

# 粒子物理与核物理实验中的 数据分析

---

杨振伟  
清华大学

第四讲：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

设置ROOT的3个环境变量\$ROOTSYS, \$PATH, \$LD\_LIBRARY\_PATH

- ROOT的语法简介

完全兼容c++语法, Int\_t, Float\_t, Double\_t, ...

- 数学函数, 直方图, 随机数, 文件, 散点图舍选法等

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

补充提示1: TH1F \*h1=new TH1F("h1", "", 100, 0., 1.);  
h1->GetBinContent(i); //i=0, 1, ..., 101.

可以得到102个值:i=0 对应underflow

i=101对应 overflow

i=1, 2, ..., 100为相应bin的值。

补充提示2: gRandom->SetSeed(seed);需要明显给定seed。如  
gRandom->SetSeed(0);

# 本讲要点

- ROOT中tree的概念(类TTree)

什么是tree, 为什么tree存取数据

- 如何定义、填充TTree并写入文件

- 如何读取实验数据填充为TTree

- 如何查看或读取ROOT文件中的tree

- TChain:

**同时处理多个相同root文件**

(root文件中含有相同的tree, 总事例不超过 $10^{12}$  个)

```
chain->Add("/data/sns/090324_01.root");
```

```
chain->Add("/data/sns/090325*.root");
```

# root 文件与它的 tree 概念

❑ 一个 root 文件就像 UNIX 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。

❑ 如要求存储大量的同类目标模块，需要引入概念：

**TTree:** 减小磁盘空间和增加读取速度方面被优化

**TNtuple:** 只能存储浮点数的TTree。尽量避免使用。

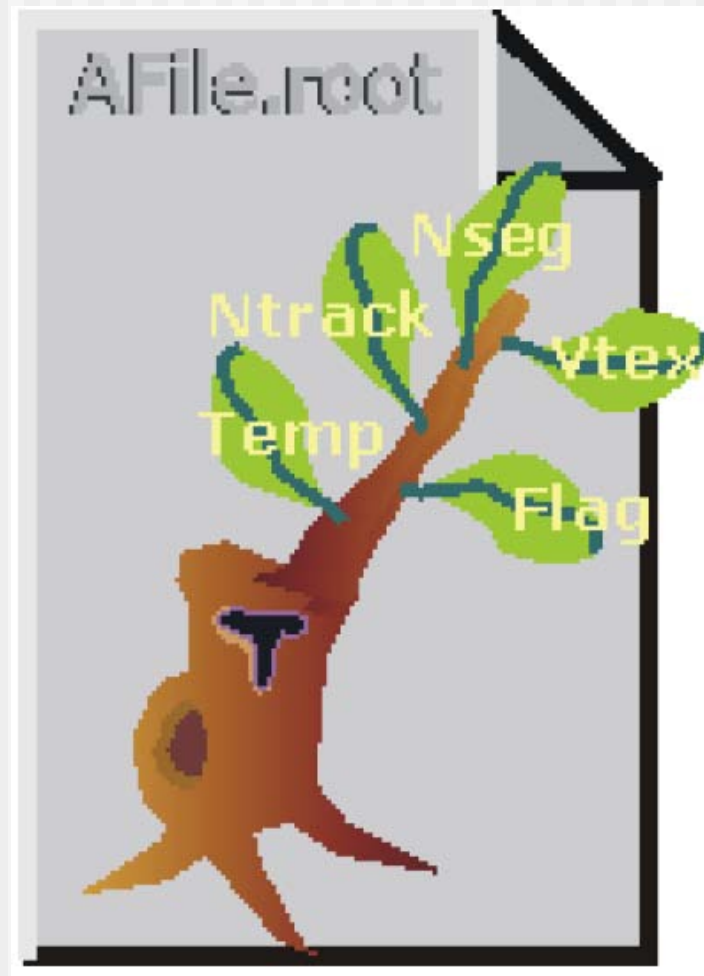
❑ TTree 减少了每个目标模块的header，但仍保留类的名字，而每个同类的目标模块名字可以被压缩。

❑ 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

- 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



**TTTree是ROOT最强大的概念之一**

# 下载本讲例子到本地计算机

```
cd <你的工作目录>
```

```
wget hep.tsinghua.edu.cn/~yangzw/CourseDataAna/examples/Lec4.tgz
```

```
tar -zxvf Lec4.tgz
```

```
cd Lec4
```

