Tutorial on ROOT and Ttree

Special Topic 63

Amit Pathak

Institute of High Energy Physics Chinese Academy of Sciences,

Date: 2019-03-15

Introduction

WHAT IS ROOT ?

• ROOT is an object oriented framework

• It has a C/C++ interpreter (CINT) and C/C++ compiler (ACLIC)

• ROOT is used extensively in High Energy Physics for "data analysis"

- Reading and writing data files

- Calculations to produce plots, numbers and fits.

WHY ROOT ?

• It can handle large files (in GB) containing N-tuples and Histograms

• Multiplatform software

• Its based on widely known programming language C++

• Its free

Outline

- This special topic is about doing data analysis using ROOT Tree.
- I will discuss writing and reading events to/from tree, where trees are defined in various formats.
- I will start fimples examples and slowly addmore features that makes data storage and analysis easier.
- The programs for the examples discussed here are available in the tutorial directory. I have understood from the website and used them for practice.
- Reference: https://root.cern.ch/root/htmldoc/guides/users-guide/ ROOTUsersGuide.html

Ttree

Tree

ROOT Tree (TTree class) is designed to store large quantities of same-class objects, and is optimized to reduce disk space and enhance access speed.
It can hold all kind of data, such as objects or arrays, in addition to simple types.

An example for creating a simple Tree

```
void tree_example1() {

  TFile *f = new TFile("myfile.root", "RECREATE");
  TTree *T = new TTree("T","simple tree");
  TRandom r;
  Float_t px,py,pt;
  Double_t random;
  UShort_t i;
  T->Branch("px",&px,"px/F");
  T->Branch("py",&py,"py/F");
  T->Branch("pt",&pt,"pt/F");
  T->Branch("random",&random,"random/D");

  for (i = 0; i < 10000; i++) {
    r.Rannor(px,py);
    pt = std::sqrt(px*px + py*py);
    random = r.Rndm();
    T->Fill();
  }

  f->Write();
  f->Close();
}
```

Run the macro:

root -l tree_example1.C

More on TBranch

- The class for a branch is called TBranch.
- A variable in a TBranch is called a leaf (TLeaf)
- If two variables are independent they should be placed on separate branches. However, if they are related it is efficient to put them on same branch.
- TTree::Branch() method is used to add a TBranch to TTree.
- The branch type is decided by the object stored in it.
- A branch can hold an entire object, a list of simple variables, content of a folder, content of TList, or an array of objects.

For example:

tree->Branch("Ev_Branch", &event, temp/F:ntrack/I:nseg:nvtex:flag/i");

where "event" is structure with one float, three integers, and one unsigned integer.

Accessing TTree

Show an entry:

```
root [ ] TFile f("myfile.root")
root [ ] T->Show(10)
======> EVENT:10
 px          = 0.680243
 py          = 0.198578
 pt          = 0.708635
 random      = 0.586894
```

Scan a few variables:

```
root [ ] T->Scan("px:py:pt")
****************************************
*  Row  *     px *     py *     pt *
****************************************
*     0 * 0.8966467 * -1.712815 * 1.9333161 *
*     1 * 1.5702210 * 0.5797516 * 1.6738297 *
*     2 * 0.6975117 * 0.1442547 * 0.7122724 *
*     3 * 0.0616207 * -1.009907 * 1.0117853 *
*     4 * -0.054552 * 1.3832200 * 1.3842953 *
*     5 * -2.017178 * 1.4682819 * 2.4949667 *
*     6 * 0.8903368 * 2.5101616 * 2.6633834 *
*     7 * -1.098390 * -0.318103 * 1.1435256 *
*     8 * 0.3865155 * 0.0235152 * 0.3872301 *
*     9 * 1.8970719 * 1.9546536 * 2.7238855 *
*    10 * 0.6802427 * 0.1985776 * 0.7086347
```

