CRIB
rdf2root
CRABAT
入門

2015-2016年 山口研究室所属 坂口裕司

はじめに

この説明書の読者には、CRIBの実験でできるrdf fileをrdf2rootによってroot fileに変換する方法を知りたい方、またROOTやC++の知識はあれど、CRABATの経験と知識がない方を想定しています。

rdf fileをroot fileへ変換する環境としては2016年2月のCRIBのパソコンであるanalys2のものを想定した説明になっています。

analys2はCRIB共用のパソコンであり、analys2を使って変換を行う場合は、ディレクトリやファイルの保全のために、バックアップはこまめにとってください。

CRABATは、この説明書と同じディレクトリにある「analysis_code.tar」として圧縮されたものになります。

CRABATの環境ができている、つまりmakeが終わっている読者を想定しています。

この説明書は初心者の入門用なので、より詳しいことが知りたい場合はREADMEを読む、ファイルを直接読み込む等してください。

コンテンツ

- rdf file をroot fileに変換する
  - rdf2root.C…ディレクトリの指定、セットアップに合わせる
  - sql file…場所、リンク貼り直し、書き換え
  - rdf2rootを動かす
- root fileをcrabatで解析する
  - CRABATのディレクトリ内ファイルの概要
  - ファイルの書き換え
  - CRABATを動かす
  - 解析の無駄な時間を減らすために
  - Analyzer.hの作り方

rdf file をroot fileに変換する

rdf file をroot fileに変換する

CRIBは基本的にAnapawの環境用にデータをとっているために、実験のデータ形式がrdfとなっています。

原子核実験の解析はROOTで行いたい方は多く、そのためにはrdf fileをroot fileへ変換する必要があります。

CRIBには「rdf2root.C」というマクロがあり、これを使って変換が行われています。

以下CRIB環境下においてrdf file をroot fileに変換するやり方を説明します。

rdf file をroot fileに変換する

rdf fileのディレクトリとroot fileのディレクトリの指定

「~/physics/rdf2root」というディレクトリには実験ごとにrdf2root.Cが入ったディレクトリが用意されています。

図が具体例を表しています。

```
(analys2:~/physics/rdf2root) crib% ls
rdf2root_be10alpha      rdf2root_f18    rdf2root_o15
rdf2root_be10alpha.tar  rdf2root_feb12  rdf2root_oct15
rdf2root_dec15          rdf2root_nov15  rdf2root_test
```

~/physics/rdf2root内

自分の実験のために他のディレクトリをコピーしておくと良いです。

例として、~/physics/rdf2root/rdf2root_be10alphaの中の「rdf2root.C」で今後説明します。エディタで開いてみてください。

rdf file をroot fileに変換する

rdf fileのディレクトリとroot fileのディレクトリの指定

inPathの部分をrdf fileがあるディレクトリに、outPathの部分を変換によってつくられたroot fileを格納するディレクトリに書き直してください。

rdf file のあるディレクトリ

```
110 // Path where the evt and root files are going to be located
111 // CHANGE ME FOR YOUR COMPUTER!
112   char inPath[100],outPath[100];
113   sprintf(inPath,
114         "/home/crib/exp/be10_a/users/crib/rdf"
115         ); // where the rdf files are at
116   sprintf(outPath,
117         "/home/crib/physics/rdf2root/rdf2root_be10alpha/root_be10"); // wher
    e to output the root data
```

