ROOT

Krzysztof Woźniak, IFJ PAN

ROOT - opis ogólny

Cele oprogramowania ROOT:
Stworzenie oprogramowania do analizy danych w fizyce wysokich energii:
- grafika przedstawiająca dane (histogramy, punkty, ...)
- struktury danych (ntuple, drzewa)
- procedury do analizy danych
- pliki skompresowane zawierające dane w formacie niezależnym od systemu operacyjnego

Cint - interpreter języka C++
- interaktywne wykonywanie kolejnych instrukcji
- kompilowanie, ładowanie bibliotek użytkownika

Oprogramowanie działa w różnych systemach: Linux, Windows, Mac OS X
jest dostępne jako kod źródłowy, który można samemu kompilować

http://root.cern.ch/drupal/content/downloading-root

ROOT - instalacja

Instalacja

http://root.cern.ch/drupal/content/downloading-root

aktualna wersja:

Production Version 5.34

dla Windows wybieramy:

VC++ 9 MSI (52.5 MB)

Standardowa instalacja tworzy skrót na pulpicie i element w Menu Start
Po uruchomieniu programu pojawia się okno typu terminal do komunikacji tekstowej z programem

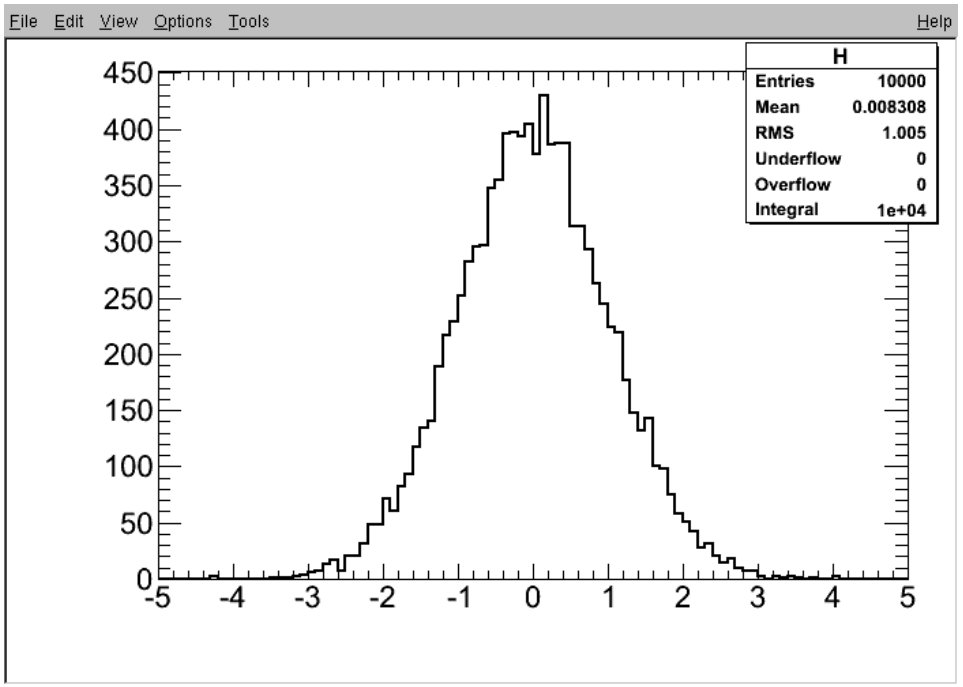
Program otwiera osobne okna graficzne do wyświetlania rysunków

