SATOROOT

松本 悟

平成 25 年 10 月 19 日

目 次

0 前書き **3**
0.1 どんな人向け . . . 3
0.2 方針 . . . 3
0.3 作業環境 . . . 3
0.4 お約束事 . . . 3

1 ROOT **4**
1.1 ROOT とは . . . 4
1.2 なぜ ROOT . . . 4
1.3 ROOT のインストール . . . 4

2 ROOT をコマンドラインで使う **6**
2.1 ROOT を起動する . . . 6
2.1.1 ROOT の起動画面を省略する . . . 6
2.1.2 ROOT の起動画面を常に省略する . . . 7
2.1.3 ROOT がデフォルトで読み込むファイル . . . 7
2.2 ROOT を終了する . . . 7

3 ROOT でマクロを読む/実行する方法 **8**
3.1 hello.cpp の実行方法 1 . . . 8
3.2 hello.cpp の実行方法 2 . . . 8
3.3 hello.cpp の実行方法 3(推奨) . . . 8

4 関数を描く **10**
4.1 練習 . . . 11
4.2 解答例 . . . 11

5 ヒストグラムを描く **13**
5.1 練習 . . . 13
5.2 解答例 . . . 14

6 TRandom と TCanvas **16**
6.1 練習 . . . 16
6.2 解答例 . . . 17

7 任意の関数に従うヒストグラムを描く **18**
7.1 練習 . . . 19
7.2 解答例 . . . 20

8 File への出力 **22**
8.1 解答例 . . . 23

9 File からの入力 **24**

10 tree に出会う **25**
10.1 meettree.cpp を実行する . . . 25
10.2 Tree を扱う . . . 26
10.3 Tree からヒストグラムを描く . . . 26
10.4 練習 . . . 27
10.5 解答例 . . . 28

11 Tree から読んで描く **28**
11.1 練習 . . . 28
11.2 解答例 . . . 28

12 ネイティブプログラム **28**

A 名前空間 **29**
A.1 名前空間 std:: . . . 29
A.2 名前空間 TMath:: . . . 29

B ROOT で使う色 **29**

C ROOT で使うスタイル **31**

0　前書き

0.1　どんな人向け

- 実験系の研究室に配属されてデータの解析をする段階になった人
- 先輩に「ROOT 使えるようになっといてね」とか言われちゃった人

そんな人の為の覚え書き。SATOROOT とは、本ドキュメントの前身を私の後輩が作業する為に作成していたディレクトリ名から拝借した名前である。

0.2　方針

とりあえず動かす。C++の細かいお作法とか正しい言葉の使い方とかは無視。とりあえず動かす上で必要なお作法やおまじないについてはその都度紹介したりしなかったりする。とにかく動かせるようにすることを目指す。ただし、自分で調べることにも重きを置くのでサンプルを示したらその都度サンプルをいじる練習問題を提供する。

0.3　作業環境

著者の作業環境は

1. OS X 10.8.5
2. ROOT version 5.34/09

0.4　お約束事

1. $ — プロンプトを表す記号。パソコンがユーザーの入力を受け入れる状態を表す。
2. root[i] — i には数字が入る。コマンドライン上で ROOT を作業している時の行番号である。i を省略することもある。
3. SATOROOT — この覚え書きで使用する全てのファイルは SATOROOT 以下のディレクトリで行う。

1 ROOT

1.1 ROOT とは

ROOT(http://root.cern.ch/drupal/) とは、高エネルギー業界で広く普及している膨大なデータを効率的に扱うためのフレームワークです。C++のお作法でプログラミングします。コマントライン上で ROOT と対話的にプロットやプログラミングを行うことが出来ます。

1.2 なぜ ROOT

世の中のいろんなニーズに応えた結果です。(投げやり)

1.3 ROOT のインストール

ROOT のインストール作業を行う。

- /usr/local/hep/root/5.34.09 — ROOT のライブラリ置き場
- /tmp — コンパイルを実行する時の場所

各ディレクトリの作成

```
$ sudo mkdir -p /usr/local/hep/root/v5.34.09
$ mkdir ~/tmp
```

ROOT のソースコードのダウンロードと展開

```
$ cd ~/tmp
$ sudo wget ftp://root.cern.ch/root/root_v5.34.09.source.tar.gz
$ ls
root_v5.34.09.source.tar.gz
$ sudo tar zxvf root_v5.34.09.source.tar.gz
$ ls
root
root_v5.34.09.source.tar.gz
```

環境変数の定義

```
$ export ROOTSYS=/usr/local/hep/root/v5.34.09
```

インストール作業

```
$ cd root
$ sudo ./configure --prefix=/usr/local/hep/root/v5.34.09
```

以下のコメントが出てくると configure は成功

```
To build ROOT type:

make
make  install
```

指示に従い、make 及び make install を行う。コンパイルする。

```
$ make
```

以下のコメントが出てくると make は成功

```
====================================================================
===                 ROOT BUILD SUCCESSFUL.                       ===
=== Run 'make install' now.                                      ===
====================================================================
```

インストールする。

```
$ su
Password:
$ make install
...
$ exit
```

一時ファイルの削除

```
$ cd ../
$ pwd
~/tmp
$ rm -rf root
```

環境変数ファイルの作成

ROOT 用の環境変数定義を書き込んだ setup.sh を準備して、ホームディレクトリに置く。

setup.sh

```
export ROOTSYS=/usr/local/hep/root/v5.34.09
export PATH=${ROOTSYS}/bin:${PATH}
export LD_LIBRARY_PATH=${ROOTSYS}/lib/root:${LD_LIBRARY_PATH}
```

ホームディレクトリ内のファイル.bash_profile に以下の一文を追加する。

```
source /usr/local/hep/root/setup.sh
```

その後、

```
source .bash_profile
```

2 ROOT をコマンドラインで使う

2.1 ROOT を起動する

```
$ root
```

ROOT の起動画面が立ち上がり、

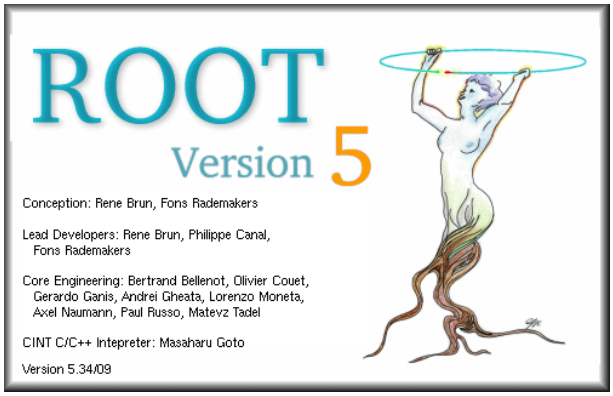


図 1: ROOT の起動画面

```
****************************************
*                                      *
*        W E L C O M E  to  R O O T    *
*                                      *
*   Version   5.34/09      26 June 2013  *
*                                      *
*  You are welcome to visit our Web site  *
*          http://root.cern.ch         *
*                                      *
****************************************

ROOT 5.34/09 (v5-34-09@v5-34-09, Jun 26 2013, 17:10:36 on macosx64)

CINT/ROOT C/C++ Interpreter version 5.18.00, July 2, 2010
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
```

