
- Xserverは起動していますか？



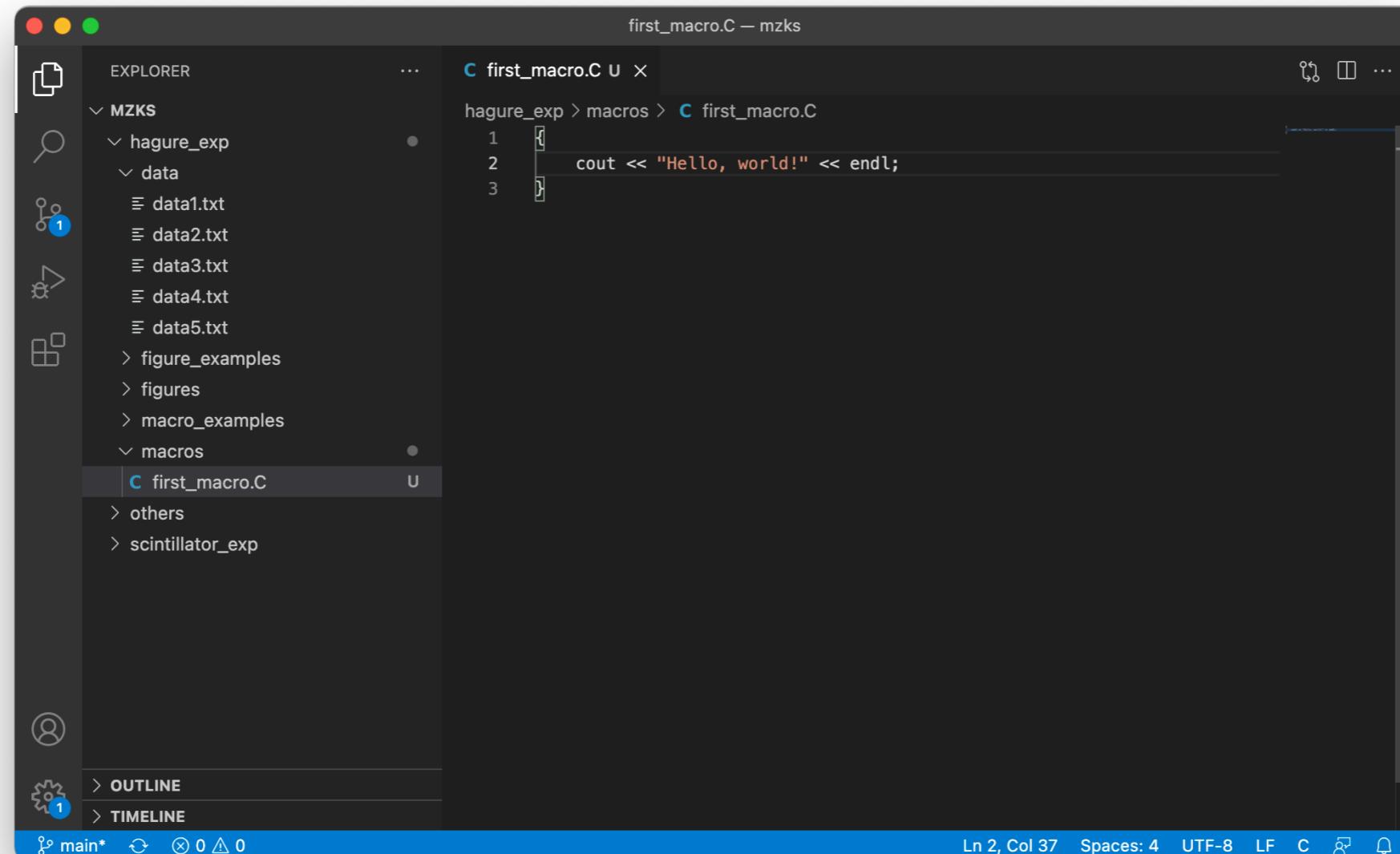
最初のマクロ

```
// first_macro.C <- ファイルの名前(これはかかなくてよい)  
1 {  
2     cout << "Hello, world!" << endl;  
3 }
```

```
> root first_macro.C  
Mon May 11 07:39:40 2020  
root [0]  
Processing first_macro.C...  
Hello, world!  
root [1]
```

- エディタを開いて`first_macro.C`という名前でファイルをつくる
 - 全ページでつくったmacrosというディレクトリ内でつくる
- first_macro.Cを入力して,保存
- 1,2,3というのは行番号. エディタが勝手に表示していることがほとんど(自分で入れる必要はない)
- ファイルがあるディレクトリまでcdで移動して,rootで実行

画面例 (Visual studio code)



The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left displays a project structure under 'MZKS'. The 'first_macro.C' file is selected in the Explorer. The main editor window shows the following C++ code:

```
1 cout << "Hello, world!" << endl;
```

The status bar at the bottom indicates the file is 2 lines long, 37 columns wide, with 4 spaces per line, in UTF-8 encoding, and is a C++ file.

```
> cd ~/root_tutorial/root_lecture/macros/mzks/macros
> root first_macro.C
root [0]
Processing first_macro.C...
Hello, world!
root [1]
```

ターミナルでの実行例

- できましたか?

関数としてマクロで実行

```
first_function.C
1 void first_function(){
2     cout << "This is my first function." << endl;
3 }
```

```
> root first_function.C
Mon May 11 07:41:42 2020
root [0]
Processing first_function.C...
This is my first function.
root [1]
```

- 新しく`first_function.C`を作る
- 単に中括弧{}で括ると上から順に実行するが, これからのために
C++の関数にして部品として使えるようにする
- C++/Cの場合はmain関数から処理が始まるが, root macroの場合は
マクロのファイル名から拡張子をとった名前の関数が最初によばれる

ヒストグラムのクラス TH1D

変な名前だけどこういう名付けをするのが昔流行った

- ・ ヒストグラムのクラスはTH1D.
- ・ 新しく自力で書いてみる

```
// first_hist.c
1 void first_hist(){
2
3     int total_bin = 10;
4     int min_bin = 0;
5     int max_bin = 10;
6     TH1D* hist = new TH1D("hist", "hist",
7                           total_bin, min_bin, max_bin);
8
9     hist->Fill(2);
10    hist->Fill(1);
11    hist->Fill(2);
12    hist->Fill(3);
13    hist->Fill(4);
14
15    hist->Draw();
16 }
```