ROOT - pierwszy test

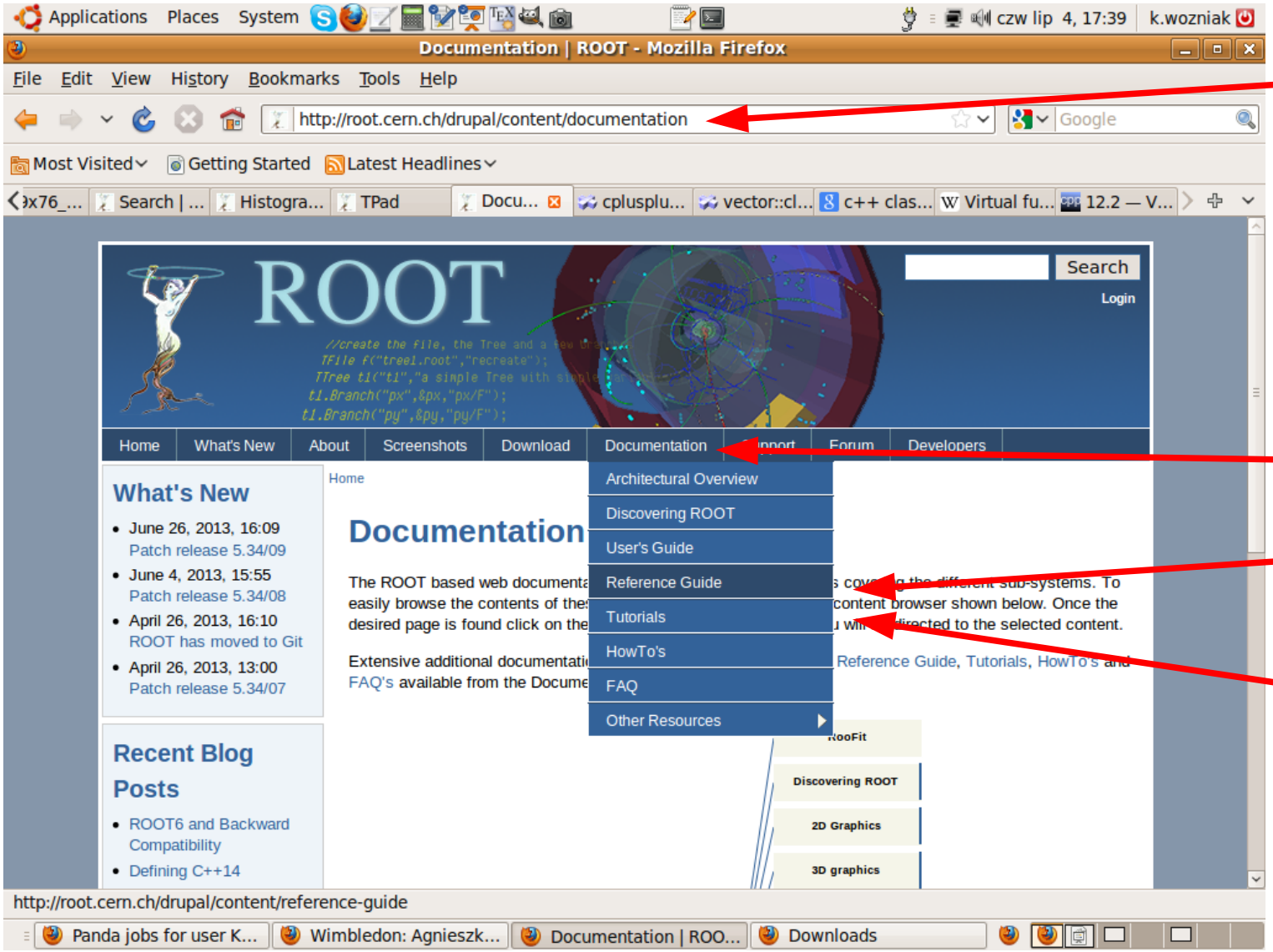
Po uruchomieniu i wpisaniu:

```
root [0] TH1F *h = new TH1F("H", "H", 100, -5, 5);
root [1] h->FillRandom("gaus", 10000);
root [2] h->Draw()
<TCanvas::MakeDefCanvas>: created default TCanvas with name c1
root [3]
```

wyświetlone jest okno



ROOT - dokumentacja



http://root.cern.ch

Documentation

Reference Guide
Opis klas

Tutorials
Przykłady

Podstawowe komendy

.h - wyświetlenie pomocy
.x plik.C - wykonanie makra zawartego w plik.C
.x plik2.C(10) - wykonanie makra zawartego w plik2.C z parametrem 10
musi być w nim zdefiniowana funkcja plik2(Int_t n)
.L plik2.C - załadowanie funkcji z plik2.C
plik2(12); i wywołanie jej z parametrem 12
.L plik2.C++ - załadowanie z kompilacją
.q - zakończenie pracy

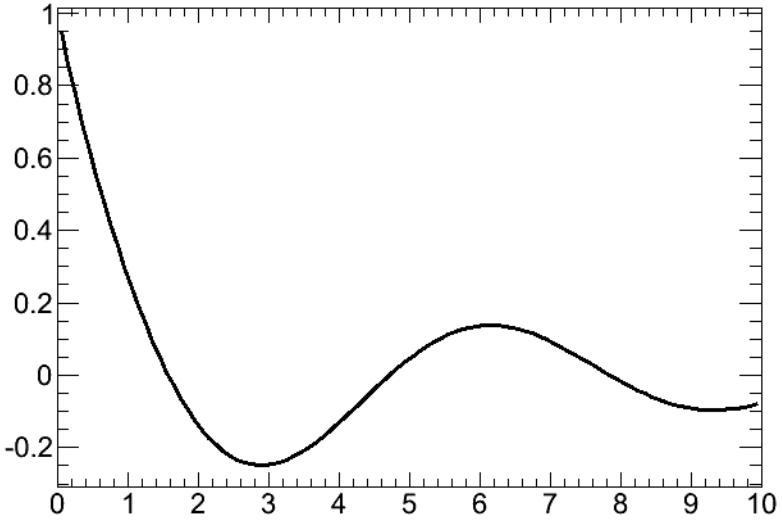
.! komenda - wykonanie komendy systemu operacyjnego

