ROOT解析入門
～アホの子でもできるn次元解析～

H.ITO

平成26年7月4日

目 次

1 ROOT 入門

ROOT は、CERN によって開発が行われている、データ解析環境および関連するライブラリ群である. グラフ作成のみならず、ヒストグラムの操作、4 元ベクトルの扱い、実験データの可視化など、高エネルギー物理学の研究に不可欠な要素が組み込まれている。開発当初は素粒子実験のデータ解析用ソフトウェアとして構築されたが、近年では高エネルギー宇宙物理学や天文学といった分野でも使用されている.

1993 年から René Brun と Fons Rademakers の手によって、CERN にて開発が開始され、1995 年に最初の版が公開された. 当時の素粒子実験では、FORTRAN がプログラミング言語の主役であり、データ解析ソフトウェアとしては PAW が使われていた。しかし、より実験が大規模化するにつれソフトウェア開発の手法も変化し、C++を用いたオブジェクト指向型のデータ解析環境が求められるようになった. 2009 年現在でも彼らを中心としたメンバーで開発が継続されている. CINT を採用することにより、C++をスクリプト言語として使用することが可能である. そのため C++を理解していれば PAW や Display 45 のような独自の文法規則を覚える必要がなく、またユーザが C++で ROOT の機能を拡張し、独自の解析やシミュレーションを行うことも容易である.(wikipedia)

1.1 使い方

起動方法

```
$root
```

画像の表示は

```
root[] TCanvas*c1 =new TCanvas("c1")
root[] TGraph *g = new TGraph("output.dat")
root[] g->Draw("AP")
```

で、output.dat の plot をしてくれる。ここで、コマンドラインは $ を、ROOT の対話式のコマンドには root[] を先頭につけることにする.

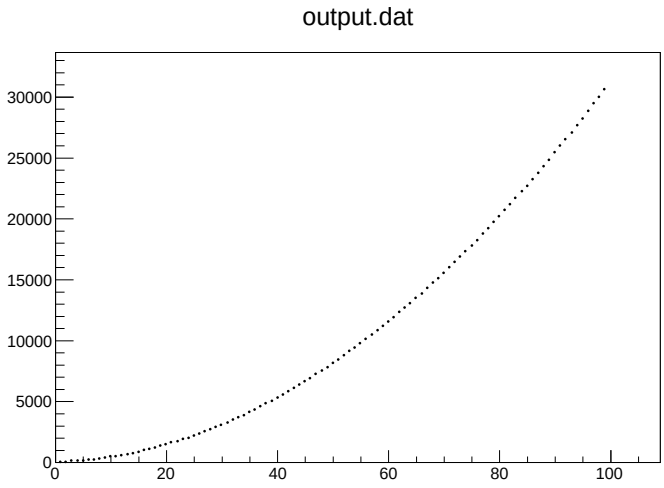

図 1: ROOT plot imaging

画像の保存方法

```
root[] c1->Print("img.png")
```

終了の仕方

```
root[] .q
```

2　インストール方法 Linux/Debian/Fedora OS

今回は Linux でのインストール方法を紹介する.MAC および Windows でもインストールは可能.Web で「how to install root」で検索、「1.1. ROOT インストール-ROOT 解体新書-」のインストール方法を参考にする.Linux OS は Ubuntu 12.10/10.04/13.04、mint mate maya 13, Fedora 13 で確認済.

2.1　CERN のサイトにアクセス

CERN の ROOT インストールサイトは以下である.

`http://root.cern.ch/drupal/content/downloading-root`

まず Pro, version 5.34/18 をクリック.Source の ROOT5.34/03 complete source tree for all system(56MB) をクリックすればインストールが開始される.

図 2: CERN ROOT インストール ページ

2.2　source の解凍

インストールが完了したら、ダウンロードしたファイルを解凍する. ターミナルを起動して、コマンドでインストールした場所までいく. ここではとりあえず、HOME/Download/に保存されていることにする.

```
$cd
$cd Download/
$tar -zxf root_v534.03.source.tar.gz
$cd root
$./configure --prefix=/usr/local/root
```

すると、***MUST be install と次のパッケージをインストールしてくれというエラーが出る.

```
http://root.cern.ch/drupal/content/build-prerequisites
```

にアクセスして OS ごとに何をインストールするか確認する. ちなみにインストールのコマンドは次のとおりである.

```
$sudo apt-get install dpkg-dev
$sudo apt-get install libxft-dev
$sudo apt-get install libxext-dev
...
```

インストール完了したらもう一度コマンド

```
$./configure --prefix=/usr/local/root
```

完了した場合、

```
------------------
To build ROOT Type;

make
make install
------------------
```

と出力される. 次に

```
$make -j4
```

と打つ. これはコンパイルを意味していて、実効ファイルを作っている. 結構時間がかかる.-j4 は CPU の性能で異なるが、マルチスレッド処理で実行するオプションである。make が終わったら

```
$sudo make install
```

と打つ. これも時間がかかる.