変換後のroot file を格納するディレクトリ

rdf2root.C 一部抜粋

rdf file をroot fileに変換する

セットアップに合わせる

セットアップによってSSDやPSDやPPACの数は変わります。
rdf2root.Cの中でセットアップに合わせて書き換えなくてはならない部分があります。

図の部分を自分の実験のセットアップに合わせてください。

```
50 const Int_t RfN = 2;
51 const Int_t PpacN = 3;
52 const Int_t SsdN = 3;
53 const Int_t PsdN = 2;
54 const Int_t CrN =3;
```

rdf2root.C一部抜粋

→ TDC上でのRFの数
→ PPACの数
→ SSDの数
→ PSDの数
→ coin, single, pileupの三つを表している。

coin…PPACとPSD等のcoincidenceのイベント
single…PPACのみのイベント
pileup…同時に粒子が二つ来たイベント

Analyzer.cxxで扱う変数としては
cr[0]…coinを判定
cr[1]…singleを判定
cr[2]…pileupを判定　という対応のはず。

rdf file をroot fileに変換する

sql fileの場所

キャリブレーションの変数や、検出器と変数の対応などを、sql fileに書き込まなくてはなりません。ここからsql fileの書き込みの説明をします。

~/physics/rdf-1.0というディレクトリ内にそれぞれの実験用のsql fileを含んだディレクトリがいくつかあります。自分の実験用にディレクトリを他のディレクトリからコピーして用意しましょう。

```
(analys2:~/physics/rdf-1.0) crib% ls
aclocal.m4              COPYING      Makefile.in        sql_june10
AUTHORS                 data         missing            sql_may07
bin                     depcomp      NEWS               sql_may08
ChangeLog               doc          README             sql.may08.tar.gz
config.guess            example      sql                sql_may15
config.h                html         sql_c11            sql_may15backup
config.h.in             include      sql_dec15          sql_may15.tar
config.log              INSTALL      sql_feb11          sql_o15
config_max_element.log  install-sh   sql_feb11.tar.gz   sql_oct15
config.status           lib          sql_feb12          sql_sept10
config.sub              libtool      sql_july08         src
configure               ltmain.sh    sql.july08.tar.gz  stamp-h1
configure~              Makefile     sql_july09         test
configure.ac            Makefile.am  sql.july09.tar.gz
```

~/physics/rdf-1.0の内部の様子

rdf file をroot fileに変換する

sql fileのリンクを貼りなおす

~/physics/rdf-1.0内で「ls –l」と入力すると図のような関係が表示されるはずです。

```
lrwxrwxrwx 1 crib crib      9 2016-02-28 16:24 sql -> sql_may15
```

「ls –l」後の端末の一部抜粋

「sql」がシンボリックリンクとして「sql_may15」に結び付けられているのがわかると思います。この「sql」を自分の実験用のディレクトリに貼りなおしてください。
「unlink sql」と入力した後、「ln –s 自分の実験のディレクトリ sql」と端末で入力すればできるはずです。
リンクづけが終わったら、あとはディレクトリ内部のsql fileを書き換えていく必要があります。自分の実験のディレクトリに入ってください。

rdf file をroot fileに変換する

sql fileの概要

着目すべきsql file　は以下になります。

・ppac.sql…　PPACのsql file

・ssd.sql… SSDのsql file

・psd.sql… PSDのsql file

セットアップ通りに書き換えることを説明するための例としてpsd.sqlを使います。psd.sqlをエディタで開いて見てください。

rdf file をroot fileに変換する

sql fileの具体的な書き換える部分

psd.sqlの一部は図のようになっています。上図では全部で9個のパラメタが挙げられていることがわかると思います。直接書き換えるべきパラメタは下図の部分になります。

```
 9 CREATE TABLE `PSD0` (
10   `detector channel`       int NOT NULL,
11   `adc map`                integer,
12   `adc offset`             double,
13   `adc gain`               double,
14   `adc histogram channel`  integer,
15   `tdc map`                integer,
16   `tdc offset`             double,
17   `tdc gain`               double,
18   `tdc histogram channel`  integer,
19    PRIMARY KEY (`detector channel`)
20 );
```

下図の9個の値がそれぞれ何を指しているのかを理解する必要があります。上図に挙げられているパラメタの順番と、下図のパラメタの順番は対応しています。例えばADCのgainは上図で上から4番目なので、下図の左から4番目となります。