或者

```
cd <你的工作目录>
```

```
scp -r \$USER@166.111.32.64:/home/yangzw/examples/Lec4.tgz .
```

(注意最后有个".", 表示复制到当前目录, 需要输入密码)

```
tar -zxvf Lec4.tgz
```

```
cd Lec4 (本讲所有例题都在该目录下)
```

# TTree的定义

参见 <http://root.cern.ch/root/html526/TTree.html>

构造函数:

名称

描述

**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

/home/yangzw/examples/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195-196

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。  
**好的做法是实验一个事例填充一次!!!**

将tree写入root文件中存盘

运行: root -l ex41.C 或ROOT环境中: .x ex41.C



# 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");
```

# 如何读写含有不定长数组的tree(1)

/home/yangzw/examples/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");  
...
```

void ex42r() { //读取数据, 适用于简单分析

```
TFile *f = new TFile(rootfile);  
TTree *t3 = (TTree*)f->Get("t3");  
t3->Draw("sqrt(px*px+py*py)");  
htemp->SetLineColor(2);  
t3->Draw("sqrt(px*px+py*py)",  
    "zv > 100", "sames");
```

直接画出Branch/Leaf,  
可以加很多条件。

!! 如何获取root  
文件中的tree指针

运行:进入ROOT环境后  
.  
L ex42.C  
ex42w()  
ex42r()

- 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"

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。

# 如何读写含有不定长数组的tree(2)

/home/yangzw/examples/Lec4/ex42.C

```
void ex42r2() {
    TFile *f = new TFile(rootfile);
    TTree *t3 = (TTree*)f->Get("t3");

    //步骤1:定义好必要的变量
    const Int_t kMaxTrack = 100;
    Int_t ntrack;
    Float_t px[kMaxTrack]; //[ntrack]
    Float_t py[kMaxTrack]; //[ntrack]
    Float_t zv[kMaxTrack]; //[ntrack]
    Double_t pv[3];
    //步骤2: 用SetBranchAddress函数
    //将tree的Branch与定义好的变量
    //地址联系起来。
    t3->SetBranchAddress("ntrack",
    &ntrack);
    t3->SetBranchAddress("px", px);
    t3->SetBranchAddress("py", py);
    t3->SetBranchAddress("zv", zv);
    t3->SetBranchAddress("pv", pv);
```

//获取事例总数

```
Int_t nentries = t3->GetEntries();
```

```
TH1I *hntrack = new TH1I("hntrack","trk n",25,0,50);
```

```
TH1F *hpt = new TH1F("hpt","trk pt",100,0,10);
```

//步骤3:对所有事例循环

```
for (int i=0;i<nentries;i++) {
```

```
    t3->GetEntry(i); //获取第i个事例
```

```
    hntrack->Fill( ntrack );
```

```
    for (int j=0;j<ntrack;j++) {
```

```
        Float_t pt = sqrt(px[j]*px[j]+py[j]*py[j]);
```

```
        hpt->Fill( pt );
```

```
    }
```

```
}
```

```
TCanvas *myC = new TCanvas("myC","",10,10,600,400);
```

```
hntrack->Draw("e");
```

```
hntrack->GetYAxis()->SetRangeUser(0,60);
```

```
TCanvas *myC1 = new TCanvas("myC1","",10,10,600,400);
```

```
hpt->Draw();
```

```
}
```

问题: 对ntrack和px  
的处理有什么差别? 为  
什么?

运行: 进入ROOT环境后  
.L ex42.C  
ex42w()  
ex42r2()

每获取一个事例, 这些Branch直接赋值给指定的变量。可以在循环中设定选取条件, 选择分析数据。进行复杂细致的分析推荐使用这种方法。

注: 程序中//[ntrack]的意义参见手册171-174 Streamer

# 如何将类对象设定为tree的Branch

/home/yangzw/examples/Lec4/ex43.C

```
...  
TTree *t3 = new TTree("t3","Reconst events");  
//Evt_t是已经定义好的类(详见ex43.C)。这里的myevt一定要用new的方式定义,  
//然后就可以直接将该对象设为tree的一个Branch。  
//第1个参数为Branch的名字, 第2个为类的名字(可省略), 第3个为对象指针的地址(!),  
//第4个为缓存大小, 第5个为分割级别(split-level)。  
//参见手册196-197"Adding a Branch to Hold an Object"
```

```
Evt_t *myevt = new Evt_t();  
t3->Branch("evt","Evt_t",&myevt,32000,1);
```