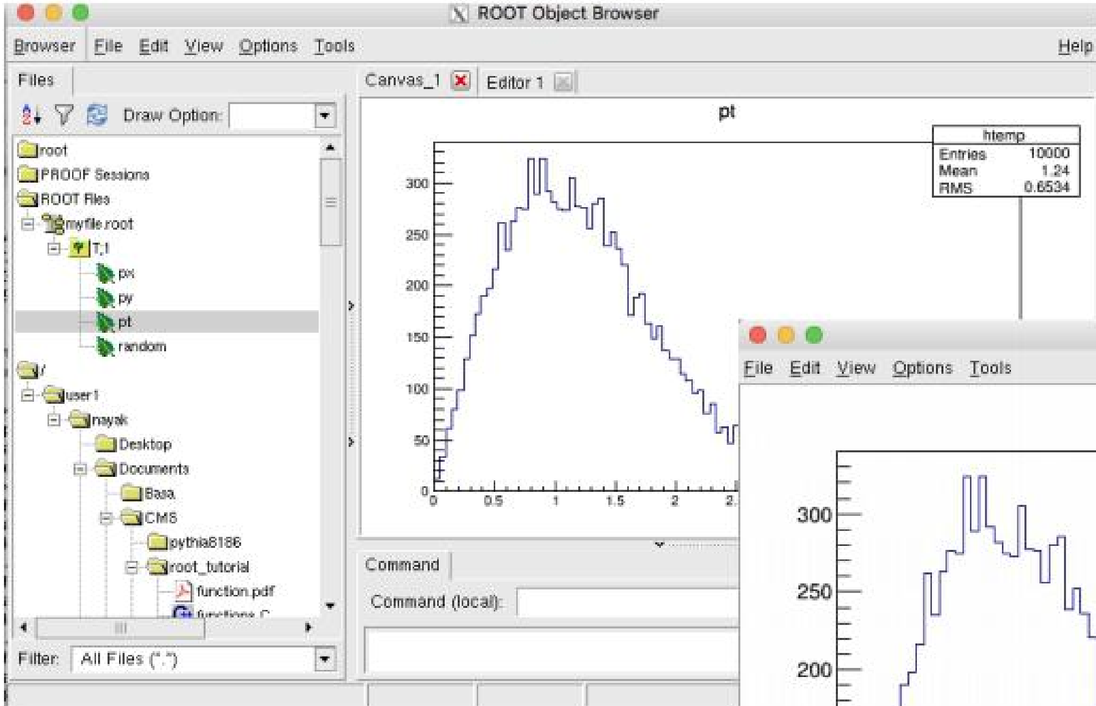
Print the Tree structure:

```
root [ ] T->Print()

******************************************************************************
*Tree    :T         : simple tree                                            *
*Entries :    10000 : Total =          202834 bytes  File  Size =     169658 *
*        :          : Tree compression factor =   1.19                       *
******************************************************************************
*Br    0 :px        : px/F                                                   *
*Entries :    10000 : Total  Size=      40594 bytes  File Size  =      37262 *
*Baskets :        2 : Basket Size=      32000 bytes  Compression=   1.08     *
*............................................................................*
*Br    1 :py        : py/F                                                   *
*Entries :    10000 : Total  Size=      40594 bytes  File Size  =      37265 *
*Baskets :        2 : Basket Size=      32000 bytes  Compression=   1.08     *
*............................................................................*
*Br    2 :pt        : pt/F                                                   *
*Entries :    10000 : Total  Size=      40594 bytes  File Size  =      35874 *
*Baskets :        2 : Basket Size=      32000 bytes  Compression=   1.12     *
*............................................................................*
*Br    3 :random    : random/D                                               *
*Entries :    10000 : Total  Size=      80696 bytes  File Size  =      58600 *
*Baskets :        3 : Basket Size=      32000 bytes  Compression=   1.37     *
*............................................................................*
```