同じ順番

```
37 INSERT INTO PSD0 VALUES ( 0,  0, 0.0, 1.0,  0,  160,  0.0, 1.0,  0);
38 INSERT INTO PSD0 VALUES ( 1,  1, 0.0, 1.0,  1,  161,  0.0, 1.0,  1);
39 INSERT INTO PSD0 VALUES ( 2,  2, 0.0, 1.0,  2,  162,  0.0, 1.0,  2);
40 INSERT INTO PSD0 VALUES ( 3,  3, 0.0, 1.0,  3,  163,  0.0, 1.0,  3);
41 INSERT INTO PSD0 VALUES ( 4,  4, 0.0, 1.0,  4,  164,  0.0, 1.0,  4);
42 INSERT INTO PSD0 VALUES ( 5,  5, 0.0, 1.0,  5,  165,  0.0, 1.0,  5);
43 INSERT INTO PSD0 VALUES ( 6,  6, 0.0, 1.0,  6,  166,  0.0, 1.0,  6);
```

psd.sql一部抜粋

rdf file をroot fileに変換する

sql fileに書き込むgain, offset

CRIBでは基本的にEnergy=gain × (channel-offset)という関係を用いてエネルギー等の計算をするのが慣習となっています。このgain とoffsetを目的の値にpsd.sql, ssd.sql, ppac.sqlそれぞれで書き換えましょう。

rdf2rootの環境を自分のパソコンに構築するのはMySQLを導入してからROOT環境を整えてそれからバグを直して…とかなり大変で、ほとんどの方はCRIBの構築済のrdf2rootを使うかと思います。

sql fileにキャリブレーションの値を書き込んでしまい、出てくるroot fileの値をキャリブレーション後のものにしてしまうのは可能です。

しかし、もしキャリブレーションの値が間違っていた場合はCRIBにまた来てrdfをrootへと変換するのをやり直す、という事態が有り得ます。

よってCRIBのrdf2rootを使うと決めた場合は、gain=1, offset=0とすべてして、raw dataのrdf fileをraw dataの root file　に変換し、キャリブレーションはCRABAT上で行うとした方が賢明かと思われます。

rdf file をroot fileに変換する

sql fileを変更後の更新

detector channel, adc map, adc histogram channel, tdc map, tdc histogram channel, などの対応には気をつけてください。