略过, 但并非不重要

```
...  
//读取tree时, 可以直接将对象指针的地址的赋给相应的Branch  
//需要提醒的是, 第1个参数为Branch的名称, 必须与写入时指定的名称相同。
```

```
Evt_t *myevt = 0 ;  
t3->SetBranchAddress("evt",&myevt);
```

```
...  
//GetEntry(i)获得第i个entry后, 通过myevt->ntrack (或其它成员变量)可以获得  
//相应的数据。
```

```
t3->GetEntry(i);  
hntrack->Fill( myevt->ntrack );
```

语法更严格, 必须include需要的头文件

详见ex43.C。运行时必须通过外部编译器如: **root -l ex43.C+**  
或者在ROOT环境中 **.x ex43.C+** 这里的"+"是必须的。

# 如何从ASCII文件中读取数据转为ROOT中的TTree

## /home/yangzw/examples/Lec4/ex44.C

有时候记录的实验数据是**ASCII**格式，或者二进制格式。我们可以读取这些数据转换成**ROOT**中的**tree**，方便进行数据分析。**ex44.C**读取**basic.dat**(ASCII码)，转成最简单的**TTree**。**basic.dat**中的数据分3列，分别为**x,y,z**坐标。

```
void ex44() {  
    ifstream in; // 定义文件流对象, i表示in, f表示file, 即从某文件中读取数据  
    in.open("basic.dat"); // 打开该文件  
    Float_t x,y,z;  
    Int_t nlines = 0;  
    TFile *f = new TFile("ex44.root","RECREATE");  
    TH1F *h1 = new TH1F("h1","x distribution",100,-4,4);  
    //TNtuple看成特殊的TTree, 只可以存放浮点型数据。  
    TNtuple *ntuple = new TNtuple("ntuple","data aus ascii","x:y:z");  
    while (1) {  
        in >> x >> y >> z; //从文件中读取一行, 分别赋值给x,y,z。  
        if (!in.good()) break;  
        if (nlines < 5) printf("x=%8f, y=%8f, z=%8f\n",x,y,z);  
        h1->Fill(x);  
        ntuple->Fill(x,y,z); //填充TNtuple  
        nlines++;  
    }  
    printf(" found %d points\n",nlines);  
    in.close();  
    f->Write();  
}
```

注：3个Branch  
分别为x, y, z

每个事例填充一次

运行: **root -l ex44.C**  
或者在root环境中:  
**.x ex44.C**

# 改成TTree方式:

```
void ex44() {
    Float_t x,y,z;
    Int_t nlines = 0;
    TFile *f = new TFile("ex44.root","RECREATE");
    TH1F *h1 = new TH1F("h1","x distribution",100,-4,4);
    //TNtuple看成特殊的的TTree，只可以存放浮点型数据。
    TTree *mytree = new TTree("mytree","aaaaaaaaaaaaa");
    mytree->Branch("x",&x,"x/F");
    mytree->Branch("y",&y,"y/F");
    mytree->Branch("z",&z,"z/F");
    mytree->Branch("w",&w,"w/C");
    while (1) {
        in >> x >> y >> z; //从文件中读取一行，分别赋值给x,y,z。
        if (!in.good()) break;
        if (nlines < 5) printf("x=%8f, y=%8f, z=%8f\n", x, y, z);
        h1->Fill(x);
        mytree->Fill(); //填充TNtuple
        nlines++;
    }
    printf(" found %d points\n",nlines);
    in.close();
    f->Write();
}
```

略过，但并非不重要

# 如何从ASCII文件中读取数据转为ROOT中的TTree

/home/yangzw/examples/Lec4/ex44a.C

读取**ASCII**格式文件还有个更简便的方式，即用**TTree**的**ReadFile**函数：  
**tree->ReadFile(parameter1,parameter2);**

参见手册**212**页**Example5**

```
void ex44a() {  
    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();  
}
```

