KumaROOT

しょーた

2015 年 1 月 28 日

目次

1 はじめに 3
1.1 ROOT について 3

2 参考サイト 3

3 ROOT を使いたい 4
3.1 オススメのインストール方法（for Mac ユーザ） 4
3.2 ROOT6 を使いたい 6
3.3 ROOT5 と ROOT6 を試してみたい 6
3.4 PyROOT を使いたい 8
3.5 Emacs で ROOT を編集したい 8
3.6 ROOT の tutorial を使いたい 9

4 ROOT tutorial 編 9
4.1 とりあえず起動 10
4.2 とりあえず終了 10
4.3 demos.C を実行してみる 11

5 全体設定編 12
5.1 初期設定したい 12
5.2 キャンバスを無地にしたい 13
5.3 統計情報を表示したい 13

5.4 フィットの結果を表示したい . . . 13
5.5 ヒストグラムの線の太さを一括で変更したい . . . 13
5.6 デフォルトの色を変更したい . . . 14
5.7 横軸に時間を使いたい . . . 14
5.8 キャンバスに補助線を描きたい . . . 15
5.9 グラフの軸を一括してログ表示にする . . . 15
5.10 軸の目盛り間隔を変更したい . . . 15

6 ヒストグラム編 16
6.1 1次元ヒストグラムを作成したい . . . 16
6.2 タイトルを変更したい . . . 17
6.3 統計ボックスを表示したい . . . 17
6.4 X 軸名を設定したい . . . 17
6.5 タイトルを中心にしたい . . . 17
6.6 平均値、RMS を知りたい . . . 17
6.7 値を詰めたい . . . 17
6.8 面積でノーマライズしたい . . . 17

7 TTree 編 17
7.1 テキストファイルを TTree に変換したい . . . 17

8 TChain 編 20

9 TFile 編 20

10 TCanvas 編 20

11 TLegend 編 20

12 TString 編 20
12.1 フォーマット文字列を作りたい . . . 20
12.2 文字列を取り出す . . . 20
12.3 使い方の一例 v . . . 20

13 その他 21
13.1 Emacs . . . 21

13.2 LaTeX 24
13.3 KiNOKO 25
13.4 Geant4 25

1 はじめに

これまで古巣の研究室に設置した DokuWiki に ROOT などの情報を書きためていましたが、そろそろ引っ越しをしなくてはいけないかもしれません。せっかく書きためたのでどうにかして残しておきたいのですが、いまの職場のサーバでは PHP などが一切動かないため、GitHub にて公開することにしました。

GitHub で公開するということで、動作するサンプルコードも作成していこうと思います。

「くま ROOT」と名づけたのは、私が ROOT を使い始めたときに、最初に目を通した「猿にも使える ROOT」（通称：さる ROOT）の知名度にあやかりたかったためです。「さる」の次は「くま」を読んでもらえるように頑張りたいと思います。

ということで、読者は、ちょっとだけ ROOT を使ったことがある学生／研究者が対象になると思います。できるだけ「逆引き辞典」として使えるようにしたいと思います。

1.1 ROOT について

「ROOT」とは高エネルギー物理学業界で広く使われている解析フレームワークです。CERN が中心になって開発されています。

「解析」フレームワークと聞くとたいそうに聞こえますが、お絵かき（＝グラフ作成をそれなりにきれいに出力）ソフトと思えば気構えずに使えると思います。（当たり前ですが「解析」するのは自分ですしね）

あと、ROOT を使うにあたり注意事項は一つです。「決して深入りしないこと」。どうにもならん部分は「仕様」と思って諦めましょう。

2 参考サイト

この KumaROOT に載っている情報は、下記のサイトから集め、自分で試してみたものです。本当に詳しいことは下記の情報を自分で読んでみるのが一番かもしれません。

ROOT 公式ユーザーズガイド　困ったらとりあえず読みましょう！「ROOT User's

Guide」の項目から自分の読みたいドキュメントを選べばよいです。とてーも長いので全部読もうとしてはいけません。必要な時に、必要は箇所を、必要なだけ読むようにしましょう。あと、これを書いているときに気づいたのですが、初心者は「ROOT Primer」に目を通してみると良いかもしれません。

ROOT 公式リファレンスガイド　クラス名やそのメソッド名、内容、使い方を知りたい場合に使いましょう！といっても、このページから辿らなくても、「cern root ttree」[*1]などと Google 検索すればたいていヒットします。「クラスの説明＋メソッド一覧」という構成になっています。最初は読むのに苦労するかもしれませんが、使いこなせるように頑張りましょう。ROOT 中級者はみんな使っています。

ROOT 公式チュートリアル　付属のサンプルコードの説明です。ドキュメントやレファレンスを読むより、実際にコードを動かし、ソースを読むほうが早く身につきます。初めて使うクラスなどはとりあえずサンプルを動かしてみましょう。

