粒子物理与核物理实验中的数据分析

王喆
清华大学

第五讲：ROOT在数据分析中的应用(2)

上讲回顾

- 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， cint
- 数学函数, 直方图, 随机数, 文件等

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

本讲要点

- 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。

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。

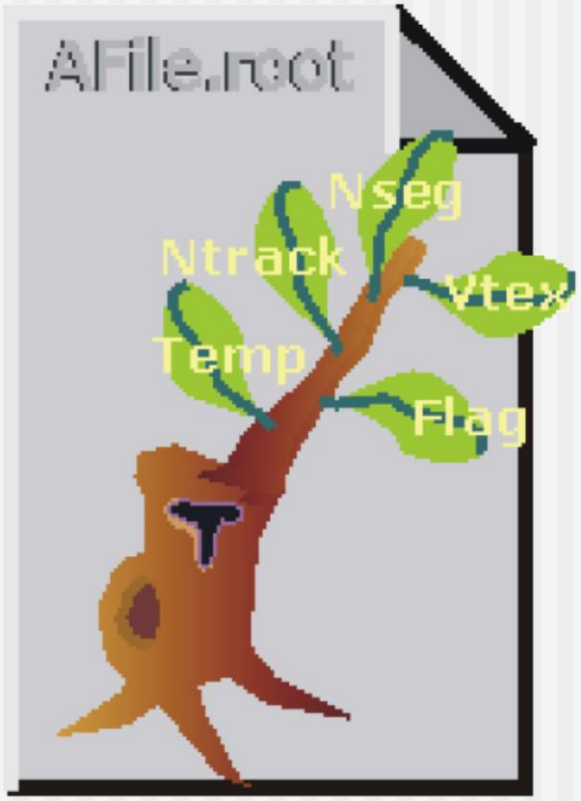
  可以把tree看成root文件中的子目录，
  branch看成子目录中的文件或者子目录。

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。
- 适用于大量的类型相同的对象
- 可以存储包括类对象、数组等各种类型数据
- 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