.p wyrażenie - wyświetlenie wartości wyrażenia (a zwykle jednej zmiennej)

fun(3); > f.out - przekierowanie wypisywanych wyników do pliku f.out
fun(5); 2> e.out - przekierowanie komunikatów o błędach do pliku e.out
fun(8); >& fe.out - przekierowanie do pliku fe.out wypisywanych wyników i komunikatów o błędach

Obliczenia natychmiastowe

```
root [0] 2+3*5-40/8                                  // operacje arytmetyczne
(const int)12                                        // wynik
root [1] sin(3.14158/6.0)                            // sin(30o)
(const double)4.9999817361051973 6e-01               // wartość (przybliżona)
root [2] TMath::Gaus(1.0)                            // funkcja Gaussa
(Double_t)6.0653065971263342 4e-01

root [3] TF1 *f = new TF1("f", "cos(x)/(1+x)", 0, 10);    // obiekt - funkcja
root [4] f->Draw();                                       // rysowanie
```



ROOT - opis ogólny

Typy niezależne od systemu operacyjnego

Int_t
Float_t
Char_t
...

Zmienne globalne

```
gROOT        - (TROOT type) przechowuje aktualne ustawienia
gROOT->SetStyle("Plain");

gStyle->SetOptStat(111111);   // opcje grafiki
gStyle->SetOptFit(111);
gStyle->SetPalette(1, 0);

gPad                          // wskaźnik do obszaru rysowania
gRandom->Rndm();              // standardowy zestaw funkcji losowych (TRandom)

gDirectory                    // wskaźnik do aktualnie wybranej kartoteki (TDirectory)
                              // przechowującej obiekty ROOT'a
```

TObject, TNamed

TObject - podstawowa klasa w ROOT, której własności dziedziczy wiele innych klas
TNamed - podstawowa klasa dziedzicząca z TObject, pozwalająca na nadanie objektowi nazwy

```
TObject *o = new TObject();
o->ClassName();          // zwraca nazwę klasy tego obiektu
o->Print();              // wypisanie informacji o obiekcie
o->ls();                 // wypisanie zwartości obiektu (w formie skróconej)
o->Dump();               // wypisanie pełnej zwartości obiektu

o->Write();              // zapisanie obiektu do aktualnej TDirectory
                         // zazwyczaj jest to zapisywanie do pliku

TNamed *a = new TNamed(”nazwa”, ”tytul”);
cout << nn->GetName() << " " << nn->GetTitle() << endl;
        // wypisuje:  nazwa tytul

TNamed *a = new TNamed();
a->SetName(”nazwa”);     // nadanie nazwy
a->SetTitle(”Tytul”);    // nadanie tytułu
```

obiekty TNamed zapisywane do pliku mogą być z niego potem wydobywane na podstawie ich nazwy

TCanvas, TPad

TCanvas - okno do rysowania
TPad - obszar rysowania

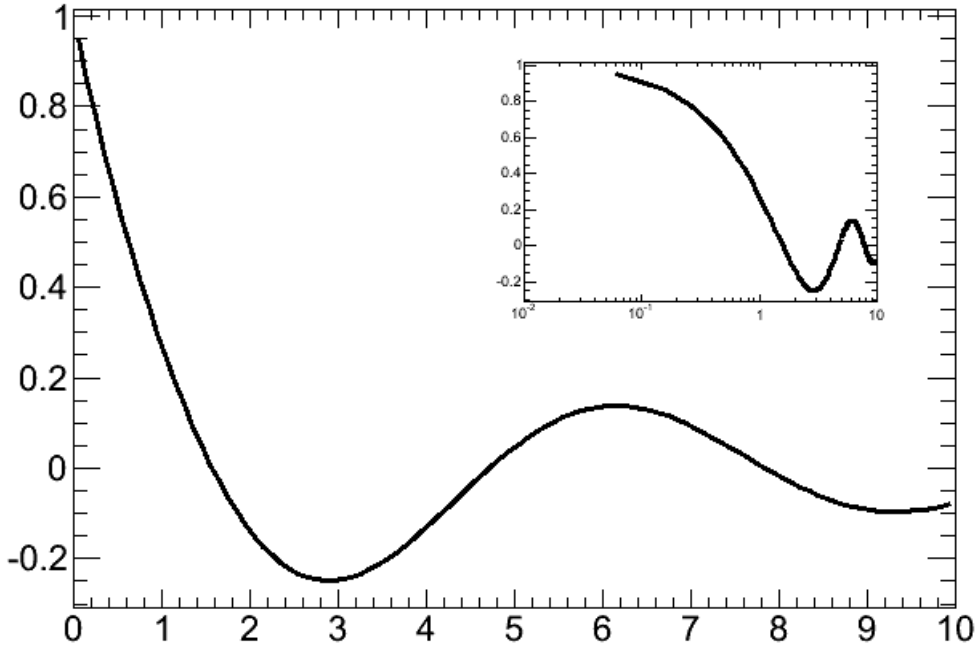
```
TCanvas *c = new TCanvas(”nazwa”,”tytul”, xsize, ysize);       // rozmiar w pikselach
c->cd();                          // ustawienie rysowania w tym oknie
```