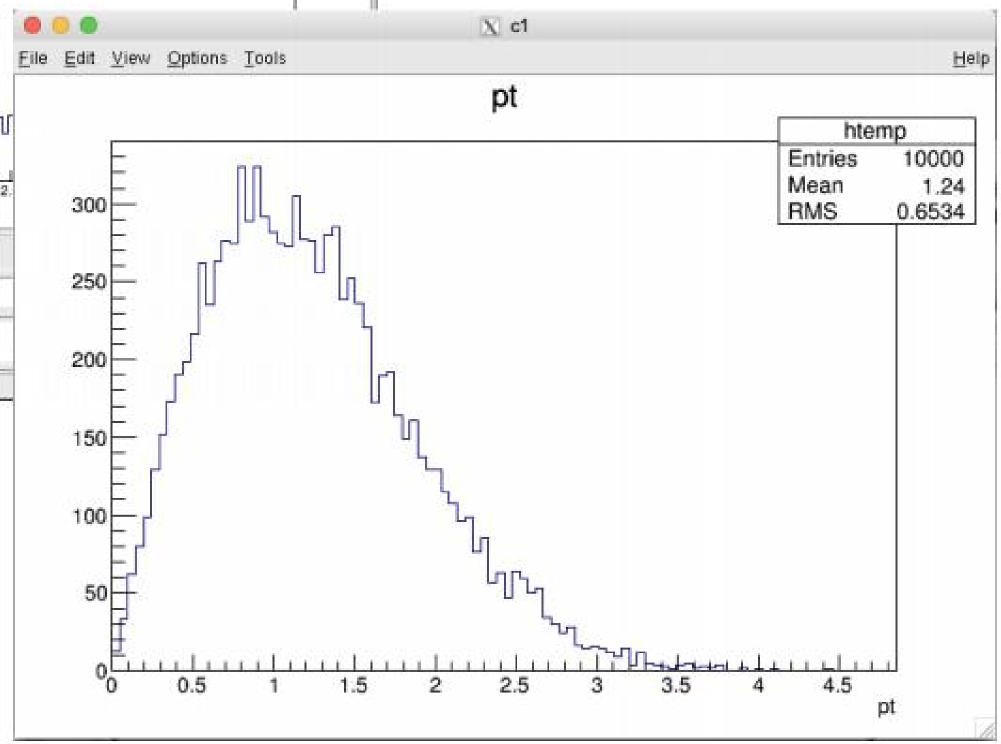
Accessing TTree

Open and Draw branches using TBrowser:



Using TTree::Draw()

T->Draw("pt")



2D plot:
T->Draw(“py:px”)

Simple Analysis using TTree::Draw()

Applying cuts:
T->Draw("pt", "px>0&&py>0")

For applying "weights" to the distribution:
Selection = "weight * (boolean expression)"
Where "weight" is a event-by-event weight stored as a branch in the ntuple (e.g. cross section * luminosity for the normalization)

Using TCuts:
TCut c1 = "px > 0"
TCut c2 = "py < 0"
TCut c3 = c1 && c2
T->Draw("pt", c3)
T->Draw("pt", c1 && c2)
T->Draw("pt", c1 && "py > 0")
TCut c4 = c1 || c2
T->Draw("pt", c4)
T->Draw("pt", c1 || c2)

Filling Histograms:
TH1F *hist = new TH1F("hist", "", 100, 0., 2.);
T->Draw("pt>>hist");
hist->SetLineColor(2);
hist->Draw("same")

Adding Branch with Arrays

```
//Tree with fixed size arrays
void tree_example3() {

  TFile *f = new TFile("myfile.root", "RECREATE");
  TTree *T = new TTree("T","tree with fixed array");
  TRandom r;
  Float_t px[5],py[5],pt[5];

  T->Branch("px",px,"px[5]/F");
  T->Branch("py",py,"py[5]/F");
  T->Branch("pt",pt,"pt[5]/F");


  for (Int_t i = 0; i < 10000; i++) {
    Float_t x, y;
    for(Int_t j = 0; j < 5; j++){
      r.Rannor(x, y);
      px[j] = x;
      py[j] = y;
      pt[j] = std::sqrt(x*x + y*y);
    }
    T->Fill();
  }

  f->Write();
  f->Close();
}
```