ROOT 公式コーディングガイド　ROOT のソースコードを読むときに役に立つかも。なんとなく知っておくと良いです。覚える必要は全くありません。

ROOT 解体新書 by 楠本さん[*2]　gROOT や、gStyle の設定など、他のページ／マニュアルでは見られない項目がすごく詳しい。

宇宙線実験の覚え書き（大学院生版） by 奥村さん　PyROOT のことをググっていてこのページに辿り着きました[*3]。いまは 宇宙線実験の覚え書き（社会人版） にお引っ越ししたみたいです。

3　ROOT を使いたい

3.1　オススメのインストール方法（for Mac ユーザ）

```
$ sudo port install root      ## 実際にはroot5 がインストールされる
```

さて、まずはインストールしないとはじまりません。Mac の場合は MacPorts からインストールできるようになっています[*4]。すごく楽ちんなのでオススメです。環境変数（$ROOTSYS、$LD$_{\text{LIBRARYPATH}}$、$DYLD$_{\text{LIBRARYPATH}}$ など）の設定も不要です。

[*1] 検索の際「cern」と付けるのが重要です。でないと、管理者の意味の「root」がたくさんヒットします。
[*2] ページのリンク切れを確認（*[2015-01-27 Tue]*）
[*3] 青い色のページにお世話になってた気がするんだけど、気のせいかな
[*4] Homebrew は使っていないので分かりません。誰か情報をくださいな

3.1.1 Git を使ったインストール方法

Linux を使っている場合は、Git を使うとよいでしょう。方法は「installing-root-source | ROOT 公式ページ」に載っている通りです。

```
## STEP1 : 先のディレクトリを作成する（ここはお好み）clone
[any]  $ mkdir ~/repos/git
[any]  $ cd ~/repos/git

## STEP2 : 本体をするROOTclone
[git]  $ git clone http://root.cern.ch/git/root.git
[git]  $ cd root

## STEP3 : 目的のバージョンのタグを探す
[root] $ git tag -l

## STEP4 : 目的のバージョンのタグをチェックアウトする
[root] $ git checkout -b v5-34-08 v5-34-08

## STEP5 : を指定しPREFIXconfigure -> make -> make install する
[root(v5-34-08)] $ ./configure --prefix=/usr/local/heplib/ROOT/v5-34-08
[root(v5-34-08)] $ make 2>&1 | tee make.log                    ## 時のログを残すようにするmake
[root(v5-34-08)] $ make install 2>&1 | tee makeinstall.log     ## で設定したにインストールされるconfigureprefix
```

まず、本体をどこかに clone します（今回は、~/repos/git/ 以下）。次にどんなタグがあるのかを調べます。目的のバージョンのタグ名が分かったら、そのタグをチェックアウトします。ブランチ名は好きにしてかまいません（今回は、タグ名と同じ名前）。あとは、従来通り PREFIX を指定して configure します。configure の内容は たしか config.status に書きだされます。make、make install の際、PREFIX で指定したディレクトリによっては sudo が必要です。また、失敗した場合に備えてログを残しておくとよいです。今回は、tee コマンドを使うことで、端末に表示しながらログファイルに保存しています。

あとは、~/.bashrc に環境変数の設定を書いておきます。

3.1.2 従来のインストール方法

ググればたくさん出てきますが、一応紹介しておきます。

```
## STEP1 : ソース（tar.）を保存するディレクトリに移動し、で取ってくるgzwget
```

```
$ cd /usr/local/heplib/tarballs
$ wget ftp://root.cern.ch/root/root_v5.30.06.source.tar.gz     ## は公式から探してくるURLROOTWEB

## STEP2 : ソース（tar.）を展開するgz
[ROOT] $ cd /usr/loca/heplib/ROOT
[ROOT] $ tar zxvf ../tarballs/root_v5.30.06.source.tar.gz     ## 先ほどしたファイルを展開wget

## STEP3 : を指定しPREFIXconfigure -> make -> make install
[ROOT] $ cd root
[root] $ ./configure --prefix=/usr/local/heplib/ROOT/v5-30-06
[root] $ make 2>&1 | tee make.log
[root] $ make install 2>&1 | tee makeinstall.log
```

3.1.3 インストール方法 for Windows ユーザ

Windows はよく分かりません。ごめんなさい。たぶん「downloading-root | ROOT 公式ページ」から目的のバージョンを選び、そこからバイナリを落としてくるのが一番簡単だと思います。

3.2 ROOT6 を使いたい

```
$ sudo port install root6
```

2014 年に ROOT6 がリリースされました。MacPorts の場合「root6」というパッケージ名[*5]でインストールすることができます。

ただし、現在遂行中の実験は ROOT5 系を使っていることが多いです[*6]。バージョンを変えてしまうとうまく動作しないこともあるので、実験で推奨されているバージョンには従ってください。

3.3 ROOT5 と ROOT6 を試してみたい

[*5] 正確には「ポート名」かも。あまり気にしないでください。

[*6] LHC 実験の要求に応えるため（？）、最近は ROOT の開発が非常に活発に行われています。しかし、その他の実験では、安定性を求め古いバージョン使い続けていることがあります。あと OS が古いと最新バージョンはうまくインストールできないこともあります

```
$ sudo port select --set root root6    ## にするROOT6
$ sudo port select --set root root5    ## にするROOT5
```