w tym momencie obiekty graficzne mogą być rysowane przez wywołanie funkcji Draw(), choć faktyczna realizacja tego zadania dokonywana jest w funkcji Paint()

```
c->Divide(2,3);                   // podział okna na 6 części: dwie kolumny i 3 wiersze
c->cd(3);                         // przejście do części 3  (numeracja tych części od 1)
                                  // (numeracja tych części od 1, a 0 oznacza całe okno)

c->Update();                      // Informacja, że należy odświeżyć okno

c->SaveAs(”Plik.gif”);            // Zapisanie okna jako rysunku

gPad->SetLogx(kTRUE);   // zmiana skali w kierunku X na logarytmiczną

TPad *p = new TPad(”p”, ”p”, 0.5, 0.6, 0.9, 0.8);   // obszar od (0.5,0.6) do (0.9,0.8)
                                                    // całe okno: od (0.0, 0.0) do (1.0, 1.0)
p->Draw();                // dołączenie do TCanvas c
p->cd();                  // odtąd rysowanie w obszarze p
```

TCanvas, TPad

```
root [0] TF1 *f = new TF1("f", "cos(x)/(1+x)", 0, 10);
root [1] f->Draw();
<TCanvas::MakeDefCanvas>: created default TCanvas with name c1
root [2] TPad *p = new TPad("p", "p", 0.5, 0.5, 0.9, 0.9);
root [3] p->Draw();
root [4] p->cd();
(class TVirtualPad*)0x98c3670
root [5] f->Draw();
root [6] p->SetLogx();
```



ROOT - histogramy

Podstawowy sposób prezentacji wyników analizy
pozwalają badać częstość występowania jakichś wartości

Definicja histogramu jednowymiarowego

```
TH1F *h = new TH1F(”nazwa”, ”tytul”, nbins, xmin, xmax);       // zakres (xmin, xmax)
                                                                // podzielony na nbins przedziałów
Float_t xlim[6] = {0.0, 1.0, 4.0, 10.0, 30.0, 200.0};          // granice przedziałów
TH1F *h = new TH1F(”nazwa”, ”tytul”, 5, xlim);                 // zakres (xlim[0], xlim[5])

TH1F *h = new TH1F(”nazwa”, ”tytul; opis_osi_X; opis_osi_Y”, nbins, xmin, xmax);
                                // zdefiniowanie oprócz tytułu opisu osi X i osi Y
```

Wypełnianie

```
h->Fill(xval);                  // dodaje jeden przypadek z wartością xval
h->Fill(xval, weight);          // dodaje przypadek z wartością xval z wagą  weight
```

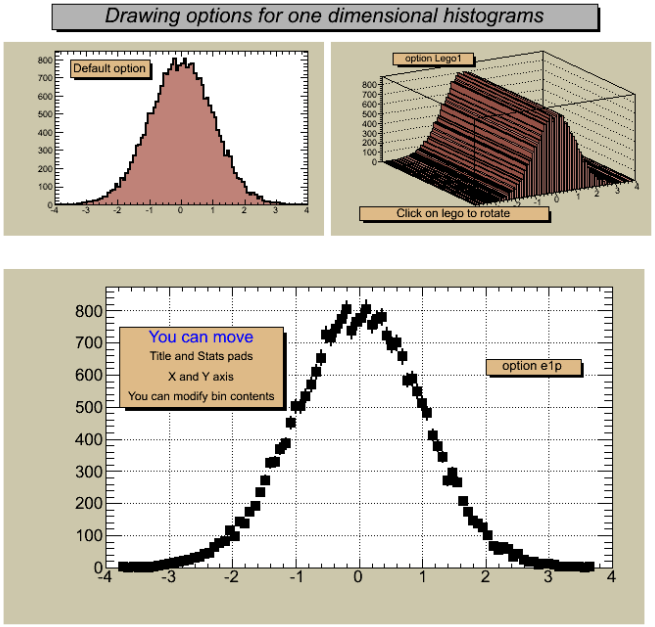
Wykorzystanie

```
h->Draw();                      // wyświetlenie histogramu
y = h->GetBinContent(ib);       // wydobycie zawartości dla przedziału (binu)  ib
                                // numeracja: 0  -  poniżej dolnej granicy (underflow)
                                //            1..nbin  -  biny wewnątrz zakresu
                                //            nbin+1  -  powyżej górnej granicy (overflow)
```