ヒストグラムのクラス, TH1Dのインスタンス(実体)をコンストラクタ(初期化用メソッド)を使って作成する。

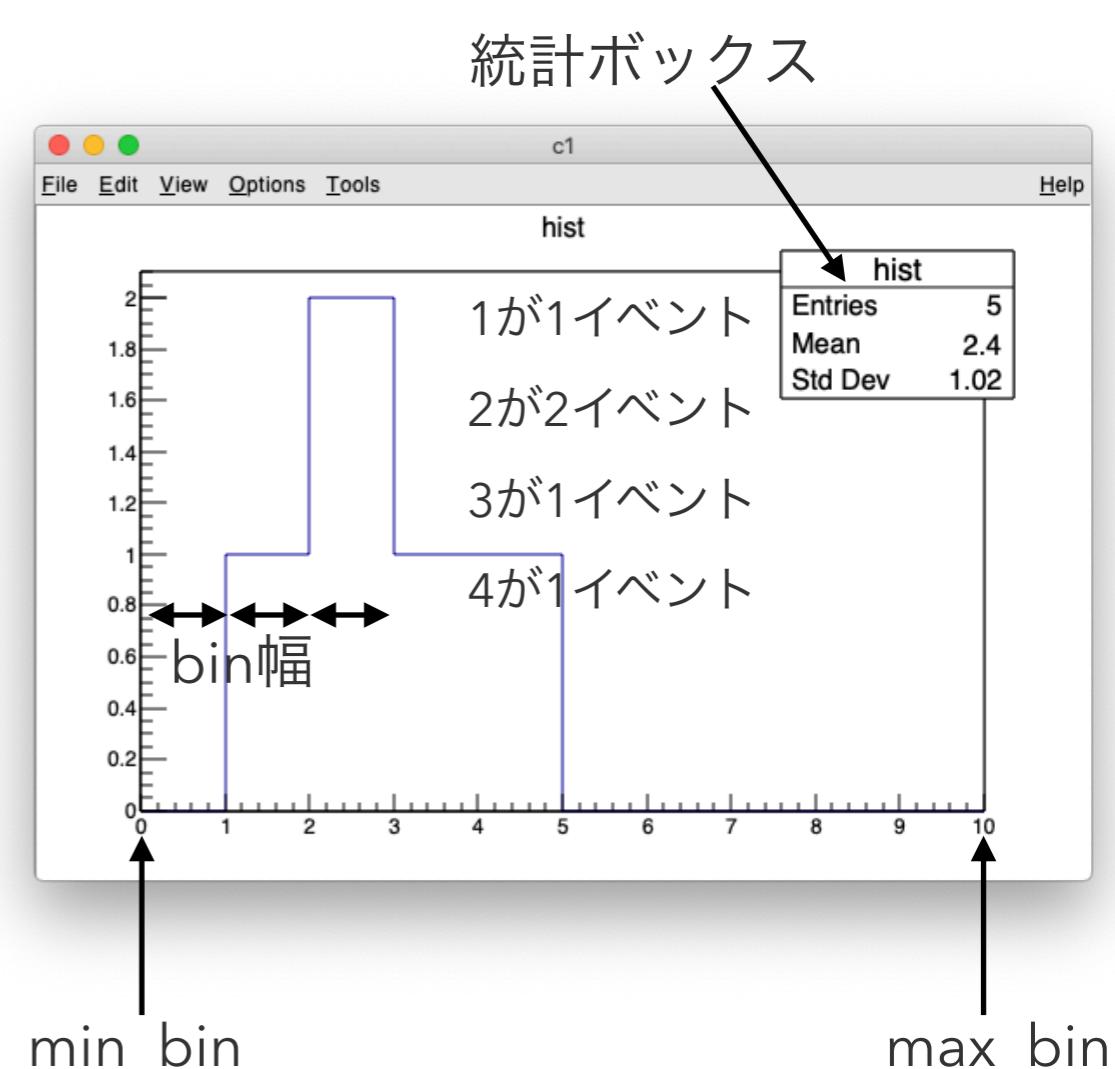
形式は TH1D("名前", "タイトル",
Bin数, 最小値, 最大値);
ポインタをつかってnewで定義する。

ヒストグラムに対してデータを詰める
Fill()というメソッドを使う
ポインタで定義した時は.(ドット)ではなく
->(アロー)でメソッドを呼ぶ

最後にDraw()メソッドを呼べば
図が出てくるはず

実行してみる

- 前ページの`first_hist.C`を実行すると、ヒストグラムが表示される
- 自分でFill()した5イベントが入れられたヒストグラムが表示されるはず
- ヒストグラムのどの部分がコードのどの部分に対応しているのか確認する。
 - 変更してみて確かめる(重要!)



*本例の様な離散値を扱う際はBin境界に注意する
うまく割り切れないBinを設定すると
凹んだり出っぱったりしてしまう

$$\text{bin幅} = (\text{max_bin} - \text{min_bin}) / \text{total_bin}$$

*Bin境界では大きい方のBinに入れられる

補足など

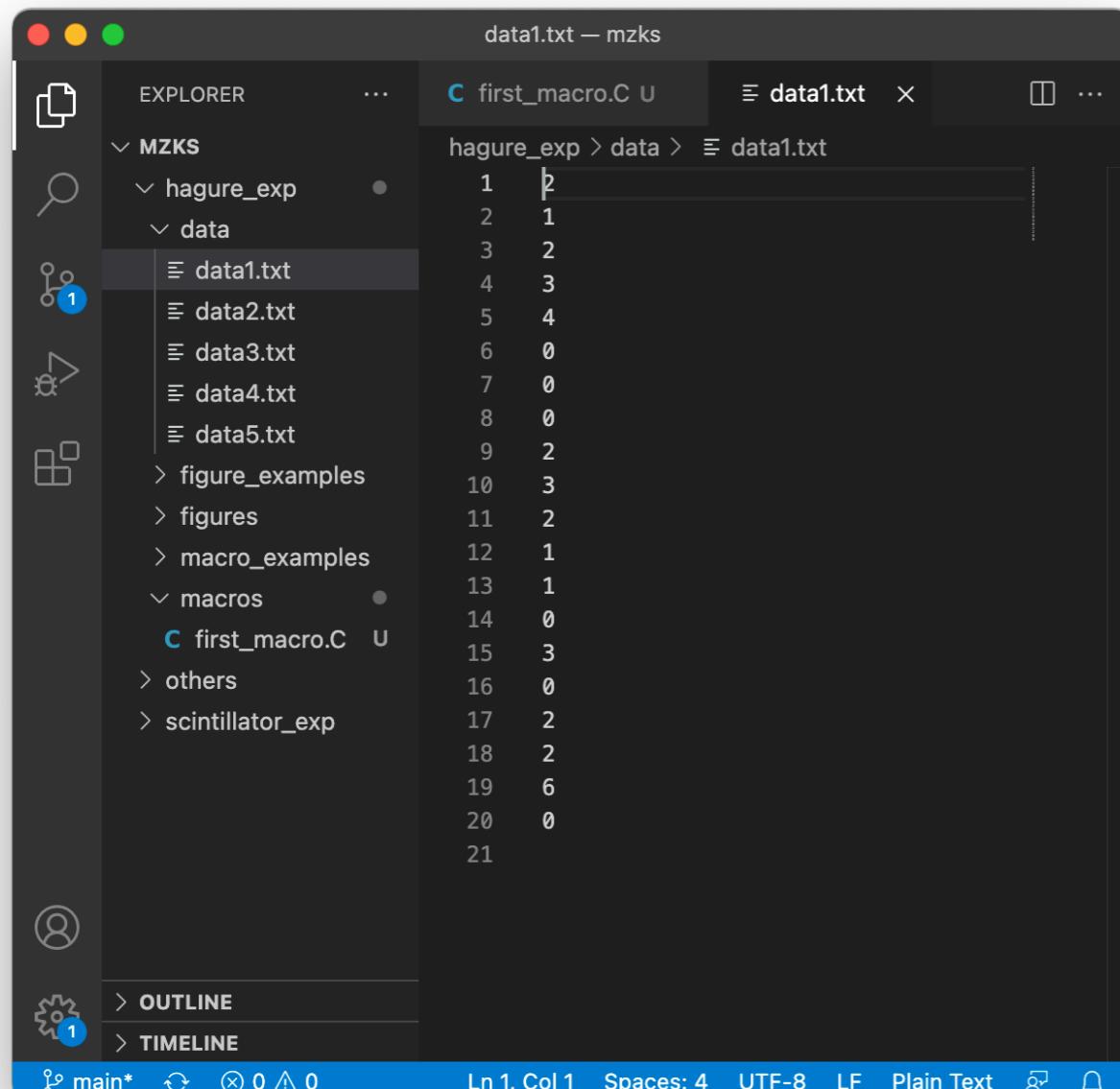
コードをたくさん書く様になると関数の引数の順番とかを忘れてしまうので、自分で分かりやすい名前の変数を定義して代入する。

絶対コード内で変更しないと心に誓う際はconstをつけることで変更されていないことを確認できる
(変更するとコンパイルエラーになって教えてくれる)

```
// first_hist.c すこし変更
1 void first_hist(){
2
3     const int total_bin = 10;
4     const int min_bin = 0;
5     const int max_bin = 10;
6     TH1D* hist = new TH1D("hist", "hist",
7                           total_bin, min_bin, max_bin);
8
9     hist->Fill(2);
10    hist->Fill(1);
11    hist->Fill(2);
12    hist->Fill(3);
13    hist->Fill(4);
14
15    hist->Draw();
16 }
```

- TH1DはROOTにあるヒストグラムのクラスのひとつ
 - ROOTではクラスの名前のはじめにTをつける
 - HistogramなのでH
 - 1次元なので1
 - double型を扱うのでD
- 他にもTH1F, TH1C, TH1S, TH1Iなどがあるが初心者はつかわない
- これからコードを書く人は自分の変数名などには略号などを使わず、誰が見てもわかりやすい名前をつけることを勧める。

