第3回 ROOT講習会 資料
- TF1 と TTree -

細川佳志（YMAP, 東北大学 RCNS）

27th May, 2021

YMAP
Youth Meeting for AstroParticle

自己紹介

- 細川 佳志（東北大学 ニュートリノ科学研究センター）
- カムランド, カムランド禅@神岡 などに参加
  - ニュートリノを伴わない二重ベータ崩壊探索実験
    ニュートリノ検出，暗黒物質探索など
- 暗黒物質探索実験 XMASS@神岡 で博士号取得
- まだ若い。水越さん同様，気軽になんでも聞いてください

はじめに

- わからないことがあれば，**講義を遮って良いのですぐ質問**してください。
  zoomの「手を挙げる」機能や，slackへの書き込みでもよいです。
- 簡単すぎる場合は途中退室してもよいです。
  わからない/ついてこれていない場合は質問してください。

- 本資料は，ROOTのClass Referenceを参考にしています。
  - https://root.cern.ch/doc/master/classTF1.html
  - https://root.cern.ch/doc/master/classTTree.html

- 2019年度までの講師 奥村さんの資料も有用です。
  - https://github.com/akira-okumura/RHEA

準備

- コード・データの最新版をダウンロードする

```
[keishi@mac ~] $ cd ROOT2021   ←前回までででgit cloneして作ったdirectory
[keishi@mac ~] $ git pull
```

- これでダメなら,

```
[keishi@mac ~] $ cd [ROOT2021を含むdirectory]
[keishi@mac ~] $ mv ROOT2021 ROOT2021_day2
[keishi@mac ~] $ git clone https://github.com/ymap-team/ROOT2021.git
```

- ROOT2021のhskw directoryに移動する

```
[keishi@mac ~] $ cd ROOT2021/hskw
```

YMAP
Youth Meeting for AstroParticle

第3回講習会の到達目標

- **TF1**
  - TF1で関数を自作/描画できるようになる
  - 定義したTF1関数で，TH1のフィッティングができるようになる

- **TTree**
  - 概念をざっくり理解する
  - TTreeを自分で作って，保存できるようになる
  - 保存したTTreeを開いて，簡単なお絵描きができるようになる

- **(統計は各々教科書読んで勉強してください)**
  - https://pdg.lbl.gov/2020/reviews/rpp2020-rev-statistics.pdf **無料**
  - John R. Taylor『計測における誤差解析入門』**日本語，わかりやすい**

この資料以外にもPDGは便利

MAP Youth Meeting for AstroParticle

TF1

- 1次元の関数を定義できる
  - TFormulaであらかじめ定義された関数
    - gaus, expo, pol1, pol2, pol3 など
  - 変数xに加えて，パラメータを含めた関数を自分で定義可能
  - C, C++の関数，オブジェクト関数の使用
  - ROOT6からはラムダ式も使えるらしい（使ったことないが）

- 定義した関数を用いて，1次元ヒストグラムをフィットできる

TF1を使ってみよう 1

下線部をターミナルに打って，Enter.

hskw/TF1/examples/func1.C

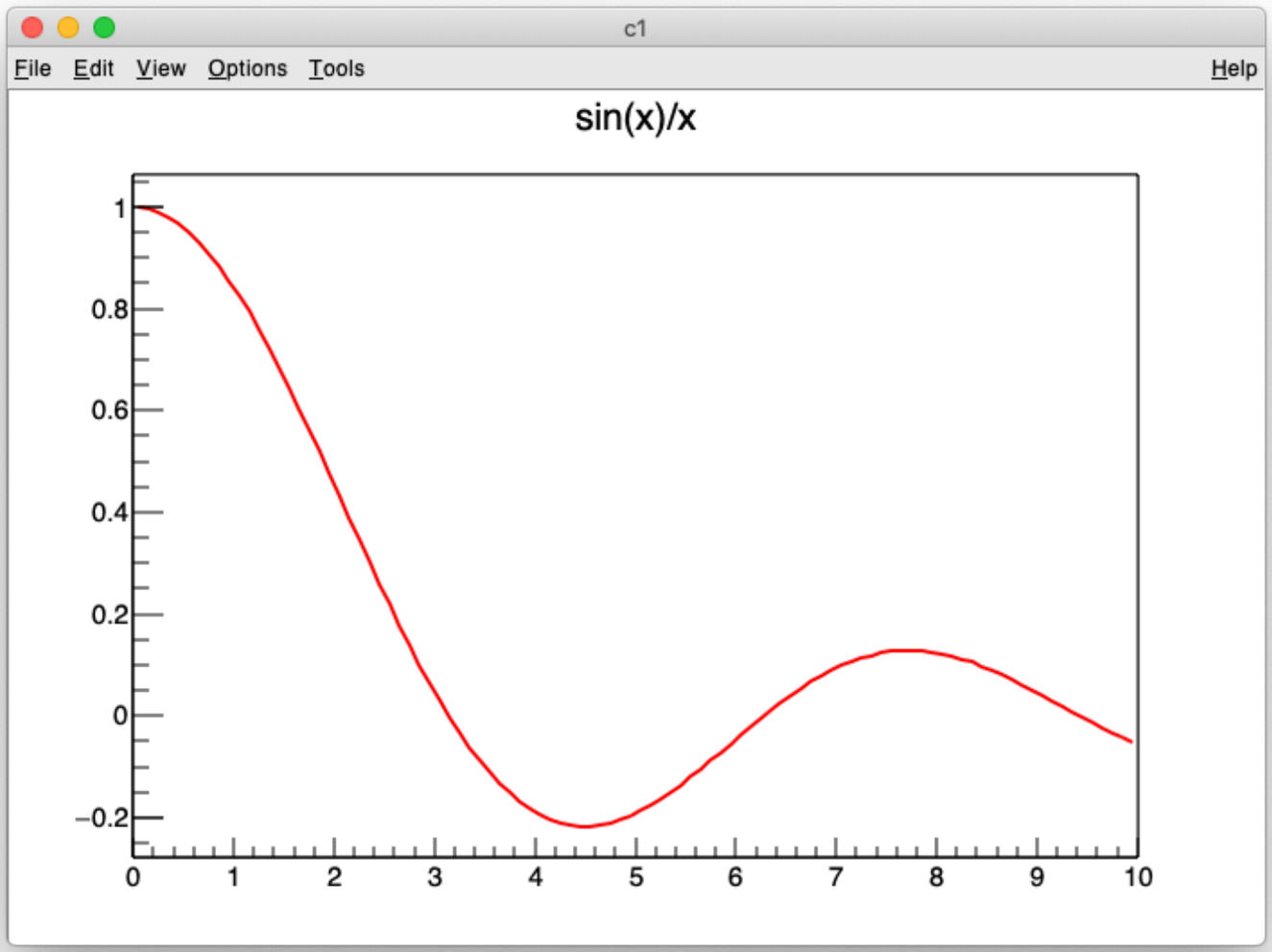
関数名　関数の定義　関数の範囲 (0~10)

```
[keishi@mac ~] $ root
root [0] TF1 *func1 = new TF1("func1", "sin(x)/x", 0, 10);
root [1] func1->Draw();
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
root [2]
```

←TF1を定義

←描画

← .q と打って，EnterでROOTを終了できます



- こんな絵が描けましたか？

TF1を使ってみよう 2

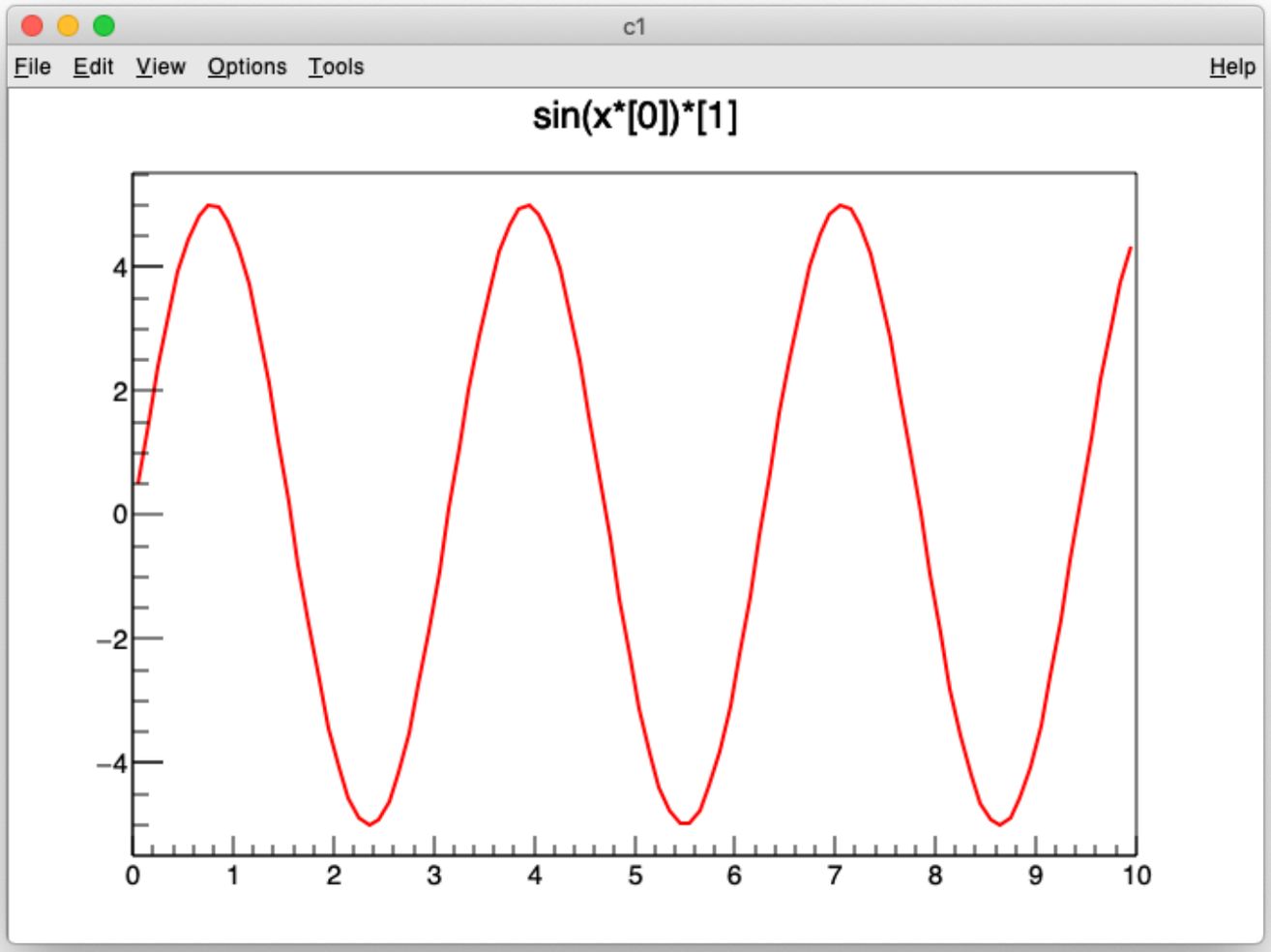
hskw/TF1/examples/func2.C

```
[keishi@mac ~] $ root
root [0] TF1 *func2 = new TF1("func2","sin(x*[0])*[1]",0,10);
root [1] func2->SetParameters(2.,5.);
root [2] func2->Draw();
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
root [3]
```