という一連の情報が表示された後、

```
root[0]
```

となって ROOT が立ち上がり操作可能になる。

2.1.1 ROOT の起動画面を省略する

root 起動時に-l オプションをつければ起動画面は省略される。

```
$ root -l
```

2.1.2 ROOT の起動画面を常に省略する

いちいち -l オプションをつけるのが面倒くさい人は bash 起動時に自動的に読み込まれるファイル ~/.bash_profile に次の文章を追加する。

```
alias root="root -l"
```

次に source コマンドを使用すればよい。

```
$ source ~/.bash_profile
```

この状態でも ROOT の起動画面を省略しない場合には

```
$ \root
```

2.1.3 ROOT がデフォルトで読み込むファイル

2.2 ROOT を終了する

```
root[] .q
```

3 ROOT でマクロを読む/実行する方法

ROOT への命令文が書かれたプログラムのソースコード hello.cpp を準備しよう。今はソースコードを理解しなくてもいいが、気になる人は付録 A などを参考にして理解せよ。

hello.cpp

```
#include <iostream>
void hello(){
std::cout << "Hello, world" << std::endl ;
}
```

3.1 hello.cpp の実行方法 1

hello.cpp を ROOT 起動時に実行するには、

```
$ root hello.cpp
```

上記のコマンドを実行すると、

```
root [0]
Processing hello.cpp...
Hello, World
```

3.2 hello.cpp の実行方法 2

別の方法は ROOT をいったん起動して、プログラムを”ロード”して実行するという手段。

```
$ root
root [0] .L hello.cpp
root [1] hello()
Hello, World
```

3.3 hello.cpp の実行方法 3(推奨)

ロードする時に C++コンパイラを通してマクロに含まれるエラーメッセージなどを表記してくれる方法。やり方は’+’ をロードするファイル名の末尾につける。(http://root.cern.ch/drupal/content/compiling-macros)

```
$ root
root [0] .L hello.cpp+
```

例えば hello.cpp の std::endl; のセミコロンが無い時にはどうなるかというと、下のようにエラーメッセージと何が悪いのかをコンパイラが返してくれる。

```
Info in <TUnixSystem::ACLiC>: creating shared library /Users/SATOROOT/hello_cpp.so
In file included from /Users/SATOROOT/hello_cpp_ACLiC_dict.cxx:17:
In file included from /Users/SATOROOT/hello_cpp_ACLiC_dict.h:34:
/Users/SATOROOT/hello.cpp:3:42: error: expected ';' after expression
std::cout << "Hello, World" <<std::endl
^
;
```

```
In file included from /Users/SATOROOT/hello_cpp_ACLiC_dict.cxx:17:
In file included from /Users/SATOROOT/hello_cpp_ACLiC_dict.h:18:
/usr/local/hep/root/v5.34.09/include/root/G__ci.h:971:7:  \
warning: private field 'type' is not used [-Wunused-private-field]
int type;
^
/usr/local/hep/root/v5.34.09/include/root/G__ci.h:972:7: \
warning: private field 'tagnum' is not used [-Wunused-private-field]
int tagnum;
^
/usr/local/hep/root/v5.34.09/include/root/G__ci.h:973:7: \
warning: private field 'typenum' is not used [-Wunused-private-field]
int typenum;
^
/usr/local/hep/root/v5.34.09/include/root/G__ci.h:975:19: warning: \
private field 'isconst' is not used
[-Wunused-private-field]
G__SIGNEDCHAR_T isconst;
^
/usr/local/hep/root/v5.34.09/include/root/G__ci.h:977:29: warning: \
private field 'dummyForCint7' is not used
[-Wunused-private-field]
struct G__DUMMY_FOR_CINT7 dummyForCint7;
^
5 warnings and 1 error generated.
clang: error: no such file or directory: '/Users/SATOROOT/hello_cpp_ACLiC_dict.o'
Error in <ACLiC>: Compilation failed!
```

4 関数を描く

早速、関数を ROOT で関数を描こう。sinfunction.cpp を見てほしい。

sinfunction.cpp

```
#include "TF1.h"
#include "TMath.h"
TF1 *sinfunction(){
TF1 *f = new TF1("f","TMath::Sin(x)") ;
f->Draw() ;
return f ;
}
```

そしてとりあえず実行して欲しい。

```
$ root
root [0] .L sinfunction.cpp+
root [1] sinfunction()
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
(class TF1*)0x7fe64437b580
```

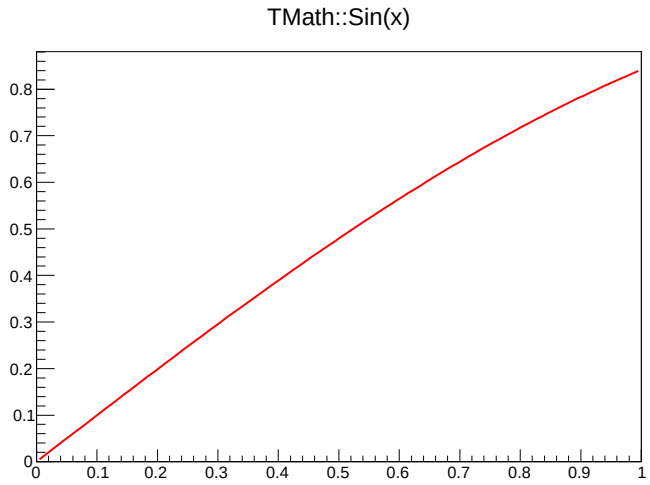


図 2: sinfunction.cpp の実行結果

おそらく、思っていたような定義域でない、軸に名前がついていないなどいろいろな不満があるだろう。それらは全て人間が指定してあげる必要がある。そういった手法も紹介していく。

これまであまりにすっ飛ばしてきたので、sinfunction.cpp を一行目から簡単に見ていこう。(C++のお作法をしっかり学ぶつもりは無いので説明はそれなりの質なので気になる箇所は自分でググるべし)

1. #include "TF1.h"
   #include とは、C++のお作法であってヘッダファイル/ライブラリを読み込むということである。今の場合 ROOT で関数を描く為に必要となるライブラリ"TF1.h"を読み込むという意味

2. #include "TMath.h"
   ROOT で特定の定数 π や e などの値や、関数 $\sin(x)$、$\exp(x)$ などを使用する時に必要となるライブラリ"TMath.h"を読み込む。

3. `TF1 *sinfunction(){`
   マクロの名前を準備するかを記述している箇所である。C++では関数を定義する時に
   ＜型名＞ ＜関数名＞ (＜引数＞){HogeHoge}
   という書き方をする。今の場合だと
   h ＜型名＞ = TF1、＜関数名＞=sinfunction、＜引数＞=無し、といった具合である。暫くの間はマクロ名とファイル名を同じにする。*sinfunction というのは sinfunction という関数のポインタを意味するが、これ以上は触れない。

4. `TF1 *f = new TF1("f","TMath::Sin(x)") ;`
   ここからが sinfunction.cpp の本文。この行の左辺では"TF1"という型のポインタ f を定義している。new とは C++のお作法でメモリを動的に確保する為のものである。TF1("f","TMath::Sin(x)") とは左辺で定義した f というポインタの名前を f として、その間数は TMath というライブラリの中で定義された Sin(x) という関数にするという意味。