2.3 環境設定

ホームディレクトリの隠しファイル.bashrc に起動時のパスを通す.

```
$cd
$emacs .bashrc
```

エディタは何でもよい。.bashrc に次の文章を追加する.

.bashrc

```
export ROOTSYS=/usr/local/root
export PATH=$ROOTSYS/bin:$PATH
export LD_LIBRARY_PATH=$ROOTSYS/lib/root:$LD_LIBRARY_PATH
```

再起動もしくはログインしなおしてターミナルを起動し、コマンド

```
$root
```

と打つと、ROOT 起動画面が立ち上がる. また、ログインしなおさなくても、bahrc を実行することですぐに ROOT が起動可能な状態になる。

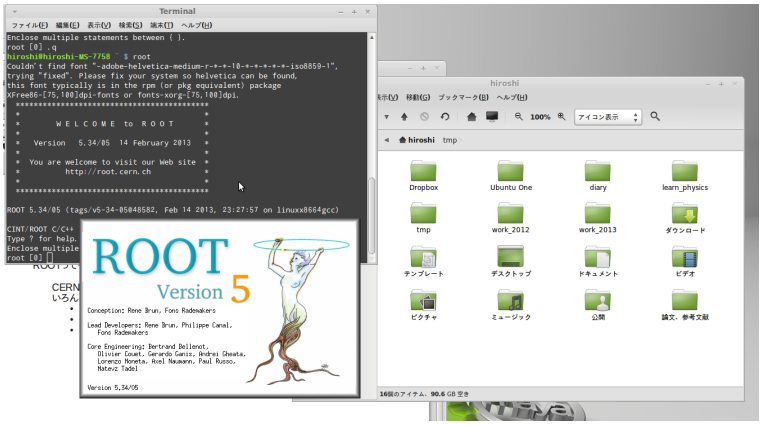
Terminal
ROOT
Version 5
hiroshi
Dropbox
Ubuntu One
diary
learn_physics
tmp
work_2012
work_2013

図 3: ROOT 起動画面

3　ヒストグラム

3.1　1 次元ヒストグラム

output データを読みだして 1 次元のヒストグラムを生成する. 方法はいくつかあり、紹介する. コマンドラインで作成することも可能だが毎回は面倒なので、マクロで解析すると楽である.

out1.dat

```
0 0
1 2
2 1
3 3
4 6
5 10
6 23
7 45
8 33
9 20
10 3
```

hist1.cpp

```
{
 gROOT ->Reset();
 gStyle ->SetOptStat(1001110);
 double x,y;
 TCanvas*c1=new TCanvas("c1");
 TH1S*hist=new TH1S("h1","h1",10,0,10);
 ifstream ifs("./out1.dat");
 while(ifs>>x >>y)
  hist->Fill(x,y);
 hist->Draw();
}
```

マクロを起動するコマンドは

```
root[] .x hist1.cpp
```

または、

```
$root -l hist1.cpp
```

である.-l は起動画面を表示しないで ROOT を起動するコマンドのオプションである. 同じディレクトリ内の out1.dat のテキストデータをヒストグラム表示してくれるプログラムである. ソースの主に 6 行目の”./out1.dat”を変更すれば違うデータをヒストグラムに可能. 5 行目はヒストグラムの名前、ビン数、min、max を表している.TH1S*hist=new TH1S(”h1”,”h1”,10,0,10); は 0 から 10 の間を 10 等分してヒストグラムを生成することを意味する. ソースは C/C++ 言語で書かれるため、行末には; が必要.out1.dat には x 座標 0 から 10 に含まれるカウント数を 2 列で表示したテキストファイルである.