←TF1を定義

←変数を割当て

←描画



- x 以外にもパラメータを与えることができる
- SetParameters()やSetParameter()でパラメータ指定できます

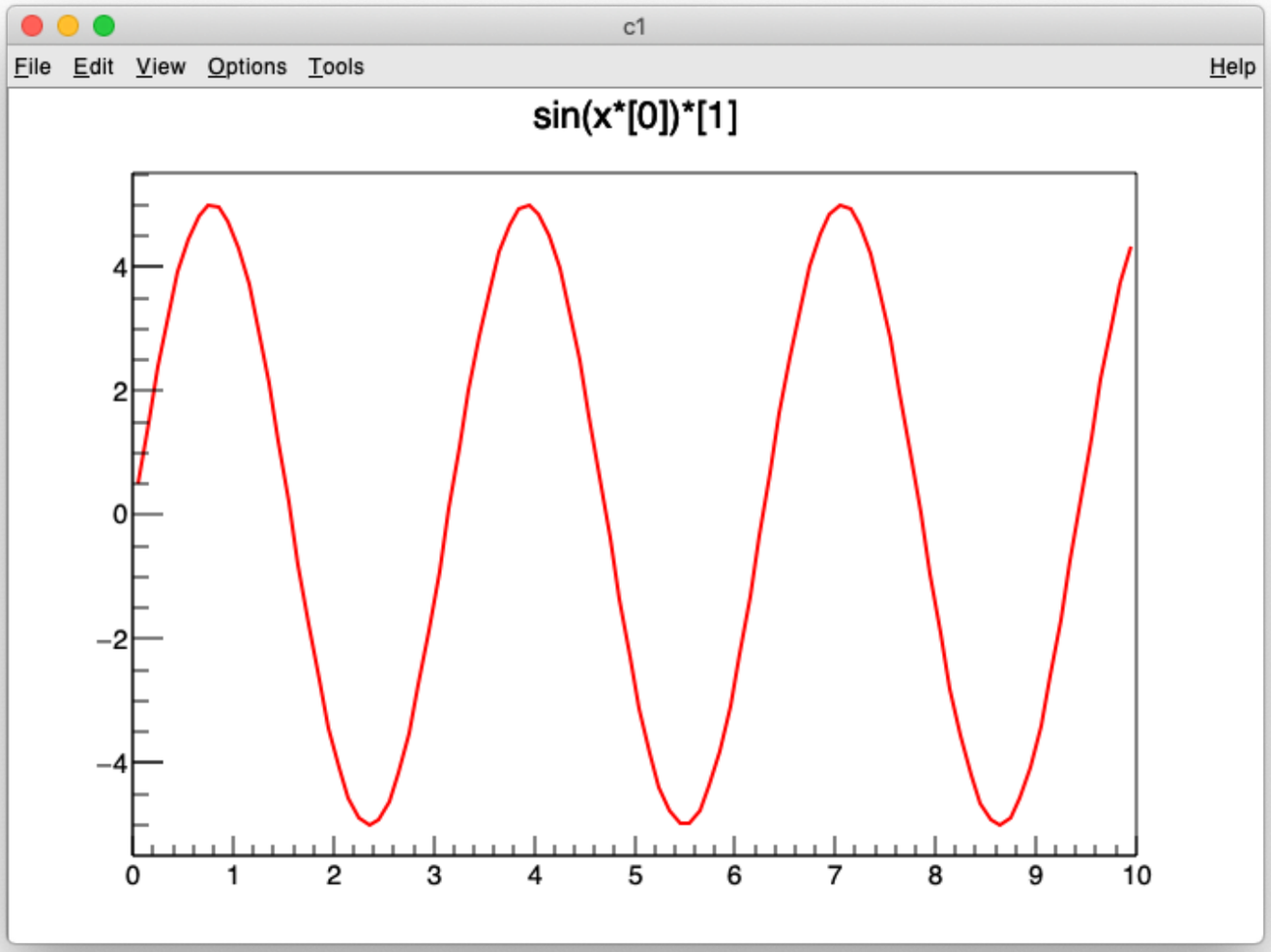
YMAP Youth Meeting for AstroParticle

TF1を使ってみよう 2

hskw/TF1/examples/func2.C

```
[keishi@mac ~] $ root
root [0] TF1 *func2 = new TF1("func2","sin(x*[0])*[1]",0,10);
root [1] func2->SetParameter(0, 2.);
root [2] func2->SetParameter(1, 5.);
root [3] func2->Draw();
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
```

←この書き方で，変数一つ一つを指定
SetParameters()はまとめて指定



- x 以外にもパラメータを与えることができる
- SetParameters()やSetParameter()でパラメータ指定できます

YMAP Youth Meeting for AstroParticle

TF1を使ってみよう 3

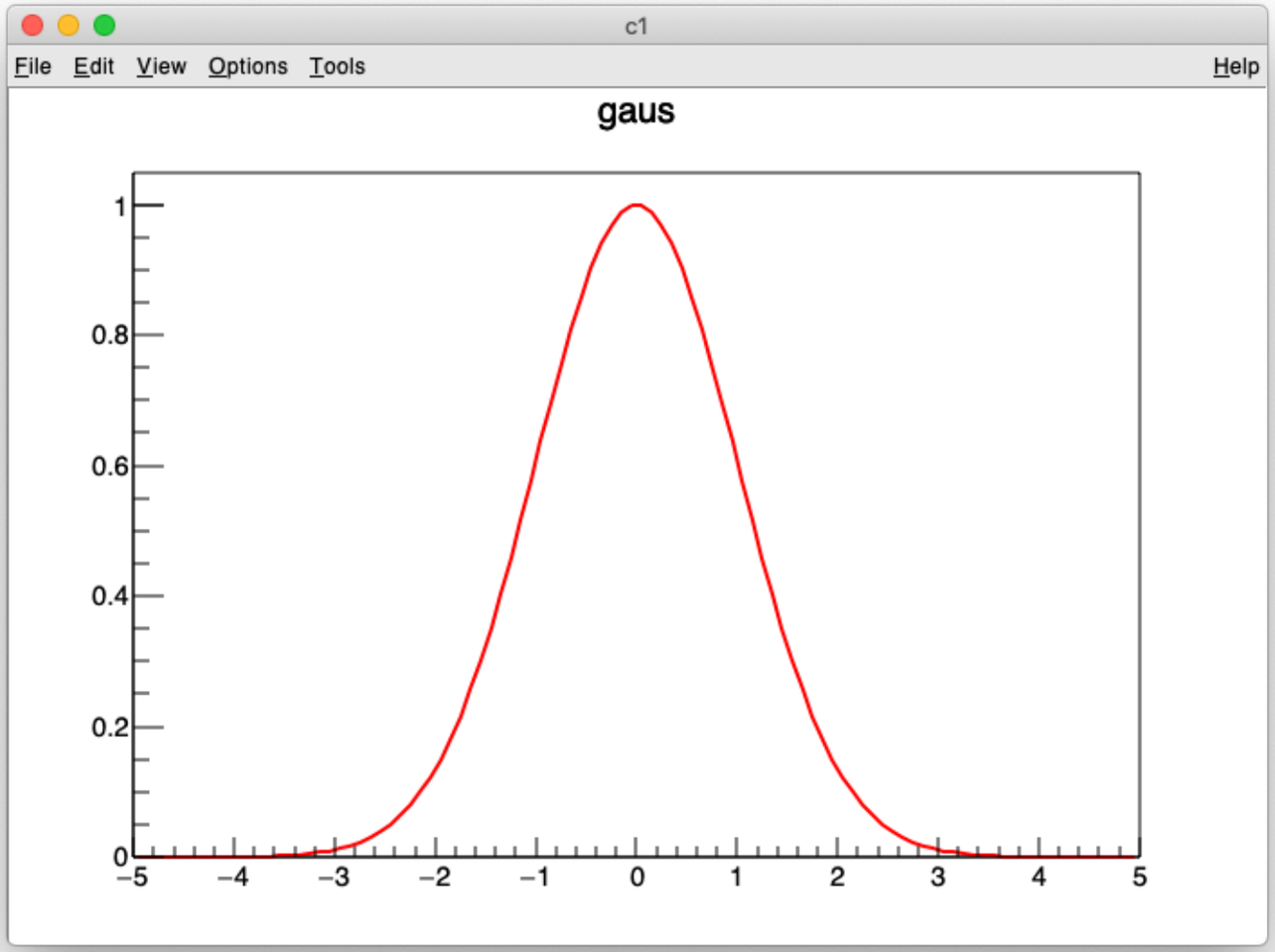
hskw/TF1/examples/func3.C

```
[keishi@mac ~] $ root
root [0] TF1 *func2 = new TF1("func2","gaus", -5, 5);
root [1] func2->SetParameters(1., 0., 1.);
root [2] func2->Draw();
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
root [3]
```

←TF1を定義

←変数を割当て

←描画



- TFoumulaが定義した式も使える
  - gaus, expo, pol1など
- SetParameters();
  で，パラメータを指定できます
- この場合だと，TFoumulaで

$$\mathrm{gaus} = [0] \times \exp\left(-0.5 \times \left\{\frac{x-[1]}{[2]}\right\}^2\right)$$

と定義されているので，
SetParameters(高さ，中心，幅)

YMAP Youth Meeting for AstroParticle

TF1を使ってみよう 4

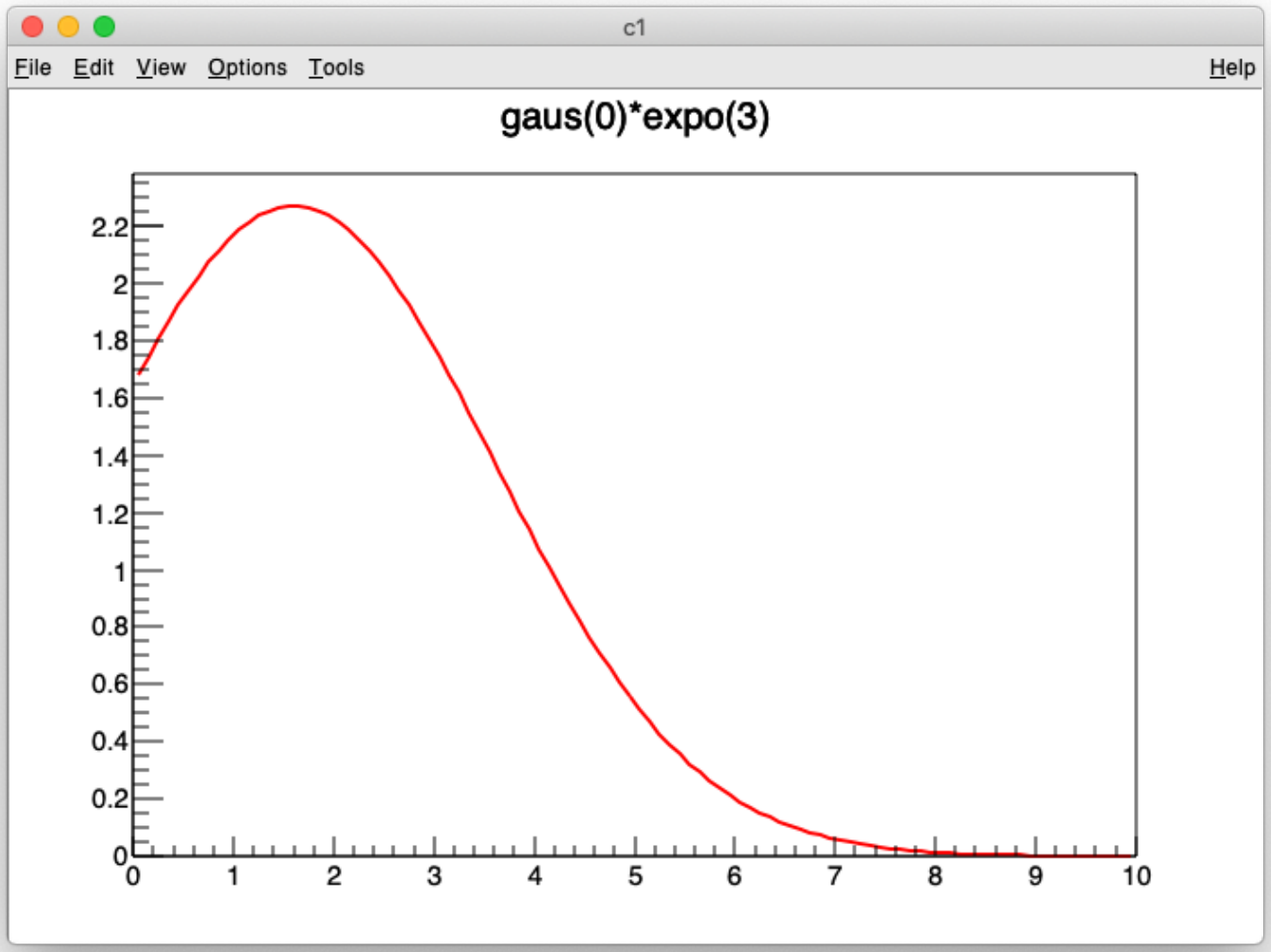
hskw/TF1/examples/func4.C

```
[keishi@mac ~] $ root
root [0] TF1 *func3 = new TF1("func3", "gaus(0)*expo(3)", 0, 10);
root [1] func3->SetParameters(1., 2., 2., 1., -0.1);
root [2] func3->Draw();
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
root [3]
```