5. `f->Draw() ;`
   f というポインタを描く。4 行目のやり方で定義されたポインタ f に対して命令を与える時には->というアロー演算子を用いる。

6. `return f ;`

7. `}`
   3 行目の{に対応する括弧。

4.1 練習

1. 定義域を $-\pi$ から π に変更せよ。

   ヒント http://root.cern.ch/root/html/TF1.html#TF1:TF1@1

2. $2\sin(x/2)$ を描け。

   ヒント http://root.cern.ch/root/html/TF1.html#TopOfPage の B - Expression using variable x with parameters

   ヒント http://root.cern.ch/root/html/TFormula.html#TFormula:SetParameter

3. $\sin(x)$ と $\cos(x)$ を一緒に描け。また $\sin(x)$ の線を赤色、$\cos(x)$ の線を緑色にせよ。

   ヒント http://root.cern.ch/root/html/TF1.html#TF1:Draw

   ヒント http://root.cern.ch/root/html/TAttLine.html#TAttLine:SetLineColor

4.2 解答例

1. 定義域を $-\pi$ から π に変更せよ。

sinfunctionsol1.cpp

```
...
TF1 *sinfunctionsol1(){
double pi = TMath::Pi() ;

TF1 *f = new TF1("f","TMath::Sin(x)", -pi, pi) ;
...
}
```

2. $2\sin(x/2)$ を描け。

sinfunctionsol2.cpp
```
...
TF1 *sinfunctionsol2(){
double pi = TMath::Pi() ;

TF1 *f = new TF1("f","[0]*TMath::Sin([1]*x)", -pi, pi) ;
f->SetParameter(0, 2.) ;
f->SetParameter(1, 0.5) ;
...
}
```

3. $\sin(x)$ と $\cos(x)$ を一緒に描け。また $\sin(x)$ の線を赤色、$\cos(x)$ の線を緑色にせよ。

sinfunctionsol3.cpp
```
#include "TF1.h"
#include "TMath.h"
TF1 *sinfunctionsol3(){
double pi = TMath::Pi() ;

TF1 *f = new TF1("f","TMath::Sin(x)", -pi, pi) ;
TF1 *f2 = new TF1("f2","TMath::Cos(x)", -pi, pi) ;

f->SetLineColor(kRed) ;
f2->SetLineColor(kGreen) ;

f->Draw() ;
f2->Draw("same") ;
return f ;
}
```

5 ヒストグラムを描く

hist1.cpp

```
#include "TH1.h"
#include <iostream>

TH1D *hist1(){
std::cout << "Start!!" << std::endl;
TH1D *h = new TH1D("h","h",100,-5.,5.);
h->FillRandom("gaus") ;
h->Draw();
return h;
}
```

まずは実行してみてほしい。すると、

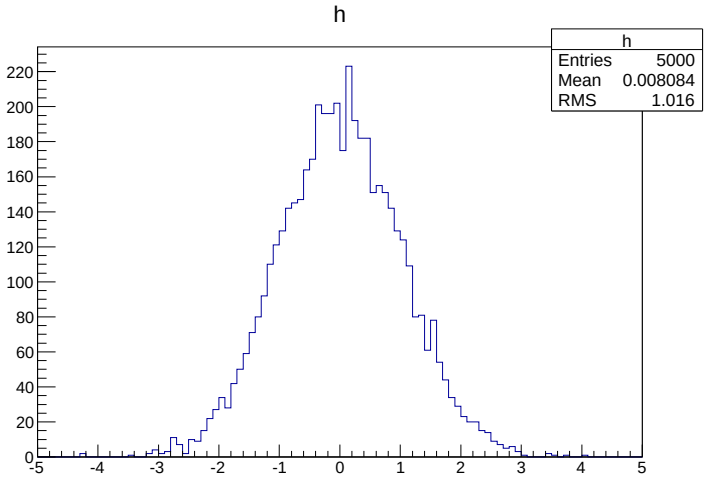


図 3: ヒストグラムの図

5.1 練習

1. ヒストグラムを統計誤差付きで評価せよ

   ヒント http://root.cern.ch/root/html/TH1.html#TH1:Draw

   ヒント http://root.cern.ch/root/html/THistPainter.html#HP01a

   ヒント http://root.cern.ch/root/html/THistPainter.html#HP01b

2. ヒストグラムの最大値を取得せよ。

   ヒント http://root.cern.ch/root/html/TH1.html#TH1:GetMaximum

3. ヒストグラムの最大値が納められた bin の bin 番号を取得せよ。

   ヒント http://root.cern.ch/root/html/TH1.html#TH1:GetMaximumBin

4. ヒストグラムの最大値が納められた bin のエラーの値を取得せよ。

ヒント http://root.cern.ch/root/html/TH1.html#TH1:GetBinError

5. GUI を用いてグリッドを描け。最終的に図 4 のように描け。

   ヒント GUI では< View >から< Editor >を選択する。するとキャンバスの左側に様々な編集ツールが表示される。

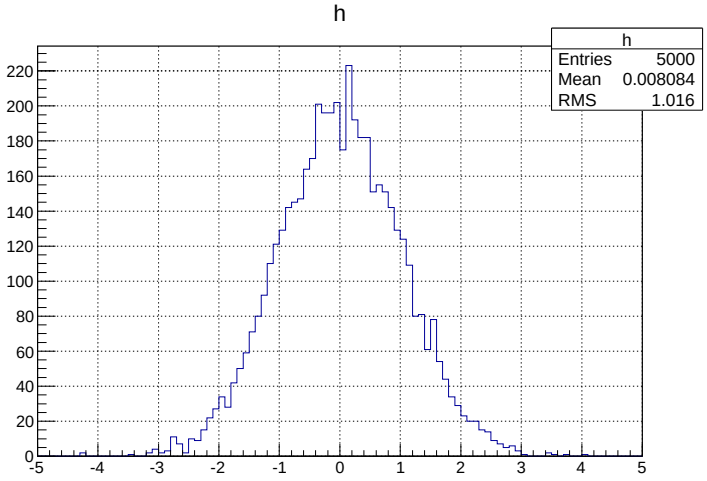


図 4: グリッドを描いたヒストグラム

6. ROOT で h->Draw(); を行ったとき、
   Info in <TCanvas::MakeDefCanvas>: created default TCanvas with name c1
   というメッセージが出ただろう。文字どおり、c1 というキャンバスが作られ、そこに h が描画されている。c1 を”c1.eps”として保存せよ。

   ヒント ROOT のキャンバスなどは全て TObject からの派生である。
   http://root.cern.ch/root/html/TObject.html#TObject:SaveAs

5.2 解答例

1. ヒストグラムを統計誤差付きで評価せよ

hist1sol1.cpp

```
...
TH1D *hist1sol1(){
...
h->Draw("E");
...
}
```

2. ヒストグラムの最大値を取得せよ。

```
root [0] .L hist1.cpp+
root [1] hist1()
Start!!
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
```

```
(class TH1D*)0x7f97b4b7bda0
root [2] h->GetMaximum()
(const Double_t)2.23000000000000000e+02
```

