粒子物理与核物理实验中的数据分析

张黎明
清华大学

第四讲：ROOT在数据分析中的应用(2)

上讲回顾和本节要点

- ROOT 基本概念(C++, 实验数据处理)
- 安装与登录ROOT以及体验
- ROOT的语法简介

  完全兼容c++语法， 解析运行
- 随机数， gRandom
- 接下来讲

  直方图, 数学函数, 文件， 树等

  TH1I， TH1F， TH1D， TH2F， TMATH， TF1, TF2， TF3， TFile,

ROOT中统计直方图

- 定制一维直方图

```
TH1F *hist_name = new TH1F("hist_name","hist_title",
num_bins,x_low,x_high);
```

- 定制二维图

```
TH2F *hist_name = new TH2F("hist_name","hist_title",
num_bins_x,x_low,x_high,num_bins_y,y_low,y_high);
```

- 定制三维图

```
TH3F *hist_name = new TH3F("hist_name","hist_title",
num_bins_x,x_low,x_high,num_bins_y,y_low,y_high,
num_bins_z,z_low,z_high);
```

- 填充统计图

```
hist_name->Fill(x);
hist_name->Fill(x,y);
Hist_name->Fill(x,y,z);
```

绘图：

```
root[0]hist_name->Draw();
```

ROOT脚本文件示例:直方图，随机数

```
void histw() {
   gROOT->Reset();
   const Int_t NEntry = 10000 ;
   TFile *file = new TFile("Landau.root","RECREATE");
   TH1F *h1 = new TH1F("h1","A simple histo",100,0,1);
   for (int i=0;i<NEntry;i++)  {
     h1->Fill( gRandom->Landau(0.2,0.1) );
   }
   h1->GetXaxis()->SetTitle("X Title, #sqrt{#Delta_{2}}");
   h1->GetXaxis()->CenterTitle();
   h1->GetYaxis()->SetTitle("Y Title, Entries/0.01");
   TCanvas *c1 = new TCanvas("c1","");
   h1->Draw();
   file->Write();
   c1->SaveAs("Landau.pdf");
}
```

1. 文件准备，写入
2. 生成直方图
3. 填图
4. 生成图片

打开ROOT文件

1. 打开已有的root文件，如刚刚生成的Landau.root:
   终端提示行下:
   > root –l Landau.root

2. ROOT环境下 :
   > root
   root [0] TFile f1("Landau.root");
   root [1] .ls
   root [2] h1->Draw();

3. 利用TBrowser 课上练习
   > root
   root [0] TBrowser b
   利用你的鼠标 ……

4. 当然也可以利用C++代码执行

打开一个文件，查看一个直方图

```
//
//   open a root file and get a histogram
//
void histr() {

   TFile *file = new TFile("Landau.root","READ");
   TH1F *h1 = (TH1F*) file->Get("h1");
   h1->Draw();

}
```

1. 文件打开
2. 得到直方图的指针
3. 画图

很多情况下，这些命令可以在**root**的提示符下逐行键入，并进行测试。

ROOT中的数学函数

画图时采用
root[1]fun_name.Draw();

- 制作一维函数曲线图

```
TF1 *fun_name = new TF1("fun_name","expression",
x_low,x_high);
```

```
root[0]TF1 *f1 = new TF1("f1","x*sin(x)",-5,5);
```

- 制作二维函数曲线图

```
TF2 *fun_name = new TF2("fun_name","expression",
x_low,x_high, y_low,y_high);
```

```
root[0]TF2 *f2 = new TF2("f2","x*sin(x)+y*cos(y)",
-5,5,-10,10);
```

- 制作三维函数曲线图

```
TF3 *fun_name = new TF3("fun_name","expression",
x_low,x_high,y_low,y_high,z_low,z_high);
```

```
root[0]TF3 *f3 = new TF3("f3","x*sin(x)+y*cos(y)
+z*exp(z)",-5,5,-10,10,-20,20);
```