Run Macro:

root -l tree_example3.C

See the contents:

```
root [0] TFile f("myfile.root")
root [1] T->Show(0)
======> EVENT:0
 px              = 0.896647,
                   0.685336, 1.58414, 0.697512, -0.318623
 py              = -1.71282,
                   0.0155619, 0.879379, 0.144255, -0.442143
 pt              = 1.93332,
                   0.685512, 1.81185, 0.712272, 0.544988
```

```
//Tree with variable size arrays
void tree_example4() {

  TFile *f = new TFile("myfile.root", "RECREATE");
  TTree *T = new TTree("T","tree with fixed array");
  TRandom r;
  Float_t px[10],py[10],pt[10];
  Int_t np;

  T->Branch("np",&np,"np/I");
  T->Branch("px",px,"px[np]/F");
  T->Branch("py",py,"py[np]/F");
  T->Branch("pt",pt,"pt[np]/F");


  for (Int_t i = 0; i < 1000; i++) {

    Float_t x, y;
    np = (Int_t)(r.Rndm()*10);
    for(Int_t j = 0; j < np; j++){
      r.Rannor(x, y);
      px[j] = x;
      py[j] = y;
      pt[j] = std::sqrt(x*x + y*y);
    }
    T->Fill();
  }

  f->Write();
  f->Close();
}
```

```
root [1] T->Show(4)
======> EVENT:4
 np              = 7
 px              = -0.924715,
                   0.567014, 0.261468, -0.545032, 0.156111, -0.698435,
                   0.297523
 py              = -1.52197,
                   0.0129729, -1.429, -0.23653, 1.7405, -1.03075,
                   0.217231
 pt              = 1.78087,
                   0.567162, 1.45273, 0.594144, 1.74749, 1.2451,
                   0.368388
```

Branch with Vectors

Advantage: No need to define arrays with MAX SIZE

```
//Tree with vectors
#include <vector>

void tree_example6() {

  TFile *f = new TFile("myfile.root", "RECREATE");
  TTree *T = new TTree("T","tree with fixed array");
  TRandom r;
  std::vector<Float_t> px, py, pt;
  Int_t np;

  T->Branch("px", "std::vector<Float_t>", &px);
  T->Branch("py", "std::vector<Float_t>", &py);
  T->Branch("pt", "std::vector<Float_t>", &pt);

  for (Int_t i = 0; i < 1000; i++) {

    //clean vectors for each event
    px.clear(); py.clear(); pt.clear();
    //Fill vectors for each event
    Float_t x, y;
    np = (Int_t)(r.Rndm()*10);
    for(Int_t j = 0; j < np; j++){
      r.Rannor(x, y);
      px.push_back(x);
      py.push_back(y);
      pt.push_back(std::sqrt(x*x + y*y));
    }
    T->Fill();
  }

  f->Write();
  f->Close();
}
```