3. ヒストグラムの最大値が納められた bin の bin 番号を取得せよ。

```
root [3] h->GetMaximumBin()
(const Int_t)52
```

4. ヒストグラムの最大値が納められた bin のエラーの値を取得せよ。

```
root [4] int maxbinnum = h->GetMaximumBin()
root [5] h->GetBinError(maxbinnum)
(const Double_t)1.49331845230680784e+01
```

5. GUI を用いてグリッドを描け。最終的に図 4 のように描け。

6. c1 を”c1.eps”として保存せよ。

```
root [6] c1->SaveAs("c1.eps")
Info in <TCanvas::Print>: eps file c1.eps has been created
```

6 TRandom と TCanvas

乱数とキャンバスの操作に出会う。

canran.cpp

```
#include "TCanvas.h"
#include "TH1.h"
#include "TRandom3.h"
#include "TStyle.h"
TCanvas *canran(){
TCanvas *c1 = new TCanvas("c1","c1",600,600) ;
TRandom3 *r = new TRandom3();
TH1D *h = new TH1D("h","h-title;x;y",100,-5,5) ;
for(int i = 0 ; i<100000 ; i++){
h->Fill(r->Uniform(-3.,3.)) ;
}
h->Draw("HE") ;
c1->SaveAs("c1.eps") ;
return c1 ;
}
```

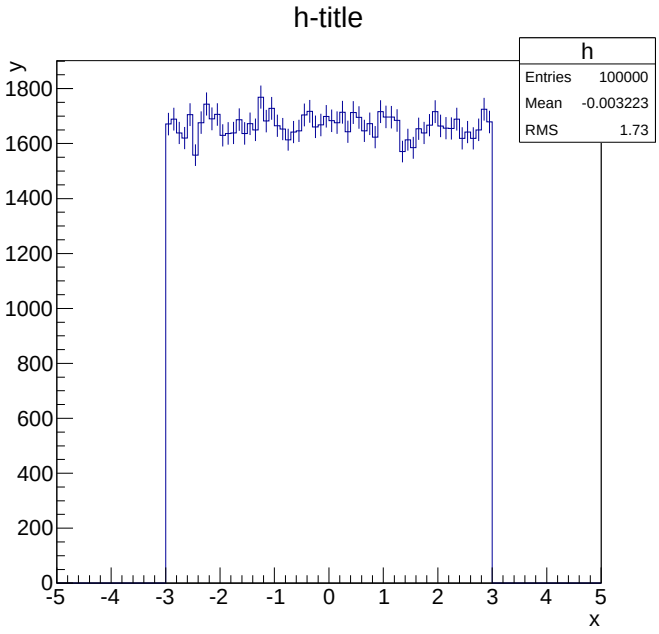


図 5: canran.cpp の実行結果

6.1 練習

1. プログラムの各行を説明せよ。

   ヒント メルセンヌツイスタとはメルセンヌ数という数の特徴を用いた乱数生成子のこと。

   ヒント http://root.cern.ch/root/htmldoc/TRandom.html

ヒント http://root.cern.ch/root/htmldoc/TRandom3.html

2. 今は h->Fill(r->Uniform(-1.,1.)) ; としているが、h->Fill(r->Exp(1.)) ;、h->Fill(r->Gaus(1.,1.)) ;、h->Fill(r->Binomial(3,0.3)) ;、h->Fill(r->PoissonD(1)) ; などとした時の挙動を確かめよ。

ヒント これを気に幾つかの基本的な確率分布について調べよ。

ヒント http://root.cern.ch/root/htmldoc/TRandom.html#TRandom:Exp

ヒント http://root.cern.ch/root/htmldoc/TRandom.html#TRandom:Gaus

ヒント http://root.cern.ch/root/htmldoc/TRandom.html#TRandom:Binomial

ヒント http://root.cern.ch/root/htmldoc/TRandom.html#TRandom:Poisson

3. hist1 を繰り返し実行した時にヒストグラムに変化があるかどうか検証せよ。変化がない場合、この原因を突き止めて実行毎に違うヒストグラムが出来上がるような仕様へ変更せよ。

ヒント http://root.cern.ch/root/htmldoc/TRandom.html#TRandom:TRandom

ヒント http://root.cern.ch/root/htmldoc/TRandom.html#TRandom:SetSeed

6.2 解答例

1. プログラムの各行を説明せよ。

2. 今は h->Fill(r->Uniform(-1.,1.)) ; としているが、h->Fill(r->Exp(1.)) ;、h->Fill(r->Gaus(1.,1.)) ;、h->Fill(r->Binomial(3,0.3)) ;、h->Fill(r->PoissonD(1)) ; などとした時の挙動を確かめよ。

3. hist1 を繰り返し実行した時にヒストグラムに変化があるかどうか検証せよ。変化がない場合、この原因を突き止めて実行毎に違うヒストグラムが出来上がるような仕様へ変更せよ。

canransol1.cpp

```
...
TCanvas *canransol1(){
TCanvas *c1 = new TCanvas("c1","c1",600,600) ;
TRandom3 *r = new TRandom3() ;
r->SetSeed(unsigned (time(NULL))) ;
...
}
```

7 任意の関数に従うヒストグラムを描く

自分で定義した関数に従って乱数を生成してヒストグラムを描こう。

ranfun.cpp

```
#include "TCanvas.h"
#include "TF1.h"
#include "TH1.h"
#include "TMath.h"
TH1D *ranfun(){
double range_min = 0. ; // 0 [ns]
double range_max = 8000.e-9 ;// 8000 [ns]
double tau       = 2.2e-6 ; // it means lifetime
double bgd = 0.5 ; // Background
int nbin = 100 ; // histgram bin num
int imax = 100000 ; // event

TCanvas *c1 = new TCanvas("c1","c1",600,600) ;
c1->SetGrid(1,1) ; // Canvas c1 にグリッドを描く
c1->SetLogy(1) ;  // Canvas c1 の縦軸を log で

TF1 *f = new TF1("f","TMath::Exp(-x/[0])+[1]", range_min, range_max) ;
f->SetParameter(0, tau) ;
f->SetParameter(1, bgd) ;
//  f->SetParameters(tau,bgd) ; // <--上の二行はこの一行と等価

TH1D *h = new TH1D("h","Decay curve (Muon);TDC [s] ; Counts ", nbin, range_min, range_max) ;
for(int i=0; i < imax; i++){
h->Fill(f->GetRandom()) ;
}
h->Draw("HE") ;

/*****************************************************************************\
* Decay Curve Fitting
\*****************************************************************************/
/* If you want to use fit, then please uncomment
gStyle->SetOptFit(1101) ;
TF1 *muon = new TF1("muon","[0]*TMath::Exp(-x/[1])+[2]") ;
muon->SetParameters(2e+3, 2e-6, 1e+2) ;
muon->SetLineColor(kBlue) ;
muon->SetLineWidth(4) ;
h->Fit(muon) ;
*/
return h ;
}
```