sql fileを直接読み込む、VMEのラックを調べる、Anapawのマッピングを調べる、等をして推測して順番の付けられ方を書き込んで行きましょう。

sql fileを書き換えたら、MySQLを更新する必要があります。

もしssd.sqlを書き換えたら、更新の仕方は以下のようになります。

「mysql –u crib –p < ssd.sql」

これを端末から入力してください。他のsql fileに対しても同じです。

入力するとパスワードが求められます。

この説明書にパスワードを書きたいところですが、セキュリティ上問題があるかと思いますので、諸先輩方に口頭で教わりましょう。

パスワードを打ち込んだら更新終了となります。

rdf file をroot fileに変換する

rdf2rootを動かす

rdf2root.C とsql fileをセットアップに調整することができたらやっと変換自体ができるようになります。

自分のrdf2root.Cがあるディレクトリに移ってから、以下のように入力してください。

「~/physics/root/bin/root –l」

MySQLの環境に適合したROOTでなければ変換ができません。このコマンドでMySQLに対応したROOTが立ち上がります。

ROOTが立ち上がったら

「.x rdf2root.C+(変換したいrdf fileの番号)」

と入力してください。

変換の画面の例は次のページにあげられています。

rdf file をroot fileに変換する

rdf2rootを動かす

```
Channel:   Map:     Gain:   Offset:  Histogram Map:
       0    116         1         0              0
Bit-mask = 0x00000000000100000
         = 00000000000000000000000000000000 00000000000000000000000000000000 000
00000000000000000000000000000000 00000000000000000100000000000
TDC calibration:                                  0
Channel:   Map:     Gain:   Offset:
       0      0     0.122         0)0000000
Bit-mask = 0x0001
[illegible]
OpenFileUncompressed: Opening file /home/crib/exp/be10_a/users/crib/rdf/0100.rdf
File info:
Number   : RUN-0100
Start    : START => 17:50:31
Stop     :  STOP => XX:XX:XX
Time     : Print -> XX:XX:XX
Date     :  28-May-15
RemarkA  : continue
RemarkB  :

DecodeFera: ERROR - subaddress and pattern-word mismatch
DecodeFera: ERROR - subaddress and pattern-word mismatch
DecodeFera: ERROR - subaddress and pattern-word mismatch
DecodeFera: ERROR - subaddress and pattern-word mismatch
DecodeFera: ERROR - subaddress and pattern-word mismatch
DecodeFera: ERROR - subaddress and pattern-word mismatch
DecodeFera: ERROR - subaddress and pattern-word mismatch
DecodeFera: ERROR - subaddress and pattern-word mismatch
DecodeFera: ERROR - subaddress and pattern-word mismatch
DecodeFera: ERROR - subaddress and pattern-word mismatch
DecodeFera: ERROR - subaddress and pattern-word mismatch
DecodeFera: ERROR - subaddress and pattern-word mismatch
```

rdf2root.C で変換中の端末

ERRORがたくさん出てきますが、大した問題ではありません。

変換が終わるまで待ちましょう。

変換が終わるとrdf2root.CのoutPathで指定したroot fileを格納するディレクトリに目的のroot fileができているはずです。

rdf file をroot fileに変換する

rdf2rootを自動的に動かす

```
5  #include "rdf2root.C"
6
7  void run_rdf2root()
8  {
9  //  TTree *t=new TTree("ri
10 int run_no;
11   bool status;
12
13  for (int i=11 i<14;i++)
14    {
15
16      run_no=i;
17      status=true;
18      //a loop to skip junk
19 //      if(run_no==5)
20   //        status=false;
21 //      TTree *t=new TTree("
22      if(status)
23        rdf2root(run_no);
24
25
26    }
```

解析したいrdfの初めの番号

終わりの番号

run_rdf2root.C一部抜粋

実験にもよりますが、rdf fileは何十個～何百個ほどの数にのぼります。

一つ一つのrdf fileに対して、前のファイルの変換が終わってから次のファイルの変換を手で入力してパソコンに指示する、としていたのでは、かなりの時間の無駄になります。これを自動化によって省いてくれるのがrun_rdf2root.Cです。

rdf2root.Cと同じディレクトリにrun_rdf2root.Cというマクロがあるはずです。

これによって指定したrdf fileの初めから終わりまで自動的に変換をしてくれます。run_rdf2root.Cは左図のようになっており示した部分を書き換えましょう。

あとはROOTを立ち上げ、

「.x run_rdf2root.C+」と入力すれば指定した通りに自動的に変換が行われるはずです。

rdf file をroot fileに変換する

参考

CRIBのグループに参加していたDaidのページが参考になります。

自分のパソコンにrdf2rootの環境を作りたい方は大いに参考になると思います。

http://www.cns.s.u-tokyo.ac.jp/~daid/physix/rdf2root-in-Linux.html

CRABATで解析

CRABATで解析

CRABATはROOTの環境を想定した解析システムとなります。CRABATを用いることで様々な手間が自動化され、貴重な時間を研究そのものにもっと割くことができるでしょう。

以下CRABATで解析をするための初歩を説明します。

CRABATで解析

原子核実験では目的に合わせて複数の物理量を複数回に分けて記録します。「キャリブレーションは5回測定され5個のファイルが存在し、共鳴散乱実験は20個のファイルに分けて記録されている。」という様な状況となっています。