←TF1を定義

←変数を割当て

←描画



- このように，複数の TFoumulaの組合せも可能
- ここで,

$$\mathrm{gaus} = [0] \times \exp\left(-0.5 \times \left\{\frac{x - [1]}{[2]}\right\}^2\right)$$

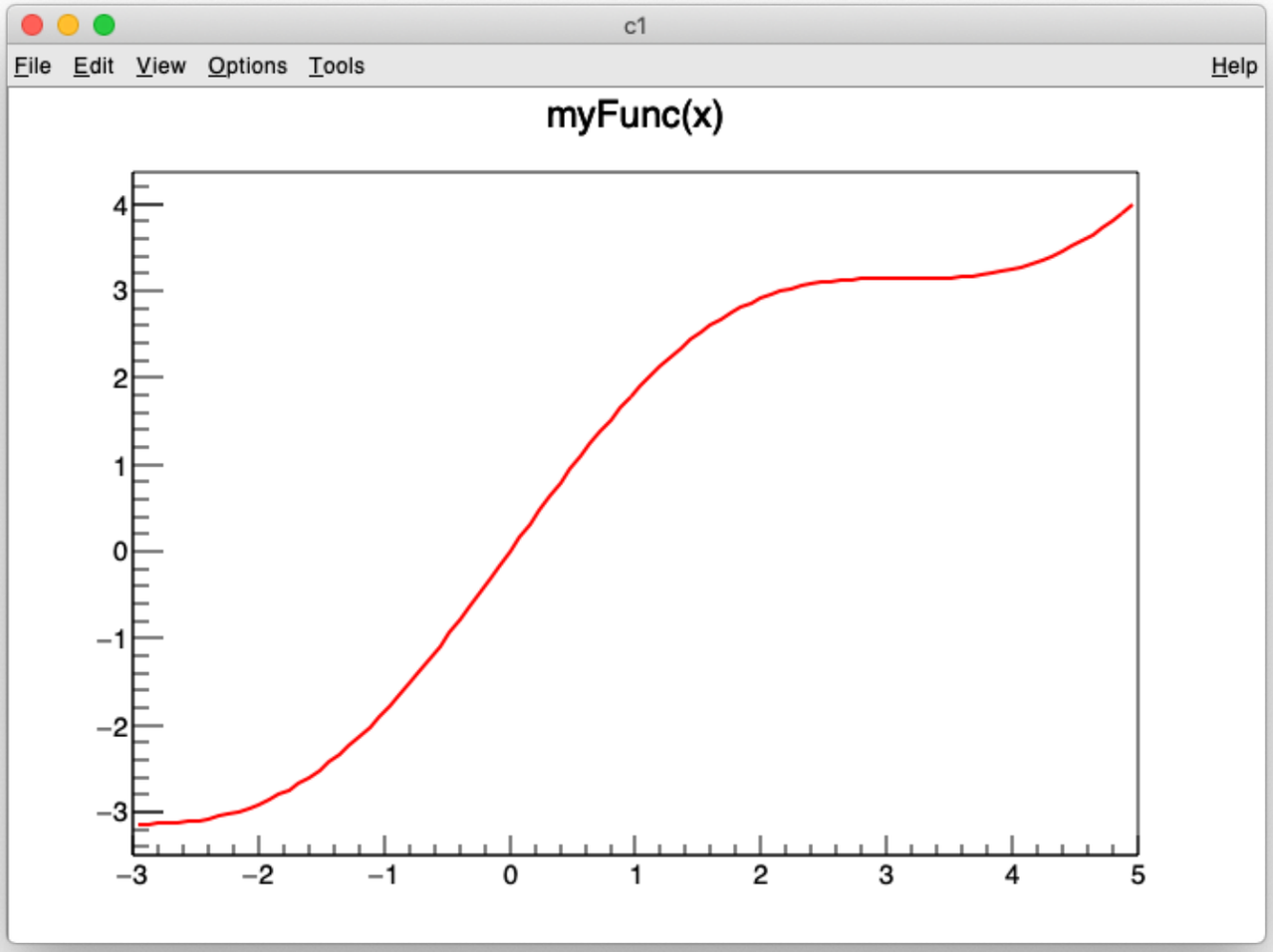
$$\mathrm{expo} = \exp\left([3] + [4] \times x\right)$$

TF1を使ってみよう 5

hskw/TF1/examples/func5.C

```
[keishi@mac ~] $ root
root [0] double myFunc(double x) { return x+sin(x); }
root [1] TF1 *func5 = new TF1("func5","myFunc(x)",-3,5);
root [2] func5->Draw();
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
root [3]
```

←TF1を定義



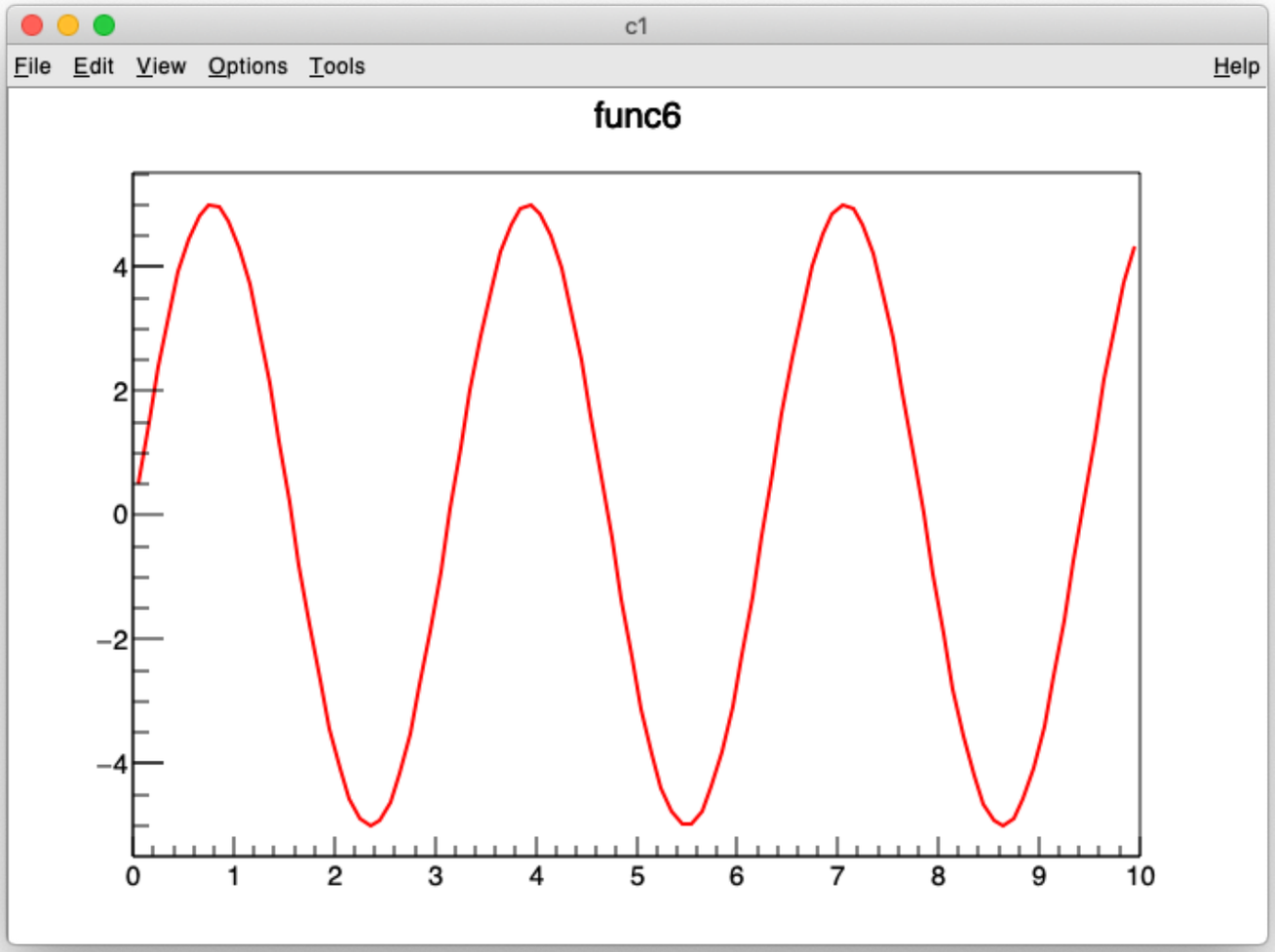
- こんな感じで，Cの関数機能を使う方法もある

TF1を使ってみよう 6

hskw/TF1/examples/func6.C

```
[keishi@mac macros] $ root
root [0] double myFunc(double *x, double *par) { return sin(x[0]*par[0])*par[1]; }
root [1] TF1 *func6 = new TF1("func6", myFunc, 0, 10, 2);
root [2] func6->SetParameters(2., 5.);
root [3] func6->Draw();
```