7.1　練習

1. プログラムの各行の役割を理解せよ。

   ヒント http://root.cern.ch/root/html/TF1.html#TF1:GetRandom

2. 図 6 のようなおしゃれをしたヒストグラムを描け。

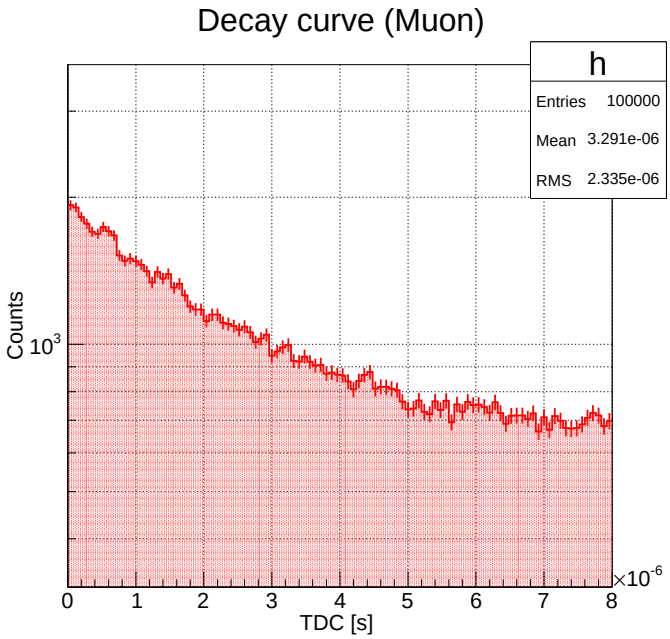


図 6: 各種の設定をいじったヒストグラム

   ヒント http://root.cern.ch/root/html/TH1.html#TH1:GetXaxis

   ヒント http://root.cern.ch/root/htmldoc/TAxis.html#TAxis:CenterTitle

   ヒント http://root.cern.ch/root/html/TH1.html#TH1:SetTitleOffset

   ヒント http://root.cern.ch/root/html/TAttFill.html#TAttFill:SetFillColor

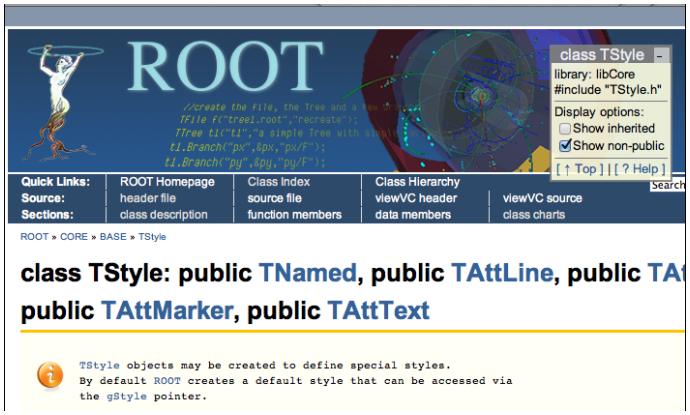
   ヒント http://root.cern.ch/root/html/TAttFill.html#TAttFill:SetFillStyle

3. ranfun.cpp の下部 Decay Curve Fitting 以下のコメントアウトされている箇所/* ... */をアンコメントして実行してみよ。

   ヒント この状態で ranfun.cpp をコンパイルオプション付きでロードしたらエラーメッセージが出るだろう。何を include すべきかはコンパイラが認識し兼ねている単語と’ライブラリ’などの言葉と一緒に検索せよ。それが ROOT に関連しているものであれば各 class を説明しているページの右上に何を include すべきかが書いてある。
   http://root.cern.ch/root/html/TStyle.html

4. フィッティングの情報が誤差付きで描かれるように変更せよ。

   ヒント http://root.cern.ch/root/html/TStyle.html#TStyle:SetOptFit



7.2　解答例

1. プログラムの各行の役割を理解せよ。
2. 図 6 のようなおしゃれをしたヒストグラムを描け。

ranfunsol1.cpp

```
...
TH1D  *ranfunsol1(){
...
h->GetXaxis()->CenterTitle() ;
h->GetYaxis()->CenterTitle() ;
h->GetYaxis()->SetTitleOffset(1.4) ;
h->SetFillStyle(3002) ;
h->SetFillColor(kRed-4) ;
h->SetLineColor(kRed) ;
h->SetLineWidth(2) ;
...
}
```

3. ranfun.cpp の下部 Decay Curve Fitting 以下のコメントアウトされている箇所/* ... */をアンコメントして実行してみよ。
4. フィッティングの情報が誤差付きで描かれるように変更せよ。

ranfunsol2.cpp

```
...
TH1D  *ranfunsol2(){
...
// If you want to use fit, then please uncomment
gStyle->SetOptFit(1111) ;
TF1 *muon = new TF1("muon","[0]*TMath::Exp(-x/[1])+[2]") ;
muon->SetParameters(2e+3, 2e-6, 1e+2) ;
muon->SetLineColor(kBlue) ;
muon->SetLineWidth(4) ;
h->Fit(muon) ;
...
}
```

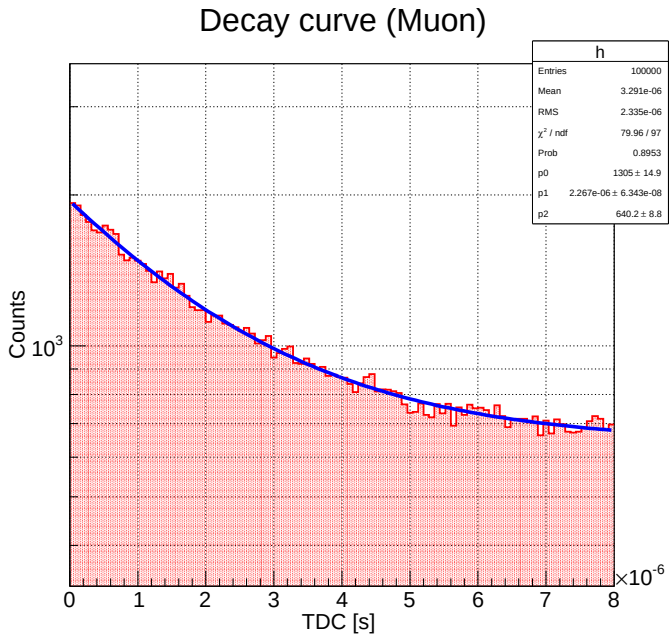


図 7: ranfunsol2.cpp の実行結果

8 File への出力

fileout.cpp

```
#include "TCanvas.h"
#include "TH1.h"
#include "TMath.h"
#include "TRandom3.h"
#include <fstream>
#include <iostream>
TCanvas *fileout(){
std::ofstream outputfile; // file の出力先
outputfile.open("output.plt" );// file を開く

TRandom3 Random(unsigned(time(NULL)));

TCanvas *c1 = new TCanvas("c1","c1") ;

int imax = 100000; // event 数の最大値
double start ; // start の時刻を擬似的に与える。
double stop  ; // stop の時刻を擬似的に与える。
double tdc   ; // TDC でのカウント数を擬似的に与える。
double life = 2.2e-6 ; // Muon の平均寿命を入力する。

TH1D *h1 = new TH1D("h1","h1",200,0,8000e-9) ; // [s]
for(int i = 0; i < imax; i++){
start = Random.Uniform(0., 1.e-9) ; // start の時刻を擬似的に与える。
stop = Random.Exp(life) ; // 指数関数に従った乱数だけ立った時刻を stop とする
stop += start ; // start してから 指数関数に従った乱数だけ立った時刻を stop とする
tdc = stop - start ; // このプログラムだけを見るとこの処理は余分な処理だが、TDC の理の為
std::cout << tdc << " [s] :: " << start << "" << stop << std::endl;
// output という出力先に start と stop をスペース区切りで出力する
outputfile << start << " " << stop << " " << endl ;
h1->Fill(tdc) ; // ヒストグラムに詰める
}
h1->Draw("H") ; // 確認用にヒストグラムを出力する。
outputfile.close(); // ファイルを閉じる return c1 ;
}
```

output.plt

```
4.8318e-10 1.59013e-06
8.88674e-11 2.45466e-06
8.04413e-10 1.65805e-06
...
```

