Pracownia Specjalistyczna

Wprowadzenie do pakietu ROOT
Część 4

- drzewa
- algebra w ROOT – wektory

dr Katarzyna Rusiecka
Zakład Fizyki Hadronów IF UJ
katarzyna.rusiecka@uj.edu.pl

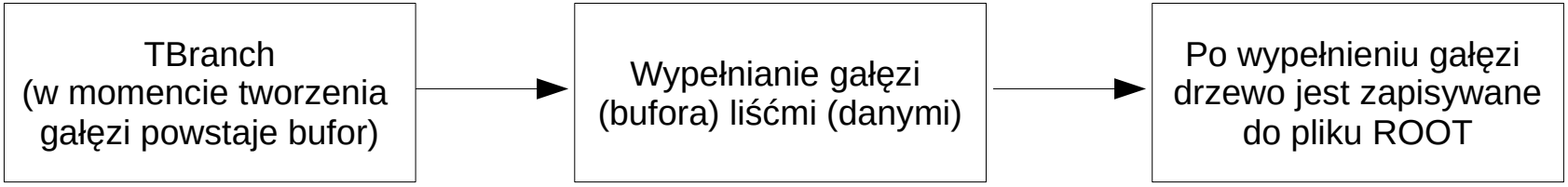
Drzewa – informacje ogólne

- Drzewa to obiekty zaprojektowane do przechowywyanie dużych ilości danych
- Oferują wydajną kompresję, łatwy oraz szybki dostęp do przechowywanych danych
- Obiekt typu TTree może zawierać wszystkie rodzaje danych – obiekty, kolekcje, proste typy
- Obiekt typu TNtuple może zawierać tylko liczby zmiennoprzecinkowe

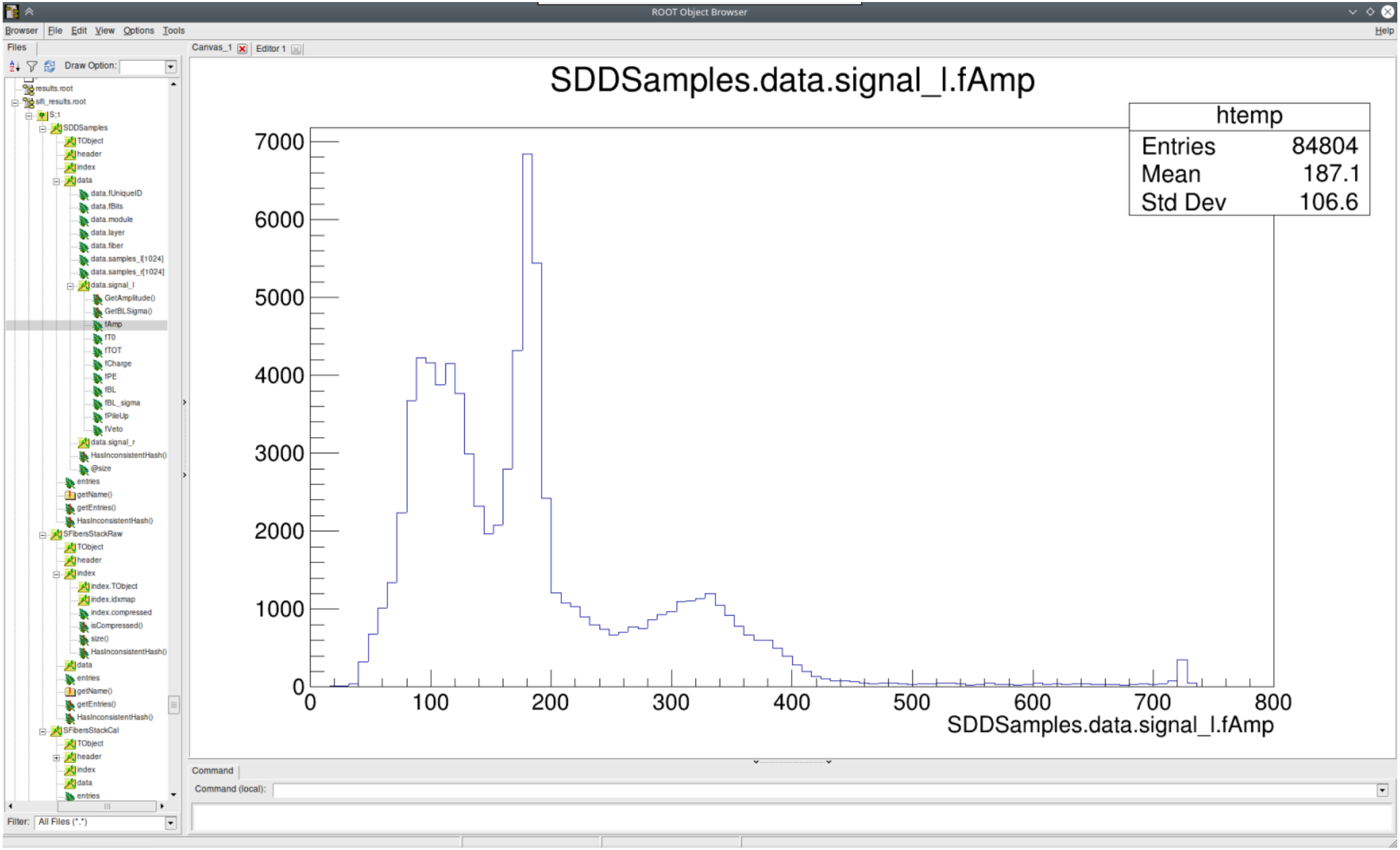
Architektura drzew:

- **TBranch** – czyli gałąź. Jeśli dwie zmienne są niezależne i nie będą używane razem powinny zostać zapisane w osobnych gałęziach. Jeśli zmienne są ze sobą powiązane, np. współrzędne punktu, powinny zostać zapisane w tej samej gałęzi.
- **TLeaf** – czyli liść. Dane zapisywane w gałęziach.

Schemat działania drzew:



Drzewa – informacje ogólne



Drzewa – informacje ogólne

Przykład

Zmienne Px anraz Py opisują pewne zdarzenie. Chcemy obliczyć $Px^2 + Py^2$ i wykreślić wynik na histogramie. Mamy 10^6 zdarzeń.

Plik tekstowy	**Drzewo**
1) wczytanie zdarzenia do pamięci	1) wczytanie gałęzi Px
2) obliczenie $Px^2 + Py^2$	2) wczytanie gałęzi Py
3) wypełnienie histogramu	3) wypełnienie histogramu sumą kwadratów
4) powtórzenie sekwencji milion razy	

Przykład: makro TreesExample.C (zakładka Pliki → Materiały z zajęć → ROOT_4)

TTree i TNtuple – konstruktory

TTree - konstruktor

```
TTree *tree = new TTree(“name”,”title”)
```

`name` – nazwa drzewa
`title` – tytuł drzewa

Dodawanie gałęzi do drzewa

```
tree->Branch(“name”,&address,”leaflist”,buffsize)
```

`address` – adres zmiennej zapisywanej do gałęzi
`leaflist` – lista liści, tzn. lista nazw zmiennych zapisywanych do danej gałęzi i ich typów. Składnia jest następująca: `variable_name/type:variable_name/type`... Jeśli `&address` wskazuje na prostą zmienną lista liści nie jest potrzebna
`buffsize` – rozmiar bufora w bajtach. Domyślnie wynosi 32000.

TNtuple - konstruktor

```
TNtuple *ntuple = new TNtuple(”name”,”title”,”varlist”,buffsize)
```

`varlist` – lista zmiannych (zmienne powinny być rozdzielone dwukropkami). Dla każdej zmiennej zostanie utworzona osobna gałąź.

Drzewa – rysowanie i cięcia

Rysowanie drzew

```
tree->Draw(”branch”,”cut”,”option”,entries,first)
```

`branch` – nazwa gałęzi
`cut` – cięcie
`option` – opcja rysowania, czyli np. “colz”
`entries` – liczba zdarzeń, jeśli nie jest określona domyślnie rysowane są wszystkie zdarzenia
`first` – pierwsze narysowane zdarzenie, jeśli nie jest określone domyślnie zdarzenia rysowane są od pierwszego

Drzewo można przejrzeć w TBrowser a także w TreeViewer:

```
tree->TreeViewer()
```

TCut - przykłady

TCut, czyli cięcie, to obiekt który precyzuje jaki zakres danych przechowywanych w drzewie chcemy narysować

```
tree->Draw(”x”,”x>0”,””);
TCut cut = ”x>3.3 && x<16.5”
tree->Draw(”x”,cut,””)
TCut cut = ”x>0 && y>1”;
tree->Draw(”x:y”,cut,”colz”)
```

Przykład tworzenia i zapisywania drzewa

```
...
 Int_t nev = 1E6;
 Double_t px, py;

 TFile *rootfile = new TFile("events.root","RECREATE");
 TTree *tree = new TTree("events","events");
 tree->Branch("px",&px);
 tree->Branch("py",&py);

 for(Int_t i=0; i<nev; i++){
     gRandom->Rannor(px,py);
     tree->Fill();
 }

    tree->Write();
    rootfile->Close();
...
```

TreesExample.c

Przykład odczytywania drzewa

TreesExample.c

Pobieranie drzewa z pliku

```
TFile *rootfile = new TFile("events.root","READ");
TTree *tree = (TTree*)rootfile->Get("events");
```