运行: **root -l ex44a.C**  
或者在**root**环境中:  
**.x ex44a.C**

**TTree**提供的**ReadFile()**函数  
更简洁，更强大？



# 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(5.26)231页Chains以及  
<http://root.cern.ch/root/html526/TChain.html>

```
void ex45() {  
    //定义TChain, t3为root文件中tree的名称!!!!!!!  
    TChain* fChain= new TChain("t3");  
    //添加所有文件至fChain, 或根据需要添加部分root文件  
    fChain->Add("rootfiles/*.root");  
  
    //画出t3的某个leaf, 如ntrack  
    fChain->Draw("ntrack");  
}
```



**问题:** root文件中  
多个tree怎么办?

**注意:** 此时, fChain等同于一个大root  
文件中的一个类"t3", 该文件包含的事例  
数为所有文件中事例数之和( $10^{12}$ 以内)

# 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();
```

## 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

由此可以得到，当ROOT文件中存在多个TTree，比如t3，t4时，需要哪个就使用哪个，其它的与我们无关。

## TChain: 分析多个root文件的利器(3)

/home/yangzw/examples/Lec4/ex45.C

```
void ex45() {  
    TChain* fChain= new TChain("t3"); //t3为文件中tree的名称  
    fChain->Add("rootfiles/*.root"); //将需要的root文件加入fChain  
    fChain->Draw("ntrack"); //可以直接把fChain当成TTree t3  
    //也可以跟读取root文件中的tree类似, 设定Branch的地址, 进行细致分析。  
    const Int_t kMaxNum = 50;  
    Int_t      ntrack;  
    Float_t    px[kMaxNum]; // [ntrack]  
    Float_t    py[kMaxNum]; // [ntrack]  
    Float_t    zv[kMaxNum]; // [ntrack]  
    Double_t   pv[3];  
    fChain->SetBranchAddress("ntrack", &ntrack); //设定Branch  
    fChain->SetBranchAddress("px", px);  
    fChain->SetBranchAddress("py", py);  
    fChain->SetBranchAddress("zv", zv);  
    fChain->SetBranchAddress("pv", pv);  
    cout << "Entries=" << fChain->GetEntries() << endl;  
    for (int i=0;i<10;i++) {  
        fChain->GetEntry(i);  
        cout << "ntrack = " << ntrack << endl;  
    }  
}
```

运行: **root -l ex45.C**  
或者在root环境中:  
**.x ex45.C**

## TChain: 分析多个root文件的利器(4)

/home/yangzw/examples/Lec4/ex45a.C

假如ROOT文件的Branch是类对象，如ex43.C生成的ex43.root，在用TChain进行分析处理时，往TChain中添加文件与例子ex45.C完全相同。只是在设定Branch进行详细分析时，需要用例子ex43.C中给出的读取tree信息的方式。详见ex45a.C

```
void ex45a() {  
    TChain* fChain= new TChain("t3");  
    fChain->Add("rootfilesclass/*.root");  
    fChain->Draw("ntrack");  
  
    Evt_t *myevt = 0;  
    fChain->SetBranch("evt", &myevt);  
    cout << "Entries=" << fChain->GetEntries() << endl;  
    for (int i=0;i<10;i++) {  
        fChain->GetEntry(i);  
        cout << "ntrack = " << myevt->ntrack << endl;  
    }  
}
```

略过，但并非不重要

运行: **root -l ex45a.C+** 或者在root环境中:  
**.x ex45a.C+** **"+"是必需的。**

与ex43.C中的函数ex43r2()进行比较，进一步理解如何读取tree中存储类对象的Branch。

注意脚本中**include**了很多头文件。

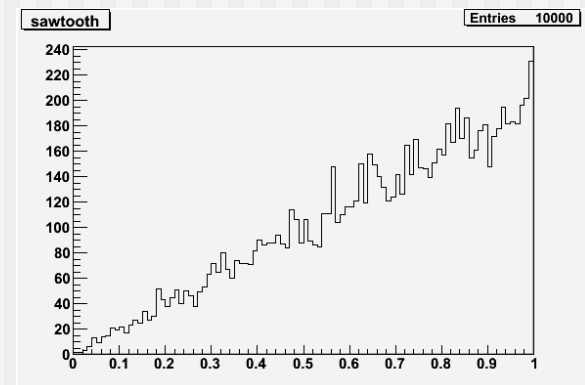
# SDA第三章练习(3.3a)

/home/yangzw/examples/Lec4/ex4SDA/exercise33a.C

根据计算可得若 $r$ 满足 $(0,1)$ 区间均匀分布, 则 $x(r) = x_{\max} * \sqrt{r}$ 满足 $(0, x_{\max})$ 之间的锯齿分布。

下面这段程序先产生随机数 $r$ , 然后计算 $x(r)$ , 填充到直方图中, 验证 $x(r)$ 是否为锯齿分布。