1. プログラムの各行を理解せよ

   ヒント <fstream> なるものを include している。これはファイル操作の時に今回使用するライブラリである。

(今回は使用していないが、ROOT には TFile.h なるクラスも存在する。)

2. 出力先のファイル名を今はマクロ内に直書きしているが、出力先のファイルを output2.plt にしたい時に

```
root[0] .L fileoutsol1.cpp+
root[1] fileout("output2.plt")
```

としたら、自動的に出力先が output2.plt に出来る仕様に変更せよ.

ヒント TCanvas *fileout(){HogeHoge }において引数の中身 (() の中身) は今まで何も書いてきませんでしたが、この中には引数を入れることができます。引数を int 型の数字_intnum にしたいのであれば TCanvas *fileout(int _intnum){HogeHoge }などとして、{}内では_intnum を定義された int 型数字だと扱えばいいだけである。では文字列ではどうすればよいか。

8.1 解答例

1. プログラムの各行を理解せよ

2. 出力先のファイル名を今はマクロ内に直書きしているが、出力先のファイルを output2.plt にしたい時に

```
root[0] .L fileoutsol1.cpp+
root[1] fileout("output2.plt")
```

としたら、自動的に出力先が output2.plt に出来る仕様に変更せよ.

fileoutsol1.cpp

```
...
TCanvas *fileoutsol1(char *outputfilename){
std::ofstream outputfile; // file の出力先
outputfile.open(outputfilename );// file を開く
...
}
```

```
root[0] .L fileoutsol1.cpp+
root[1] fileoutsol1("output2.plt")
```

9 File からの入力

先程出力したファイルからデータを読み込んでヒストグラムを描くことを経験しよう。

filein.cpp

```
#include "TCanvas.h"
#include "TF1.h"
#include "TH1.h"
#include "TMath.h"
#include "TStyle.h"
#include <fstream>
TCanvas *filein(char *file_name){
double range_min = 0. ;
double range_max = 8000.e-9 ;// 8000 [ns]
int nbin = 100 ;
double start ; // TDC start
double stop  ; // TDC stop
double tdc   ; // delta T

TCanvas *c1 = new TCanvas("c1","c1",600,600) ;
c1->SetGrid(1,1); // Canvas c1 にグリッドを描く
c1->SetLogy(1) ;  // Canvas c1 の縦軸を log で
gStyle->SetOptFit(1) ;
TH1D *h = new TH1D(file_name,"Decay curve (Muon);TDC [s] ; Counts ",
nbin, range_min, range_max);
ifstream fin(file_name) ;
while(fin >> start >> stop){
tdc = stop - start ;
h -> Fill(tdc) ;
}
h->Draw("HE");

/******************************************************************************* \
* Decay Curve Fitting
\*******************************************************************************/
TF1 *muon = new TF1("muon","[0]*(TMath::Exp(-x/[1])+[2])") ;
muon->SetParameters(2e+3, 2e-6, 0.5) ;
muon->SetLineColor(kBlue) ;
muon->SetLineWidth(4) ;
h->Fit(muon) ;

return c1 ;
}
```

10 tree に出会う

ROOT には拡張子に.root を拡張子としたファイルがある。(以下 root ファイルと呼ぶ。) ここでは root ファイルにデータを詰める方法とその使い方を示す。やることは

1. start と stop の 2 行が書かれたデータを開く
2. start と stop、及びその差を読んで TDC の値とする。
3. start、stop、TDC の値を Tree というものに格納する。
   http://root.cern.ch/drupal/content/ttree-and-its-data
4. Tree を root ファイルに書き出す。

meettree.cpp

```
#include <fstream>
#include "TFile.h"
#include "TTree.h"

TTree *meettree(char *datafile, char *rootfile = "output.root"){

double start ; // TDC start
double stop  ; // TDC stop
double tdc   ; // delta T

TTree *tree = new TTree("tree","tree");  //TTree 作成
//Branch 準備
tree->Branch( "start", &start, "start/D" ); // start を格納する為のブランチ
tree->Branch( "stop" , &stop , "stop/D" );  // stop を格納する為のブランチ
tree->Branch( "tdc"  , &tdc  , "tdc/D" );   // start と stop の時間差 を格納する為のブランチ

ifstream fin(datafile) ;
while(fin >> start >> stop){
tdc = stop - start ;
tree->Fill() ;
}

TFile *fout = new TFile(rootfile, "recreate");
tree->Write(); // tree を書き込む
fout->Close(); // file close

return tree ;
}
```

10.1 meettree.cpp を実行する

```
$ root
root [0] .L meettree.cpp+
root [1] meettree("output.plt","test.root")
```

```
(class TTree*)0x7fc1d22a1b70
```

すると、作業ディレクトリに test.root というファイルが出来ている。(出力ファイルのデフォルト名は output.root なので、meettree 実行時に第二引数を指定しなかった場合、出来上がる root ファイルは output.root である。) これが root ファイルである。

10.2 Tree を扱う

以降、前節で出力したファイル名が test.root だとして話を進める。test.root に収められている Tree の扱いの走りを紹介する。

```
$ root  test.root
root [0]
Attaching file test.root as _file0...
```

次に、今我々が扱えるものが何かを表示する。

```
root[1] .ls
TFile** test.root
TFile* test.root
KEY: TTree tree;1 tree
```

tree というのが存在するのがわかる。tree の情報を表示するには、Print を使う。

```
root [2] tree->Print()
******************************************************************************
*Tree    :tree      : tree                                                   *
*Entries :   100000 : Total =         4808328 bytes  File  Size =    2406559 *
*        :          : Tree compression factor =   2.00                       *
******************************************************************************
*Br    0 :start     : start/D                                                *
*Entries :   100000 : Total  Size=    1602694 bytes  File Size  =     801872 *
*Baskets :       26 : Basket Size=      32000 bytes  Compression=   2.00     *
*.............................................................................*
*Br    1 :stop      : stop/D                                                 *
*Entries :   100000 : Total  Size=    1602664 bytes  File Size  =     801846 *
*Baskets :       26 : Basket Size=      32000 bytes  Compression=   2.00     *
*.............................................................................*
*Br    2 :tdc       : tdc/D                                                  *
*Entries :   100000 : Total  Size=    1602634 bytes  File Size  =     801820 *
*Baskets :       26 : Basket Size=      32000 bytes  Compression=   2.00     *
*.............................................................................*
```