MacPorts で ROOT をインストールする利点は、ROOT5 と ROOT6 が簡単に切り替えられることです。

実はこの「port select」は ROOT だけでなく、Python のバージョン切り替えなどもできます。どのパッケージが使えるかは以下のコマンドで確認できます

```
$ port select --summary
```

3.3.1 ROOT5 と ROOT6 の違いについて

ROOT マクロなどを実行する際に使うインタプリタが変更されたみたいです[*7]。細かい違いは全く分かりませんが、文法のチェックが厳密になったみたいです。

実は ROOT5 では C 言語／ C++ 言語の文法的には間違っているマクロでも動いてくれました[*8]。そのため、テストで作ったマクロで動作確認した後、より多くのデータを解析するためにコンパイルするとエラーが多出。そのデバッグに追われるということは日常茶飯事でした。

ROOT6 では、このマクロの文法チェックも厳しくなったみたいです。ひぇぇ。でも心配しなくて大丈夫。エラーの内容を詳しく教えてくれるようになりました。よくある行末のセミコロンのつけ忘れなども指摘してくれます。これで場所の分からない segmentation fault に悩まされることも減るでしょう。

試しに、ROOT5 のチュートリアルを ROOT6 で実行してみてください。「warning」や「error」がたくさん表示されます。

```
## STEP1 : を指定ROOT6
$ sudo port select root root6

## STEP2 : ## のROOT5tutorialsに移動/
$ cd /opt/local/libexec/root5/share/doc/root/tutorials/

## STEP3 : (ROOT=)を起動ROOT6
$ root
```

*7 CINT → CINT++ に変更

*8 よく知られていると思われるのは、a.b でも a->b でも動いちゃうことでしょうか

```
## ===  warning の例 ===
/opt/local/libexec/root5/share/doc/root/tutorials/rootalias.C:7:13:
 warning: using the result of an assignment as a condition without
 parentheses [-Wparentheses]
      if (e = getenv("EDITOR"))
          ~~^~~~~~~~~~~~~~~~~~

## === error の例 ===

/opt/local/libexec/root5/share/doc/root/tutorials/rootalias.C:39:12
 error: cannot initialize return object of type 'char *' with an rvalue of
 type 'const char *'
    return gSystem->WorkingDirectory();
           ^~~~~~~~~~~~~~~~~~~~~~~~~~~


$ .q
```

3.4 PyROOT を使いたい

```
$ sudo port install root5 +python27    ## の場合、ROOT5+pythonXX の
 variants が必要
$ sudo port install root6              ## の場合、ROOT6variants 不要
```

CERN には「へびつかい」が多いらしく、「PyROOT[*9]」というモジュールを使えば、Python 上で ROOT が使えるようになっています。

その場合は、MacPorts でインストールする際に variants で指定する必要があります。しかも、この variants は自分の使っている Python のバージョンに合わせる必要があります。ミスマッチな場合は、動作しません（＝クラッシュします)。

ROOT6 の場合は python27 がデフォルトで ON になっています。

3.5 Emacs で ROOT を編集したい

```
$ locate root-help.el     # どこにあるかを探す
```

[*9] 実は rootpy というのもあります。こっちのほうが Python native な感じです。前に試そうとしてたのですがインストールでコケてしまいました。動かせたら項目を作るかも

これもあまり知られていないと思うのですが、Emacs 上で ROOT のソースを編集するのを簡単にする Elisp パッケージが一緒にインストールされます。locate コマンドでどこにあるか調べておきましょう。

ちなみに、僕の場合（= MacPorts の場合)、以下にありました。

```
/opt/local/libexec/root5/share/emacs/site-lisp/root-help.el
/opt/local/libexec/root6/share/emacs/site-lisp/root-help.el
```

これの使い方に関しては、あとできちんと調べて書くことにします。

3.6 ROOT の tutorial を使いたい

```
/opt/local/libexec/root5/share/doc/root/tutorials/      ## の 場 合
ROOT5
/opt/local/libexec/root6/share/doc/root/tutorials/      ## の 場 合
ROOT6
```

実は ROOT をインストールすると、たくさんのサンプルコードもついてきます。使い方をウェブで検索してもよく分からない場合は、このサンプルコードを動かしながら中身をいじくってみるのが一番です。

とりあえず、いつでも使えるようにテスト用ディレクトリを作成しコピーしておきましょう。以下に一例を示しましたが、自分の環境に合わせて適宜変更してください。

```
$ cp -r /opt/local/libexec/root5/share/doc/root/tutorials ~/TEST/root5/
$ cp -r /opt/local/libexec/root6/share/doc/root/tutorials ~/TEST/root6/
```

cp コマンドを使う際には、-r オプションを付けることでサブディレクトリもコピーできます。その際、コピー元（＝第 1 引数）の最後に「/」付けてはいけません。コピー先（＝第 2 引数）の最後には「/」を付けてもよいです（もしかしたらなくてもよいのかも)[*10]。

4 ROOT tutorial 編

この章では ROOT に付属している tutorial の使い方を簡単に紹介します。前節の最後にも書きましたが、手元にコピーを作っておきましょう。

```
$ cp -r /opt/local/libexec/root6/share/doc/root/tutorials ~/TEST/root6/
```