```
void exercise33a() {  
    const Int_t NEntry = 10000 ; //填充直方图10000次  
    const Int_t NBin   = 100 ; //直方图分100个bin  
    Float_t xMax = 1.0 ;  
    gStyle->SetOptStat("e"); //只给出entries统计信息  
    //定义直方图, 100bin, 区间(0,1)  
    TH1F *h1 = new TH1F("h1","sawtooth",NBin,0,xMax);  
    gRandom->SetSeed(0);  
    for (int i=0;i<NEntry;i++) {  
        float r = gRandom->Rndm() ; //产生(0,1)之间的随机数  
        float xr = xMax*sqrt(r); //根据公式计算x(r)  
        h1->Fill( xr );           //填充直方图  
    }  
    h1->Draw();  
    c1->SaveAs("pic_ex33a.gif");  
    c1->SaveAs("pic_ex33a.eps");  
}
```

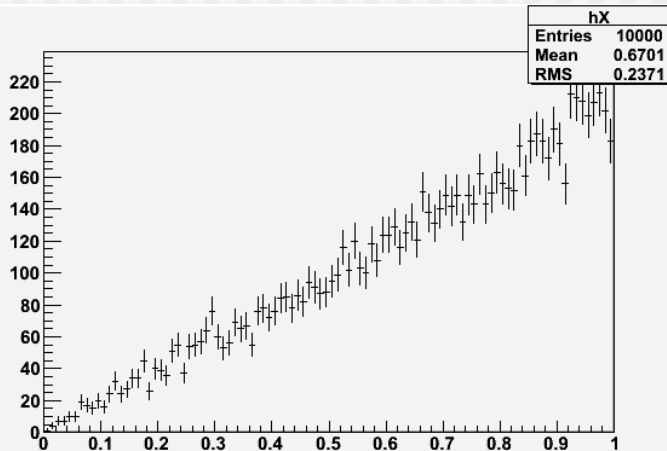


# SDA第三章练习(3.3b)

/home/yangzw/examples/Lec4/ex4SDA/exercise33b.C

## 用舍选法获得锯齿分布

```
void exercise33b() {
    const Int_t NEntry = 10000;
    const Int_t NBin = 100 ;
    Double_t x[NEntry] ;
    Double_t xMin = 0.0 ;
    Double_t xMax = 1.0 ;
    gStyle->SetOptStat(1111);
    gRandom->SetSeed(0);
    //用舍选法产生锯齿分布
    mypdf(x,NEntry,xMin,xMax);
    TH1F *hX =
        new TH1F("hX","",NBin,0,xMax);
    for (int i=0;i<NEntry;i++) hX->Fill( x[i] );
    hX->Draw("e");
    c1->SaveAs("pic_ex42.gif");
    c1->SaveAs("pic_ex42.eps");
}
```



```
void mypdf( Double_t *x, const Int_t NEntry,
Double_t xMin, Double_t xMax ) {
    Double_t mycut = xMax ;
    Double_t fmax = -999.;
    for (Int_t i=0;i<100; i++) {
        Double_t r = gRandom->Rndm();
        r = r*xMax;
        Double_t f = 2.0*r/xMax/xMax; //分布函数 f
        if (fmax<f) fmax = f;
    } //该循环寻找分布函数的最大值。
    fmax = 1.2*fmax; //放宽最大值
    Int_t nevt = 0;
    while(nevt<NEntry){
        Double_t r = gRandom->Rndm();
        r = r*xMax;
        Double_t z = 2.0*r/xMax/xMax; //f(z)
        if(z>fmax) {
            fmax=z;
            cout<<"z > fmax, find again!!!"<<endl;
        }
        Double_t u = gRandom->Rndm();
        u = u*fmax;
        if(u<=z) { //如果u*fmax<=f(z), 选取, 否则舍弃。
            if(TMath::Abs(r)<mycut){
                x[nevt]=r;
                nevt++;
            }
        } //end of if u<=z
    } //end of while nevt<NEntry
    return;
}
```

# SDA第三章练习(3.4b)

/home/yangzw/examples/Lec4/ex4SDA/exercise34b.C

当n很大时， n个均匀分布随机数之和满足高斯分布。练习ROOT脚本的参数