これらが tree で扱える情報である。

10.3 Tree からヒストグラムを描く

tree に入った情報を書き出すのは簡単である。

```
root [3] tree->Draw("start")
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
root [4] tree->Draw("stop")
```

などとすれば、ヒストグラムが描かれる。(このテキストの `fileout.cpp` でどのような乱数で `start` や `stop` を与えたのかを思い出せ。)

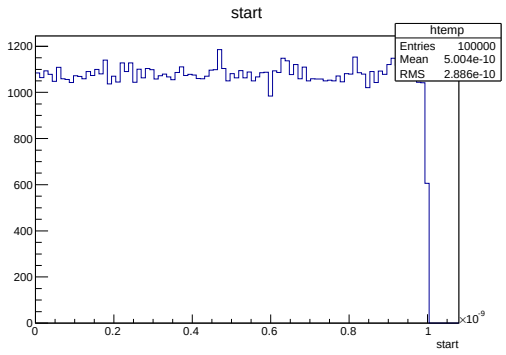


図 8: `tree->Draw("start")` 実行によって表示される `start` のヒストグラム

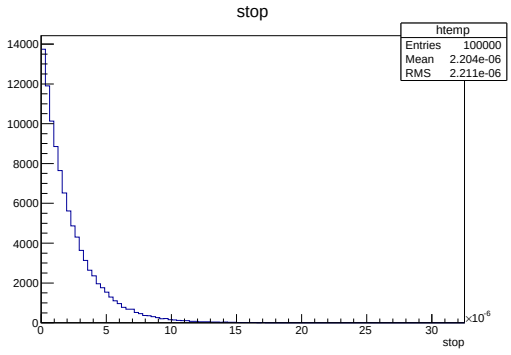


図 9: `tree->Draw("stop")` 実行によって表示される `stop` のヒストグラム

何もしなければ bin 数やヒストグラムの領域は自動で決まるが設定することももちろん出来る。

```
root [5] tree->Draw("start>>h(100, 0., 1e-9)")
```

などとすればわかるだろう。

10.4　練習

1. コマンドライン上でで tdc の値格納用のヒストグラムのを用意した後、TDC のヒストグラムを描け。

   ヒント `http://root.cern.ch/root/html/TTree.html#TTree:Draw@2`

2. tdc を x 軸、start を y 軸とした図 10 のような 2 次元ヒストグラムをコマンドラインから描け。

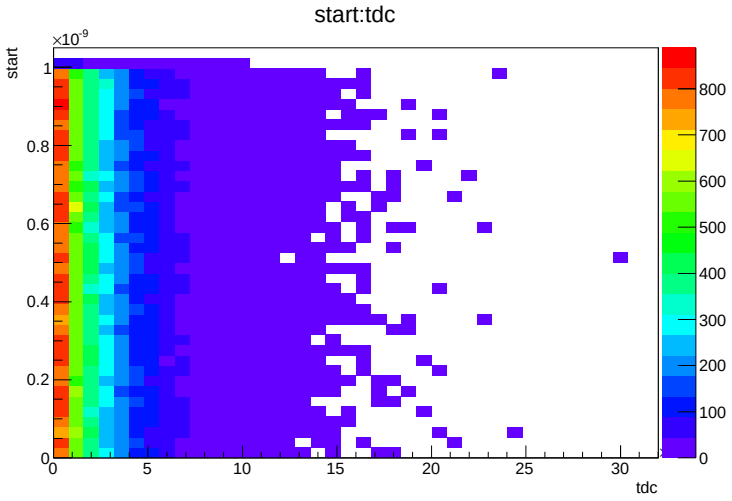


図 10: `start` 実行によって表示される `start` のヒストグラム

ヒント `http://root.cern.ch/root/html/THistPainter.html#HP01c`

10.5 解答例

1. コマンドライン上でで tdc の値格納用のヒストグラムのを用意した後、TDC のヒストグラムを描け。

```
root [] TH1D *h = new TH1D("h", "h", 100, 0., 8000e-9)
root [] tree->Draw("tdc>>h")
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
```

2. tdc を x 軸、start を y 軸とした図 10 のような２次元ヒストグラムをコマンドラインから描け。

```
root [] tree->Draw("start:tdc","","colz")
```

11 Tree から読んで描く

meettree2.cpp

```
#include "TCanvas.h"
#include "TFile.h"
#include "TH1D.h"
#include "TTree.h"

TH1D *meettree2(char *InputRootFileName){

TCanvas *c1 = new TCanvas("c1", "c1") ;
TFile *file = new TFile(InputRootFileName,"READ") ;
TTree *t = (TTree*)file->Get("tree") ;

TH1D *h = new TH1D("h","TDC",100, 0., 8000e-9) ; // 8000[ns]
t->Draw("tdc>>h") ;

c1->cd() ;
h->Draw() ;

return h ;
}
```

11.1 練習

1. プログラムの挙動を理解せよ。

   ヒント http://root.cern.ch/root/html/TFile.html#TFile:TFile@2

   ヒント http://root.cern.ch/root/html/TDirectoryFile.html#TDirectoryFile:Get

2. これまでの知識を動員して、meettree2.cpp を改良せよ。具体的には軸に単位を追加したり、自動的に Fit したりするなどせよ。

11.2 解答例

12 ネイティブプログラム

A 名前空間

「あめ」といった時にそれがどういう内容を表すだろうか？識別する方法としては、

1. 「「気象現象」に属する「あめ」」
2. 「「食べ物」に属する「あめ」」

などとしてしまえば、「あめ」の表す内容は明確になる。この時の「気象現象」や「食べ物」のようなくくりに当たる概念が名前空間である。この時使用した「属する」という言葉を C++ではスコープ演算子"::"で表す。つまり、先程の例を C++風に表現すると下記のようになる。

1. 気象現象::あめ
2. 食べ物::あめ

A.1 名前空間 std::

プログラム中で

```
#include <iostream>
```

と宣言すれば、名前空間 std が使用可能となる。具体的な使い方としては、

```
std::cout << "abc" << std::endl;
```

などである。意味としては、abc という文字列と std::endl という改行命令を std::cout で設定されている標準出力外面へ出力する。

A.2 名前空間 TMath::

プログラム中で

```
#include "TMath.h"
```

と宣言すれば、名前空間 TMath が使用可能となる。http://root.cern.ch/root/html/TMath.html またコマンドライン上で ROOT を使用する時には宣言の必要はない。

```
root [] TMath::C()
(Double_t)2.99792458000000000e+08
root [] TMath::Pi()
(Double_t)3.14159265358979312e+00
root [] TMath::Power(2,3)
(Double_t)8.00000000000000000e+00
root [] TMath::Abs(-2.)
(Double_t)2.00000000000000000e+00
root [] TMath::Sin(1.)
(Double_t)8.41470984807896505e-01
```