工夫がなければ「共鳴散乱の解析用にコードを書いてコンパイルし、20回実行ファイルを端末で手で入力する。」という状況になり、限りある時間で最高の結果を要求される研究においては、この手間は避けるべきです。加えて「コードのある部分をミスしていたので、直してからまた20回実行ファイルをまわす。」という事態も容易に発生し、これも時間の無駄です。

CRABATで解析

CRABATの一つの機能によって、同一目的の測定が記録されたファイルを、パソコンの能力を最大限に引き出しながら、実行ファイルをデータファイル分だけ人の手を使うことなく、すべて自動的に解析することが可能となります。

この機能が使えれば入門は完了したといえるので、この機能を使うためにどのようにすれば良いかを説明します。

CRABATで解析

ファイルの役割の概要

まずcrabatのスクリプトがあるディレクトリの中のファイルの概要を説明します。

- run.h　ヒストグラムの定義
- run.conf　root fileのディレクトリの指定、グループ分けされたデータファイルのどのグループを解析するかを指定
- run.cxx データファイルのグループ分けを指定
- Analyzer.h 変数の宣言等
- Analyzer.cxx 具体的な計算や変数をヒストグラムに詰める

より詳しいことはREADMEやそれぞれのファイルに載っているので是非目を通してください。

CRABATで解析

データファイルのディレクトリを指定

まず解析をするroot fileが格納されたディレクトリと解析後のroot fileを格納するディレクトリを指定する必要があります。(CRABATでは解析結果をroot fileとして保存します。)
run.confを任意のエディターで開いてみると画像のようになっています。
図に示した部分を自分の環境に合うように書き換えましょう。

run.conf 一部抜粋

コメントアウトを外して自分の環境下における解析前のroot fileが格納されたディレクトリを指定しましょう。

解析結果のroot fileを格納するディレクトリをここで指定してください。

```
# crabat run configuration file
# last modified daid 24 Apr 2015 21:01:04
# follow comments to edit
#
# location of the input ROOT tree
#/home/daid/data/
#../clone/root/Tree
#/tmp
#../root/Treelocal
#/mnt/tiny/o15_jan14_root
../../root
#../root/Tree
#/mnt/h2o/18neap/Tree
#
# location of the output ROOT tree
Tree
```

CRABATで解析

同一目的なデータファイル

run.confの下の方を見てみると図のようになっています。

```
Hflag=1
#       0: alpha calib (PPAC or SSD)
#       1: physics (alpha scattering)
#       2  F3 beam measurement
#       3: background
#       4: F2 beam measurement
# set run header value here
```

run.conf一部抜粋

これは著者の環境となっています。
「alpha calib (PPAC or SSD)」という「αソースを用いたPPACとSSDのキャリブレーション」という目的の測定のデータのグループのIDが「0」とされ、
「physics (alpha scattering)」という「α粒子を用いた散乱を観測する」という目的の測定のデータのグループのIDが「1」とされています。

このグループとIDの指定の方法は次のページで説明します。
CRABATは「同一目的のデータたちをいっぺんに解析できる。」と上で説明しましたが、どのデータのグループを解析するかはこの「Hflag」で行います。この画像では「Hflag=1」となっているので、「physics (alpha scattering)」というグループのデータが解析されることになります。解析の際はここで指定しましょう。
どの数字がどのグループに対応するかあらかじめ決めて、コメントアウトしてメモっておきましょう。グループとIDを指定するのに必要になります。

CRABATで解析

同一目的なデータファイル

同一目的のデータのグループ分けとID指定はrun.cxxで行います。
実験中、基本的に測定のデータは2時間ごとなど、決まった時間で記録をされます。この一回一回のデータはrdf fileとして記録されます。rdf fileの名前は記録される順番によって決まります。記録される順番が100回目なら、そのrdf file は「0100.rdf」と名前がつけられています。以下に例があります。