↑パラメータの数

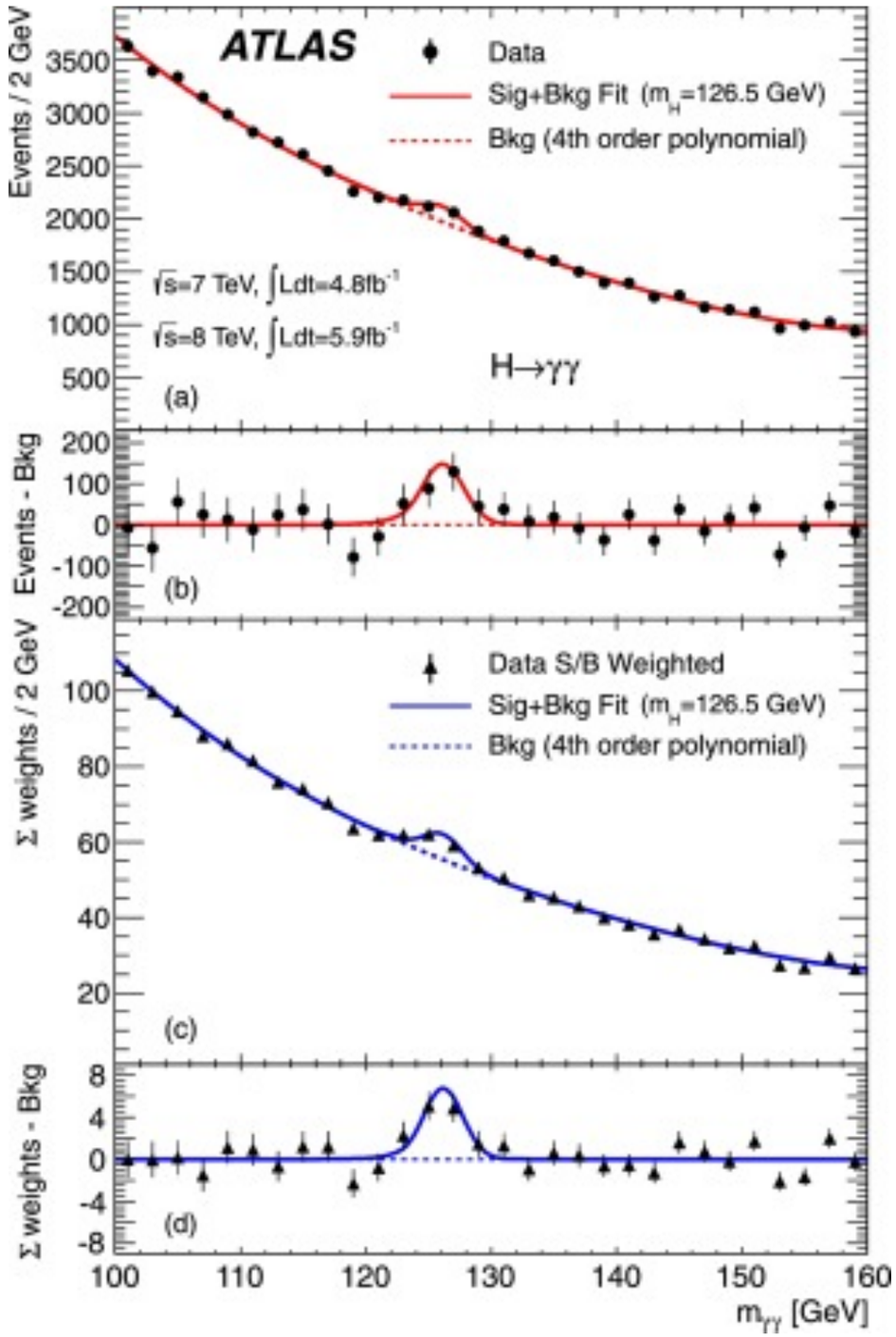


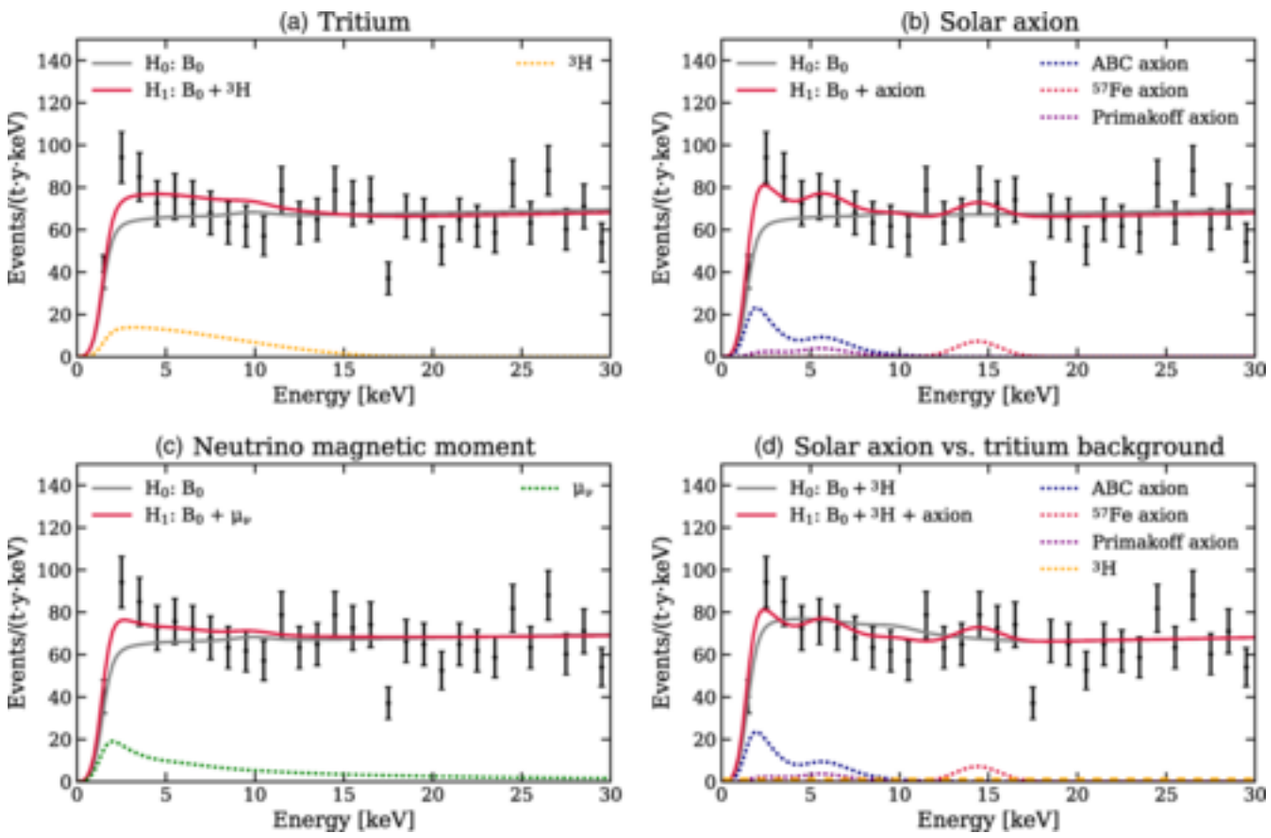
- x以外のパラメータも与えれる

YMAP Youth Meeting for AstroParticle

フィッティングとは？

- なんとなくいい感じの線を引いているだけじゃない
- カイ二乗などを計算して，最小二乗法などで誤差を評価しながらベストフィットの値を探す
- ここでは，ROOTを用いたフィット方法を紹介する

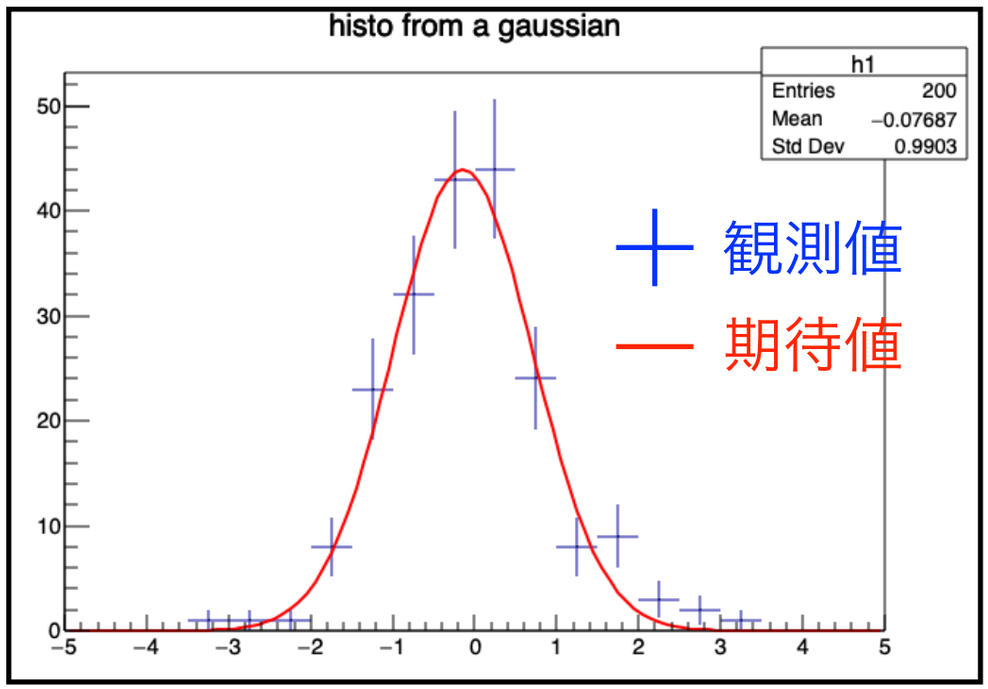




カイ二乗 フィッティング

- 測定を繰り返して，結果をn個のビンに分け，k番目のビンに入った数(観測値)を O_k とする $(k = 1, \cdots, n)$。同様に，正規分布などを仮定した時に同じくk番目のビンに入ると期待される数(期待値)を E_k とする。この時，一般的にカイ二乗は次の式で定義される。

$$\chi^2 = \sum_{k=1}^{n} \frac{\left(O_k - E_k\right)^2}{E_k}$$



(TH1*)->Fit(TF1*); などとすると，このカイ自乗が最小の値を取るパラメータ（正規分布なら，高さ・平均・幅）を探してくれる。デフォルトではカイ二乗フィットで，"L" オプションをつけるとLikelihoodフィット。さらに複雑な事をしたい場合などは，TMinuitを使えば，カイ二乗など最小化したい値を自身で定義できる。

TH1をフィットしてみよう

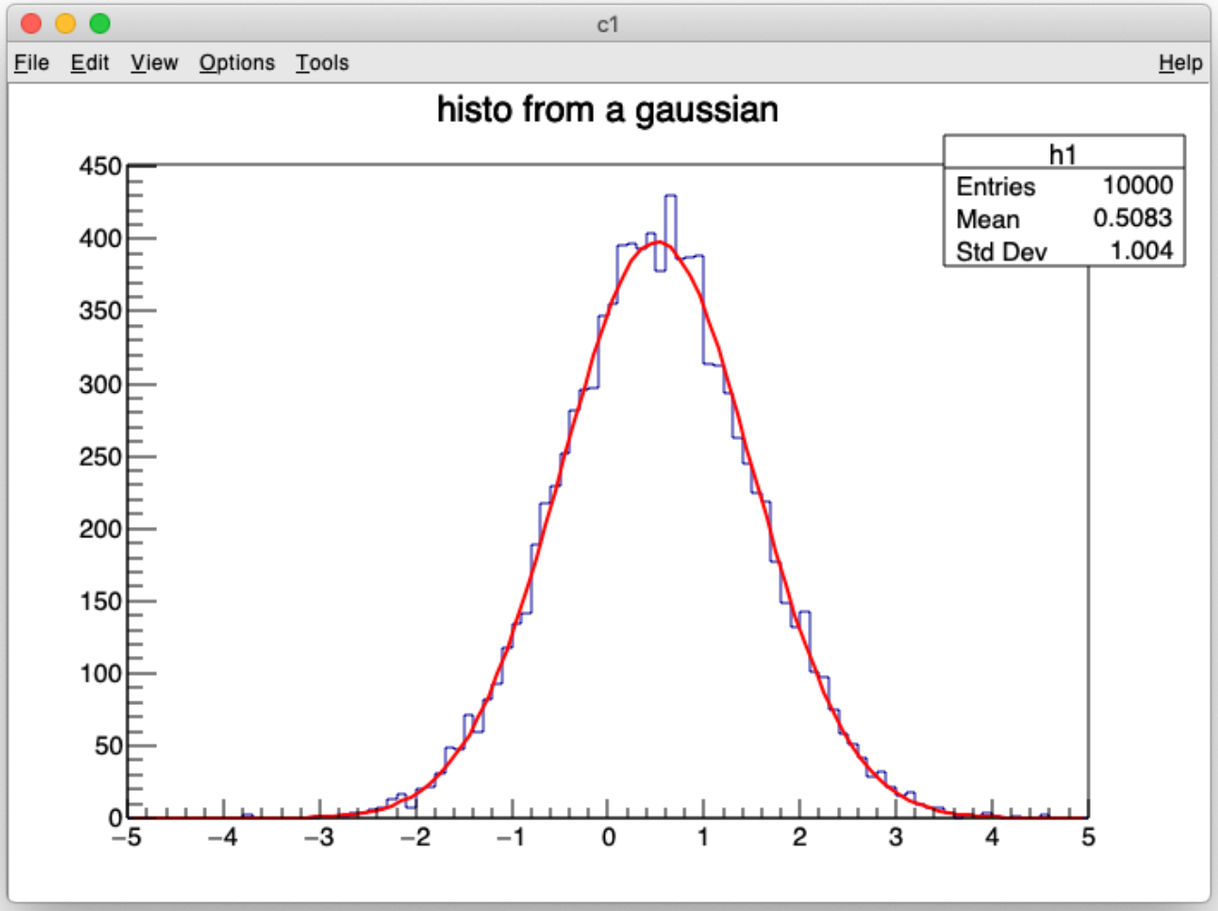
hskw/TF1/examples/fitting1.C

```
[keishi@mac macros] $ root
root [0] TH1D *h1 = new TH1D("h1", "histo from a gaussian", 100, -5, 5);
root [1] TF1 *f1 = new TF1("f1","gaus", -10, 10);
root [2] f1->SetParameters(1., 0.5, 1.);
root [3] for( int i=0; i<10000; i++ ) h1->Fill(f1->GetRandom());
root [4] h1->Fit("gaus");
（右図のようなフィット結果の値がここに出る）
root [5]
```