データの確認

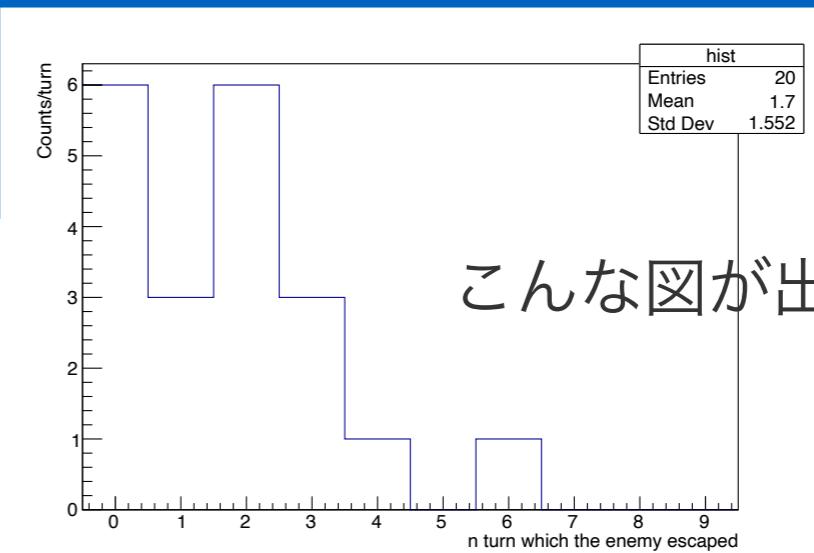


```
data1.txt — mzks
EXPLORER ... C first_macro.C U data1.txt ...
MZKS
hagure_exp > data > data1.txt
1 2
2 1
3 2
4 3
5 4
6 0
7 0
8 0
9 2
10 3
11 2
12 1
13 1
14 0
15 3
16 0
17 2
18 2
19 6
20 0
21
```

- エディタでデータファイルを開いて,
どのようにデータが入っているか
確認する
 - 今回は1列に1試行の結果が一つずつ
記述してある.
 - 実際の実験でも解析の前にまずは
生データを確認すること
 - ただし, 実際の実験ではデータが
膨大になるので, エディタで開けない
ことも多い
- (以降のROOT講習会で解説予定)

ファイルからFillする

- 手でいちいちFill()なんて普通はしない.
 - first_hist.Cは単なる例示
- Macro中でデータを読み込ませて,loopでFillする.
- 他にもいろいろとそれっぽくなる様に追加



離散値をヒストグラムにする際に値がBin端に来るようにする流儀と中心に来るようする流儀がある.
ここでは将来Fittingすることを考えて中心に来るように0.5ずらして調整

```
// fill_from_file.C
1 void fill_from_file(){
2
3     int total_bin = 10;
4     int min_bin = 0;
5     int max_bin = 10;
6     TH1D* hist = new TH1D("hist", "Title",
7                           total_bin, min_bin-0.5, max_bin-0.5);
8
9     ifstream ifs("../data/data1.txt");
10    int n_escaped_turn;
11    while(ifs >> n_escaped_turn){
12        hist->Fill(n_escaped_turn);
13    }
14
15    hist->SetTitle(";n turn which the enemy escaped;
16                          Counts/turn");
17
18 }
```

ファイルから値を読み込む際のイディオム
ifs >> n_escaped_turnでファイルから値が
1行ずつ読み込まれる
うまく読み込まれた時にこれはtrueになる
ので,失敗するまで(ファイル終端に来るまで)
whileループが回ってFillする
n_escaped_turnの型に注意

ヒストグラムのタイトル,軸タイトルの設定
';'で区切ってタイトル, Xタイトル, Yタイトル

マクロに関数を付け足していく

- data1.txtをマクロから読み込むことができた.
- しかし, data2.txt – data5.txtも, 同種のデータなので,同じヒストグラムに合わせて描きたい.
 - 実際の実験では, データ量が大きく, 比較的小さなファイルに分けてデータが入っていることはよくある.
 - もっとファイル数がたくさんあった場合, ファイルのサイズが大きい場合を想定
- ifstream とwhile文を5個書いても良いが, もう少しマシなやり方を考える
 - この方法だとすぐに限界が来る
- さっそく作ったfill_from_file.Cを部品として活用することを考える

```
// fill_from_files.C
1 void fill_from_file(TH1D* hist, TString filename){
2
3     ifstream ifs(filename);
4     int n_escaped_turn;
5     while(ifs >> n_escaped_turn){
6         hist->Fill(n_escaped_turn);
7     }
8 }
9
10 void fill_from_files(){
11
12     int total_bin = 10;
13     int min_bin = 0;
14     int max_bin = 10;
15     TH1D* hist = new TH1D("hist", "Title",
16                           total_bin, min_bin-0.5, max_bin-0.5);
17
18     fill_from_file(hist, "../data/data1.txt");
19     fill_from_file(hist, "../data/data2.txt");
20     fill_from_file(hist, "../data/data3.txt");
21     fill_from_file(hist, "../data/data4.txt");
22     fill_from_file(hist, "../data/data5.txt");
23
24     hist->SetTitle(";
25             n turn which the enemy escaped;Counts/turn");
26 }
```

マクロに関数を付け足していく(脱初心者むけ)

```
// fill_from_files.C (再掲)
1 void fill_from_file(TH1D* hist, TString filename){
2     ifstream ifs(filename);
3     int n_escaped_turn;
4     while(ifs >> n_escaped_turn){
5         hist->Fill(n_escaped_turn);
6     }
7 }
8 }

10 void fill_from_files(){
11     int total_bin = 10;
12     int min_bin = 0;
13     int max_bin = 10;
14     TH1D* hist = new TH1D("hist", "Title",
15                           total_bin, min_bin-0.5, max_bin-0.5);
16     fill_from_file(hist, "../data/data1.txt");
17     fill_from_file(hist, "../data/data2.txt");
18     fill_from_file(hist, "../data/data3.txt");
19     fill_from_file(hist, "../data/data4.txt");
20     fill_from_file(hist, "../data/data5.txt");
21
22     hist->SetTitle("");
23     n turn which the enemy escaped;Counts/turn");
24     hist->Draw();
25
26 }
```

名前にsをつけて複数形にしたマクロ

引数としてヒストグラムのポインタとファイル名を取ることにする
このファイル名を変えれば開くファイルが変わる
TStringはROOTで文字列を扱うクラス

さっき作ったfill_from_file()のヒストグラムにFillする部分だけ抜き出してきた。
ifstreamのコンストラクタは生書きではなく引数で与えたファイル名にする。

ファイル名と同じなのはこっちの関数なので,
root -l fill_from_files.C とした際にはこの関数
がまず呼ばれる

上で定義したfill_from_file()を, filenameを
変えて繰り返し呼び出すとwhileを繰り返し
書く必要がない

デフォルト引数(脱初心者むけ その2)

```
// fill_from_file.c (少し変更して再掲)  
1 void fill_from_file  
    (TString filename="../data/data1.txt"){  
2  
3     int total_bin = 10;  
4     int min_bin = 0;  
5     int max_bin = 10;  
6     TH1D* hist = new TH1D("hist",  
                           "Title",total_bin,  
                           min_bin-0.5,max_bin-0.5);  
7  
8     ifstream ifs(filename);  
9     int n_escaped_turn;  
10    while(ifs >> n_escaped_turn){  
11        hist->Fill(n_escaped_turn);  
12    }  
13  
- 省略 -  
18 }
```

最初から将来関数を使い回すことを考える。
変更するであろうパラメータを引数にしておく。
そこに、パラメータ="なんとか"とかいて、この
パラメータが指定されなかった時に自動で使われる
値を入れておく(デフォルト引数)

引数のパラメータを関数内では使う。
この関数だけを実装している際も、デフォルト引数が
使われるため、実行ごとに指定する必要がない。

```
// ターミナルで  
  
> root fill_from_file.C  
// デフォルト引数の"../data/data1.txt"が使われる。  
  
> root 'fill_from_file.C("../data/data2.txt")'  
// 指定したので、デフォルトの代わりに"../data/data2.txt"が使われる。
```

マクロ内で関数を呼ぶ際も同様

コメント

- // 以降は計算機に無視される.
- なんでも書くことができる = コメント
- 関数の処理を一時的にスキップする
 - 別の関数を試してみたりする時に有効
 - 全部消す必要がない
- 一時的に使ったデバッグ用の処理をスキップする
- わかりづらい処理に対する注釈
- マクロ自身の説明を書いておく
 - 私の場合はマクロファイルの最初に右の様な説明を加えることがおおい.

```
{  
    //function_a();  
    function_b();  
}
```

```
{  
    //cout << "Debug Comment :" << value << endl;  
}
```

```
1 // -----//  
2 // comment_example.C  
3 // - ROOT macro to show  
4 //           comment example  
5 // Author: Keita Mizukoshi  
6 // Date  : May 12 2020  
7 // Usage :  
8 // >root comment_example.C  
9 // -----//
```

Printf デバッグ(実際に使うのはcoutだけど...)