[*10] この辺はよく忘れます。失敗したらコピー先を削除すればいいだけなので、失敗もしてみてください

とりあえず ROOT6 の tutorial を使います。気が向いたら ROOT5 との比較もしようかと思います。

4.1 とりあえず起動

```
$ cd ~/TEST/root6/tutorials/
$ root

Welcome to the ROOT tutorials

Type ".x demos.C" to get a toolbar from which to execute the demos

Type ".x demoshelp.C" to see the help window

==> Many tutorials use the file hsimple.root produced by hsimple.C
==> It is recommended to execute hsimple.C before any other script

root [0]
```

tutorials をコピーしたディレクトリで ROOT を起動すると、上のメッセージが表示されます。これは同じディレクトリに rootlogon.C というのがあるからです。

ROOT を起動すると、"system.rootrc"、"~/.rootrc"、"./rootlogon.C" の順番に設定が読み込まれます。なので、個人的な全体設定は"~/.rootrc"へ、そのプログラムだけの設定は"./rootlogon.C"に書いておけばよいです。

4.2 とりあえず終了

```
root [0] .q

Taking a break from ROOT? Hope to see you back!
```

さて、さきほど ROOT セッションを終了してみましょう。ROOT セッション内で「.q」を入力します[*11]。するとまたまたメッセージが表示されます。これは./rootlogoff.C があるからです。

[*11] 自作マクロが途中で止まってしまい、.q では抜けれないことがあります。そんなときは .qqq のように q をたくさん入力してみましょう。それでもダメな場合は、Ctrl-z で一時中断してプロセスを kill します（「$ ps aux | grep root」で PID を調べ、「$ kill -9 PID」で強制終了）

このようにして、ROOT 起動時および終了時の動作を細かく設定することができます。個人的には、数ヶ月ぶりに触るプログラムなんてほとんど忘れてしまっているので、rootlogon.C に手順を書いて残したりしています。

4.3 demos.C を実行してみる

さて、ROOT を起動して表示されたメッセージにしたがって、demos.C を実行してみましょう。ROOT 内で実行する場合は、「.x ファイル名」と入力します。ファイル名の部分は TAB 補完ができます。これを bash で実行する場合は以下のようにします。

```
$ root demos.C
```