0001.rdf～0015.rdf・・・キャリブレーション
…
0096.rdf～0122.rdf・・・α粒子の共鳴散乱（本測定）
…
0130.rdf～0132.rdf・・・実験後のキャリブレーション

rdf file の順番と測定の関係例

例としては本測定は0096.rdf～0122.rdfのデータが該当します。
どのrdf fileがどの目的に当たるのかをログノートを見ながら書き出しましょう。
rdf file↔測定の目的↔自分で決めたID、という関係が決まったはずです。
この関係がわかったら、それをCRABATに教えてやる必要があります。
それを書き込むのはrun.cxxです。run.cxxをエディタで開いてください。

CRABATで解析

run.cxxは図のような部分があります。ここでrdf file↔測定の目的↔自分で決めたID、の関係を指定しましょう。CRABATは、目的に合わないrdf fileは自動的な解析からはずす、というやり方をするので、ここでは「目的に合わないrdf file」を指定してください。先ほど挙げた本測定を例とすると具体的な指定の仕方は以下のようになります。

IDを指定

本測定は0096.rdf～122.rdfなので、それと違うrdf fileを指定してください。

本測定を表す「physics」を記入してください。

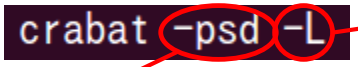
```
char file_run[150];
// to exclude a run, you can do it like this
if (flag_Header) {
  if (header_no == 0){
    if (!(run_no<=16 || run_no>=129)){
    cout << "Run number " << run_no << " is not alpha calibration...skipping" << endl;
    return 0;}
  }
  if (header_no == 1){
    if (run_no>=123 || run_no<=95){
    cout << "Run number " << run_no << " is not physics...skipping" << endl;
    return 0;}
  }
  if (header_no == 2){
    if (!(run_no==18 || run_no==20 || run_no==22 || run_no==23 || run_no==24 || run_no==25 || ru
n_no==26 || run_no==27 || run_no==28 || run_no==29 ||
      run_no==30 || run_no==31 || run_no==32 || run_no==33 || (run_no>=78 && run_no<=84 ))){
    cout << "Run number " << run_no << " is not F3 beam measurement...skipping" << endl;
    return 0;}
  }
  if (header_no == 3){
    if (run_no>=129 || (run_no>=95&&run_no<=122) || run_no<=88){
    cout << "Run number " << run_no << " is not background run...skipping" << endl;
    return 0;}
  }
  if (header_no == 4){
    if (!(run_no==17 || run_no==19 || run_no==21)){
    cout << "Run number " << run_no << " is not F2 run...skipping" << endl;
    return 0;}
  }
}
```

run.cxx一部抜粋

CRABATで解析

CRABATを動かす

root fileのあるディレクトリの指定、解析後のroot fileのディレクトリの指定、rdf file↔測定の目的↔自分で決めたIDの指定、が終われば、CRABATの同一目的の測定が記録されたファイルを、パソコンの能力を最大限に引き出しながら、自動的に解析する機能を使うことができます。
CRABATの実行ファイルがあるディレクトリ上で図のように入力してください。オプションの意味も書いてあります。



ここでAnalyzer.cxxに記入されたどの部分の解析をするかを指定できます。
p…PPAC
s…SSD
d…detailed data
詳しくはREADMEやAnalyzer.cxxに書かれています。

「L」と指定するとrun.conf内のHflagで指定するIDに該当する同一目的のrdf fileを自動的にすべて解析してくれます。
パソコンにはコアがあり、その数はパソコンごとに決まっています。「-L」オプションをつけるとそのコアの数分の能力を最大限使ってパソコンが同時に解析を進めてくれます。