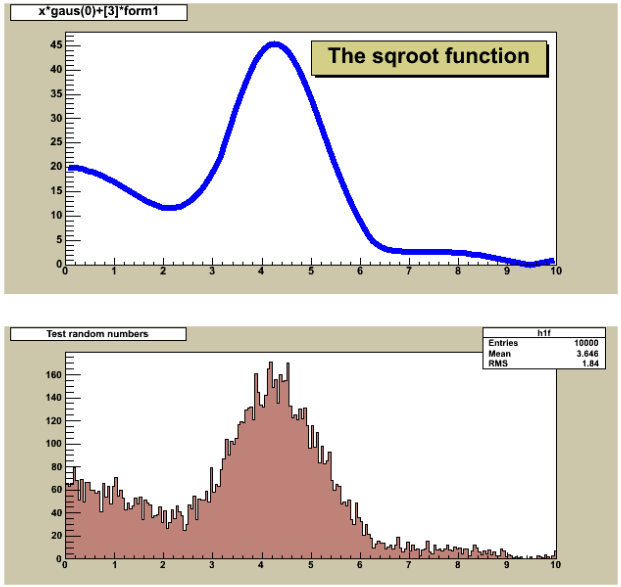
ROOT - histogramy

Przykłady rysowania (ROOT Tutorials: Histograms)

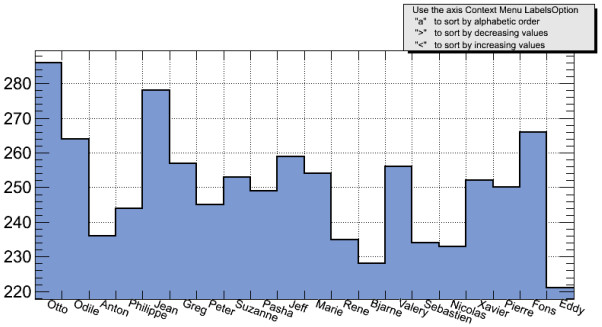
h1draw.C



fillrandom.C



hlabels1.C



Funkcje wydobywajace informacje z histogramów

```
n = h->GetEntries();                // liczba wejść (t.j. wywołań funkcji  Fill)
sum = h->Integral();                // suma zawartości binów (bez under- overflow)
sum = h->Integral(3, 5, „width”);   // całka: suma zawartości binów 3-5
                                    // mnożonych przez ich szerokość

ymax = h->GetMaximum();             // maksymalna wartość w jakimś binie
ibmax = h->GetMaximumBin();         // numer binu z maksymalną wartością
ymin = h->GetMinimum();             // minimalna wartość w jakimś binie
xmean = h->GetMean();               // wartość średnia x
xw = h->GetRMS();                   // dyspersja rozkładu x

nbins = h->GetNbinsX();                  // liczba binów w x
xunder = h->GetBinContent(0);            // zawartość histogramu poniżej xmin
xover = h->GetBinContent(h->GetNbinsX()+1);    // powyżej xmax
yerr = h->GetBinError(ib);               // błąd statystyczny w binie ib

xl = h->GetBinLowEdge(ib);               // dolna granica binu ib
xh = h->GetBinLowEdge(ib+1);             // górna granica binu ib
xc = h->GetBinCenter(ib);                // środek binu ib
xw = h->GetBinWidth(ib);                 // szerokość binu ib
```

Operacje na histogramach

```
h->Scale(a);                    // przemnożenie zawartości każdego binu przez a
h->Add(h2);                     // dodanie histogramu h2 (czyli  h = h+h2)
h->Add(h1, h2, c1, c2);         // zastąpienie zawartości h przez h1*c1+h2*c2
h->Divide(h2);                  // podzielenie zawartości h przez h2
h->KolmogorovTest(h2);          // statystyczna zgodność (prawdopodobieństwo)
h->Chi2Test(h2);                // test zgodności χ²
h->Reset();                     // wyzerowanie zawartości histogramu

h->Fit("1 ++ x*x", "I", "l", 2, 7);   // dopasowanie funkcji a+bx² do histogramu
    // wzór funkcji zawiera dopasowywane elementy (1, x*x) przedzielone przez  ++
    // mnożone przez  odpowiadające im zmienne parametry
    // mogą być wykorzystywane typowe funkcje: sin(x), abs(x+5), exp(x/(1+x*x))
    //  oraz wartości logiczne: (x>0)*sin(x) definiuje funkcję == 0 dla x<0
    // drugi parametr zawiera opcje dopasowywania (fitowania)
    // trzeci parametr podaje opcje rysowania
    // dwa ostatnie parametry definiują zakres fitowania

h->Fit("gaus", "", "", -3, 3);        // wykorzystanie funkcji predefiniowanej
                                      // inne:  expo, pol0, pol1, pol2, ...
h->Fit(fun, "", "",  0, 10);          // dopasowanie funkcji zdefiniowanej jako  fun
```

