KumaROOT

しょーた

2015 年 1 月 22 日

目次

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 に集めた情報は、下記のサイトにたいへんお世話になっています。

ROOT 公式ユーザーズガイド 困ったらとりあえず読む！

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うべし！

ROOT 公式チュートリアル 付属のサンプルコードを試すべし！

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

ROOT 解体新書 by 楠本さん gROOT とか、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（ライブドアブログ） by 奥村さん PyROOT のことをググるとたいていこのページに辿り着くはず。いまは 宇宙線実験の覚え書き（はてブ） にお引っ越ししたみたいです [*1]。

3 ROOT を使いたい

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

[*1] 実際はもっと青い色の頁にお世話になってたはずなんだけど、うまく探せず

さて、まずはインストールしないとはじまりません。従来の面倒な方法はググってください。山ほどでてくるはずです。ここでは、まだあまり紹介されてない、すごく簡単な方法を紹介します。

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。(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。)

Linux を使っている場合は、Git を使うとよいでしょう。方法は公式ページに載っています。

Windows はよく分かりません。ごめんなさい。(たぶん公式ページ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)

3.1 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[*2]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得てして ROOT5 系であることが多いです。郷に入っては郷に従え、と言うように周りで使われている環境に合わせるのが無難です。

3.2 ROOT5 と ROOT6 を試してみたい

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

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
```

[*2] 正確には「ポート名」かも。あまり気にしないでください。

3.2.1 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[*3]。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[*4]。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。

3.3 PyROOT を使いたい

```
$ sudo port install root5 +python27    ## ROOT5 の場合は variants が必要
$ sudo port install root6              ## ROOT6 の場合、実は variants 不要
```

CERN には「へびつかい」が多いらしく、「PyROOT」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。

3.4 Emacs で ROOT を編集したい

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

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

[*3] CINT->CINT++ に変更

[*4] 広く知られていると思われるのは、構造体の変数とクラスのメンバーの区別がなかったり。(i.e. クラスのメンバーに.（ドット）でアクセスできたり、逆もまた然り)

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。

3.5 ROOT の tutorial を使いたい

```
/opt/local/libexec/root5/share/doc/root/tutorials/      ## ROOT5 の場合
/opt/local/libexec/root6/share/doc/root/tutorials/      ## ROOT6 の場合
```

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てみるとよいでしょう。

4 全体設定編

4.1 初期設定したい

4.1.1 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
```

4.1.2 rootlogon.C

4.2 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。

4.3 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。

4.4 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。

4.5 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)
```

4.6 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。

4.7 横軸に時間を使いたい

4.8 キャンバスに補助線を描きたい

4.9 グラフの軸を一括してログ表示にする

4.10 軸の目盛り間隔を変更したい

5 ヒストグラム編

5.1 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

5.2 タイトルを変更したい

5.3 統計ボックスを表示したい

5.4 X 軸名を設定したい

5.5 タイトルを中心にしたい

5.6 平均値、RMS を知りたい

5.7 値を詰めたい

5.8 面積でノーマライズしたい

6 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[*5]。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。

6.1 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
...
```

6.1.1 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。

[*5] 「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

```
void tree_using_readfile()
{
  TString ifn = "inputfilename";
  TTree *tree = new TTree("tree", "tree using ReadFile()");
  tree->ReadFile(ifn, "row1/I:row2/I:row3/I:row4/D:row5/I");

  TString ofn = "out.root";
  TFile *fout = new TFile(ofn, "recreate");
  tree->Write();
  fout->Close();

  return;
}
```

TTree::ReadFile() の第 2 引数には、branchDescriptor の変数をコロン（:）で区切って記述すします。「ブランチ名/I」、「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「ブランチ名/F」になるみたいです。

6.1.2 TTree::Branch() を使う方法

一番ありがちな方法です。ググればいっぱい見つかります。

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

    // データを格納するための変数
    int val1, val2, val3, val4;

    // TTree を作成する
    // 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；int 型は I，float 型は F，double 型は F など
    TTree *tree = new TTree("name", "title);
    tree->Branch("val1", &val1, "val1/I");
    tree->Branch("val2", &val2, "val2/I");
    tree->Branch("val3", &val3, "val3/I");
    tree->Branch("val4", &val4, "val4/I");

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

    // 作成した TTree を保存するための ROOT ファイルを準備する
    TString ofn = "outputfilename";
    TFile *fout = new TFile(ofn, "recreate");
    tree->Write();  // TFile を開いた状態で、TTree::Write() すれば書き込みできる；違うファイルに書き込みたい場合は後述するかも
    fout->Close();  // 最後にファイルを閉じる；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
    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。