数学函数的定义方式(1)

ROOT中定义数学函数的方式多种多样

- 利用C++数学表达式

```
TF1* f1 = new TF1("f1","sin(x)/x",0,10);
```

- 利用TMath定义的函数

```
TF1 *f1 = new TF1("f1","TMath::DiLog(x)",0,10);
```

- 利用自定义C++数学函数

```
Double_t myFun(double x) {
    return x+sqrt(x);
}
TF1* f1 = new TF1("f1","myFun(x)",0,10);
```

以上函数都不含参数，但在数据拟合时，我们往往需要定义含未知参数的函数

数学函数的定义方式(2)

ROOT中定义含未知参数的数学函数

- ROOT已经预定义了几种常用的含参函数

```
gaus:3个参数
      f(x)=p0*exp(-0.5*((x-p1)/p2)^2))
expo:2个参数 f(x)=exp(p0+p1*x)
polN:N+1个参数 f(x)=p0+p1*x+p2*x^2+...
     其中N=0,1,2,...，使用时根据需要用pol0,pol1,pol2...
chebyshevN:N+1个参数
        切比雪夫多项式（一个更好的多项式）
landau:3个参数
     朗道分布，没有解析表达式
gausn:3个参数，归一化的
       f(x)=p0*exp(-0.5*((x-p1)/p2)^2))/(p2*sqrt(2PI))
```

这些预定义函数可直接使用，比如

```
histogram->Fit("gaus"); //对直方图进行高斯拟合
TF1 *f1=new TF1("f1","gaus",-5,5);
```

数学函数的定义方式(3)

ROOT中自定义含未知参数的数学函数

- 利用C++数学表达式

```
TF1* f1 = new TF1("f1","[0]*sin([1]*x)/x",0,10);
f1->SetParameters(2.,1.5);
```

- 利用C++数学表达式以及ROOT预定义函数

```
TF1* f1 = new TF1("f1","gaus(0)+[3]*x",0,3);
```

- 利用自定义的C++数学函数

```
Double_t myFun(Double_t *x, Double_t *par) {
  Double_t f=par[0]*exp(-x[0]/par[1]);
  return f;
}

TF1* f1 = new TF1("f1",myFun,0,10,2);
f1->SetParameter(0,100);
f1->SetParameter(1,2);
```

ROOT脚本文件示例：Macro文件

课上练习

```
// Macro myfunc.C
Double_t myfunction(Double_t *x, Double_t *par)
{
   Float_t xx =x[0];
   Double_t f = TMath::Abs(par[0]*sin(par[1]*xx)/xx);
   return f;
}
void myfunc()
{
   TF1 *f1 = new TF1("myfunc",myfunction,0,10,2);
   f1->SetParameters(2,1);
   f1->SetParNames("constant","coefficient");
   f1->Draw();
}
void myfit()
{
   TH1F *h1=new TH1F("h1","test",100,0,10);
   h1->FillRandom("myfunc",20000);
   TF1 *f1 = (TF1 *)gROOT->GetFunction("myfunc");
   f1->SetParameters(800,1);
   h1->Fit("myfunc");
}
```