Histogramy wielowymiarowe

```
TH2F *h2d = new TH2F(”nazwa”, ”tytul”, nbinsX, xmin, xmax, nbinsY, ymin, ymax);
                                        // histogram dwuwymiarowy
TH3F *h3d = new TH3F(”nazwa”, ”tytul”, nbinsX, xmin, xmax, nbinsY, ymin, ymax,
nbinsZ, zmin, zmax);                    // histogram trójwymiarowy
```

Możliwe jest użycie podziału na biny o różnej szerokości, analogicznie jak dla TH1

```
h2d->Fill(x, y);
h2d->Fill(x, y, weight);
h3d->Fill(x, y, z);
h3d->Fill(x, y, z, weight);

val = h2d->GetBinContent(ibx, iby);
val = h3d->GetBinContent(ibx, iby, ibz);

ib = h2d->GetBin(ibx, iby);             // „globalny” numer binu
val2 = h2d->GetBinContent(ib);
ib = h3d->GetBin(ibx, iby, ibz);
val3 = h3d->GetBinContent(ib);
```

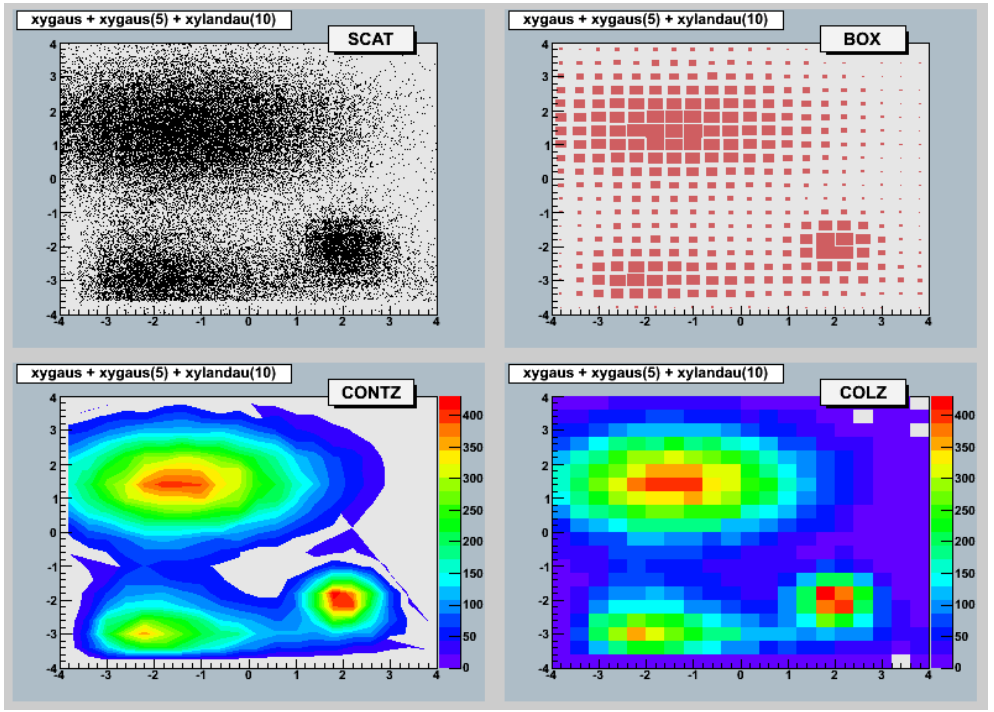
ROOT - histogramy

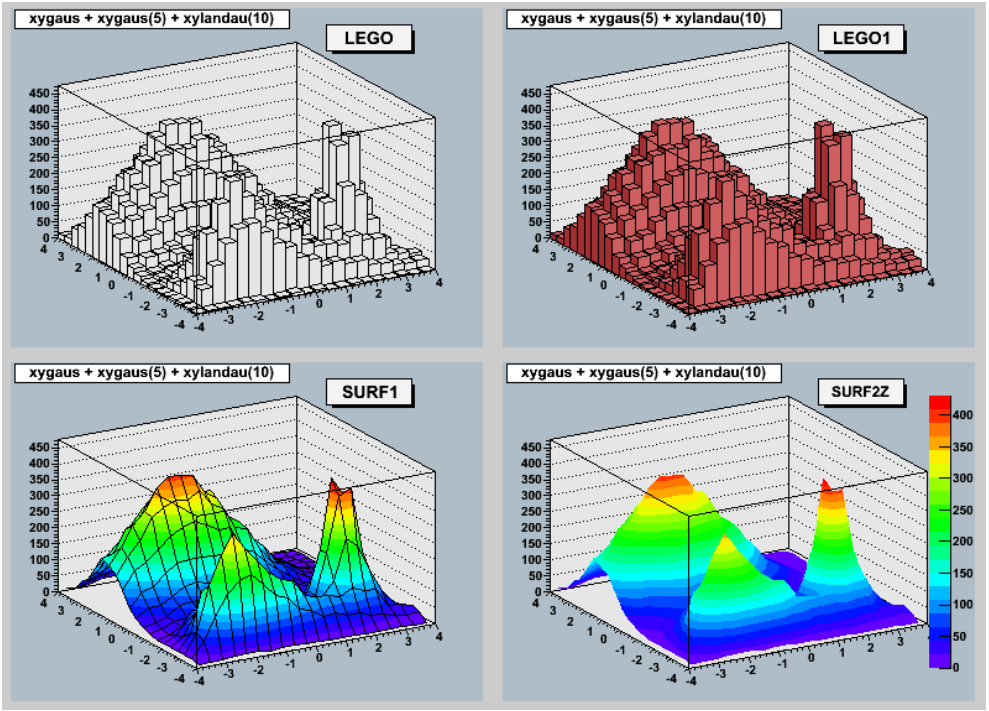
Histogramy dwuwymiarowe - rysowanie

ROOT Tutorials: draw2dopt.C (przedstawione tylko wybrane opcje)

h2d->Draw(); h2d->Draw(”box”); h2d->Draw(”lego”); h2d->Draw(”lego1”);

h2d->Draw(”contz”); h2d->Draw(”colz”); h2d->Draw(”surf”); h2d->Draw(”surf2z”);





ROOT - funkcje

Definicja funkcji za pomocą wzoru

```
TF1 *f1 = new TF1(”f1”, ”wzór”, xmin, xmax);       // f1 - nazwa funkcji
```
wzór jest wyrażeniem, które może zawierać parametry w postaci [0], [1], ...

Definicja funkcji za pomocą kodu

```
TF1 *f = new TF1(”f2”, xmin, xmax, npar);          // f2 - nazwa funkcji znanej Cint
// funkcja f2 musi być zdefiniowana jako:
Double_t f2(Double_t *x, Double_t *params)

TF1 *f = new TF1(”fun2”, f2, xmin, xmax, npar;     // analogicznie, ale ze zmiana nazwy
```

Parametry

```
f->SetParName(ip, ”nazwa_par”);          // nadanie parametrowi ip nazwy
f->SetParameter(ip, val);                // nadanie parametrowi wartości (początkowej)
f->SetParameter(”nazwa_par”, val);

yf = f->GetParameter(ip);                // wartość parametru (np. po fitowaniu)
```

ROOT - zestawy punktów TGraph

Definicja

```
TGraph *gr0 = new TGraph();          // bez określenia liczby punktów
TGraph *gr1 = new TGraph(5);         // ustalona liczba punktów, ale nie ich położenie

Float_t xtab[5] = {0,    1,    2,    3,    4 };
Float_t ytab[5] = {3,  2.5,  2.2,  2.1,  2.4};
TGraph *gr = new TGraph(5, xtab, ytab);      // pełna definicja
```

Punkty

```
Float_t x, y;
gr->GetPoint(2, &x, &y);                 // pobranie współrzędnych punktu 2 (czyli trzeciego)
gr->SetPoint(3, 7, 4.1);                 // nadanie współrzędnych punktowi 3
      // gdy dotąd było mniej punktów niż 3, dodawane są brakujące jako (0, 0)
```

Rysowanie

```
gr->Draw("alpc");                  // a  - narysowanie osi
                                   //  l  - rysowanie łamanej po punktach
                                   // p  - rysowanie symboli w punktach
                                   // c  - rysowanie linii gładkiej przechodzącej przez punkty

gr->Fit(...);                      // dopasowywanie funkcji analogiczne jak dla histogramu
```