↓用意した正規分布からランダムに選んでh1にFill

←単純な正規分布なら，これだけでフィットできる

```
root [4] h1->Fit("gaus");
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
 FCN=57.7629 FROM MIGRAD    STATUS=CONVERGED      60 CALLS          61 TOTAL
                     EDM=3.29504e-09    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                   STEP          FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  Constant     3.97891e+02   4.93046e+00   1.49364e-02   3.65167e-06
   2  Mean         5.11388e-01   1.00492e-02   3.75730e-05   7.94987e-03
   3  Sigma        9.97455e-01   7.30906e-03   7.36429e-06   3.98602e-03
```



- TH1を使って正規分布ヒストグラムを作り，TF1でフィット
- Fillする値によって，フィット関数を変えるべき
  - 今回は，正規分布を正規分布でフィットした

YMAP Youth Meeting for AstroParticle

TH1をフィットしてみよう

hskw/TF1/examples/fitting2.C

```
[keishi@mac macros] $ root
root [0] gStyle->SetOptFit(1111111);
root [1] TF1 *f1 = new TF1("f1","gaus", 0, 20);
root [2] TF1 *f2 = new TF1("f2","expo", 0, 20);
root [3] f1->SetParameters(1., 10., 1.5);
root [4] f2->SetParameters(1., -0.1);

root [5] TH1D *h1 = new TH1D("h1", "summed up histogram", 200, 0, 20);
root [6] h1->Sumw2();
root [7] for(int i=0;i<10000;i++) h1->Fill(f1->GetRandom());
root [8] for(int i=0;i<10000;i++) h1->Fill(f2->GetRandom());

root [9] TF1 *fit1 = new TF1("f1","gaus(0)+expo(3)", 0, 20);
root [10] fit1->SetParameters(100., 8., 1., 1., -0.1);
root [11] h1->Fit(fit1);

root [12] TF1 *fit2=new TF1("fit2","[0]*exp(-0.5*pow(((x-[1])/[2]),2.))+exp([3]+[4]*x)",0,20);
root [13] fit2->SetParameters(100., 8., 1., 1., -0.1);
root [14] h1->Fit(fit2);
```

←結果の絵にフィット値を表示するためのおまじない

←パラメータを指定

←histogramに統計誤差のエラーバーを付ける

←ふたつのTF1からランダムにFill

←用意した関数でフィット

←なんとなくそれっぽい値をセット
初期値の設定は必ず行う
プレフィットして持ってきたりもする

←9-11行目の代わりにこんな自作の関数
でもフィット可能。書き方が違うだけで
中身は同じなので結果も同じ

TH1をフィットしてみよう

hskw/TF1/examples/fitting2.C

```
[keishi@mac macros] $ root
root [0] gStyle->SetOptFit(1111111);
root [1] TF1 *f1
root [2] TF1 *f2
root [3] f1->Set
root [4] f2->Set

root [5] TH1D
root [6] h1->Su
root [7] for(int
root [8] for(int

root [9] TF1 *fit
root [10] fit1->
root [11] h1->F

root [12] TF1 *fit2=new TF1("fit2","[0]*exp(-0.5*pow(((x-[1])/[2]),2.))+exp([3]+[4]*x)",0,20);
root [13] fit2->SetParameters(100., 8., 1., 1., -0.1);
root [14] h1->Fit(fit2);
```

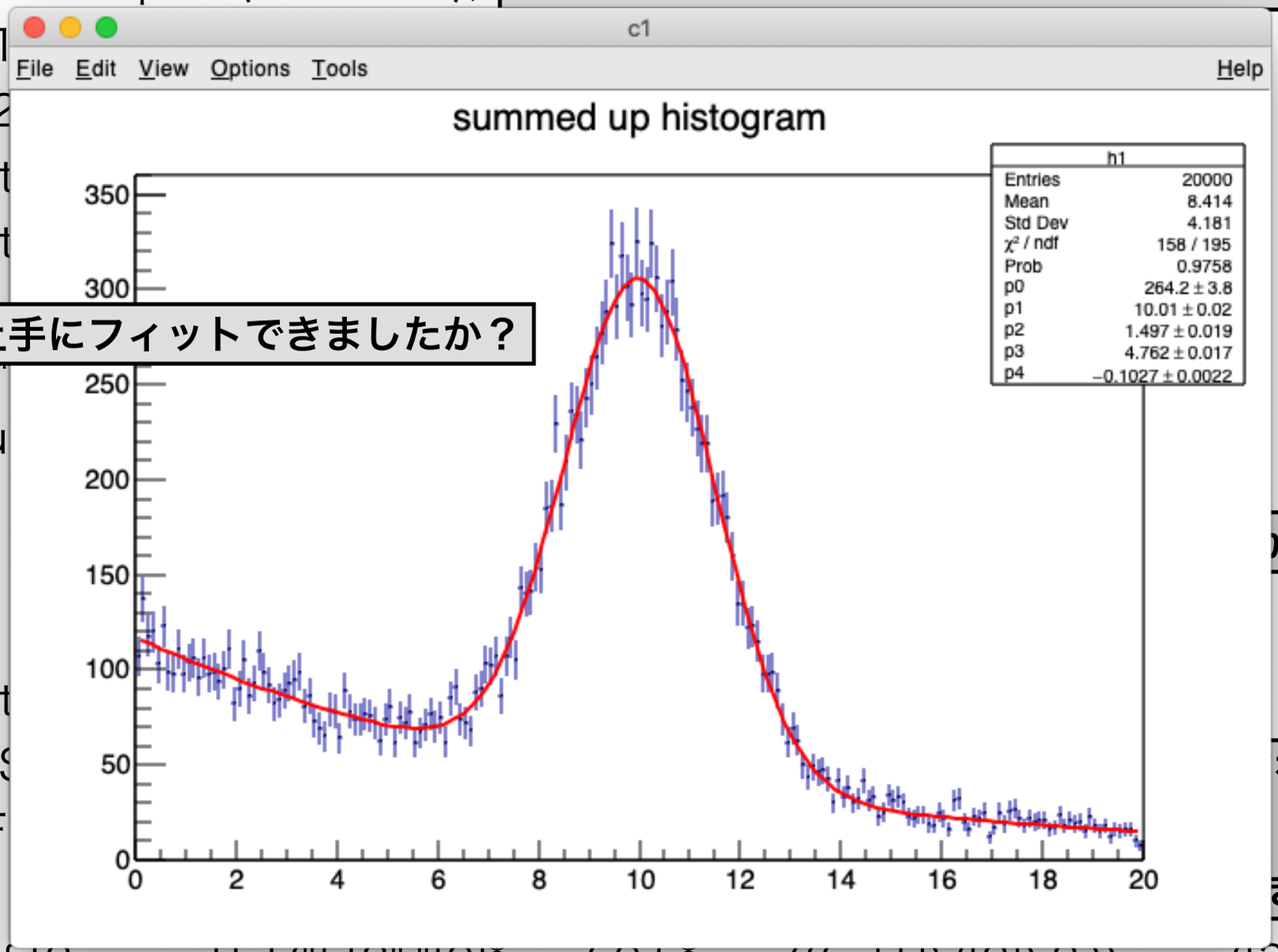
←結果の絵にフィット値を表示するためのおまじない

上手にフィットできましたか？

からランダムにFill

をセット
ット出来ない事も
きたりもする

←9-11行目の代わりにこんな自作の関数
でもフィット可能。書き方が違うだけで
中身は同じなので結果も同じ



TH1をフィットしてみよう

hskw/TF1/examples/fitting3.C

```
root [12] TF1 *fit2=new TF1("fit2","[0]*exp(-0.5*pow(((x-[1])/[2]),2.))+exp([3]+[4]*x)",0,20);
root [13] fit1->SetParameters(100., 8., 1., 1., -0.1);
root [14] h1->Fit(fit2);
```

↑前ページの続き

```
root [15] double params[5], parerrs[5];
root [16] fit2->GetParameters(&params[0]);
root [17] for(int i=0; i<5; i++) parerrs[i]=fit2->GetParError(i);
root [18] for(int i=0;i<5;i++) cout << params[i] << " +/- " << parerrs[i] << endl;
```

フィット結果を取り出す

```
root [45] TF1 *f3 = new TF1("f3","gaus", 0, 20);
root [46] TF1 *f4 = new TF1("f4","expo", 0, 20);
root [47] f3->SetParameters(params[0], params[1], params[2]);
root [48] f4->SetParameters(params[3], params[4]);
root [49] f3->SetLineColor(kMagenta); f4->SetLineColor(kBlue);
root [51] f3->Draw("same"); f4->Draw("same");
```