- 謎のエラーが出た時、まず最初にやるべきことはエラーメッセージの確認。
 - 大抵の場合親切にエラーの原因を教えてくれる。
- それでもわからなかった時、コードにcoutを挿入して原因究明する。
 - どこまで動いていたのか？
 - 失敗する時に変数には具体的にどういう値が入っていたのか？
- 初心者ではない人は任意のデバッガ使ってください。私は初心者なので。

my_function_B()に原因がありそうだとわかる

```
1 int my_macro(){  
2  
3     cout << "reach my_function_A()" << endl;  
4     int value_A = my_function_A();  
5     cout << "value_A :" << endl;  
6  
7     cout << "reach my_function_B()" << endl;  
8     double value_B = my_function_B();  
9     cout << "value_B :" << endl;  
10  
11    cout << "reach my_function_C()" << endl;  
12    bool value_C = my_function_C();  
13    cout << "value_C :" << endl;  
14  
15    return 0;  
16 }
```

```
> root -l my_macro.C  
Mon May 11 07:41:42 2020  
root [0]  
Processing my_macro.C...  
reach my_function_A()  
value_A : 4  
reach my_function_B()  
ERROR!!!
```

モデルとFitting(次のステップのチラ見せ)

- はぐれメタルが一定の確率 p で逃げる時,
 n ターン逃げない確率は、このターン
逃げない確率 $q = 1 - p$ を用いて式(1)
- この分布の規格化定数から…式(2),
分布は試行数 C を用いて式(3)と
表すことができる(とモデル化する).
- ROOTで式(3)でFittingを行うと,
赤点線の結果を得た(図1).
- すぐ逃げると言われているが検証の
結果、モデルが正しければ3/4程度の
はぐれメタルは逃げていない。

$$(1 - p)^n p = q^n - q^{n+1} \dots (1)$$

$$\int_0^\infty q^n - q^{n+1} dn = \frac{q - 1}{\log q} \dots (2)$$

$$C \times \frac{\log q}{q - 1} (q^n - q^{n+1}) \dots (3)$$

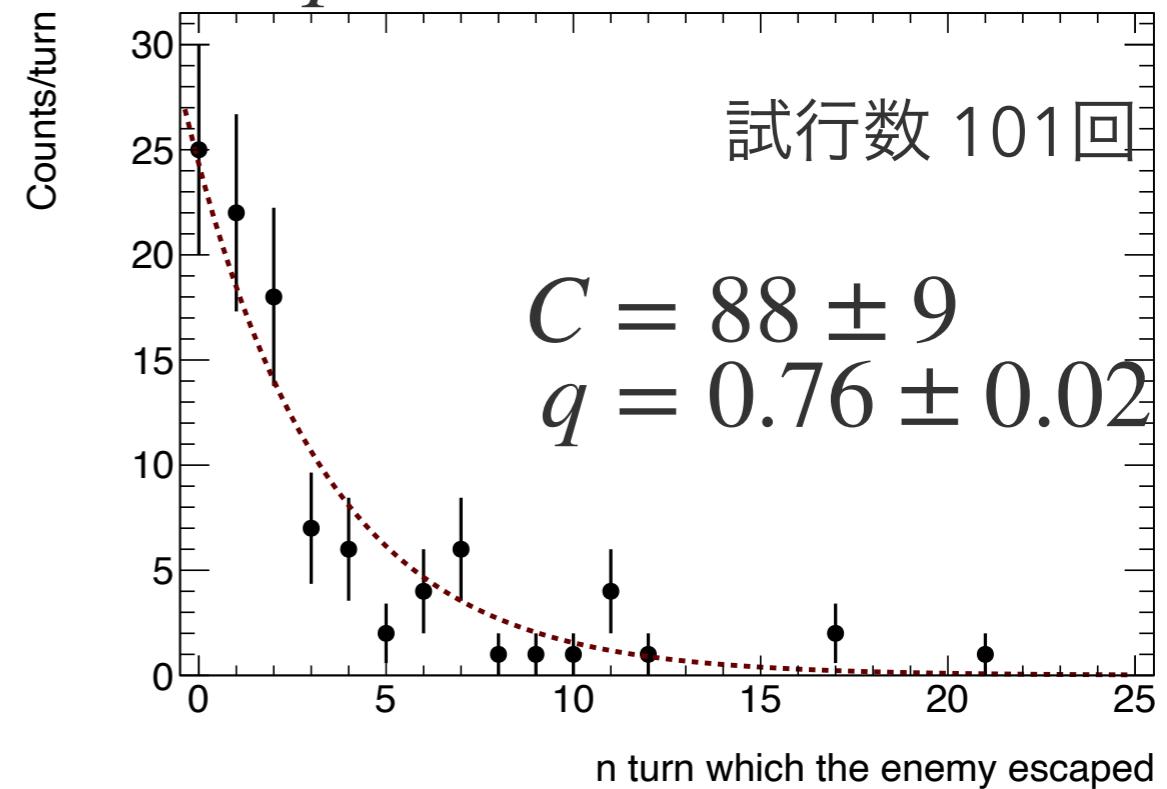


図1:はぐれメタルが逃げるターン数分布
見た目が変わっているが、本質的には
Fittingの線を除けばさっき書いたヒストグラムと等価

* Fittingと検定は次回の内容 29

図について補足

- ヒストグラムのBin中心にマーカーで表示することもある
 - Bin境界は自明なので省略
- 縦の線は誤差棒 (Error bar).
 - デフォルトだとBin内の事象数 N に \sqrt{N} のエラーがつく
 - `hist->Draw("E")`で表示できる.
- Y軸のタイトル Counts / turn
 - 事象数をBin幅で割り算した値を詰めることもよくある.
 - 異なるBin幅のヒストグラムを比較できる
 - この場合はBin幅1turnなので何もしていない(割る1した)
 - Bin幅で規格化しない場合も, Y軸にBin幅を書いておくと親切