解释执行的条件下，要与文件名同名

```
Root > .L myfunc.C
Root > myfunc();
Root > myfit();
```

TTree

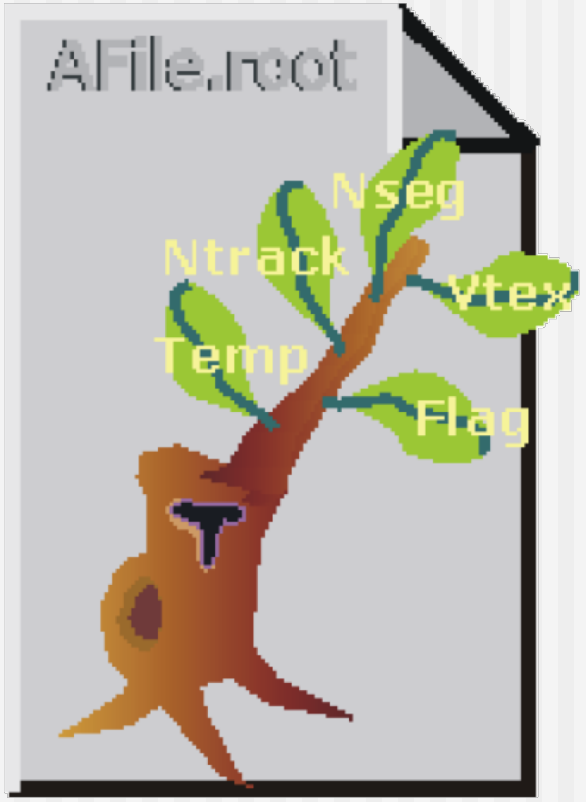
- ROOT中tree的概念(类TTree)

  什么是tree，为什么用tree存取、分析数据、使用演示
- 如何定义、填充、读取TTree
- 如何将ASCII数据转化为TTree
- TChain：把很多TTree连接在一起
- ROOT自动生成Tree的分析框架

深入数据分析的关键技巧之一：考察数据各个角度的关联性。

为什么使用TTree

- 一维，两维，三维的直方图是远远不够的，需要研究的物理内容经常的多维的，更复杂的。
- TTree采用了branch的体系，每个branch 的读取可以与别的branch无关。
- 适用于大量的类型相同的对象
- 一般情况下，tree的Branch，Leaf信息就是一个事例的完整信息
- 有了tree之后，可以很方便对事例进行循环处理。
- 占用空间少，读取速度快