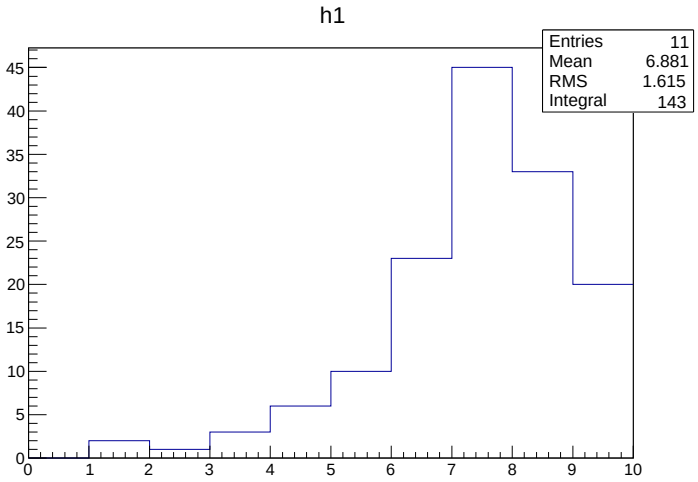

図 4: ROOT plot imaging

out2.dat

```
1
1
4
2
1
6
5
2
1
3
1
4
5
3
1
6
2
```

hist2.cpp

```
{
 gROOT ->Reset();
 gStyle ->SetOptStat(1001110);
 double x,y;
 TCanvas*c1=new TCanvas("c1");
 TH1S*hist=new TH1S("h2","h2",10,0,10);
 ifstream ifs("./out2.dat");
 while(ifs>>x)
  hist->Fill(x);
 hist->Draw();
}
```

out2.dat にはある物理量の数値を 1 列で羅列したテキストファイルである. これを蓄積させてヒストグラムを生成する.

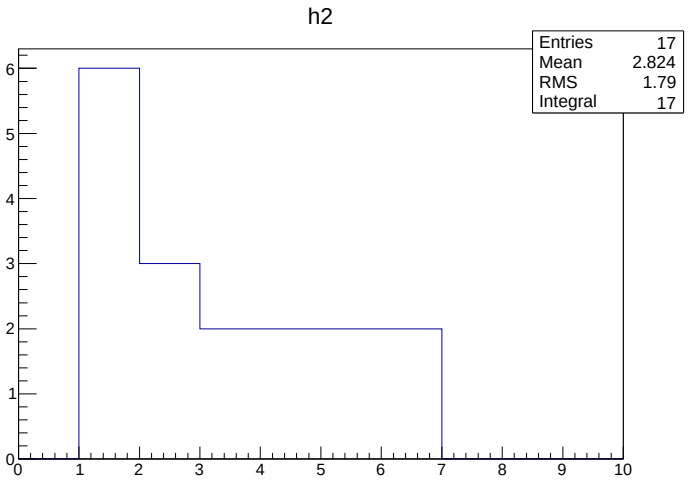

図 5: ROOT plot imaging

3.2　2 次元ヒストグラム

2 次元ヒストグラムの作成も 1 次元とほぼ同じマクロで生成可能.

out3.dat

```
1 5
1 2
4 1
2 3
1 1
3 1
1 2
3 4
5 2
2 3
3 1
3 5
1 6
```

hist3.cpp

```
{
 gROOT ->Reset();
 gStyle ->SetOptStat(1001110);
 double x,y;
 TCanvas*c1=new TCanvas("c1");
 TH2S*hist=new TH2S("hist","hist",10,0,10,10,0,10);
 ifstream ifs("./out3.dat");
 while(ifs>>x>>y)
  hist->Fill(x,y);
 hist->Draw("LEGO");
}
```

2 次元の場合は TH1S の代わりに TH2S を使用する. 変数は (ポインタ, 名前, x_bin, x_min, x_max, y_bin, y_min, y_max) で、3 つ増えた。2 次元では Draw() の中はオプションを入れないと書いてくれない.LEGO は立体的に表示、COLZ は色の違いで高さを表現する.