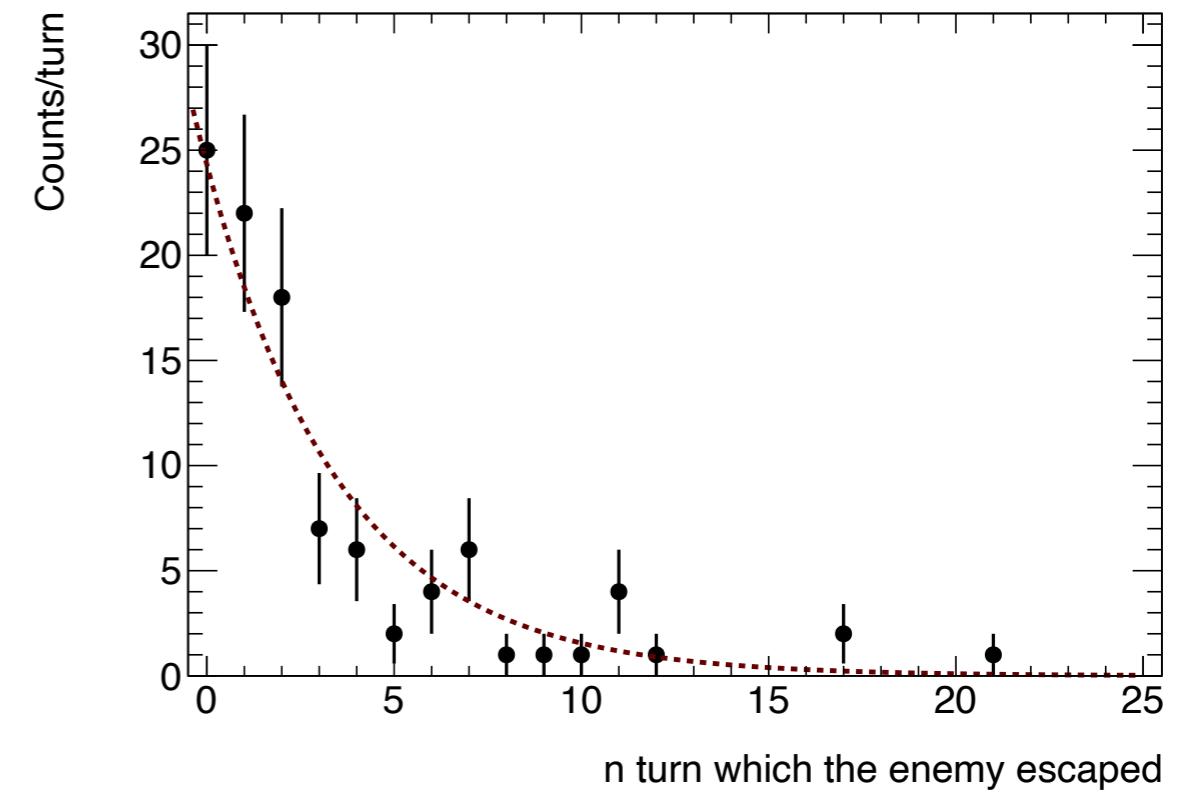
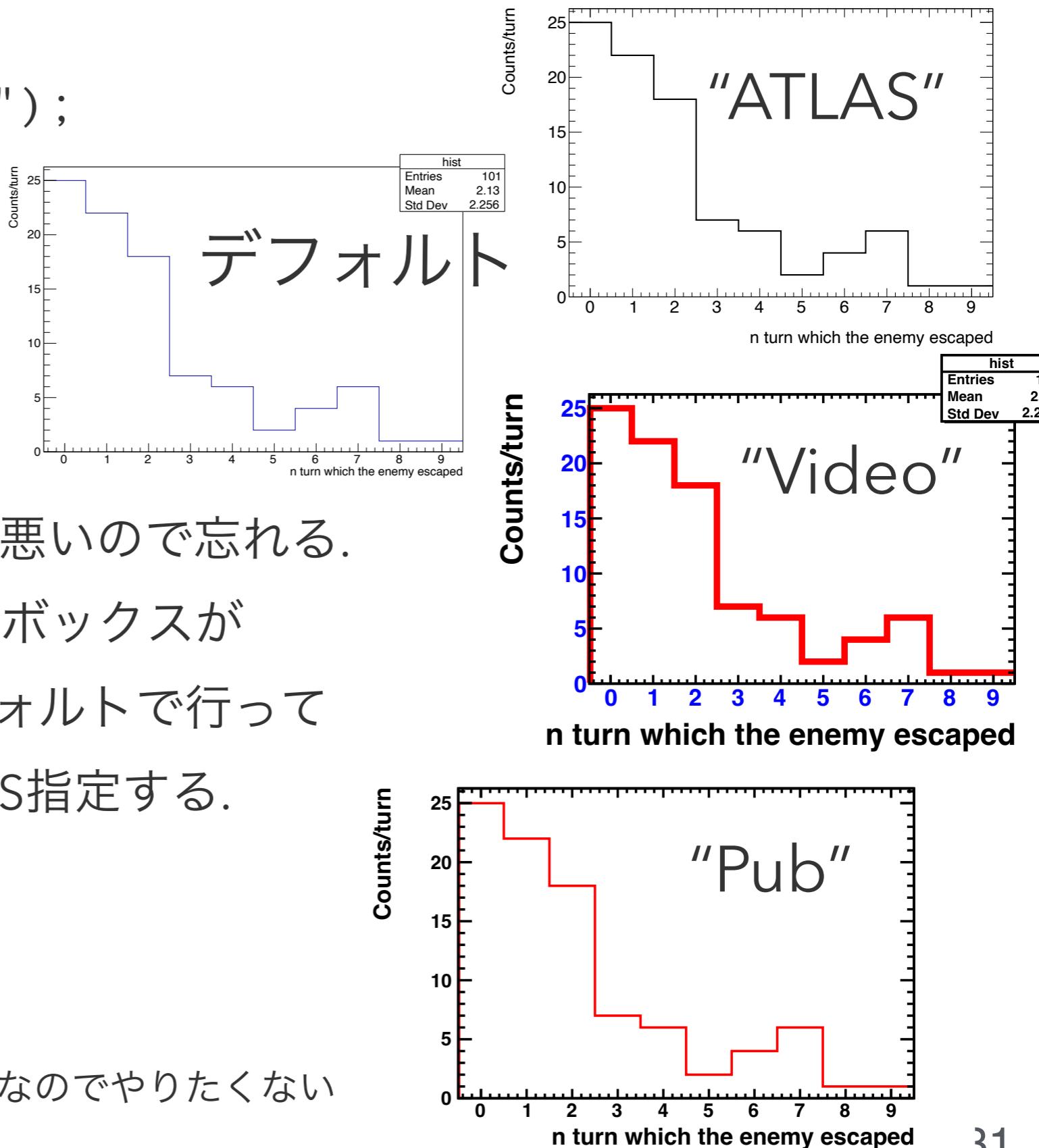


図 再掲

Styleの一括変更

- `gROOT->SetStyle("ATLAS");`
- 関数の上の方でするとStyleを一括で変えることができる.
- デフォルトはスライドに貼るには文字が小さすぎる
- 他にもStyleがあるが,センスが悪いので忘れる.
- ただ, ATLASを指定すると統計ボックスが表示されないので,解析はデフォルトで行って人に見せる図を出す時にATLAS指定する.
- ROOTの惜しいところ