さてさて、実行すると図 1 のようなツールバーが出てきます。

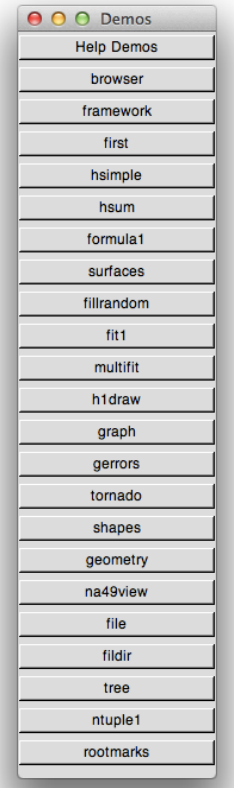

図 1　demos.C を実行した時に出てくるツールバー的なもの

そして、一番上にある、「Help Demos」をクリックすると、図 2 のようなキャンバスが表示されます。

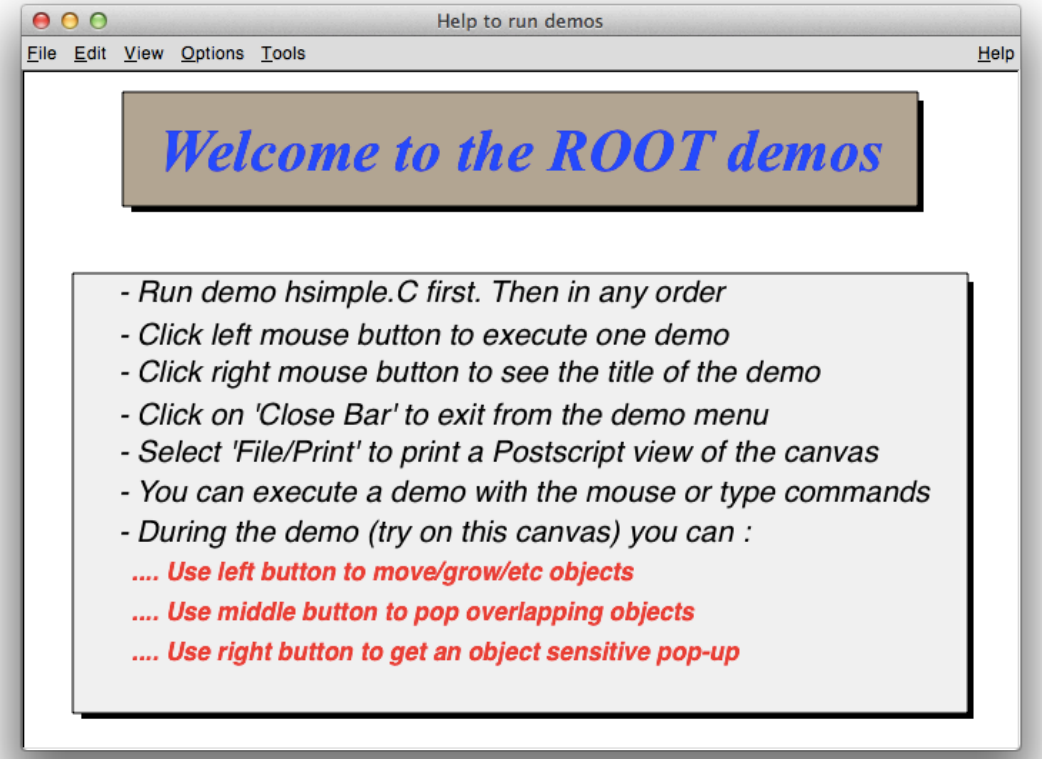

図 2　Help Demos を実行すると出てくるキャンバス

5　全体設定編

5.1　初期設定したい

5.1.1　rootrc

bash の設定を ~/.bashrc に書くように、ROOT の設定は ~/.rootrc に書きます。デフォルト値は、{ROOT をインストールしたパス}/etc/system.rootrc に書かれているので、とりあえずこれをホームディレクトリにコピーして編集したら OK です。

```
$ locate system.root
# $ROOTSYS/etc/system.rootrc
```

```
$ cp $ROOTSYS/etc/system.rootrc ~/.rootrc
```

5.1.2 rootlogon.C

5.2 キャンバスを無地にしたい

```
gROOT->SetStyle("Plain");
```

ROOT v5.30 からキャンバスの色がデフォルトで無地になりました。なので、それ以降のバージョンを使っている場合、特に設定は必要ありません。

(いないと思いますが) 昔のキャンバス (=灰色っぽいやつ) を使いたい場合は、"Classic" を指定します。

5.3 統計情報を表示したい

ヒストグラムを描画すると、右上にそのヒストグラムの情報が表示されます。デフォルトだと 3 つしか表示されないので、少し増やしておきます。

```
gStyle->SetOptStat(112211)
```

引数はビットのようなものを表しています。このビットは右から読まれます。最大で 9 くらいまでいける気がする。0 or 書かなければ「非表示」、1 は「表示」、2 は「エラー表示」です。

5.4 フィットの結果を表示したい

```
gStyle->SetOptFit(1111111)
```

ビットの使い方は、ひとつ前の「統計情報を表示したい」と同じです。

5.5 ヒストグラムの線の太さを一括で変更したい

ヒストグラムの外枠線の太さは、一括で設定しておくことができます。デフォルトだと少し細い気がするので、太くしておくとよいと思います。ただし、たくさんのヒストグラムを描く際は、見えにくくなってしまうので細くします。その辺りは臨機応変にお願いします。

```
gStyle->SetHistLineWidth(2)
```

5.6 デフォルトの色を変更したい

```
gROOT->GetColor(3)->SetRGB(0., 0.7, 0.); // Green  (0, 1, 0)->(0, 0.7, 0)
gROOT->GetColor(5)->SetRGB(1., 0.5, 0.); // Yellow (1, 1, 0)->(1, 0.5, 0)
gROOT->GetColor(7)->SetRGB(0.6, 0.3, 0.6); // Cyan (0, 1, 1)->(0.6, 0.3, 0.6)
```

デフォルトは（1:黒, 2:赤, 3:黄, 4:青, 5:黄緑, 6:マゼンダ, 7:シアン）なのですが、この中で、(3:黄, 5:黄緑, 7:シアン）は明るすぎてとても見えづらいので、もう少し見やすい色に変更します。

上２つは奥村さんのページのコピペ、最後のはシアンを紫っぽい色に変更しました。

RGB の度合いは自分の好みで選んでください。手順としては、RGB の値を検索（Wikipedia 使用すると良い）→ その値を 256（ほんとは 255 かも？）で割るだけです。

おまけとして、ROOT 公式ブログの「虹色カラーマップを使うこと」 の記事もリンクしておきます。

5.7 横軸に時間を使いたい

```
gStyle->SetTimeOffset(-788918400);      // 時間と時間のの差UnixROOTepoch
graph->GetXaxis()->SetTimeDisplay(1);
graph->GetXaxis()->SetTimeFormat("%Y\/%m\/%d");
graph->GetXaxis()->SetTimeOffset(0, "gmt");      // GMTに設定+0
```

Unix の epoch time は 1970 年 01 月 01 日 00 時 00 分 00 秒から始まるのに対し、ROOT の epoch time は 1995 年 01 月 01 日 00 時 00 分 00 秒から始まるので、その差をオフセットとして設定する必要がある。

5.7.1 Unix epoch と ROOTepoch の差を計算する

簡単な計算なので確かめてみる

```
 |年
25[] * 日年365[/] * 時間日24[/] * 分時間60[/] * 秒分60[/]
+ 日6[] * 時間日24[/] * 分時間60[/] * 秒分60[/]      // この年間に閏年は回256
= 秒788918400[]
```