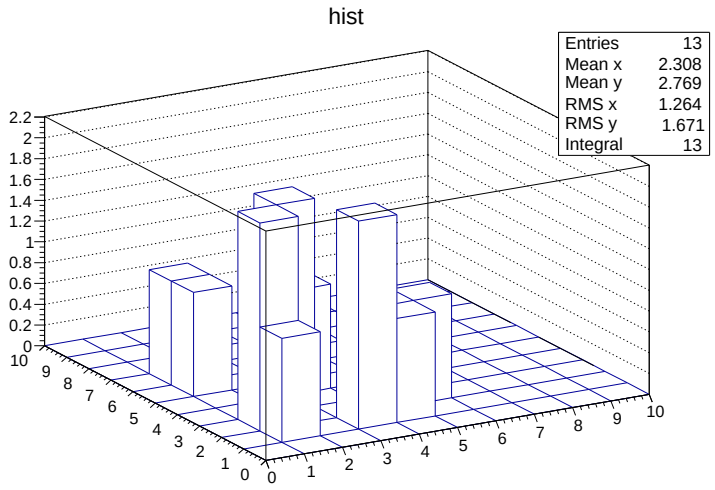

図 6: ROOT plot imaging

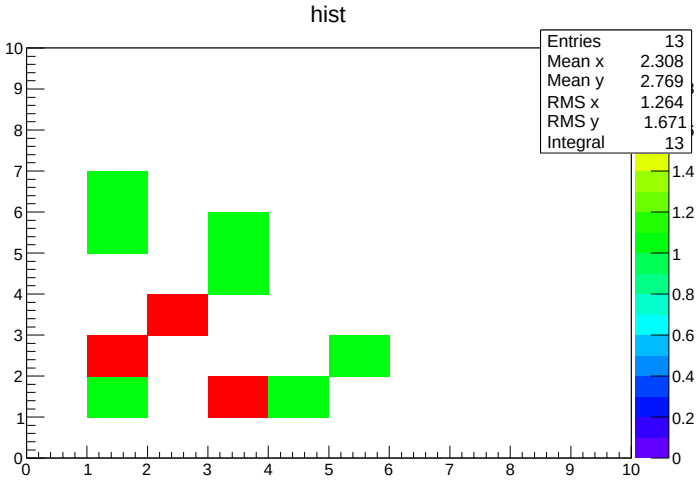

図 7: ROOT plot imaging

3.3 統計情報の表示

デフォルトではヒストグラムを描画すると統計情報のボックスも表示される。統計情報のボックスを取り除くには TH1::SetStats(kFALSE) を使うとよい. 統計情報のボックスを表示するとき、gStyle->SetOptStat(mode) を使って表示される情報の種類を選択することができる. パラメータ mode は 9 桁までで、次のように OFF(0) または ON とすることができる: mode = ksiourmen (デフォルト = 000001111)

- n = 1 ヒストグラムの名前の表示
- e = 1 エントリーの数
- m = 1 平均値
- m = 2 平均値と平均誤差
- r = 1 二乗平均平方根 (RMS)
- r = 2 RMS と RMS の誤差
- u = 1 アンダーフローの数
- o = 1 オーバーフローの数
- i = 1 ビンの積分値
- s = 1 歪度
- s = 2 歪度およびその誤差
- k = 1 尖度
- k = 2 尖度およびその誤差

SetOptStat(0001111) のようにコールしてはいけない。SetOptStat(1111) とすること。というのも、0001111 は 8 進数と解釈されてしまうからである. メソッド TStyle::SetOptStat(Option_t *option) も、文字列をパラメータにしてコールすることができる. パラメータ option には以下を指定することができる.

- n ヒストグラムの名前の表示
- e エントリーの数

- m 平均値
- M 平均値と平均誤差
- r 二乗平均平方根 (RMS)
- R RMS と RMS の誤差
- u アンダーフローの数
- o オーバーフローの数
- i ビンの積分値
- s 歪度
- S 歪度およびその誤差

[引用]ROOT/CINT 自社翻訳ドキュメント ヒストグラムから拝借

4 グラフ

4.1 手打ち plot

まずはマクロソースを見てもらおう。

graph1.cpp

```
{
gROOT->Reset();
gStyle->SetOptFit();
canv=new TCanvas("c1");

int n=4;
double x[n]={90,60,30,0};
double y[n]={8.9,11.2,14.6,21.1};
double exl[n]={1.5,1.2,0.8,1.1};
double exh[n]={0.2,0.4,1.1,0.5};
double eyl[n]={3.7,4.2,4.8,5.7};
double eyh[n]={3.7,4.2,4.8,5.7};

gr1 = new TGraphAsymmErrors(n,x,y,exl,exh,eyl,eyh);
gr1->SetMarkerStyle(4);
gr1->SetMarkerSize(3);
gr1->SetTitle("test for attenuation length: Y-11 + 401nm LED");
gr1->GetXaxis()->SetTitle("transfered length [cm]");
gr1->GetYaxis()->SetTitle("mean number of photoelectrons [p.e.]");
gr1->Draw("AP");

int k=1;
char name[100];
 sprintf(name,"f%d",k);
 TF1*fanc=new TF1(name,"[0]*exp(-x/[1])");
 fanc->SetParameters(30,100);
 gr1->Fit(name,"","",-10,100);
 cout<<fanc->GetParameter(0)<<" "<<fanc->GetParError(0)<<endl;
 cout<<fanc->GetParameter(1)<<" "<<fanc->GetParError(1)<<endl;
}
```

このプログラムは実験で何か結果が得られた時に x と y の情報をいれてグラフを作成する。このプログラムで実行している動作は (1) 配列に情報を代入、(2) グラフの装飾、(3) フィッティングの 3 つが動く。ちなみに、誤差棒を入れたグラフを表示する。

基本的にグラフの設定は `TGraph(n,x,y)` で可能で誤差棒なんていらないときに使用する。引数の n は plot のエントリー数 (int 型),x,y はグラフに打ち込む点の情報 (配列) でないとエラーを吐く。配列の数が n に等しいとミスが減る。`TGraphAsymmErrors(n,x,y,exl,exh,eyl,eyh)` ではそれに加えて、誤差棒の情報がいる。