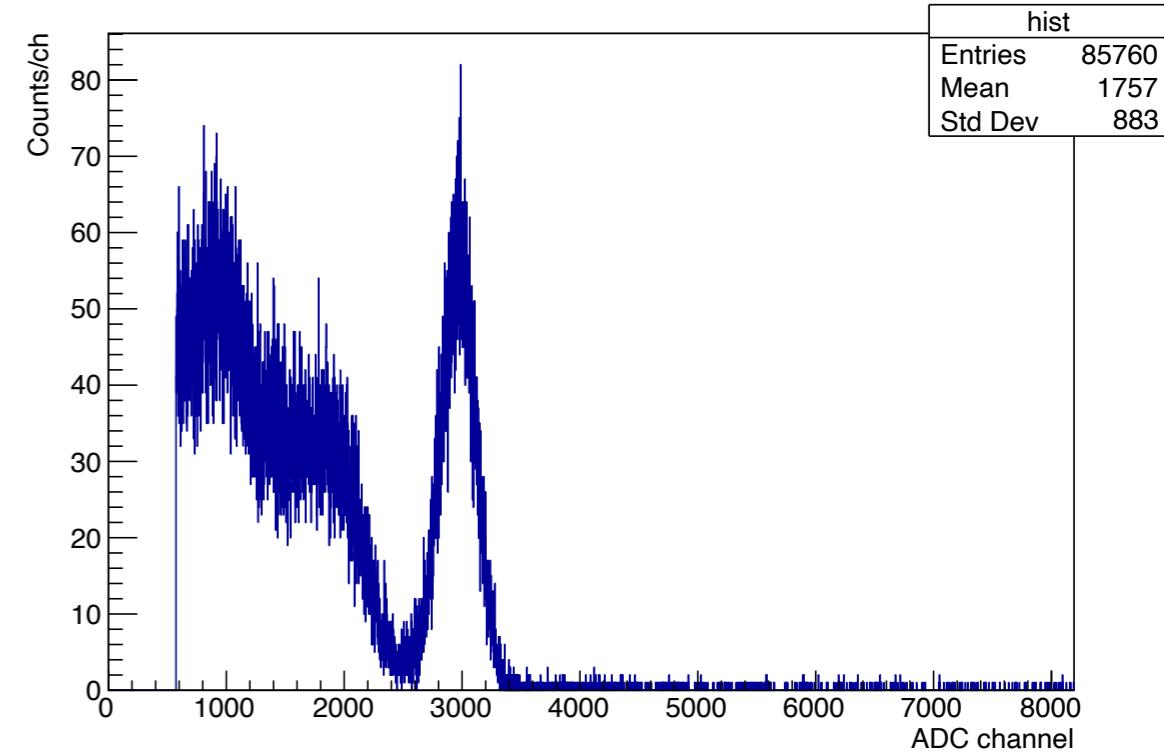
*各々の要素を細かく設定できるが,面倒なのでやりたくない

もう少し物理の実験らしい状況で練習

- CsIシンチレータとPINフォト

ダイオードでCs-137 γ 線源をあてた
データを取得した。

- ADC値の分布が得られた。
- ADC値の範囲 (0–8191)
- データの場所: root_lecture/macros/
mzks/scintillator_exp/data/data1.txt
- さっきの例を応用して自力で
ヒストグラムを表示するマクロをかく
 - 時間とります。



- hagure_expディレクトリから
scintillator_expディレクトリに移動
- macrosディレクトリをつくって
さっきの内容を参考にマクロをかく
- データを念のため目で確認
- Bin数や最大, 最小の値に注意

適切なBin幅

- Bin幅が細かすぎて上下に振れてみづらいので
Bin幅を広げたい
- やり方1: ヒストグラムを作る時にBin数を小さく
設定する.
- やり方2: クラスのメソッド, Rebin()をつかって周
囲のBinをそれぞれまとめる.

```
24
25 hist->Rebin(8); //これで8Binをあたらしい1Binに
26
27 hist->SetTitle(";ADC channel;Counts/ch");
28 hist->Draw();
```

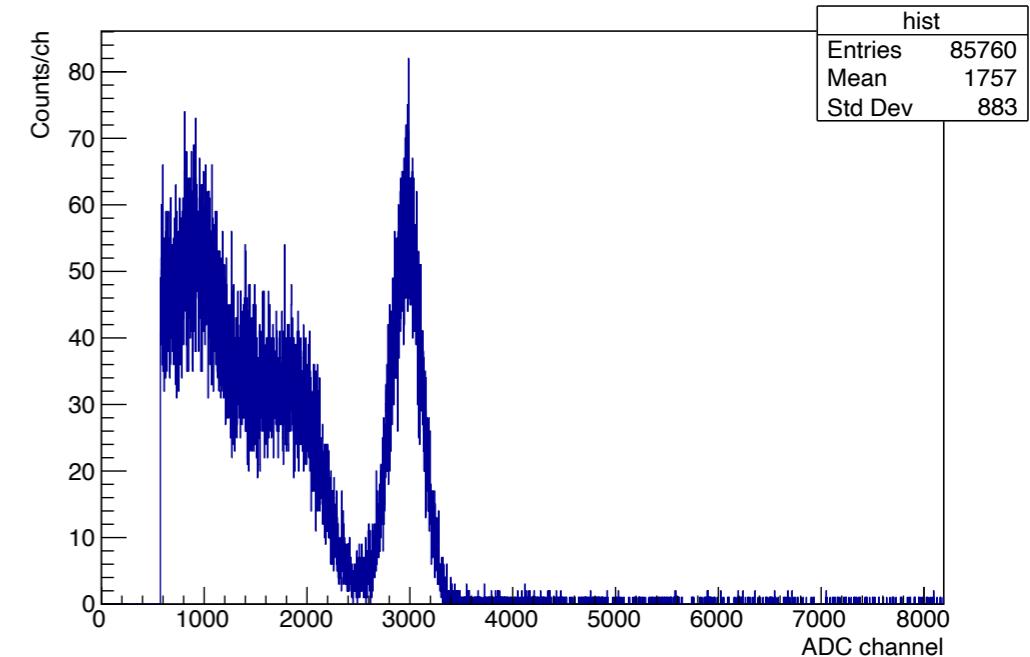
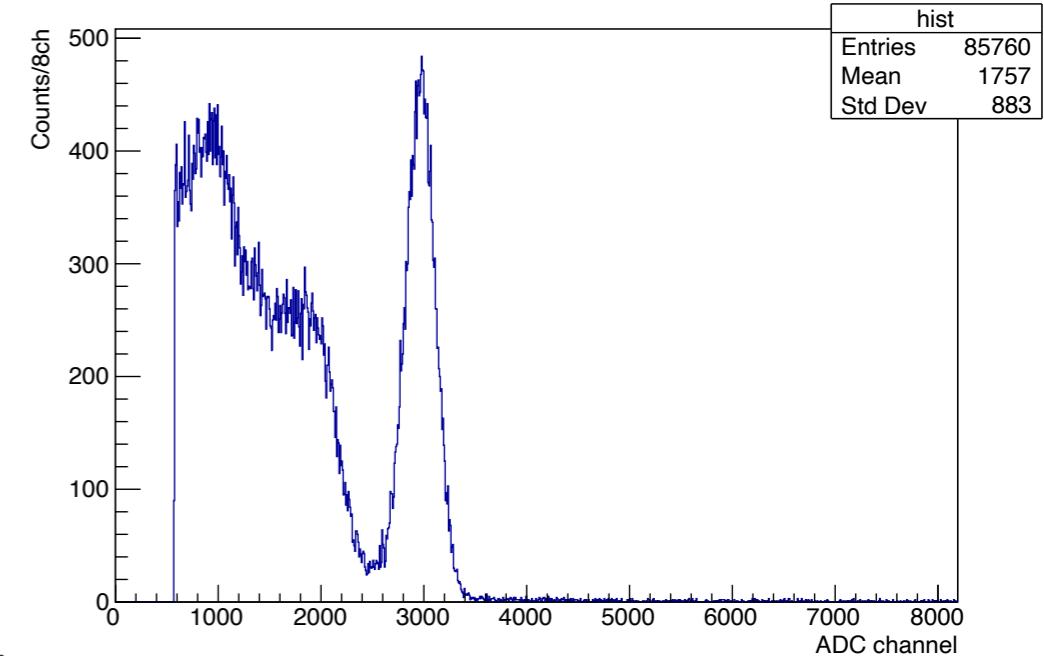


図 再掲



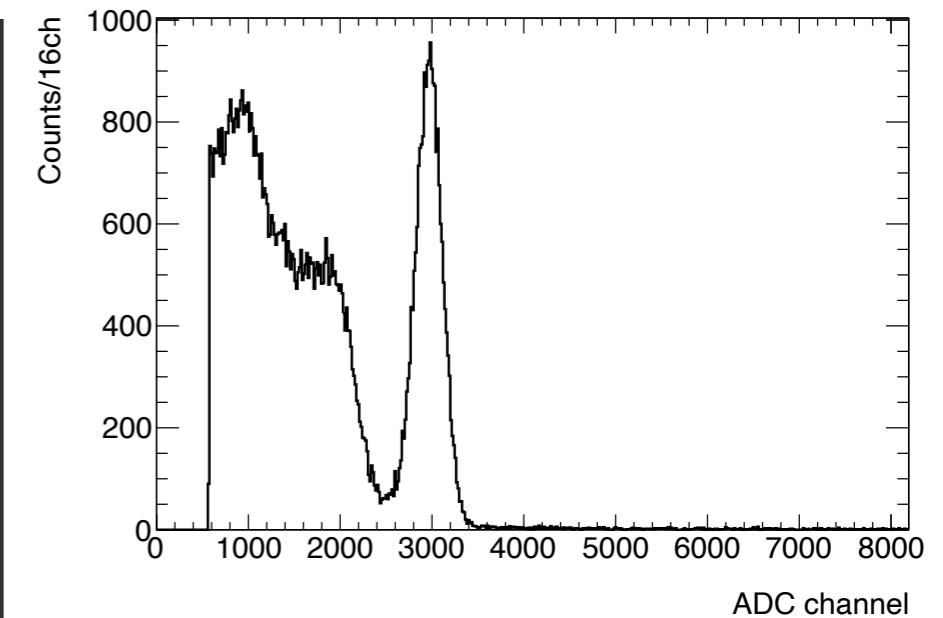
- どんな状況でも使えるBin幅決定アルゴリズムはない.
- 解析する人のセンスが要求される.