5.7.2 GMT+0 に設定する

```
graph->GetXaxis()->SetTimeOffset(0, "gmt");
```

理由は忘れてしまったが、上の設定をしないと軸の時間がずれてしまってたはず。epoch の時間ではなく、作成したグラフ／ヒストグラムの軸に対して設定する

5.7.3 月日と時刻を 2 段にして表示したい

```
graph->GetXaxis()->SetTimeFormat("#splitline{/%m\/%d}{%H:%M}");
```

時間に対する安定性を示したい場合などに使える。

5.8 キャンバスに補助線を描きたい

```
gStyle->SetPadGridX(1)      // 軸のグリッドX
gStyle->SetPadGridY(1)      // 軸のグリッドY
```

5.9 グラフの軸を一括してログ表示にする

```
gStyle->SetOptLogx(1)       // 軸の目盛をログ表示X
gStyle->SetOptLogy(1)       // 軸の目盛をログ表示Y
```

5.10 軸の目盛り間隔を変更したい

```
gStyle->SetNdivisions(TTSSPP)
```

PP　軸全体の分割数
SS　PP 分割された目盛り 1 つ分の分割数
TT　SS 分割された目盛り 1 つ分の分割数

デフォルトは 510 になっている。PP=10、SS=05、TT=00 なので、軸を 10 分割してその 1 目盛りを 5 分割、ということで全体で 50 目盛りになる。

全体を 100 目盛りにするには、20510 にすればよい。(10 分割、その 1 目盛りを 5 分割、さらにその 1 目盛りを 2 分割 ＝ 100 目盛り)

6 ヒストグラム編

6.1 1次元ヒストグラムを作成したい

```
TString hname, htitle;
hname.Form("hname");     // <------------------------ ヒストグラムの名前
htitle.Form("title;xtitle;ytitle;");     // <-------- タイトル、軸名
Double_t xmin = 0, xmax = 10;     // <--------------- 左端、右端
Int_t xbin = (Int_t)xmax - (Int_t)xmin;     // <----- ビン数

TH1D *h1 = new TH1D(hname.Data(), htitle.Data(), xbin, xmin, xmax);
```

- ヒストグラムに限らずROOTオブジェクトには「名前」をセットする必要がある
- タイトル部分を「;」で区切ることで、軸名を設定することができる（"タイトル;X軸名;Y軸名前"）
- TString::Formはprintfの書式が使えるのでとても便利

6.2 タイトルを変更したい

6.3 統計ボックスを表示したい

6.4 X 軸名を設定したい

6.5 タイトルを中心にしたい

6.6 平均値、RMS を知りたい

6.7 値を詰めたい

6.8 面積でノーマライズしたい

7 TTree 編

ROOT を使うにあたって、TTree（もしくは次の章の TChain）は基礎中の基礎です[*12]。とりあえず、取得したデータはさっさと TTree に変換してしまいましょう。

7.1 テキストファイルを TTree に変換したい

取得したデータはテキストデータとして保存するのが、一番簡単な方法です。

仮に、100 行 4 列のテキストファイルがあるとします。このファイルの「行数」はイベント数に相当し、「列数」は取得したデータの項目に相当します。

```
100     105     104     103
101     106     103     100
...
```

7.1.1 TTree::ReadFile() を使う方法

データがテキストファイルで保存されている場合、それを TTree に変換する最も簡単な方法です。

```
{
```

[*12] 「さる ROOT」や、他のウェブサイトでは「TNtuple」をサンプルとして取り上げていますが、これだけを使っている研究者はみたことがありません。TTree のベースには TNtuple があるのかもしれませんが、なんでこんな使われていないものをサンプルにするのか疑問です

```
  // STEP1: 入力ファイル名を指定する
  TString ifn = "inputfilename";

  // STEP2: を作成するTTree
  TTree *tree = new TTree("tree", "tree using ReadFile()");

  // STEP3: TTree::ReadFile(...) でデータを読み込む
  tree->ReadFile(ifn.Data(), "row1/I:row2/I:row3/I:row4/D:row5/I");

  // STEP4: 作成したを保存するためのを作成するTTreeTFile
  TString ofn = "out.root";
  TFile *fout = new TFile(ofn, "recreate");

  // STEP5: にを書き込むTFileTTree
  tree->Write();

  // STEP6: を閉じるTFile
  fout->Close();

  return;
}
```

TTree::ReadFile の第 1 引数には入力ファイル名（ifn)、第 2 引数には、branch descriptor を指定します。

「branch descriptor」は、TTree のブランチ変数になります。複数のブランチ変数を指定する場合は、コロン（:）で区切って記述します。$\mathrm{Int_t}$ 型 の場合は「ブランチ名/I」、$\mathrm{Double_t}$ 型の場合は「ブランチ名/D」といった感じで、その変数名（＝ブランチ名）とその型を指定できます。型を省略した場合は $\mathrm{Float_t}$ 型の「ブランチ名/F」になるみたいです。