`gr1->SetMarkerStyle(4);` はマーカーのスタイルを変更する。デフォルトは 1 は・、2 は+、3:※,4:○、5:×などである。また、`gr1->SetMarkerSize(3);` でマーカーのサイズも番号に応じて変更可能。マーカーの色は `gr1->SetMarkerColor(2);` で変更可能。デフォルトは 1:黒で、2; 赤、3:緑、4:青である。ただ、マーカーの色を変えても、誤差棒の色は変わらない。誤差棒はマーカーではなくラインなので。つまり、`gr1->SetLineColor(2);` で変更可能。`gr1->SetTitle("...");` はヒストグラムと同様に使用が可能である。キャンバスに Draw するとき

は引数が必要で必ず "" で囲わなければいけない。"A"は x-y 軸を書き加える引数で、重ね書きする場合は"A"は必要なくなる。"P"は各 plot を点として表示する引数だ。もし折れ線グラフを表示するなら"L"、その線をなめらかにするなら"C"を引数にする。

せっかくだからフィッティングについても簡単に触れておく。`sprintf(name,"...")` は char 型の `name` にポインタの名前を代入している。TF1 で指数関数を定義して、 `gr1->Fit(name,"","",-10,100);` で x 範囲 [-10:100] の関数でフィッティングする。詳しくは後ほど。最後はフィットしたパラメータを cout で出力している。

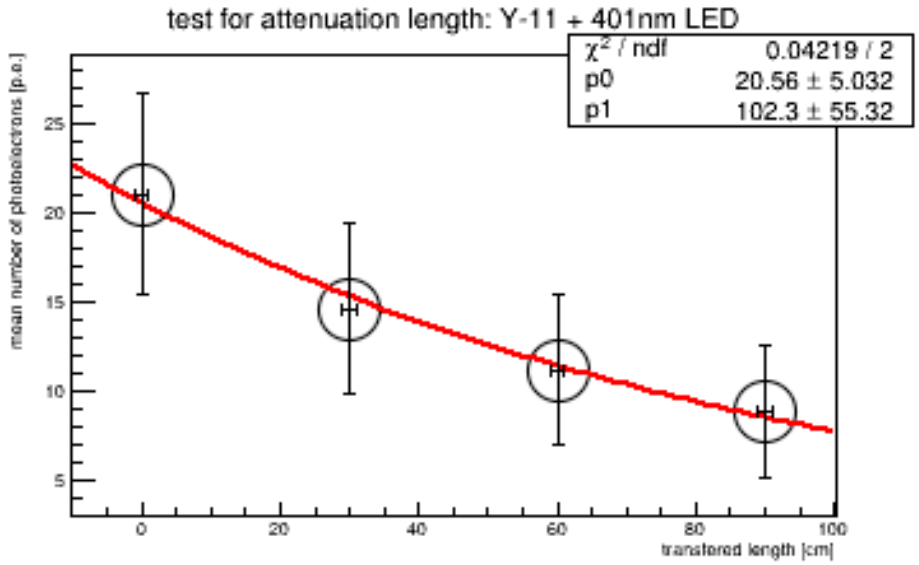

図 8: ROOT plot imaging

4.2 リスト読み込み型マクロ

data.dat

```
80.9 35.8 2.6 2.5 0.84 0.84
86.8 29.9 8.9 7.4 0.25 0.25
125.2 25.6 19.8 15 0.24 0.24
135.1 23.1 39.9 25.1 0.19 0.19
186.4 18.1 32.4 24 0.21 0.21
312.1 14.6 66.4 46.6 0.35 0.35
...
```

data2.dat

```
179 32.4 14 23 1.2 2.3
191 17.8 21 24 4.1 2.3
12 16.6 3.1 3.2 5.3 6.7
21 17.5 9.1 5.4 5.6 4.7
287 7.6 20 25 1.4 2.1
436 7.6 124 103 3.2 2.4
175 8.6 40.2 54 1.4 2.3
478 4.7 30 40 0.3 0.1
1979 41.8 125 210 10 15
...
```

graph2.cpp

```
{gROOT->Reset();
gStyle->SetOptFit();
canv=new TCanvas("c1");
int n=100;
double a,b,c,d,e,f;
double x[n],y[n],exl[n],exh[n],eyl[n],eyh[n];


int m=0;
ifstream ifs("./data.dat");
while(ifs>>a >>b >>c >>d >>e >>f){
 x[m]=a; y[m]=b; exl[m]=c; exh[m]=d; eyl[m]=e; eyh[m]=f;
 m++;}
ifs.close();


gr1 = new TGraphAsymmErrors(m,x,y,exl,exh,eyl,eyh);
TAxis *axis = gr1->GetXaxis();
axis->SetLimits(1,1e3);                 // along X
gr1->GetHistogram()->SetMaximum(1e3);   //along
gr1->GetHistogram()->SetMinimum(1);  //   Y
gr1->SetMarkerStyle(4);
gr1->SetMarkerSize(3);
gr1->SetTitle("test");
gr1->GetXaxis()->SetTitle("x-axis");
gr1->GetYaxis()->SetTitle("y-axis");
gr1->Draw("AP");


m=0;
ifstream ifs("./data2.dat");
while(ifs>>a >>b >>c >>d >>e >>f){
 x[m]=a; y[m]=b; exl[m]=c; exh[m]=d; eyl[m]=e; eyh[m]=f;
 m++;}
ifs.close();
gr2 = new TGraphAsymmErrors(m,x,y,exl,exh,eyl,eyh);
gr2->SetMarkerStyle(4);
gr2->SetMarkerSize(2);
gr2->SetMarkerColor(2);
gr2->SetLineColor(2);
gr2->Draw("P");


canv->SetGridx();
canv->SetGridy();
canv->SetLogx();
canv->SetLogy();
```