TTree是ROOT最强大的概念之一

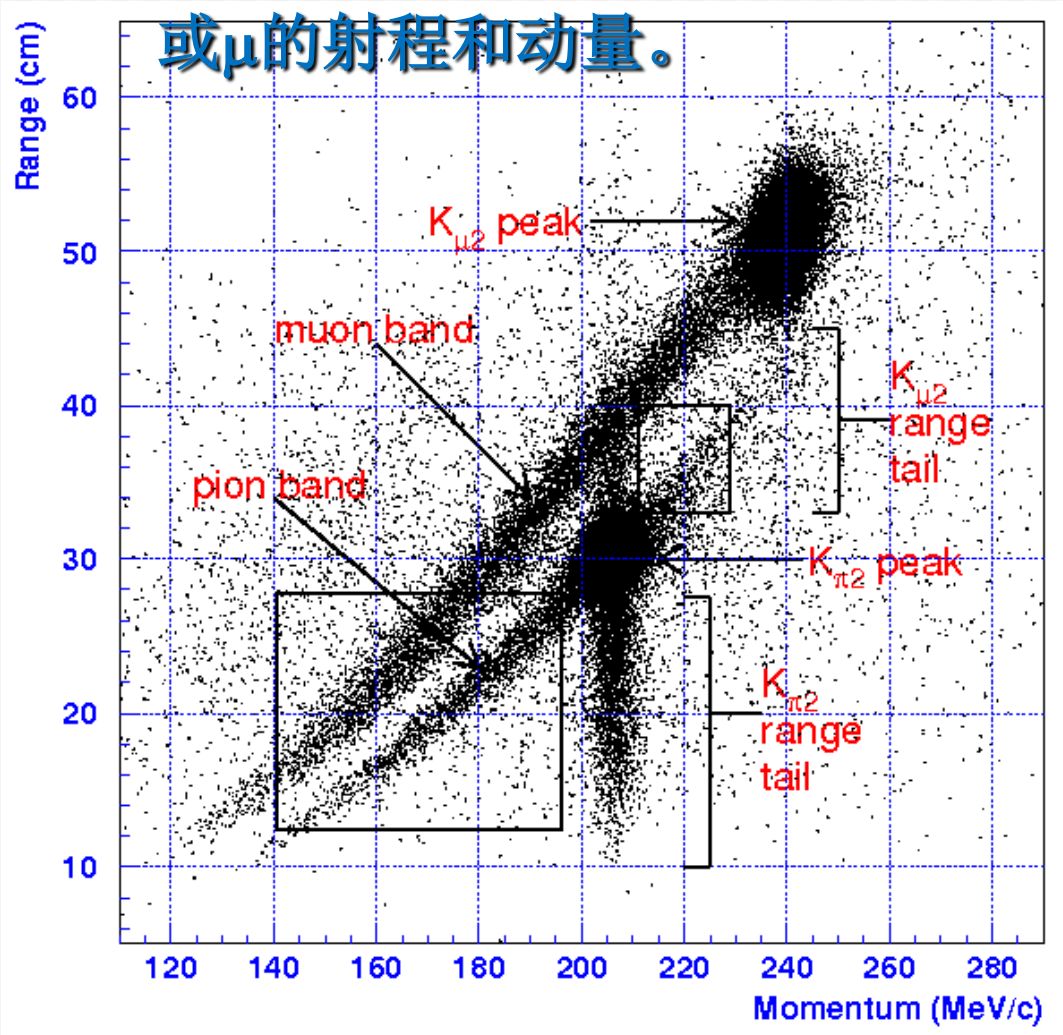
使用TTree进行分析

看一组K^+介子静止衰变的例子：

- $K^+ \rightarrow \pi^+\pi^0$ (Kπ2peak)
- $K^+ \rightarrow \pi^+\pi^0\gamma$ (pion band)
- $K^+ \rightarrow \mu^+\nu$ (Kμ2 peak)
- $K^+ \rightarrow \mu^+\nu\gamma$ (muon band)

目标：鉴别muon band事例！

图中为衰变出的带电粒子π或μ的射程和动量。



TTree的定义

参见 https://root.cern.ch/doc/master/classTTree.html

名称　　描述

构造函数：

TTree(const char* name, const char* title,
Int_t splitlevel = 99, TDirectory * dir = gDirectory);

Branch**成员函数：**

TBranch*Branch(const char* name, void* address,
const char* leaflist, Int_t bufsize = 32000);

创建TTree，并设置Branch，比如：

```
Int_t RunID;
TTree *t1 = new TTree("t1","test tree");
TBranch *br = t1->Branch("RunID",&RunID,"RunID/I");
```

Branch可以是单独的变量，也可以是一串变量，也可以是定长或不定长数组，也可以是C结构体，或者类对象(**继承自TObject**，如TH1F对象)。

如何写一个简单的TTree
参见Lec4/ex41.C

```
void ex41() {
  TFile *f = new TFile("tree1.root","recreate");
  TTree *t1 = new TTree("t1","test tree");
  gRandom->SetSeed(0);
  Float_t px,py,pz;
  Double_t random;
  Int_t i;
  //Set the Branches of tree
  t1->Branch("px",&px,"px/F");
  t1->Branch("py",&py,"py/F");
  t1->Branch("pz",&pz,"pz/F");
  t1->Branch("random",&random,"random/D");
  t1->Branch("i",&i,"i/I");
  for (i=0;i<5000;i++) {
      gRandom->Rannor(px,py);
      pz = px*px + py*py;
      random = gRandom->Rndm();
      t1->Fill();//Fill tree
  }
  t1->Write();
}
```

定义tree，参数分别为tree的名称和描述

设置Branch，参数分别为Branch的“名称”、“地址”以及“leaf列表和类型”。这里只有一个leaf，如果多个则用冒号分开。
常用类型:C,I,F,D分别表示字符串、整型、浮点型和双精度型，参见ROOT手册TTree