7.1.2 TTree::Branch() を使う方法

よくある方法です。ググればいっぱい見つかります。

```
{
    // STEP1: データファイルを読み込む
    TString ifn = "inputfilename"
    ifstream fin;
    fin.open(ifn);

    // STEP2: データを格納するための変数を定義する
```

```
    int val1, val2, val3, val4;

    // STEP3: を作成するTTree
    TTree *tree = new TTree("name", "title);

    // STEP4: TTree::Branchを使って、各変数のブランチを作成する(...)
    // 第一引数：ブランチ名；なんでも良い；用意した変数名と違っていても構わない
    // 第二引数：変数のアドレス；変数が実体の場合は、を先頭につけてアドレスを指定する；事前に変数を用意しておかないと怒られる&
    // 第三引数：変数の型；"変数／型"の形で記述する；型はintI, 型はfloatF, 型はなどdoubleF
    tree->Branch("val1", &val1, "val1/I");
    tree->Branch("val2", &val2, "val2/I");
    tree->Branch("val3", &val3, "val3/I");
    tree->Branch("val4", &val4, "val4/I");

    // STEP5: Cでファイルを読み込むときの常套手段++
    while (fin >> val1 >> val2 >> val3 >> val4) {
        // STEP6: データのエントリの区切りで必ずTTree::Fillする()
        tree->Fill();
    }

    // STEP7: 作成したを保存するためのを作成するTTreeTFile
    TString ofn = "outputfilename";
    TFile *fout = new TFile(ofn, "recreate");

    // STEP8: にを書き込むTFileTTree
    tree->Write();  //

    // STEP9: を閉じるTFile
    // プログラム（やマクロ）終了時に勝手に閉じてくれるらしいが一応
    fout->Close();

    return;
}
```

前述した ReadFile を使った方法と比べると、コードの行数がぐーんと多いことが分かります。(ReadFile の場合、肝となる部分はたったの一行です)。

行数が増えた分、汎用性が高くなっています。こちらの方法だと、ブランチに「配列」を設定することも可能です。

8 TChain 編

9 TFile 編

10 TCanvas 編

11 TLegend 編

12 TString 編

C/C++ では文字とか文字列の扱いは面倒くさいのですが、ROOT には TString という便利なクラスがあります。使わない手はないでしょう、ということで紹介しておきます。

12.1 フォーマット文字列を作りたい

```
TString str;
str.Form("Hist%d", i);
```

12.2 文字列を取り出す

```
str.Data();
```

12.3 使い方の一例 v

複数のヒストグラムをループで生成したいときなどによく使います。

```
const Int_t nhist = 10;
TString hname, htitle;
for (Int_t i = 0; i < nhist; i++) {
    hname.Form("h%02d", i);
    htitle.Form("%s;%s;%s", hname.Data(), "x", "y");
    h[i] = new TH1D(hname.Data(), htitle.Data(), xbin, xmin, xmax);
}
```

13 その他

ここでは、ROOT 以外の研究で役に立ちそうなことについてまとめます。

13.1 Emacs

最初に断っておくと、特に Emacs 信者というつもりはないのです。修士でプログラミングを始めた時に、最初に使ったのが Emacs だったというだけの理由[*13]です。Vim も基本操作はできるようにしといた方がよいです。

13.1.1 ページ操作

Emacs	vim	less	操作内容
C-n	j, RET	j, RET	次の行
C-p	k	k	前の行
C-v	C-f	SPC	1 ページ進む
M-v	C-b	S-SPC	1 ページ戻る
M-<	gg	g	ファイルの先頭
M->	G	G	ファイルの最後
M-g g 数値	数値 G	:数値	指定した数値の行へジャンプ
		d	半ページ進む
		u	半ページ戻る
C-x C-c	:q, :q!	q	ファイルを閉じる

13.1.2 エディタ共通

Emacs	vim	操作内容
C-f	l, SPC	次の文字
C-b	h	前の文字
M-f	w, e	次の単語[*14]
M-b	b	前の単語
C-a	0	行頭

[*13] Mac の Cocoa アプリも Emacs キーバインドで使えるのも大きな理由だったかもです

	^	文頭（行頭にある文字）
C-e	$	行末
C-i		タブ（インデント？）
C-l		画面の移動（上-中-下）
C-x C-s	:w	ファイルを保存
C-x C-w	:w ファイル名	ファイル名を指定して保存
C-x C-i	:r ファイル名	ファイル名の中身を挿入
C-d	x	カーソルの下の文字を削除（Delete）
C-h[*15]	Backspace	カーソルの左の文字を削除（Backspace）
C-k	d$	カーソルの位置から行末までを切り取り
C-w	d$, dd, dw	選択範囲を切り取り
	dd	一行削除（切り取り）
	dw	1 単語を切り取り
	d$, d^, d0	それぞれ切り取り
M-w	y	選択範囲をコピー（yank）
	yy	一行コピー（yank）
	yw	1 単語をコピー（yank）
	y$, y^, y0	それぞれコピー（yank）
C-y	p	貼り付け
C-s	/文字, n, C-i	前方検索 [*16]
C-r	?文字, N, C-o	後方検索
C-@	v	マーカーのセット
M-%	:s/old/new	現在行の最初の文字を置換（old -> new）
	:s/old/new/g	現在行のすべての文字を置換（old -> new）
	:%s/old/new/gc	ファイル全体のすべての文字を、確認しながら置換