graph2.cpp の続き

```
leg = new TLegend(0.12,0.12,0.35,0.25,"");
leg.AddEntry(gr1,"data.dat","p");
leg.AddEntry(gr2,"data2.dat","p");
leg.SetTextSize(0.03);
leg.SetBorderSize(1);
leg.SetFillColor(0);
leg.Draw();}
```

このプログラムは座標と誤差の情報が詰まった dat ファイルを複数読み込みグラフ化してくれるものだ。結構無理やり書いている気がするのはご愛嬌で...。ファイル読み込みは ifstream を用いて、変数 a,b,c,d,e,f はバッファとして x[m],y[m],exl[m],exh[m],eyl[m],eyh[m] にそれぞれ代入する。while 文でファイル内のデータを 1 行読み終わるごとに m++; している。グラフは先程と同様にエラーバーを含んだものだ。

```
TAxis *axis = gr1->GetXaxis();
axis->SetLimits(1,1e3);                  // along X
gr1->GetHistogram()->SetMaximum(1e3);    //along
gr1->GetHistogram()->SetMinimum(1);
```

以上ではグラフの座標の幅を設定する。グラフを重ね書きするときは、gr2->Draw("P) と変数 A はいらない。

```
canv->SetGridx();
canv->SetGridy();
canv->SetLogx();
canv->SetLogy();
```

以上ではキャンバスにスケールごとに点線を書き、スケールを log にする。

```
leg = new TLegend(0.12,0.12,0.35,0.25,"test");
leg.AddEntry(gr1,"data.dat","p");
leg.AddEntry(gr2,"data2.dat","p");
leg.SetTextSize(0.03);
leg.SetBorderSize(1);
leg.SetFillColor(0);
leg.Draw();
```

以上は凡例を加えてくれる。TLegend の変数は (double) 右下 x 座標、(double) 右下 y 座標、(double) 左上 x 座標、(double) 左上 y 座標、(char) タイトルで、AddEntry でグラフのポインタを追加する。

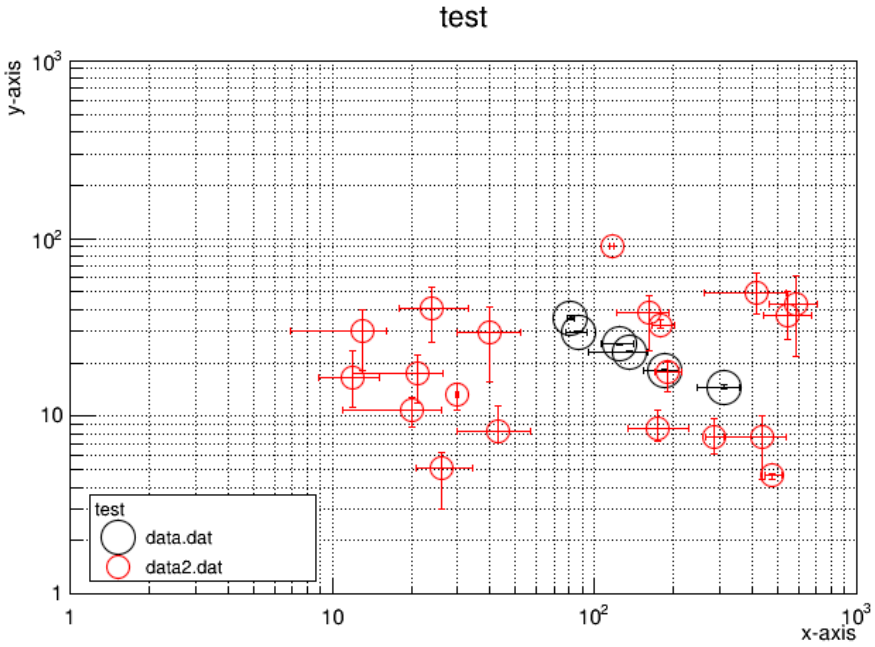

図 9: ROOT plot imaging

5 ROOT ファイル

5.1 input/output

ROOT ファイルは ROOT 専用のバイナリファイルである．もし、拡張子.root のファイル test.root があったら

```
root[] f=TFile("./test.root")
root[] .ls
TFile** ./test.root
 TFile* ./test.root
  KEY: TH2S hist;1 hist
```