フィット結果を成分に分けて見せる

- フィット対象をモデル化したTF1関数を定義して，TH1をフィットした
- 対象によってフィットする関数を変えれば，様々な信号に対応可能

YMAP Youth Meeting for AstroParticle

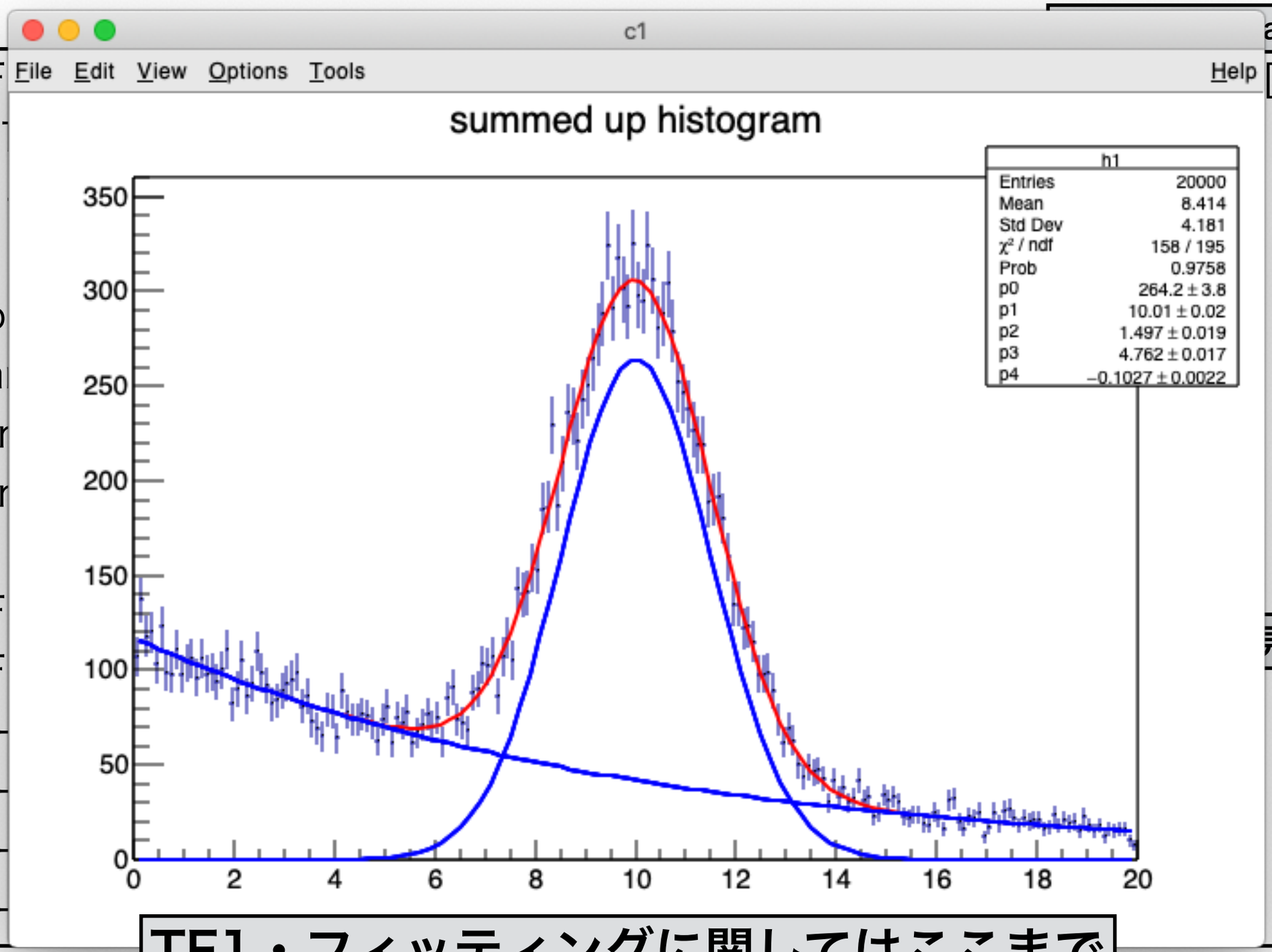
TH1をフィットしてみよう

root [12] TF
root [13] fit
root [14] h1

root [15] do
root [16] pa
root [17] for
root [18] for

root [45] TF
root [46] TF
root [47] f3-
root [48] f4-
root [49] f3-
root [51] f3-

amples/fitting3.C
[4]*x)",0,20);
見せる



TF1・フィッティングに関してはここまで
質問ありますか？

- フィット対象を フィットした
- 対象によってフィットする関数を変えれば，様々な信号に対応可能

TTree

- ROOTが選ばれる大きな理由のひとつ
- 膨大な量/種類のデータを扱うのに適している
  - 様々な型が保存可能。オブジェクトも
  - 小規模のデータでもかなり便利

- 加速器実験（ATLASなど）などでは，複数検出器からのデータを，事象再構成（位置・エネルギー・粒子種類等の推定）や背景事象除去に利用
- 神岡地下で行われているカムランドやスーパーカミオカンデでも，大量のPMTからの情報(時間と電荷量)を元に計算された膨大な数のパラメータ
- これらを "事象毎" に便利な形でまとめて，高速に処理したい

YMAP Youth Meeting for AstroParticle

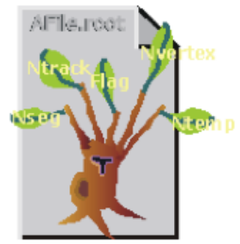
TTree とは

- TH1D や TGraph と違い、同じ概念を持つものが他のソフトウェアには（多分）存在しない
- Event の概念を持つ実験データには欠かせない
- 非常に大雑把に説明すると表計算ソフト（Excel など）のシートや FITS の table のようなもの

Fermi/LAT のガンマ線イベントデータの例

Event No.	Energy(MeV)	Gal. Longitude (°)	Gal. Latitude (°)	Time (ns)
0	44.5018	123.252	11.7771	239557417.793099
1	241.2553	61.90714	60.7625	239557417.953302
2	105.5841	49.0666	78.24632	239557418.516744
3	248.1058	184.6369	13.48609	239557419.156299

- ただし、演算機能、データの可視化、ROOT クラスの保存など、TTree でしか実現できない機能が多くある
- どうして ROOT を使うのかという問いへの 1 つの答え
- 数字以外にも、class を保存することもできる



27

名古屋大 奥村さんの2019年ROOT講習会資料より

TTreeの簡単な例 - 書込み -

```
void TTreeWrite_0(){
 TFile *file = new TFile("TTreeWrite_0.root","RECREATE");
 TTree *tree = new TTree("eventtree","Event Tree");

 int a; double b; float c;
 tree->Branch("a", &a, "a/I");
 tree->Branch("b", &b, "b/D");
 tree->Branch("c", &c, "c/F");

 for(int i=0;i<100;i++){
   a = i;  b = i*0.1;  c = sin(b);
   tree->Fill();
 }

 file->Write();
 file->Close();
}
```

↓書き込みファイルを指定。"RECREATE"は新規作成あるいは上書き

←Tree作成。名前，タイトルを指定

名前　アドレス　型指定

←Treeに入れるBranch(変数)を設定する

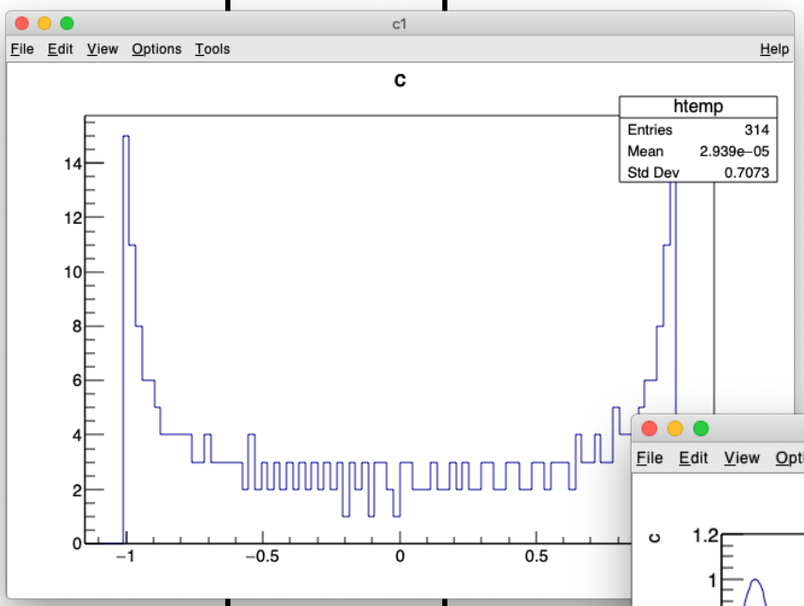
←Treeの中のBranchにそれぞれの値をFillしていく

←TFileに書きこみ，閉じて終了

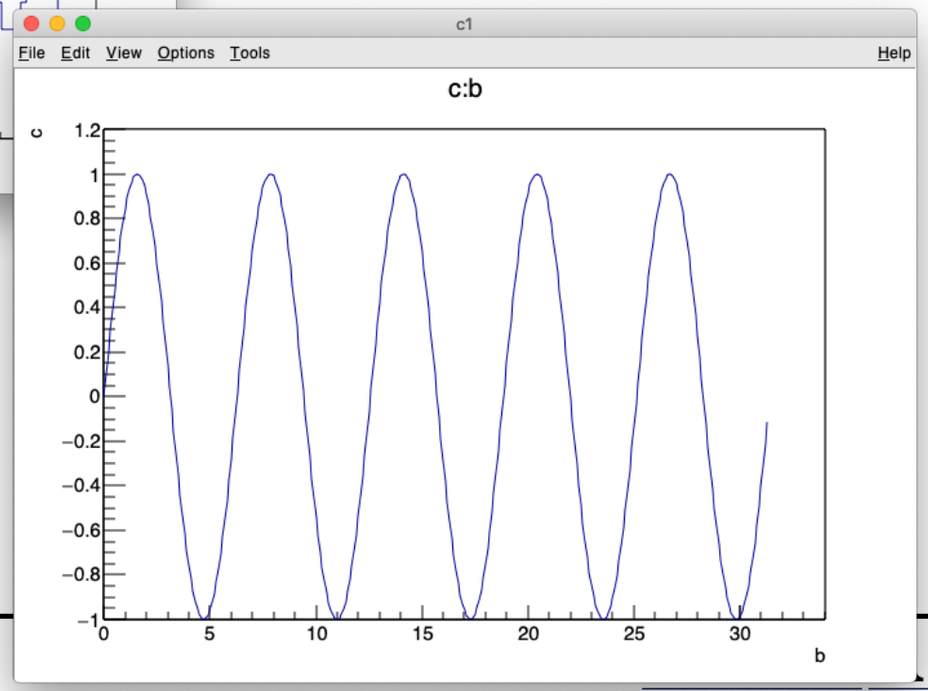
TTreeの簡単な例 - 読出し -

```
[keishi@mac macros] $ root TTreeWrite_0.root
root [0]
Attaching file TTreeWrite_0.root as _file0...
(TFile *) 0x7f82f6fdb720
root [1] .ls   ←TFile内の構造が見える
TFile**         TTreeWrite_0.root
 TFile*         TTreeWrite_0.root
  KEY: TTree    eventtree;1  Event Tree
root [2] eventtree->Scan()   ←TTreeの中身が見える
************************************************
*    Row    *       a.a *       b.b *       c.c *
************************************************
*        0 *         0 *         0 *         0 *
*        1 *         1 *       0.1 * 0.0998334 *
*        2 *         2 *       0.2 * 0.1986693 *
*        3 *         3 *       0.3 * 0.2955202 *
*        4 *         4 *       0.4 * 0.3894183 *
*        5 *         5 *       0.5 * 0.4794255 *
*        6 *         6 *       0.6 * 0.5646424 *
*        7 *         7 *       0.7 * 0.6442176 *
*        8 *         8 *       0.8 * 0.7173560 *
*        9 *         9 *       0.9 * 0.7833269 *
*       10 *        10 *         1 * 0.8414709 *
Type <CR> to continue or q to quit ==> q
************************************************
(long long) 25
root [3]
```

```
root [3] eventtree->Draw("c","","")
Info in <TCanvas::MakeDefCanvas>:  created default
TCanvas with name c1
(long long) 314
root [4] eventtree->Draw("c:a","","L")
(long long) 314
root [5] eventtree->Draw("c:b","","L")
(long long) 314
root [6] .q
[keishi@mac macros] $
```



←↓こんなのが見えるはず



Youth Meeting for AstroParticle

TTreeの書き方

- 本講習会登録時の情報の一部を抜いてデータ化した
  - 所属，学年，経験のある言語，ネットワーク環境など
- これらをTTreeに詰めて，解析してみる

TTreeの書き方

- hskw/TTree/TTreeSeed.dat
  - 登録順，登録時間，ML登録可否，
    所属，学年，コンピュータ環境，
    ネットワーク環境，言語経験
    （unix, shell, ROOT, cpp, python）
    がリスト化されている。