```
void exercise34b(Int_t n) {
    const Int_t NEntry = 10000;
    const Int_t NBin = 100;
    gROOT->SetStyle("Plain");
    gStyle->SetOptStat(1111);
    gRandom->SetSeed(0);

    TH1D *hZ = new TH1D("hZ","",NBin,-3,3);
    for (int i=0;i<NEntry;i++) {
        float z = genZ(n);
        hZ->Fill(z);
    }

    TCanvas *myC = new TCanvas("myC","",10,10,800,600);
    gPad->SetBottomMargin(0.15);
    hZ->Draw("e");
    TString xTitle( Form("z=(#sum_{i=1}^{n}x_{i}-n/2)/ #sqrt{#frac{n}{12}}, n=%i",n) );
    hZ->GetXaxis()->SetTitle(xTitle);
    hZ->GetXaxis()->CenterTitle();
}
```

//产生n个(0,1)之间均匀分布的随机数，求这n个随机数之和y  
//返回 $z = (y - n/2) / \sqrt{n/12}$ .

```
Double_t genZ(Int_t n) {
    Double_t y = 0;
    for (int i=0;i<n;i++) y += gRandom->Rndm();
    Double_t z = (y-n/2.)/sqrt(n/12.);
    return z;
}
```

运行: root -l

.L exercise34b.C

exercise34b(1)

exercise34b(2)

...

exercise34b(20)

可以看到当n变大时，z的分布越来越接近标准高斯分布

- 1)脚本的运行需要参数n，n为正整数
- 2)n=1时，z仍为均匀分布
- 3)n=2时，z接近锯齿分布
- 4)n~12时，z非常接近高斯分布
- 5)注意ROOT中Latex公式的写法。



# 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->SetBranchAddresses(...);`
- 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(...);`
- 舍选法, 含参数的ROOT脚本

# 练习

1. /home/yangzw/examples/Lec4/ascii\_data/random1.dat

该文本文件有5列数据，前2列为整型，之后两列为浮点型，最后一列为字符串型。读取该文件，把这些数据写入tree，并存到random1.root文件中。为tree设置5个Branch: npart, ntrack, px, py, ch，分别为整型、整型、浮点型、浮点型、字符串

提示：参照ex41.C和ex44.C，将二者综合起来。

设置字符串的Branch可以如下：

```
Char_t ch[4];
```

```
tree1->Branch("ch",ch,"ch/C");
```

2. 用练习1的程序将ascii\_dat/random2.dat, random3.dat分别读取为random2.root, random3.root。加上习题1的root文件，共3个。用这3个root文件数据，画出npart,ntrack,px以及py的分布。画出npart:ntrack的散点图。

提示：参照ex45.C，用TChain将这3个root文件连起来。