コマンドラインで root ファイルを読出し、.ls では root ファイルの中身を確認している．TH2S ということは 2 次元ヒストグラムが格納されていることがわかる．では、実際に root ファイルをつくろう．

```
root[] TFile*f=new TFile("./test.root","recreate")
root[] .x hist3.cpp
root[] .ls
TFile** ./test.root
 TFile* ./test.root
  OBJ: TH2S hist;1 hist
root[] f->Write()
root[] .ls
TFile** ./test.root
 TFile* ./test.root
  OBJ: TH2S hist;1 hist
  KEY: TH2S hist;1 hist
```

TFile の recreate を入れることで書き込み可能になる．コマンド.x はマクロを起動する．OBJ は ROOT を終了すると消えてしまう tmp ファイルであり、保存する必要がある．f->Write() で ROOT ファイルを保存する.KEY と出たら保存完了．ROOT を終了して確認してもらいたい.ROOT には GUI によって ROOT ファイルを確認することも可能．

```
root[] TBrowser tb
```

tb は何でもいい.root ファイルを視覚的に検索できる．

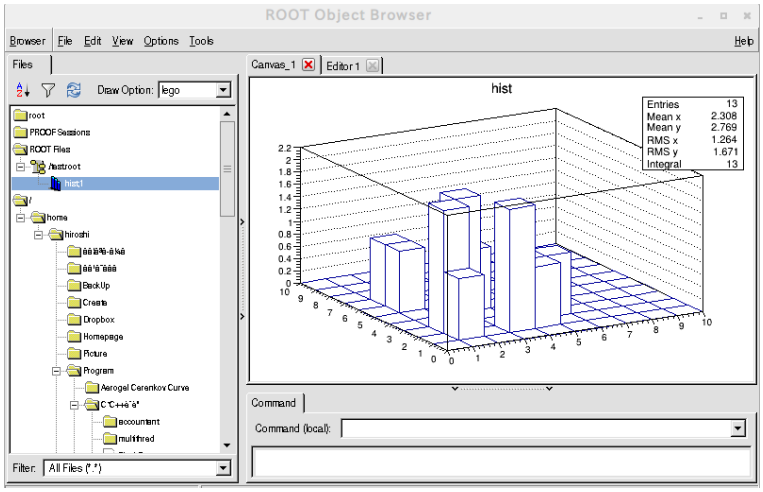

図 10: TBrowser 画面

5.2 TREE

ROOT は n 次元解析の目的で開発されたアプリケーションで、木 (tree) から枝 (branch) が伸びる形をイメージした構造を持っている. 4 次元以上のヒストグラムは 2 次元上には表示が不可能だから射影して解析する. ある変数に条件を指定した場合の解析も可能.

output.dat

```
1 1 2 3
2 1 4 2
0 0 3 1
1 0 3 1
3 4 1 5
1 4 2 1
4 1 5 3
1 0 0 0
2 4 2 0
3 1 4 1
3 1 2 4
```

raw-root.C

```
{
gROOT ->Reset();
gStyle ->SetOptStat(1001110);

TFile* f = new TFile("./test.root","recreate");
TTree* tree = new TTree("t1","t1:raw data");

double x1,x2,x3,x4;

tree->Branch("x1",&x1,"x1/D");
tree->Branch("x2",&x2,"x2/D");
tree->Branch("x3",&x3,"x3/D");
tree->Branch("x4",&x4,"x4/D");

ifstream data ("./output.dat");
while (data >> x1 >> x2>> x3>> x4)tree->Fill();

f->Write();
}
```