- 今回はこいつを読んで，TTreeに書き込んで rootファイルに保存。簡単な解析をしてみる。

hskw/TTree/TTreeSeed.dat

```
1 0 1 1 6 3 3 1 1 1 1 1
2 760.7439997 1 2 1 1 3 1 1 1 1 1
3 1007.335 1 3 3 2 3 1 1 1 1 0
4 1048.01 0 4 6 5 3 1 1 1 1 0
5 1295.243 1 5 2 3 3 1 1 1 1 0
6 1520.173 1 6 0 5 3 0 0 1 0 0
7 1577.752 1 5 0 0 3 0 0 0 0 1
8 1816.487 1 8 1 3 3 0 0 0 1 0
9 2075.415 1 4 0 5 4 1 1 1 1 1
10 2450.442 1 3 3 3 0 0 1 0 1 1
11 2479.401 1 11 0 4 1 0 0 0 1 1
12 2793.208 1 10 1 1 3 1 1 0 0 0
13 2894.193 1 12 0 4 1 0 0 0 1 1
14 3131.372 1 3 1 1 3 0 0 0 0 1
15 3473.696 1 9 0 3 1 1 0 1 1 1
16 3818.842 1 2 1 3 3 1 1 1 1 1
17 4357.803 1 4 1 2 4 0 0 0 1 0
18 4406.251 1 7 0 3 1 1 1 0 0 1
19 5058.993 1 11 0 4 3 0 0 0 1 0
20 6362.288 0 4 1 5 3 0 0 1 1 0
21 6439.324 0 3 1 4 3 0 0 0 0 1
22 7194.975 0 13 1 3 1 0 0 0 1 0
23 7274.854 1 3 1 1 4 1 0 0 0 0
24 9195.559999 0 9 1 2 3 0 0 1 1 0
25 9578.085 1 3 1 3 0 0 0 0 0 1
26 10816.741 0 3 1 1 1 0 0 0 1 0
27 11036.589 1 2 6 1 3 1 1 1 1 1
28 11524.608 0 2 2 3 3 0 0 0 0 1
29 13983.103 1 14 1 1 3 1 0 0 1 0
30 19682.209 1 3 1 3 3 0 0 1 1 0
31 27539.268 1 7 0 2 3 1 0 1 1 1
32 28479 1 2 0 4 1 0 0 0 0 0
33 30017.428 1 1 0 4 1 0 0 0 1 1
34 34851.905 1 3 2 1 3 1 1 1 1 1
35 62339.63 0 2 6 3 1 1 1 0 0 0
36 63823.454 1 15 0 4 1 0 0 0 1 0
37 66018.933 1 4 1 1 3 0 0 0 1 0
38 67111.337 1 8 1 1 3 0 0 0 0 0
39 69069.535 1 1 0 4 1 0 0 0 0 1
40 69355.029 0 4 0 4 1 0 0 0 0 0
41 69682.024 0 2 0 1 1 0 0 0 0 1
42 71501.536 1 0 0 4 1 0 0 0 0 0
43 72773.701 1 12 5 2 3 1 1 1 1 1
44 73823.928 0 5 0 3 3 0 0 1 1 0
45 77450.746 1 3 1 1 3 0 0 0 1 1
```

TTreeの書き方

```
void TTreeWrite_1(){
 TFile *file = new TFile("test.root","RECREATE");
 TTree *tree = new TTree("tree","Event Tree");

  int id, institute, year, env, network, nskill;
  double dt;
  bool ML, unix, shell, ROOT, cpp, python;

  tree->Branch("id",         &id,         "id/I");
  tree->Branch("institute", &institute, "institute/I");
  tree->Branch("year",       &year,       "year/I");
  tree->Branch("env",        &env,        "env/I");
  tree->Branch("network",    &network,    "network/I");
  tree->Branch("nskill",     &nskill,     "nskill/I");
  tree->Branch("dt",         &dt,         "dt/D");
  tree->Branch("ML",         &ML,         "ML/O");
  tree->Branch("unix",       &unix,       "unix/O");
  tree->Branch("shell",      &shell,      "shell/O");
  tree->Branch("ROOT",       &ROOT,       "ROOT/O");
  tree->Branch("cpp",        &cpp,        "cpp/O");
  tree->Branch("python",     &python,     "python/O");

  ifstream ifs("./TTreeSeed.dat");
  if(!ifs.is_open()) {
    cout << "File not opened." << endl;
    return 0;
  }

  while(ifs >> id >> dt >> ML >> institute >> year >> env >> network >> unix >> shell >> ROOT >> cpp >> python){
   nskill = unix + shell + ROOT + cpp + python;
    tree->Fill();
  }

  file->Write();
  file->Close();
}
```

←書き込みファイルを指定。"RECREATE"は新規作成あるいは上書き

←Tree作成. treeの名前，タイトルを指定

パラメータにBranchを設定する
tree->Branch("名前", アドレス, “型指定")

←データファイルを呼出し

←データファイルがちゃんと開けているか確認

↓データファイルを1行ずつ読む

←各ブランチに各変数を詰める

←書き込んでファイル閉

型リスト(リンク)

- `C` : a character string terminated by the 0 character
- `B` : an 8 bit signed integer (`Char_t`)
- `b` : an 8 bit unsigned integer (`UChar_t`)
- `S` : a 16 bit signed integer (`Short_t`)
- `s` : a 16 bit unsigned integer (`UShort_t`)
- `I` : a 32 bit signed integer (`Int_t`)
- `i` : a 32 bit unsigned integer (`UInt_t`)
- `F` : a 32 bit floating point (`Float_t`)
- `f` : a 24 bit floating point with truncated mantissa (`Float16_t`)
- `D` : a 64 bit floating point (`Double_t`)
- `d` : a 24 bit truncated floating point (`Double32_t`)
- `L` : a 64 bit signed integer (`Long64_t`)
- `l` : a 64 bit unsigned integer (`ULong64_t`)
- `G` : a long signed integer, stored as 64 bit (`Long_t`)
- `g` : a long unsigned integer, stored as 64 bit (`ULong_t`)
- `O` : [the letter o, not a zero] a boolean (`Bool_t`)

hskw/TTree/examples/TTreeWrite.C

Youth Meeting for AstroParticle

TTreeの読み方

hskw/TTree/examples/TTreeRead1.C

```
[keishi@mac macros] $ root TTreeWrite.C
root [0]
Processing TTreeWrite.C...
root [1] .q

[keishi@mac macros] $ root TTreeTest.root
root [0]
Attaching file TTreeTest.root as _file0...
.(TFile *) 0x7faf15fb98d0


root [1] .ls
TFile**         TTreeTest.root
 TFile*         TTreeTest.root
  KEY: TTree   tree;1   Event Tree

root [1] tree->Print()
******************************************************************************
*Tree    :tree      : Event Tree                                               *
*Entries :      269 : Total =            17818 bytes  File  Size =       6146 *
*        :          : Tree compression factor =   2.30                       *
******************************************************************************
*Br    0 :id        : id/I                                                     *
*Entries :      269 : Total  Size=       1618 bytes  File Size  =        516 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   2.22     *
*.............................................................................*
*Br    1 :institute : institute/I                                              *
*Entries :      269 : Total  Size=       1653 bytes  File Size  =        374 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   3.08     *
*.............................................................................*
*Br    2 :year      : year/I                                                   *
*Entries :      269 : Total  Size=       1628 bytes  File Size  =        280 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   4.10     *
*.............................................................................*
```

↑マクロでTTreeを含んだrootファイルを作成

←rootファイルを読込み

←読んだrootファイルの中身を確認

←treeの中身を確認

tree->Show(event番号) も使える。tree->Show(0); とか

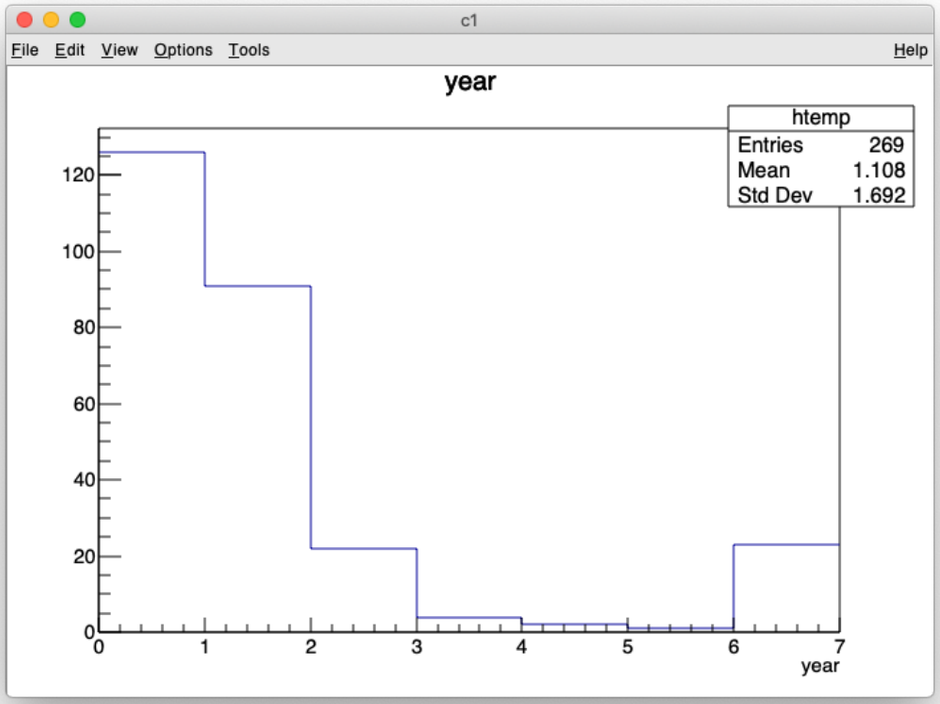
Youth Meeting for AstroParticle

TTreeの読み方

hskw/TTree/examples/TTreeRead1.C

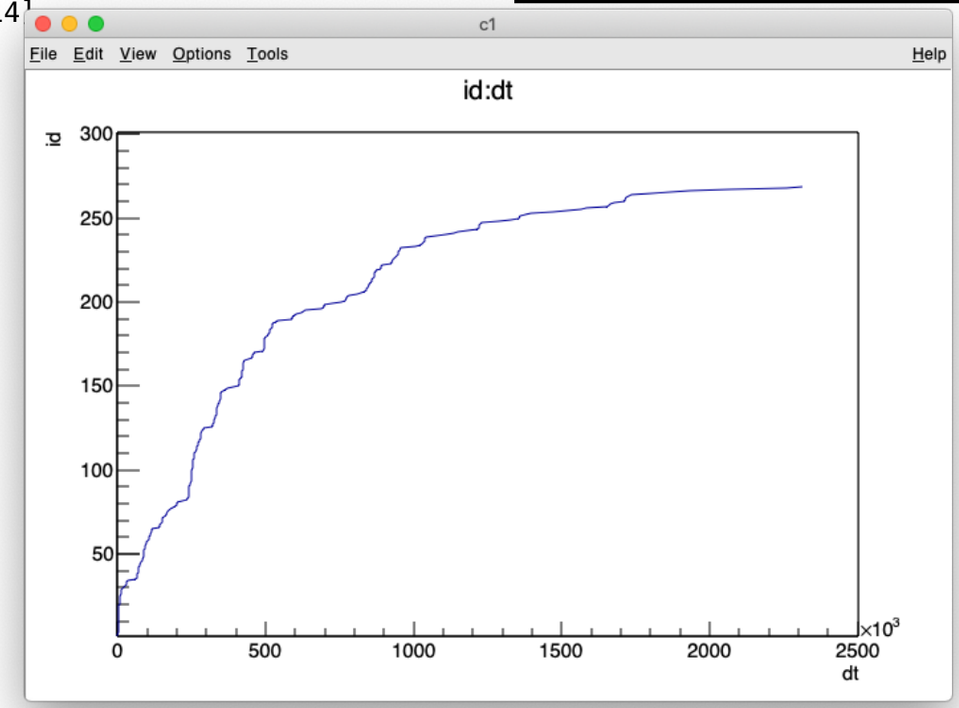
```
root [8] tree->Draw("year")
root [9]
```