You get only pointers to vectors in TTree::Show() method
TTree::Draw() works only for basic type vectors

```
root [1] T->Show(4)
======> EVENT:4
 px              = (vector<float>*)0x11a9d10
 py              = (vector<float>*)0x11abcf0
 pt              = (vector<float>*)0x11ace80
root [2] T->Draw("px")
```

```
//Read Tree with vectors as branch
#include <vector>

void tree_example7() {

  TH2F *h_pxpy = new TH2F("h_pxpy", "py Vs px", 100, -2.0, 2.0, 100, -2.0, 2.0);
  TH1F *h_pt = new TH1F("h_pt", "pt", 100, 0., 5.0);

  TFile *f = new TFile("myfile.root");
  TTree *t1 = (TTree*)f->Get("T");

  std::vector<Float_t> *px = new std::vector<Float_t>();
  std::vector<Float_t> *py = new std::vector<Float_t>();
  std::vector<Float_t> *pt = new std::vector<Float_t>();


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

  Int_t nentries = (Int_t)t1->GetEntries();

  for (Int_t i = 0; i<nentries; i++) {
    t1->GetEntry(i);

    //Find the object with highest pt and fill its distributions
    Float_t hPt = 0;
    Int_t h_index = -1;
    if(pt->size() > 0){
      for(Int_t j = 0; j < pt->size(); j++){
        if((*pt)[j] > hPt){
          hPt = (*pt)[j];
          h_index = j;
        }
      }

      h_pxpy->Fill((*px)[h_index], (*py)[h_index]);
      h_pt->Fill((*pt)[h_index]);
    }

  }
```

Lorentz Vector

ROOT provides general four-vector classes that can be used either for the description of position and time (x,y,z,t) or momentum and energy
(px, py, pz, E)

TLorentzVector

- It provides a general four vector class that has various methods to initialize, set and get various quantitiesof a four vector(e.g., pt, eta, phi, mass etc..)
- Perform arithmetic operations, Lorents transformations
- Compute angular separation between four vectors
- More details:
  https://root.cern.ch/doc/master/classTLorentzVector.html

More advanced four-vector classes are provided by ROOT MathCore Library:https://root.cern.ch/root/html518/MATHCORE_Index.html

For example:
ROOT::Math::LorentzVector<ROOT::Math::PxPyPzE4D<double>>

Example for TLorentzVector

```
//Example using TLorentzVector
#include "TLorentzVector.h"
#include <vector>

void tree_example12() {

  TH2F *h_etaVsPhi = new TH2F("h_etaVsPhi", "eta vs phi", 100, -2.5, 2.5, 100, -3.14, 3.14);
  TH1F *h_mass = new TH1F("h_mass", "mass", 100, 40., 140.);

  TFile *f = new TFile("ntuple_array.root");
  TTree *t1 = (TTree*)f->Get("ntupleProducer/tree");

  Float_t muonPx[10], muonPy[10], muonPz[10];
  Int_t nMuon;

  t1->SetBranchAddress("nMuon", &nMuon);
  t1->SetBranchAddress("muonPx", muonPx);
  t1->SetBranchAddress("muonPy", muonPy);
  t1->SetBranchAddress("muonPz", muonPz);

  Int_t nentries = (Int_t)t1->GetEntries();

  for (Int_t i = 0; i<nentries; i++) {
    t1->GetEntry(i);

    std::vector<TLorentzVector> *muons = new std::vector<TLorentzVector>();
    muons->clear();
    //Compute muon eta, phi and fill them.
    if(nMuon>0){
      for(Int_t j = 0; j < nMuon; j++){
        float muonE = sqrt(muonPx[j]*muonPx[j] + muonPy[j]*muonPy[j] + muonPz[j]*muonPz[j]);
        TLorentzVector mu(muonPx[j], muonPy[j], muonPz[j], muonE);
        //apply pT cut
        if(mu.Pt() < 20) continue;

        h_etaVsPhi->Fill(mu.Eta(), mu.Phi());

        muons->push_back(mu);
      }
    }
```

Example for TLorentzVector

```
      //Fill mass if there are two muons
      if(muons->size() >= 2){
        TLorentzVector dimuon = (*muons)[0]+(*muons)[1];
        h_mass->Fill(dimuon.M());
      }

      delete muons;
    }

    TCanvas *c1 = new TCanvas();
    h_etaVsPhi->Draw("colz");
    c1->SaveAs("tree_example12_etaVsphi.png");

    c1->Update();
    h_mass->Draw();
    c1->SaveAs("tree_example12_dimuonMass.png");

    f->Close();
  }
```

Thank You for Your attention