gROOT->Recet() はオブジェクトのリセット. ポインタ tree から枝 (Branch) が作られて、4 列で書かれた output.dat の 1 行ずつ 4 成分のベクトル (テンソル) にして詰め込む (Fill). tree->Branch(”x1”,&x1,”x1/D”); の形を見ると、3 つ目の変数の /D は double 型だからと気づく. もちろん int 型なら/I、float 型なら/F にする. 中身を確認しよう.

```
root[] .ls
TFile** ./test.root
 TFile* ./test.root
  OBJ: TTree t1 t1:raw data : 0 at: 0x1cf5530
  KEY: TTree t1;1 t1:raw data
```

```
root[] t1->Print()
******************************************************************************
*Tree    :t1        : t1:raw data                                               *
*Entries :       11 : Total =            2883 bytes  File  Size =        950 *
*        :          : Tree compression factor =   1.41                       *
******************************************************************************
*Br    0 :x1        : x1/D                                                      *
*Entries :       11 : Total  Size=        633 bytes  File Size  =        110 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   1.41     *
*.............................................................................*
*Br    1 :x2        : x2/D                                                      *
*Entries :       11 : Total  Size=        633 bytes  File Size  =        106 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   1.46     *
*.............................................................................*
*Br    2 :x3        : x3/D                                                      *
*Entries :       11 : Total  Size=        633 bytes  File Size  =        111 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   1.40     *
*.............................................................................*
*Br    3 :x4        : x4/D                                                      *
*Entries :       11 : Total  Size=        633 bytes  File Size  =        112 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   1.38     *
*.............................................................................*

root[] t1->Draw("x1")
root[] t1->Draw("x1>>h1")
root[] t1->Draw("x1:x2")
root[] t1->Draw("x1",x2>0)
```

t1->Print() は TTree の枝を確認する．1 次元の射影を見たい場合は t1->Draw(”x1”) とすれば可能．”x1¿¿h1”とすればヒストグラム”h1”に充填してくれる．ヒストグラムのビン数、min,max は自動で決めてくれる．良くも悪くも自動なので、TH1S などでビンを調節してから充填すれば綺麗だ．2 次元の場合は”x1:x2”とする．しかし順番が”y 軸:x 軸”なので注意．

raw-root.C

```
{
 gROOT ->Reset();
 gStyle ->SetOptStat(1001110);

 TFile* f = new TFile("./test.root","recreate");
 TTree* tree = new TTree("t1","t1:raw data");

 double x[4],dummy;
 tree->Branch("x",x,"x[4]/D");
 ifstream ifs ("./output.dat");

 while (!ifs.eof()){
  for(int j=0;j<4;j++)ifs>>x[j];
  tree->Fill();
  }
 f->Write();
}
```

また配列を利用すれば、データ容量を節約することができる. tree->Branch(”x”,x,”x[4]/D”); をみると、配列の場合は変数 2 つ目は & はいらないことがわかる.

```
root[] .ls
TFile** ./test.root
 TFile* ./test.root
  OBJ: TTree t1 t1:raw data : 0 at: 0x1cf5530
  KEY: TTree t1;1 t1:raw data

root[] t1->Print()
******************************************************************************
*Tree    :t1        : t1:raw data                                              *
*Entries :       12 : Total =            1266 bytes  File  Size =        523 *
*        :          : Tree compression factor =   2.73                       *
******************************************************************************
*Br    0 :x         : x[4]/D                                                   *
*Entries :       12 : Total  Size=        930 bytes  File Size  =        165 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   2.73     *
*.............................................................................*
```

今度は枝がたくさんある ROOT ファイルの中から必要な枝だけを選別して新しい ROOT ファイルを作成する.

skim.C

```
{
 gROOT ->Reset();
 gStyle ->SetOptStat(1001110);

 TFile *f = new TFile("./test.root");
 double adc[2],x[2];

 TTree *oldtree = (TTree*) f->Get("t1");
  oldtree->SetBranchAddress("x1",&x[0]);
  oldtree->SetBranchAddress("x2",&x[1]);

 TFile *g = new TFile("test_new.root","recreate");
 TTree * tree = new TTree("t1","t1");
 tree->Branch("adc",adc,"adc[2]/D");

 int LOOP = oldtree->GetEntries();
 for(int i=0; i<LOOP; i++ ){
  oldtree->GetEntry(i);
  evenum=i;
  for(int j=0; j<2; j++)adc[j]=x[j];
  tree->Fill();
 }
 g->Write();
}
```

test.root を参照し、TTree *oldtree = (TTree*) f->Get(”t1”); で TTree オブジェクトを取得、それらの欲しい枝を SetBranchAddress で読み込む. その後新しく test_new.root を作成. 入っている Entries 数の文だけループさせる. 枝の名前も x から adc に変えた.

5.3 example

いくつかの例題は実際に使用された raw data から root.file への変換プログラムである。これはほぼ自分用に覚え書きとしておいておこうと思っている。

5.3.1 raw data の文字が最初の行にある場合

数字だけでは何を示しているかわからないので、最初の行に”通し番号、チャンネル番号、電圧”みたいに書いておく場合、そんな時にでも動くプログラムを考えていこう。

raw-root.C

```
run_number adc1 tdc energy(MeV) momentum(MeV/c)
1 120 2300 305 120
2 119 2200 221 140
3 140 2324 330 142
4 110 0 406 133
5 113 0 334 251
6 124 1903 253 231
7 122 2103 312 103
8 110 0 31 135
9 134 2104 151 332
10 110 2450 412 131
11 105 0 407 184
.
.
.
```