演示：使用TTree进行分析

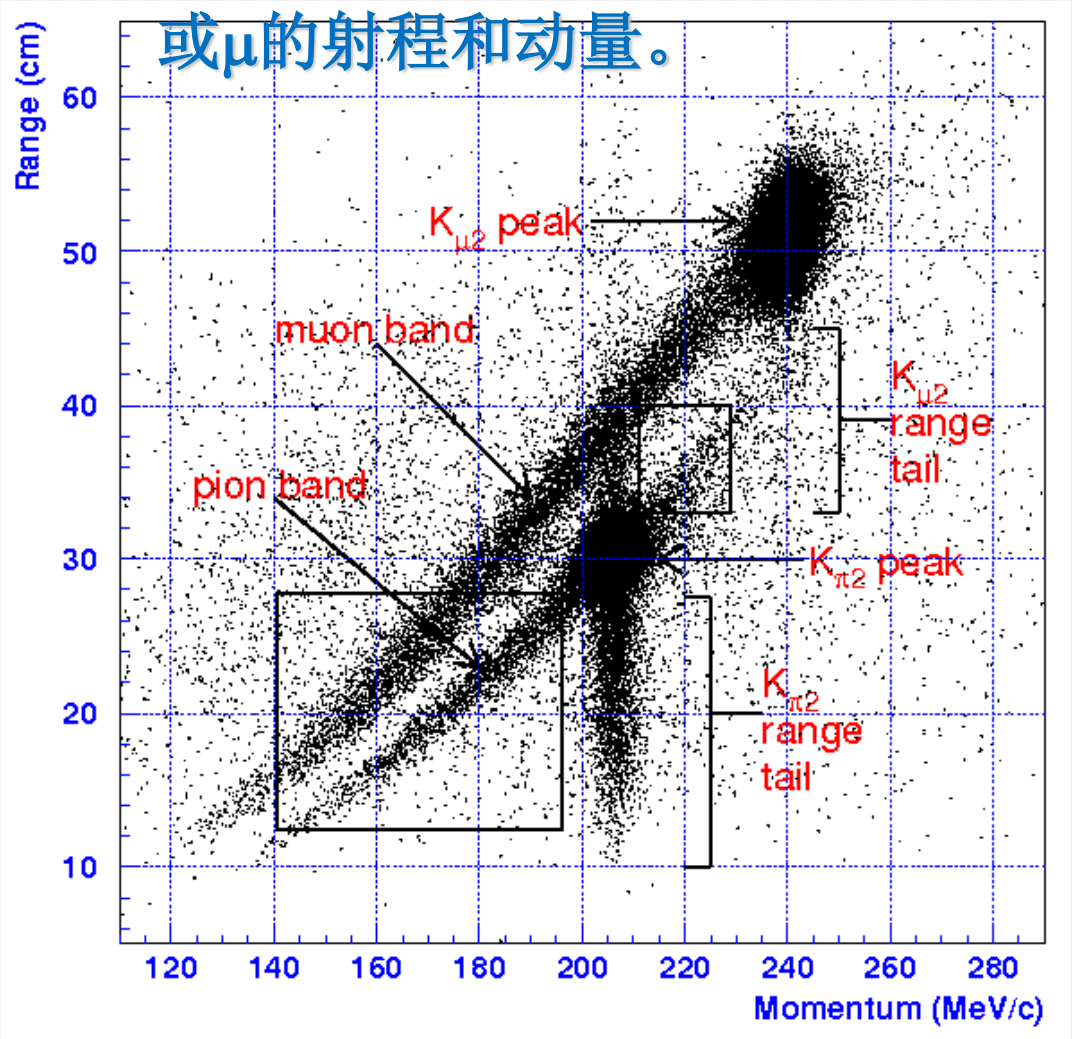
参见：kin_number_km2g_ptotmu.root

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 http://root.cern.ch/root/html526/TTree.html

名称　　描述

构造函数：

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

Branch成员函数：

virtual 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Lec5/tree1w.C

```
void tree1w() {
  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 tree1w.C 或在ROOT中: .x tree1w.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)

也可以

```
>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("py","px>0","sames");
root[5]t1->Draw("py","","sames");
root[6]TH1D h2("h2","h2",100,0,100)
root[7]t1->Project("h2","px+py")
```

做简单运算

投影到一个一维直方图

加选择条件

投影到一个一维直方图

如何读一个简单的TTree
Lec5/tree1r.C

```
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()

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.emplace_back(px);
   }
   t->Fill();
}
```

直接把**STL**的容器**vector**放入**Tree**

可以参见
tutorials/tree/hvector.C

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
```

ROOT文件中的子目录

```
//创建root文件，创建后默认目录就是在myfile.root目录中，
//实际上，root文件被当作一个目录。
TFile *fname=new TFile("myfile.root","recreate");

//gDirectory指向当前目录，即myfile.root
//可以调用mkdir函数在ROOT文件中创建一个子目录，如subdir
gDirectory->mkdir("subdir");

//进入到subdir子目录
gDirectory->cd("subdir");

//创建TTree
TTree *tree = new TTree("tree","tree in subdir");
.....
//将tree写到当前目录，即subdir中
tree->Write();
```

设置单个文件大小，自动写入新文件

- 设置文件大小要求
- TTree::SetMaxTreeSize(int size);
- 例子：aTree->SetMaxTreeSize(500000000); //500M, otherwise split into a new file

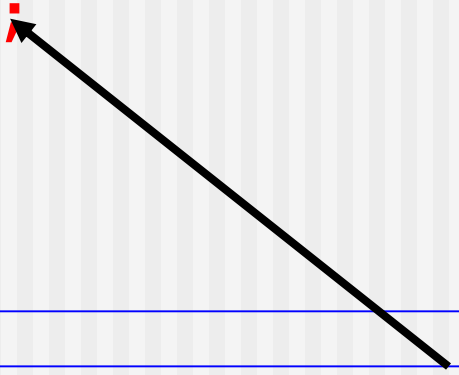
- 关闭文件
- TTree::GetCurrentFile(); //get the pointer to the current file
- OutFile->Close();