←この要領で，Drawできる



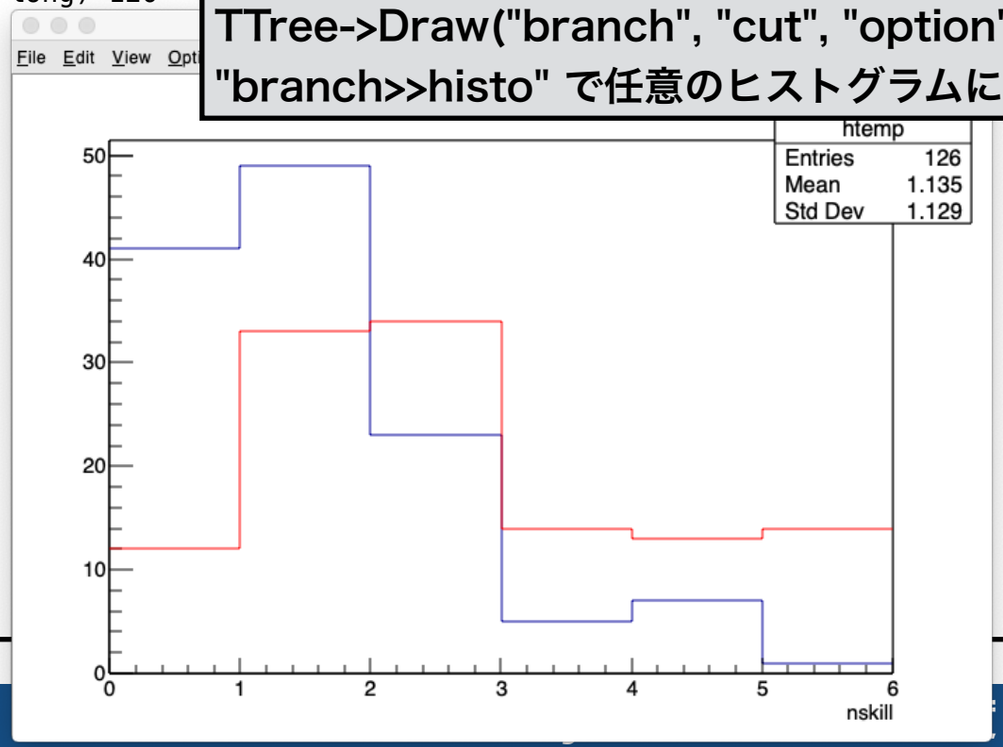

```
root [13] tree->Draw("id:dt","","L")
(long long) 269
root [14]
```

"L" optionでライン描写



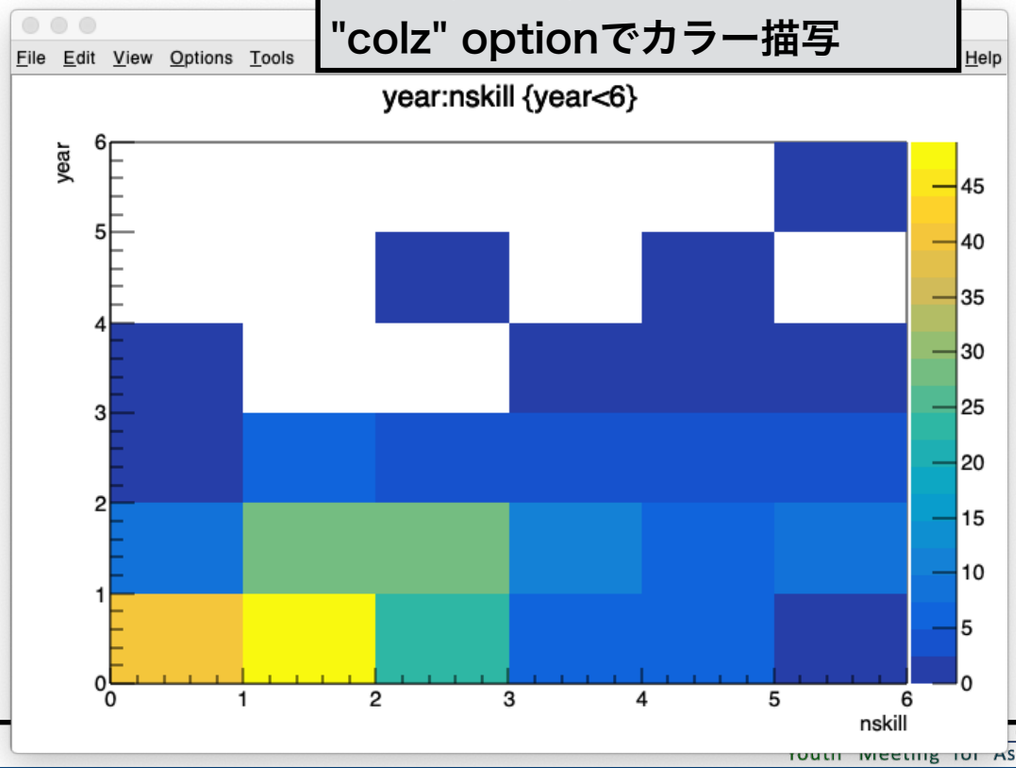

```
root [9] tree->Draw("nskill","year==0","")
(long long) 126
root [10] TH1D *h1 = new TH1D("h1","h1",6,0,6);
root [11] h1->SetLineColor(kRed);
root [12] tree->Draw("nskill>>h1","year!=0&&year!=6","same")
(long long) 120
```

TTree->Draw("branch", "cut", "option")
"branch>>histo" で任意のヒストグラムにDraw




```
root [14] tree->Draw("year:nskill","year<6","colz")
(long long) 246
root [15]
```

Draw("A:B")で二次元ヒスト描写
"colz" optionでカラー描写



TTreeの読み方

hskw/TTree/examples/TTreeRead2.C

```
void TTreeRead(){
```

↓ファイル読込み

```
  TFile file("TTreeTest.root");
  TTree* tree=(TTree*)file.Get("tree");
```

←TTreeWriteで作ったtreeを呼び出す

```
  int id, institute, year, env, network, nskill;
  double dt;
  bool ML, unix, shell, ROOT, cpp, python;
```

←treeの中身を入れるための変数を用意

```
  tree->SetBranchAddress("id",        &id);
  tree->SetBranchAddress("dt",        &dt);
  tree->SetBranchAddress("year",      &year);
  tree->SetBranchAddress("nskill",    &nskill);
  tree->SetBranchAddress("ROOT",      &ROOT);
```

←ブランチ毎にアドレス割当て
SetBranchAddress("tree内での名前", アドレス);
使いたいものだけSetBranchAddressすれば良い

```
  for( int i=0; i<tree->GetEntries(); i++ ){
    tree->GetEntry(i);
    if(year==0) cout << nskill << " " << ROOT << endl;
  }
```

←tree->GetEntries()でエントリ数がもらえる
この要領でTH1->Fill()なども可能

```
  tree->Draw("id:dt","","L");
}
```

←前ページと同様にtree->Draw()もできる
これだけしたい場合はSetBranchAddress()等は不要

TTreeの読み方

配列を含んだTreeの作り方

```
void TTreeWriteArray(){

  TFile *file = new TFile("TTreeArrayTest.root","RECREATE");
  TTree *tree = new TTree("tree","tree including array");

  double wfArray[100]={};

  tree->Branch("wfArray", wfArray, "wfArray[100]/D");

  for(int i=0;i<100;i++){

    for(int j=0;j<100;j++){

      wfArray[j] = i+j;

    }

    tree->Fill();

  }

  file->Write();
  file->Close();

}
```

配列を含んだTreeの読み方

```
void TTreeReadArray(){

  TFile file("./TTreeArrayTest.root");
  TTree *tree = (TTree*)file.Get("tree");

  double wfArray[100]={};

  tree->SetBranchAddress("wfArray", wfArray);

  for(int i=0;i<tree->GetEntries();i++){
    tree->GetEntry(i);

    for(int j=0;j<100;j++){
      cout << wfArray[j] << endl;
    }
  }
```

TTree, ROOT fileに関連した便利機能

- haddコマンドで，同じtreeを持ったrootファイルを結合できる

hskw/TTree/examples/hadd.C

```
[keishi@mac macros] $ hadd sum0.root TTreeTest.root TTreeTest1.root
TTreeTest2.root
hadd Target file: sum0.root
hadd compression setting for all output: 1
hadd Source file 1: TTreeTest.root
hadd Source file 2: TTreeTest1.root
hadd Source file 3: TTreeTest2.root
hadd Target path: sum0.root:/
[keishi@mac macros] $ hadd sum1.root TTreeTest*.root
hadd Target file: sum1.root
hadd compression setting for all output: 1
hadd Source file 1: TTreeTest.root
hadd Source file 2: TTreeTest1.root
hadd Source file 3: TTreeTest2.root
hadd Target path: sum1.root:/
[keishi@mac macros] $
```

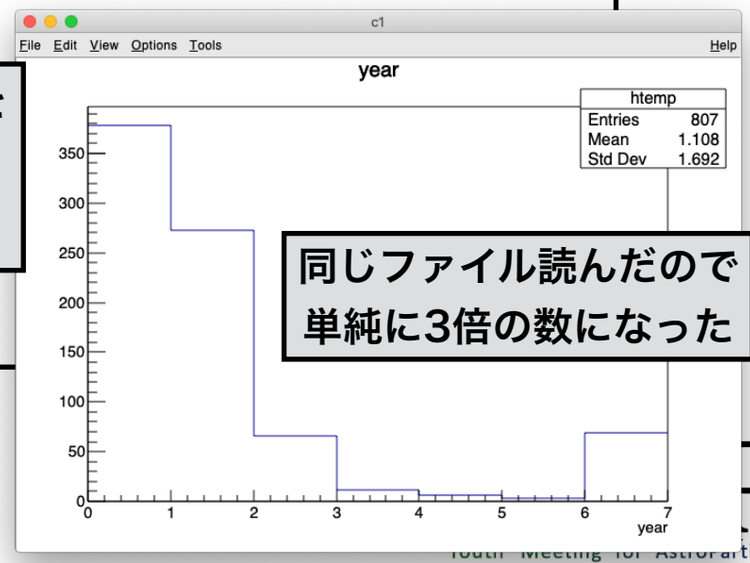
- TChain：同じ構造のTTreeを含んだroot fileを単一のtreeのように扱える

hskw/TTree/examples/TTreeRead3.C

```
void TTreeRead2(){
  TChain *chain = new TChain("tree");
  chain->Add("TTreeTest.root");
  chain->Add("TTreeTest1.root");
  chain->Add("TTreeTest2.root");

  chain->Draw("year","","");
}
```

←ここではサボってroot fileのコピーを読んだが，本来は同じ構造のTreeを持った"別の"ファイルを読むべき



同じファイル読んだので単純に3倍の数になった

おわりに

- お疲れさまでした
- 折角なので，水越宿題のデータ管理にTTreeを，解析にフィッティングを導入してみてください

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 #homework2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.

- これからも，ROOTその他の質問受け付けております。お気軽にどうぞ
- YMAPへの参加,
  ROOT講習会の運営・講師
  としての参加も歓迎します！！
  http://www.icrr.u-tokyo.ac.jp/YMAP/join.html

YMAP
Youth Meeting for AstroParticle