Rebin(8)したヒストグラム

マクロ例

```

1 // -----
2 // draw_hist2.C -- macro
3 // Author: K. Mizukoshi
4 // Date : May 12 2020
5 // -----
6
7 int draw_hist2(){
8
9   gROOT->SetStyle("ATLAS");
10
11 // Define Range and Num of bin of histogram
12 const int MCACh = 8192;
13 const double HistMin = 0.;
14 const double HistMax = 8192.;
15
16 TH1D* hist = new TH1D("hist", "hist", MCACh, HistMin-0.5, HistMax-0.5);
17
18 double BufferValue;
19 ifstream ifs("../data/data1.txt");
20 while(ifs >> BufferValue){
21   hist->Fill(BufferValue);
22 }
23
24 hist->Rebin(16);
25
26 hist->SetTitle(";ADC channel;Counts/16ch");
27 hist->Draw();
28
29 return 0;
30 }
```



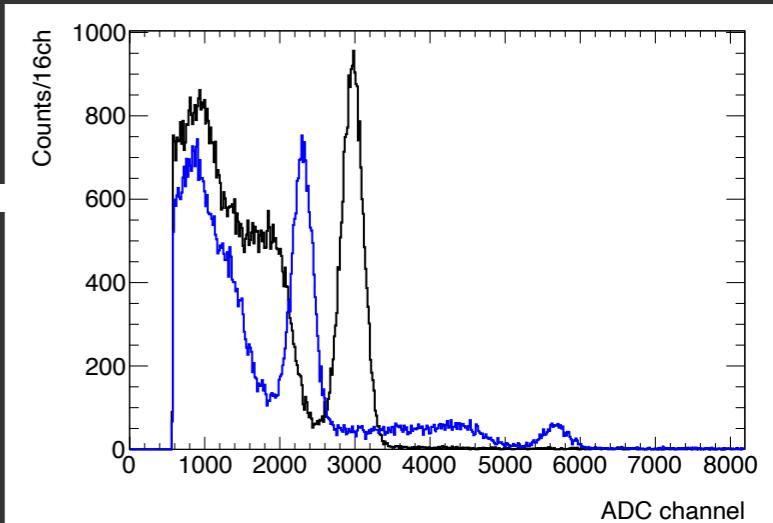
ヒストグラム重ね書き

- data1.txtとdata2.txtのヒストグラムを一つの図に表示する.
- 2つヒストグラムをつくり,1つ目は普通にDraw(), 2つ目はDraw("same");とする.
- 線が同じ色で表示されてしまうので, SetLineColor(kBlue);などをつかってどちらかの線の色を変える.
 - kWhite, kBlack, kGray, kRed, kGreen, kBlue, kYellow, kMagenta, kCyan, kOrange, kSpring, kTeal, kAzure, kViolet, kPinkなどがつかえる
 - ""でくくる必要がないことに注意

```
30     hist1->SetTitle("ADC channel;Counts/16ch");
31     hist1->Draw();
32     hist2->SetLineColor(kBlue);
33     hist2->Draw("same");
```

マクロ例(あまりよくない方法)

```
// draw_hists_not_good.C
7 int draw_hists_not_good(){
8
9 gROOT->SetStyle("ATLAS");
10 const int MCACH = 8192;
11 const double HistMin = 0.;
12 const double HistMax = 8192.;
13
14 TH1D* hist1 = new TH1D("hist1", "hist1" , MCACH, HistMin-0.5, HistMax-0.5);
15 TH1D* hist2 = new TH1D("hist2", "hist2" , MCACH, HistMin-0.5, HistMax-0.5);
16
17 double BufferValue1;
18 ifstream ifs1("../data/data1.txt");
19 while(ifs1 >> BufferValue1){
20     hist1->Fill(BufferValue1);
21 }
22 double BufferValue2;
23 ifstream ifs2("../data/data2.txt");
24 while(ifs2 >> BufferValue2){
25     hist2->Fill(BufferValue2);
26 }
27
28 hist1->Rebin(16);
29 hist2->Rebin(16);
30 hist1->SetTitle(";ADC channel;Counts/16ch");
31 hist1->Draw();
32 hist2->SetLineColor(kBlue);
33 hist2->Draw("same");
34 return 0;
35 }
```

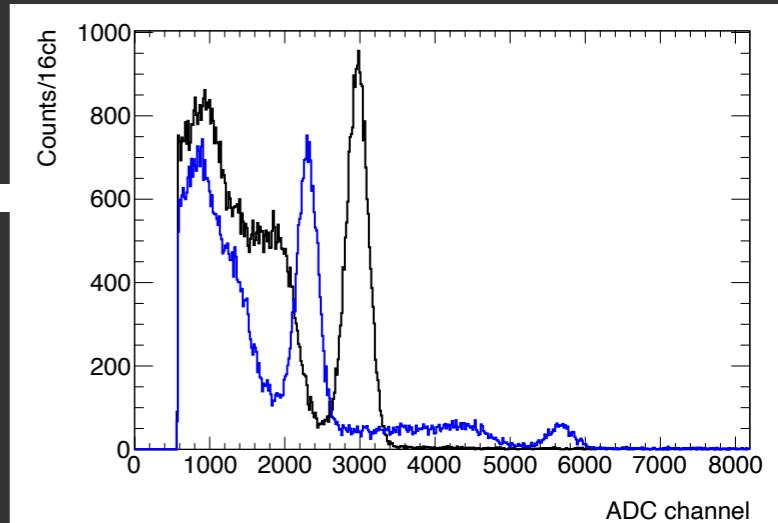


普通に2個ヒストグラムを作り,それぞれ描画する方法
定義, 値詰め, Rebin, Drawをそれぞれ2回ずつ行い,
最後にDraw(), Draw("same")する.

これでも十分動くし,図のクオリティも変わらないが,
データがたくさんになってくることを見据えると
次のページの方法を推奨する.

よいマクロ例

```
// draw_hists.C
7 TH1D* make_hist(TString filename = "data1"){
8
9     gROOT->SetStyle("ATLAS");
10    const int MCACh = 8192;
11    const double HistMin = 0.;
12    const double HistMax = 8192.;
13
14    TH1D* hist = new TH1D(filename, filename, MCACh, HistMin-0.5, HistMax-0.5);
15
16    const TString data_dir = "../data/";
17    double BufferValue;
18    ifstream ifs(data_dir + filename + ".txt");
19    while(ifs >> BufferValue){
20        hist->Fill(BufferValue);
21    }
22
23    hist->Rebin(16);
24    hist->SetTitle(";ADC channel;Counts/16ch");
25    //hist->Draw();
26    return hist;
27 }
28
29 int draw_hists(){
30
31    TH1D* hist1 = make_hist("data1");
32    TH1D* hist2 = make_hist("data2");
33
34    hist1->Draw();
35    hist2->SetLineColor(kBlue);
36    hist2->Draw("same");
37
38    return 0;
39 }
```



さっき作ったdraw_hist2.Cを関数にして再利用
引数でファイルの名前を与える様にする.
最後に,できたヒストグラムをreturnして,
外からヒストグラムが利用できる様にする

渡した先にDrawしてもらうのでコメントアウト

ここでmake_hist()関数から返されたヒストグラムを受け取る.
今は手間が増えたと思うかもしれないが,こうしておくとどれだけ
データファイルが増えても数行たすだけで実現できる.
また,ヒストグラムを作るところと,絵を描くところを構造的に
ロジックに従って分離させることができた.