为每个leaf赋值，每个事例结束时填充一次。这里一共填充5000事例。

将tree写入root文件中存盘

运行：root -l ex41.C 或在ROOT中: .x ex41.C
(一定在你有些权限的地方执行)

查看Tree的信息

>root -l tree1.root 打开root文件
root[1].ls 查看文件信息，
发现TTree t1
root[2]t1->Show(0);
显示第0个event的信息
root[3]t1->GetEntries() 总事例数
root[4]t1->Scan();
root[5]t1->Print();
root[6]t1->Draw("px");

```
[training] root -l tree1.root
root [0]
Attaching file tree1.root as _file0...
root [1] .ls
TFile**         tree1.root
 TFile*         tree1.root
  KEY: TTree    t1;1    test tree
root [2] t1->Show(0)
======> EVENT:0
 px             = -0.864926
 py             = 1.28846
 pz             = 2.40823
 random         = 0.436748
 i              = 0
root [3]
```

```
root [3] t1->Scan()
************************************************************************
*    Row   *        px *        py *        pz *    random *         i *
************************************************************************
*        0 * -0.864926 * 1.2884608 * 2.4082288 * 0.4367477 *         0 *
*        1 * 0.1207585 * 0.8442332 * 0.7273124 * 0.2977862 *         1 *
```

```
root [4] t1->Print()
************************************************************************
*Tree    :t1        : test tree                                         *
*Entries :     5000 : Total =          155503 bytes  File  Size =      91601 *
*        :          : Tree compression factor =   1.47                  *
************************************************************************
```

查看Tree的信息(续)

```
>root -l    进入root
root[0]TFile *f1=new TFile("tree1.root");
root[1]t1->Draw( "sqrt(px*px+py*py)" );
root[2]TH1F *h1;
root[3]t1->Draw("px>>h1");
root[4]t1->Draw("px+py","px>0","same");
root[5]TH2D h2("h2","h2",10,0,10,10,0,10);
root[6]t1->Project("h2","px:py","pz>0")
```

做简单运算

投影到一个一维直方图

不加>>时，生成一个名为htemp直方图

加选择条件

投影到一个二维直方图

```
Selection = "weight *(boolean expression)"
```

如何读一个简单的TTree

```
void tree1r()
{
  TFile *f = new TFile("tree1.root");
  TTree *t1 = (TTree*)f->Get("t1");
  Float_t px, py, pz;

  t1->SetBranchAddress("px",&px);
  t1->SetBranchAddress("py",&py);
  t1->SetBranchAddress("pz",&pz);

  TH1F *hpx   = new TH1F("hpx","px distribution",100,-3,3);
  TH2F *hpxpy = new TH2F("hpxpy","py vs px",30,-3,3,30,-3,3);

  Int_t nentries = (Int_t)t1->GetEntries();
  for (Int_t i=0;i<nentries;i++) {
    t1->GetEntry(i);
    hpx->Fill(px);
    hpxpy->Fill(px,py);
  }
}
```

实际工作中好多事情是不能够简单的在命令行中就完成的，需要编程，要更丰富的计算。

告诉root每读进一个值的时候该放在哪个变量里

总的事例数是多少？

拿到这一组数据，这时新拿到的数据已经按照你的指示放到该放的变量里了。
你可以在这里用px，py，pz做计算了。

如何生成含有不定长数组的tree(1)
Lec4/ex42.C

```
...
const Int_t kMaxTrack = 50;
Int_t ntrack;
Float_t px[kMaxTrack];
Float_t py[kMaxTrack];
Float_t zv[kMaxTrack];
Double_t pv[3];

TFile f(rootfile,"recreate");
TTree *t3 = new TTree("t3",
                      "Reconst events");
t3->Branch("ntrack",&ntrack,"ntrack/I");
t3->Branch("px",px,"px[ntrack]/F");
t3->Branch("py",py,"py[ntrack]/F");
t3->Branch("zv",zv,"zv[ntrack]/F");
t3->Branch("pv",pv,"pv[3]/D");
...
```