```
tree->Draw("npart:ntrack");
```

可以看2个变量的关联

3. 课后仔细阅读ROOT手册第12章，以及第11章。熟悉TTree的使用。

# 参考资料

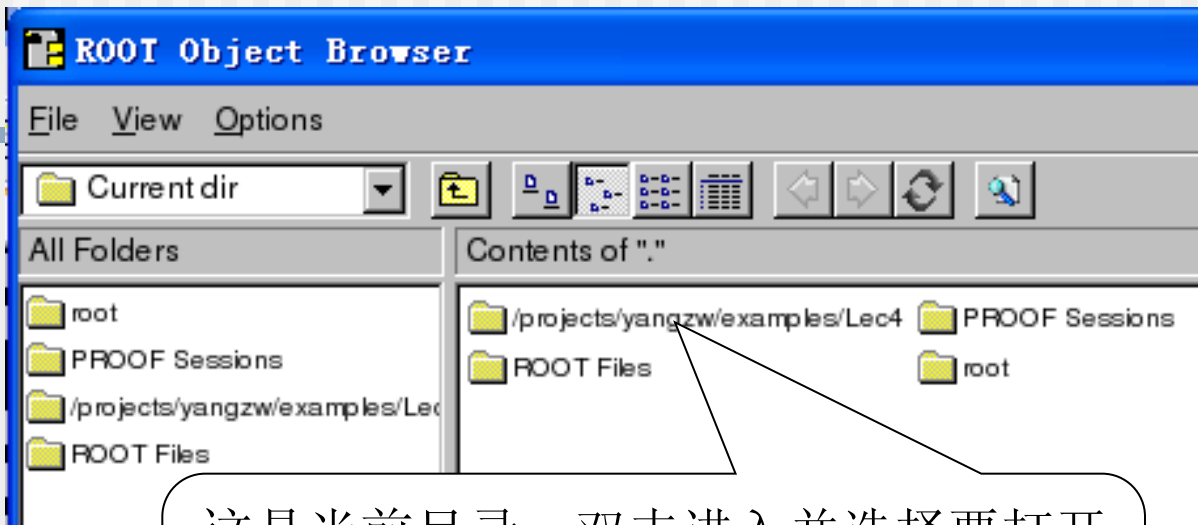
---

- ROOT手册第12章
- ROOT手册第11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/tree目录中的例子

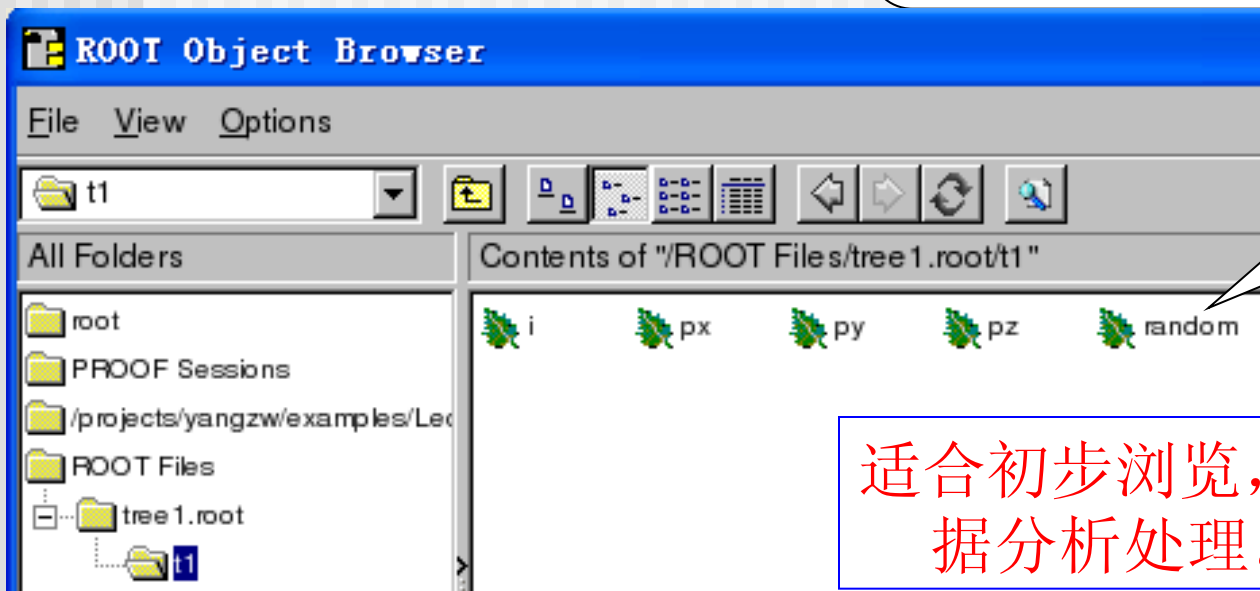
## 查看Tree的信息(2)

### TBrowser b

TBrowser打开一个浏览器，从中可以选择root文件，并一层层进入其中的tree，branch以及leaf。类似于Windows下的Explorer。



这是当前目录，双击进入并选择要打开的root文件，以及文件中的tree，最后可以看到tree的各个leaf



双击leaf可以查看leaf的直方图

适合初步浏览，但不适合具体的数据分析处理。并不推荐使用。