TChain: 分析多个root文件的利器(1)

TChain对象是包含相同tree的ROOT文件的列表。
参见手册TChain部分
http://root.cern.ch/root/html526/TChain.html

```
void exChain() {
 //定义TChain，t1为root文件中tree的名称!!!!!!!
 TChain* fChain= new TChain("t1");
 //添加所有文件至fChain，或根据需要添加部分root文件
 fChain->Add("Lec4/tree*.root");

 //画出t3的某个leaf，如ntrack
  fChain->Draw("px");
}
```



注意：此时，fChain等同于一个大root文件中的一个类"t1",该文件包含的事例数为所有文件中事例数之和（10^{12}以内）

TChain: 分析多个root文件的利器(2)

如果**root**文件中的类**t3**是在子目录**subdir**下，
则可以这样定义：

```
TChain* fChain= new TChain("subdir/t3");
```

或者将子目录和类的名字全部放在**Add()**函数中

```
TChain* fChain=new TChain();
fChain->Add("rootfiles/*.root/subdir/t3");
```

注意；从这里可以看出，

1) **ROOT**文件中的目录**subdir**与系统的子目录**rootfiles**同等地位;
2) **ROOT**文件的文件名在这里也类似于目录
3) **TTree**，即**t3**在这里也类似于目录

自动生成TTree的分析框架

- MakeClass,MakeSelector的运用

比如当前目录下有文件ex51.root，其中含有复杂的tree。可以用MakeClass或MakeSelector自动产生分析文件和头文件（非常方便）：

```
root [0] TFile f("ex51.root");
root [1] .ls
TFile**         ex51.root
 TFile*         ex51.root
  KEY: TTree    t4;1    Reconst events
root [2] t4->MakeClass();
或:          t4->MakeClass("AnaFrame");
```

自动产生以t4.h和t4.C文件，

或AnaFrame.h和AnaFrame.C文件。

类的定义以及Branch地址设定、分析框架都已经自动完成。

```
root [0] .L AnaFrame.C
root [1] AnaFrame t4
root [0] t4.Loop()
```

直接可以使用

MakeSelector的用法类似

分析框架的内容

重要的成员函数

```
virtual Int_t    Cut(Long64_t entry);
virtual Int_t    GetEntry(Long64_t entry);
virtual Long64_t LoadTree(Long64_t entry);
virtual void     Init(TTree *tree);
virtual void     Loop();
virtual Bool_t   Notify();
virtual void     Show(Long64_t entry = -1);
```

Loop的重要结构：

```
Long64_t nbytes = 0, nb = 0;
for (Long64_t jentry=0; jentry<nentries;jentry++) {
   Long64_t ientry = LoadTree(jentry);
   if (ientry < 0) break;
   nb = fChain->GetEntry(jentry);   nbytes += nb;
   // if (Cut(ientry) < 0) continue;
}
```

小结

- TTree的基本概念
- 如何创建TTree并写入root文件中
  为TTree设定Branch:
  tree->Branch(...);
- 如何查看root文件TTree信息，如何读取
  tree->Show(i), tree->Scan(), tree->Print()
  为Branch指定变量地址:
  tree->SetBranchAddress(...);
- 如何从ASCII格式文件读取数据生成TTree
  tree->ReadFile("filename","branch descriptor");
- TChain分析含相同类结构的多个root文件
  chain->Add(...);
- 自动生成分析框架

作业

- 利用MakeClass分析ex51.root（附件），画总动量分布图，画ntrack小于20的总动量分布，分别计算总动量的平均值和RMS
- 利用vector将上面ex51中的px排序，打印