などである。上記の入力や引数や返り値の意味することは各自で調べよ。

B ROOT で使う色

http://root.cern.ch/root/html/TAttFill.html#F1

図 11: ROOT で使用できる色

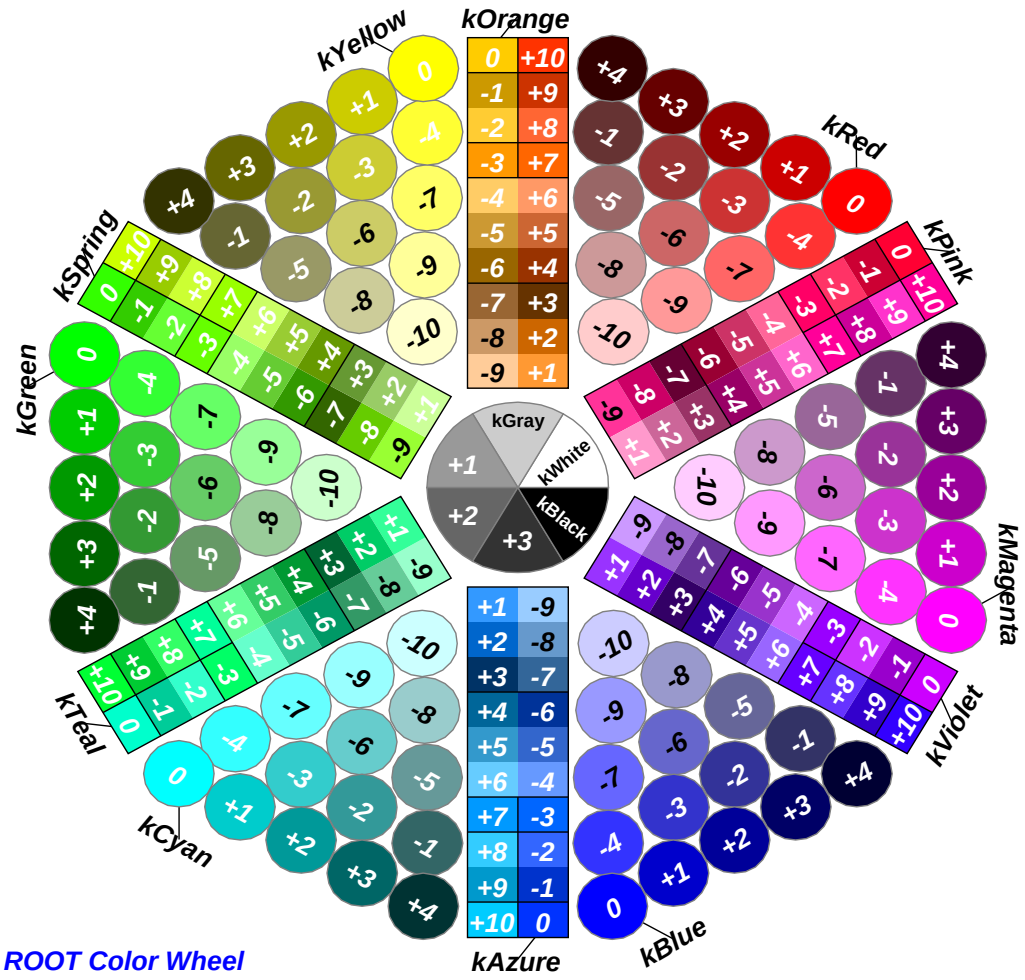


図 12: ROOT で使用できる色

C　ROOT で使うスタイル

http://root.cern.ch/root/html/TAttFill.html#F2

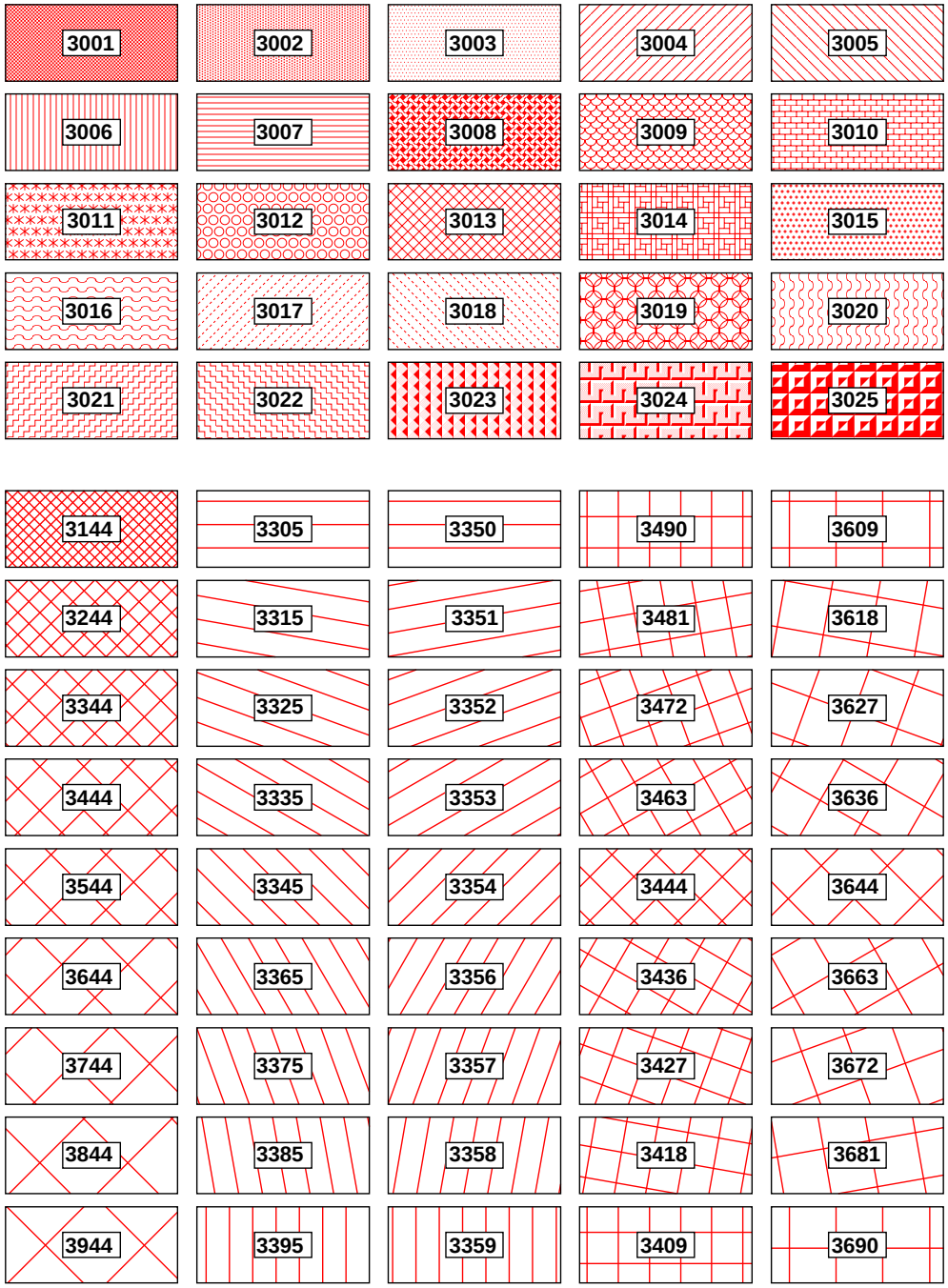


図 13: ROOT で使用できるスタイル