很多时候不定长数组是必要的，比如正负电子对撞，记录末态粒子的信息，末态粒子数目是不固定的。

1)估计不定长数组的最大维数，以该维数定义数组；如float zv[kMaxTrack]
2)定义某变量，用于存放数组的实际维数。如int ntrack,表示一个事例中实际的径迹数。
3)定义tree，设置Branch。第三个参数给出数组的实际维数。如"zv[ntrack]/F"

运行:进入ROOT环境后
.L ex42.C
ex42()

如何从ASCII文件中读取数据转为ROOT中的TTree

ASCII是一种非常普遍的格式，虽然效率不高，但常会遇到，例如示波器所记录的数据，excel转化的数据，都可以轻松的转到root里，进行分析。

```
void readasc() {
  TFile *f = new TFile("ex44.root","RECREATE");
  TTree *T = new TTree("ntuple","data from ascii file");
  //第1个参数为要打开的文件名称
  //第2个参数是Branch的描述，即设定3个Branch x,y,z
  Long64_t nlines = T->ReadFile("basic.dat","x:y:z");
  printf(" found %lld points\n",nlines);
  T->Write();
}
```

TTree提供的ReadFile()函数很简洁且实用

运行：root -l readasc.C
或者在root环境中:
.x readasc.C

basic.dat is like this:

```
-1.102279 -1.799389 4.452822
1.867178 -0.596622 3.842313
-0.524181 1.868521 3.766139
-0.380611 0.969128 1.084074
0.552454 -0.212309 0.350281
-0.184954 1.187305 1.443902
0.205643 -0.770148 0.635417
1.079222 -0.327389 1.271904
-0.274919 -1.721429 3.038899
2.047779 -0.062677 4.197329
```

Tree vector

```
std::vector<float> vpx;
TTree *t = new TTree("tvec","Tree with vectors");
t->Branch("vpx",&vpx);

for (Int_t i = 0; i < 25000; i++) {
   Int_t npx = (Int_t)(gRandom->Rndm(1)*15);

   vpx.clear();
   for (Int_t j = 0; j < npx; ++j) {

      Float_t px,py,pz;
      gRandom->Rannor(px,py);
      vpx.push_back(px);
   }
   t->Fill();
}
```

直接把STL的容器vector放入Tree

可以参见 tutorials/tree/hvector.C

作业

1. 写一个两维高斯函数拟合上面产生的直方图(自己生成两维的直方图，利用随机数生成两维的高斯分布)

2. 把**ex41.C**改为一个**Branch**有多个**leaf**的**C**文件，保存**root**，然后读取**root**里的**leaf**信息，打印前**100**个**entries**的数字

3. 分别运行**ex42.C**和hvector.C, 存成TFile。用SetBranchAddress 打印前10个事例信息。【可用MakeClass()】

4. 在**tutorials/tree**里寻找用**class**的**object**做为**Branch**的例子，读懂

参考资料

- ROOT手册第2章，第3章
- http://root.cern.ch
- http://root.cern.ch/root/Reference.html
- http://root.cern.ch/root/Tutorials.html
- http://root.cern.ch/root/HowTo.html
- $ROOTSYS/tutorials中的各个例子
- http://hep.tsinghua.edu.cn/training/index.html （一个三段的视频教程）

root 文件与它的 tree 概念

- 目录结构：一个 root 文件就像Linux中的一个目录，它可以包含目录和没有层次限制的目标模块。

即：在可以在root文件创建不同的目录、子目录，
目录中存放不同的类对象或普通数据。
现在的root甚至支持透明的网络文件读写。

- 如要求存储大量的同类目标模块，需要：

减小磁盘空间和同时增加读取速度

- TTree 采用了 branch 的体系，每个 branch 的读取可以与别的 branch 无关。

可以把tree看成root文件中的子目录，
branch看成子目录中的文件或者子目录。