CRABATで解析

CRABATを動かす

```
Calibration file: psd1_Y_calib_time.dat
[Ch: Offset, Gain]
------------------------------
0: -10.7, 0.122
1: -241.7, 0.122
2: 8.4, 0.122
3: -61.1, 0.122
4: -46.3, 0.122
5: -116.4, 0.122
6: -76.2, 0.122
7: -153, 0.122
8: -230, 0.122
9: -93.4, 0.122
10: -207.7, 0.122
11: -72.1, 0.122
12: -34, 0.122
13: -50.3, 0.122
14: -213.8, 0.122
------------------------------
Calibration file: ssd_F2_calib.dat
[Ch: Offset, Gain]
------------------------------
0: -256.258, 0.0203259
------------------------------
Calibration file: lowgain_calib.dat
[Ch: Offset, Gain]
------------------------------
0: 20.0011, 0.00379383
1: 38.2548, 0.00564848
------------------------------
40000 / 1470989
```

「crabat –psd –L」と端末で入力すると左の図のようになります。

コアの数だけ同時進行でLoopが進んでいきます。

CRABATによって自動的に目的のrdf fileを選別し、Loopを進めていきます。

Loopが進むごとに数字が上がって行きます。

CRABATで解析

解析で無駄な時間を使わないために

後後の解析の流れは、run.hでヒストグラムの定義をして、Analyzer.cxxの内部で計算をして、変数をヒストグラムに詰めていくことになると思います。Analyzer.hには変数の定義があります。ここで恐らく関数の定義をすることにもなると思います。

解析をしていく上で留意しておくべきことは「ほとんどのイベントはノイズなどの目的の物理以外で作られてしまっていること」です。これのために関数の計算をまわしていると大変な時間の無駄となります。よって、イベントがノイズ等の無意味なもの由来であるとわかった時点で、そのイベントの計算をやめて次のイベントの計算へと移るべきです。その為に「continue」を使いましょう。以下にAnalyzer.cxx内のcontinueの例をあげておきます。

```
// Check if each PPAC is hit with valid event
PpacIsHit0 = false;
if (PPAC0_time[0]>0. && PPAC0_time[1]>0. && PPAC0_time[2]>0. && PPAC0_time[3]>0.) PpacIsHit0=true;
PpacIsHit1 = false;
if (PPAC1_time[0]>0. && PPAC1_time[1]>0. && PPAC1_time[2]>0. && PPAC1_time[3]>0.) PpacIsHit1=true;

    if(PpacIsHit0!=true || PpacIsHit1!=true)
    {
            continue;
    }
```

Analyzer.cxx一部抜粋

PPAC0とPPAC1の値がおかしい場合はcontinueでLoopから外れる、ということをやっています。

CRABATで解析

Analyzer.hの作り方

実験のセットアップによって扱う物理量も変わり、よってAnalyzer.hの変数の定義もかわるはずです。これをもとの物理量がどのroot fileの変数に該当するか一つ一つ調べて先輩方からもらったAnalyzer.hを書き換えていくというやり方をすると、そもそも調べるのに時間がかかるばかりか、何行も書いてくのでバグを含む可能性が増えてしまい、時間と体力と気力の無駄になります。

root fileから自動でAnalyzer.hを作ることが可能です。
「root Analyzer.h を作りたいrootファイル名」と入力してrootシステムに入りましょう。
そこでTreeの名前がriken_expならば、
「riken_exp->MakeSelector("Analyzer.h")」と入力すると、元いたディレクトリ内にAnalyzer.hとAnalyzer.Cができているはずです。このAnalyzer.hを使えばよいと思います。
（参考　ftp://root.cern.ch/root/doc/21ExampleAnalysis.pdf)

CRABATで解析

あとは目的通りにAnalyzer.h, Analyzer.cxxなどを書き換えて解析を進めていくだけです。
Loopの自動化だけでも大いに研究が楽になると思いますが、CRABATには他の機能もあるようです。READMEなどを読んでCRABATの力を引き出すことに挑戦してみると良いと思います。

以上の説明が読者の皆様の研究にたいして貢献できることを祈っています。