SetLineColor()で線の色を変えて表示する.
Draw("same")で前のヒストグラムを消さずに
重ねがきできる

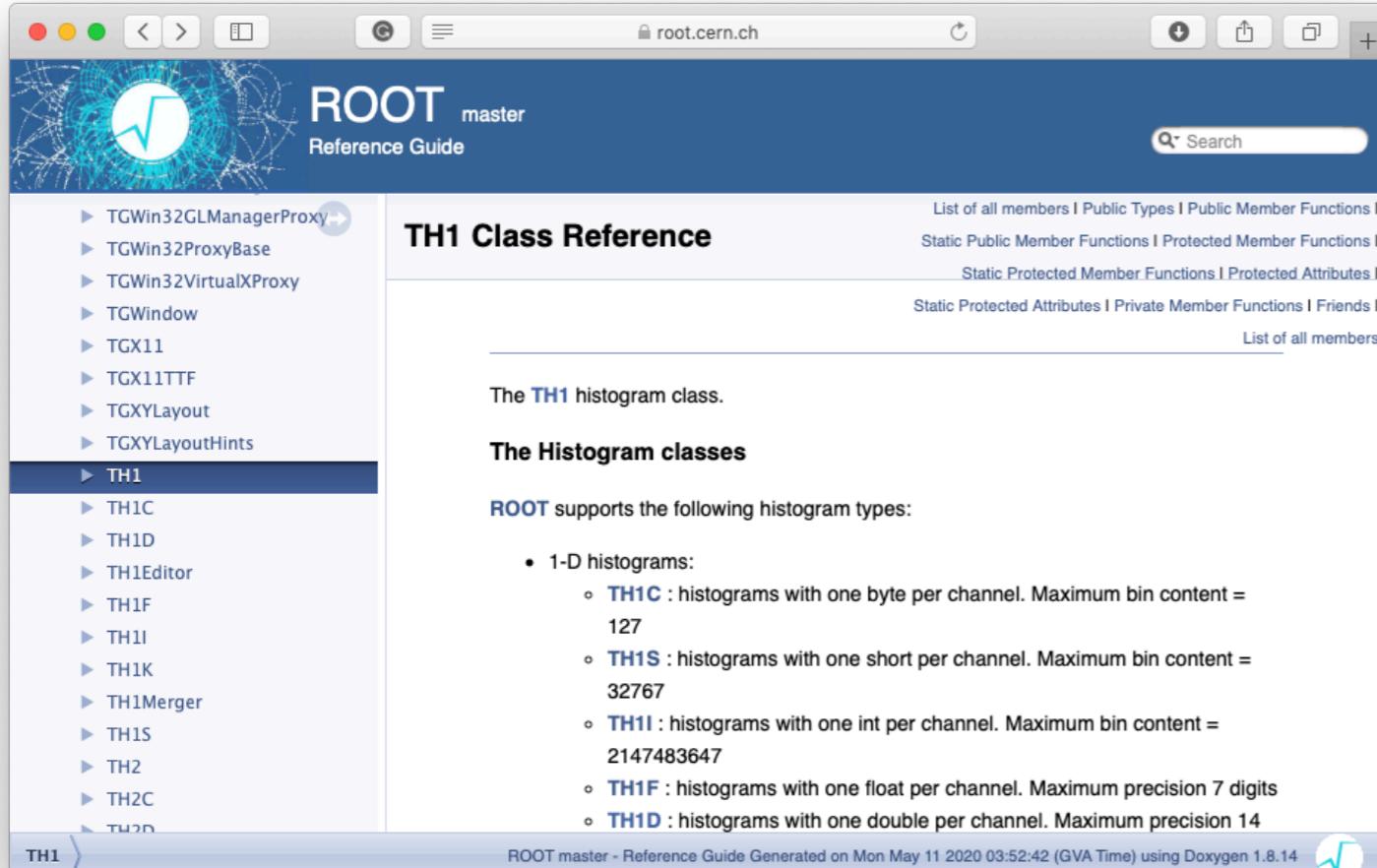
メソッドでヒストグラムのパラメータを取得する

```
1   cout << "Sample Size      : " << hist->GetEntries() << endl;
2   cout << "Maximum Value   : " << hist->GetMaximum() << endl;
3   cout << "Maximum Bin     : " << hist->GetMaximumBin() << endl;
4   cout << "Mean             : " << hist->GetMean() << endl;
5   cout << "Standard Dev    : " << hist->GetStdDev() << endl;
6   cout << "Mean Error       : " << hist->GetMeanError() << endl;
7
8   cout << "7th Bin content: " << hist->GetBinContent(7) << endl;
```

- お絵かきするだけではない便利なメソッドがある。
- Sample SizeやMeanをメソッド一発で取得できる。
- GetBinContent()で
Binの値そのものを得る
- SetBinContent()もできる

```
> root -l use_methods.C
Tue May 12 09:19:03 2020
root [0]
Processing use_methods.C...
Sample Size      : 39728
Maximum Value   : 224
Maximum Bin     : 731
Mean             : 3127.93
Standard Dev    : 1791.8
Mean Error       : 8.98962
7th Bin content: 0
Info in <TCanvas::MakeDefCanvas>: created default TCanvas
with name c1
(int) 0
root [1]
```

クラスリファレンスを見る



ROOT公式リファレンス
<https://root.cern/doc/master/index.html>

最初にここを見る!

- クラスの使い方を知りたい時は公式クラスリファレンスを参照する。
- 適切なクラスを探したり、クラスのメソッドの使い方を探したりできる。
- 微妙に痒いところに手が届かないで、慣れてきたらソースコードを見るようになるかも

マクロ内で図として保存

- `hist->SaveAs("../figures/my_great_figure.pdf");`
- これでpdfとして図を保存できる。
- ROOTのメニューからSave...としてもできるが、面倒なのでマクロを実行した際に自動で保存されるようとする
- マクロと同じディレクトリに保存するとごちゃごちゃしてくるので別のディレクトリを作ってそちらに保存することを推奨。

よく見るリンク集

- ROOT Users Guide
 - 公式のユーザーガイド 困ったらここを見る.
 - <https://root.cern.ch/guides/users-guide>
- ROOT Forum
 - フォーラム 質問してもいいし,大抵誰かが以前に同様の質問をしている
 - <https://root-forum.cern.ch>
- KamonoWiki (日本語)
 - 解析に必要なポイントについて詳細に纏まっている
 - <https://www-he.scphys.kyoto-u.ac.jp/member/n.kamo/wiki/doku.php?id=study:software:root>
- KumaROOT (日本語)
 - 話題が幅ひろい チュートリアルとしておすすめ
 - <https://kumaroot.readthedocs.io/ja/latest/index.html>
- ROOT : 便利なコマンド表と使い方の例 (日本語)
 - ちょっとしたことのやり方をわすれたときに
 - <http://atlas.kek.jp/comp/ROOT-commands.html>

Day 2 宿題

- あなた自身で設定したテーマでデータを収集し,
ROOTを使ってヒストグラムを描いて考察してください.
- テーマは物理に限りません.
- 形式はなんでもいい(レポート, スライド, その他)ので, slackの
#homeworkにアップロードしてください.
- 最低1枚はROOTで書いた図, ソースコード を載せてください.
- Day 3 はFitting, 検定, グラフなので, 集めたデータを
実際に解析することもできるようになるとおもいます.

- 今日の宿題を2023年6月10日までにSlackの #homework チャンネルにアップロードした人から 優秀な方を表彰します。(豪華副賞)
- 昨年度までの秀逸なテーマの例
PEANUTSの主人公, 抜け毛, いろはすの飲み残し, あつもり魚釣,
Switch転売価格, 人工生命体, 地震震度比較, 天気,
相席食堂(ABCテレビ)での大悟およびノブの「ちょっと待ていボタン」
- おおまかな評価基準
 - オリジナリティ ○図の取り扱いが正しいか
 - がんばってデータを集めたか ○わかりやすい資料, コードかどうか

おわりに

- 今回までの3回で水越担当分終了
 - おつかれさまでした.
 - Slack/その他でこれからもサポートしています.
 - おもしろい宿題おまちしています.
- YMAPへの参加もお待ちしております.
 - <http://www.icrr.u-tokyo.ac.jp/YMAP/join.html>
- 本講習会は100%ボランティアで研究時間を削って開催されています。
志のある方は来年度,サポート側にまわって我々を
助けてくださるとありがとうございます.
 - 講師も募集しています!!!