Odczytywanie i rysowanie danych – Metoda 1 (iteracyjna)

```
TH1F* h_tree = new TH1F("h_tree", "h_tree", 100, 0, 30);

Int_t ntree = tree->GetEntries();
Double_t px, py;
tree->SetBranchAddress("px",&px);
tree->SetBranchAddress("py",&py);

for(Int_t i=0; i<ntree; i++){
    tree->GetEntry(i);
    h_tree->Fill(px*px + py*py);
}

can->cd(1);
h_tree->Draw();
```

Przykład odczytywania drzewa

Odczytywanie i rysowanie danych – Metoda 2 (wprost z drzewa)

TreesExample.c

```
can->cd(1);

tree->Draw("px*px + py*py>>htmp(100,0,30)","","");
TH1F* h_tree = (TH1F*)gROOT->FindObjectAny("htmp");

h_tree->Draw();
```

Odczytywanie i rysowanie danych – Metoda 3 (wprost z drzewa, bez wskaźnika)

```
can->cd(1);
tree->Draw("px*px+py*py","","");
```

Wektory

Klasa TVector3 reprezentuje wektory w trzech wymiarach. **Uwaga – w najnowszych wersjach ROOTa TVector3 to już legacy code.**

Konstruktory

```
TVector3 v1;
TVector3 v2(1,2,3);
TVector3 v3(v2);
```

Ustalanie współrzędnych

```
v1.SetXYZ(1,2,3);
v1.SetX(1);
v1.SetY(2);
v1.SetZ(3);
```

Dostęp do współrzędnych

```
Double_t xx = v1.X();
Double_t yy = v1.Y();
Double_t zz = v1.Z();
```

Współrzęde nie-kartezjańskie

```
Double_t m = v1.Mag();
Double_t m2 = v1.Mag2();
Double_t t = v1.Theta();
Double_t ct = v1.CosTheta();
Double_t p = v1.Phi();
Double_t pp = v1.Perp();
Double_t pp2 = v1.Perp2();
```

```
v1.SetTheta(0.5);
v1.SetPhi(0.8);
v1.SetMag(10.0);
v1.SetPerp(3.0)
```

```
v3 = -v1;
v1 = v2+v3;
v1 += v3;
v1 = 5*v2;
if (v1==v2){ … }
```

```
Double_t a = v1.Dot(v2);
Double_t b = v1*v2;
TVector3 v = v1.Cross(v2);
```

```
Double_t ang = v1.Angle(v2);
```

Wektory i inne

Klasa `TVector3` reprezentuje wektory w trzech wymiarach. **Uwaga – w najnowszych wersjach ROOTa TVector3 to już legacy code.**

Rotacja wektora wokół osi

```
v.RotateX(0.5);
v.RotateY(TMath::Pi());
v.RotateZ(angle);
```

Rotacja wokół innego wektora

```
v1.Rotate(TMath::Pi()/4, v2);
```

Inne wektory

```
TVector3 u = v1.Unit();
TVector3 o = v1.Orthogonal();
```

- ROOT oferuje także klasę reprezentującą czterowektory (TLorentzVector), czyli wektory opisujące pozycję i czas (x, y, z, t) lub pęd i energię (px, py, pz, E). W nowszysch wersjach ROOTa - również legacy code.
- Tvector3 i TLorentzVector zostały zastąpione przez pakiet klas Physics Vectors: link
- ROOT oferuje także klasy reprezentujące macierze i umozliwiające operacje na nich, np.: TMatrixD, TMatrixDSym, TMatrixDSparse, itd.
- Klasy związane z macierzami nie są jednak tak dokładnie udokumentowane w Reference Guide na stronie internetowej ROOTa. Aby uzyskać o nich więcej informacji należy zajrzeć do ROOT Users Guide (wersja dla zaawansowanych), rodział 14: link

Zadanie

Napisz makro składające się z dwóch części (funkcji):

1) funkcja generująca zdarzenia i zapisująca je do drzewa
- argumentem funkcji ma być liczba generowanych zdarzeń
- generowane zdarzenia mają symulować emisję promieniowania o dwóch energiach z punktowego źródła w pełen kąt bryłowy
- parametry źródła: środek w punkcie (0,0,0), rozmycie gaussowskie w kierunkach X i Y, dwie emitowane energie – 1.1732 MeV (90%) oraz 1.3325 (10%)
- zdarzenia losowane jednorodnie w cos(θ) oraz jednorodnie w φ

2) funkcja odczytująca drzewo i rysująca odpowiednie histogramy
- histogram energii
- histogram dwuwymiarowy `start→X()` vs. `start→Y()`
- histogram dwuwymiarowy `versor→X()` vs. `versor→Y()`
- histogram dwuwymiarowy `versor→Theta()` vs. `versor→Phi()`