13.1.3 エディタ特有

Emacs	vim	操作内容
	ESC	ノーマルモードへ切替

[*14] Emacs の場合、jaword パッケージ を導入すると日本語の単語移動が賢くなります

[*15] 元々は Help ですが、置き換えています

[*16] Emacs の場合、cmigemo と migemo パッケージ を導入するとローマ字で日本語検索が可能になります。インストールと設定の詳細は るびきち「日刊 Emacs」 を参考にするとよいと思います

	i	カーソルの位置に追加
	a	カーソルの次の位置に追加
	A	行末に追加
	I	行頭に追加
	o	カーソルの下の行に追加
	O	カーソルの上の行に追加
C-j		改行
C-o		改行
C-m		改行
RET		改行
C-x u	u	直前の動作の取り消し
	U	行全体の変更の取り消し
	C-r	取り消しの取り消し
	r	カーソル下の 1 文字の置換
	R	カーソル下の複数文字の置換
	cw	カーソル位置の単語の変更（削除＋挿入）
	c$	カーソル位置から行末までの変更（削除＋挿入）
	c0	カーソル位置から行頭までの変更（削除＋挿入）
	c^	カーソル位置から文頭までの変更（削除＋挿入）
	C-g	ファイル内の位置の表示
	%	対応するカッコへ移動
	!コマンド	外部コマンドを実行

13.1.4 Emacs + Org

Emacs のアウトラインモードです。最近の Emacs には標準添付です。とても多機能なので、詳しくは「M-x org-info」でドキュメントを参照するとよいです。

13.1.5 Emacs + Prelude

Emacs の設定を一からするのってめんどくさいですよね。そんな場合はとりあえずググってみましょう。いろんな人が、いろんな形で公開しています。

Prelude もその 1 つで、GitHub で公開されています。いろいろあって違いがよく分からなかったので、名前がかっこいいなーと思ってこれに決めました。ほんとそれだけです。

複数のマシンで同じ Emacs 設定を使いたい場合は、Prelude を自分の GitHub に Fork して、Clone するとよいと思います。

13.2 LaTeX

ほとんどの人は修論の時に LaTeX をがしがし使うことになると思います。その時に「インストールできないー」などと焦るのはしょうもないので、簡単にまとめておきます。

13.2.1 MacTeX を使おう

日本語はマルチバイトコードであるため、LaTeX でコンパイルするのが難しかったみたいです。それに対処する歴史的な紆余曲折から日本語版 LaTeX にはさまざまな派生品が存在します。この歴史の詳細に関しては、三重大学の奥村さんのウェブサイトをはじめ、ググってみるとよいでしょう。

つい最近までは「Mac LaTeX インストール」などでググると、なんだかまとまりのない情報が溢れていました。しかし、現在はそれらを取りまとめようということで開発が進んでいるようで、これからは TeXLive 一択で良いみたいです。

TeXLive は MacPorts からインストールすることもできますが、うまく設定できた試しがありません。なので、MacTeX 公式ページ に置いてある MacTeX パッケージをダウンロード[*17]するのが一番簡単で良いと思います。

MacTeX パッケージを使う利点としては、TeX の統合環境である TeXShop[*18]や TeXworks、TeX 関連のパッケージ管理ツールである TeX Live Utility[*19]、文献管理の BibDesk、スペルチェックの Excalibur、そして、Keynote に数式を貼り付けるのに必要な LaTeXiT[*20]もついてきます。

13.2.2 YaTeX を使おう

さて、前述のように MacTeX をインストールしたとします。MacTeX には TeXShop.app や他にもいろいろな便利なアプリがついてきます。しかし、ここでは Emacs+YaTeX を使うことにします。

[*17] フルパッケージは 2GB ちょいあるので、ダウンロードに少し時間がかかります。細い回線で行うのはオススメしません

[*18] 修論の頃はお世話になりました。現在は YaTeX に移行したので全く使っていません

[*19] コマンドラインから tlmgr として使えます。パッケージのインストールがとても楽ちん。ただし、TeXLive のバージョンが上がるたびに動かなくなるのでちょっとめんどくさい

[*20] これが一番重宝してます

http://ichiro-maruta.blogspot.jp/2013/03/latex.html http://qiita.com/zr_tex8r/items/5413a29d5276acac3771

13.3 KiNOKO

CAMAC や VME でデータ収集を行うためのドライバをインストールします。詳細に関しては「KiNOKO プロジェクト」を参照してください。

```
$ cd ~/Downloads/
$ wget http://www.awa.tohoku.ac.jp/~sanshiro/kinoko-download/files/kinoko-2014
$ tar zxvf kinoko-2014-01-29.tar.gz /usr/local/heplib/
```

僕の場合、ダウンロードしたファイルはとりあえず ~/Downloads に保存することにしています。また、高エネルギー物理関連のプログラムは *usr/local/heplib* 以下にインストールすることにしています。

13.3.1 CAMAC ドライバのインストール

13.3.2 VME ドライバのインストール

13.4 Geant4