ROOT - zestawy punktów TGraphErrors

Definicja - punkty z błędami w x i y - symetrycznymi

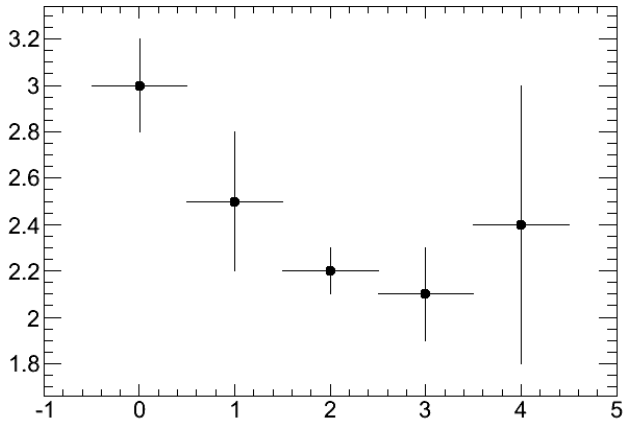
```
Float_t xtab[5] =  {0,      1,      2,      3,      4 };
Float_t extab[5] ={0.5,     0.5,    0.5,    0.5,    0.5};
Float_t ytab[5] =  {3,      2.5,    2.2,    2.1,    2.4};
Float_t eytab[5] = {0.2,    0.3,    0.1,    0.2,    0.6};
TGraphErrors *gr = new TGraphErrors(5, xtab, ytab); // wersja z zerowymi błędami
TGraphErrors *gr = new TGraphErrors(5, xtab, ytab,extab,  eytab); // pełna definicja
```

Punkty

```
ex = gr->GetErrorX(2);              // pobranie błędu X dla punktu 2 (czyli trzeciego)
ey = gr->GetErrorY(4);              // pobranie błędu Y dla punktu 4 (czyli piątego)
gr->SetPointError(3, 0.2, 0.7);     // ustawienie błędów punktu 3
```

Rysowanie

```
gr->Draw(”ap”);
```



ROOT - zestawy punktów TGraphAsymmErrors

Definicja - punkty z błędami w x i y - asymetrycznymi (różnymi w górę i w dół)

Typowe zastosowanie - liczenie wydajności selekcji:

- mamy rozkład pewnej wielkości dla wszystkich przypadków (histogram hall)
- obserwujemy jednak tylko część przypadków (histogram hrec)
- to, ile ich rejestrujemy, zmienia się wraz z tą wielkością
- prawdopodobieństwo rejestracji to stosunek hrec/hall
- błąd statystyczny liczby zliczeń w histogramie hrec to: sqrt(hrec->GetBinContent(ib))
- tego błędu nie można jednak używać bez zastanowienia przy określaniu błędu stosunku hrec/hall - bo mógłby pokazywać szansę na przekroczenie przez prawdopodobieństwo wartości maksymalnej możliwej: 1

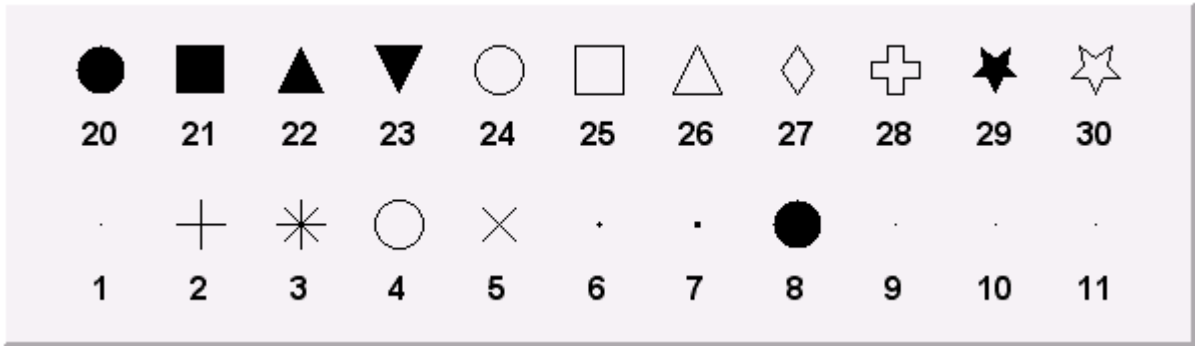
```
TGraphAsymmErrors *gr = new TGraphAsymmErrors();
gr->BayesDivide(hrec, hall);
```

TLine - odcinek

```
TLine *l1 = new TLine();
l1->DrawLine(x1, y1, x2, y2);                  // współrzędne użytkownika
TLine *l2 = new TLine(x1, y1, x2, y2);
l2->Draw();
l2->DrawLineNDC(xncd1, yndc1, xndc2, yndc2); // współrzędne (0,0) do (1,1)

l1->SetLineColor(kRed);          // kolor linii: numer lub nazwa
l1->SetLineWidth(5);             // grubość linii (1- słabo widoczna)
l1->SetLineStyle(2);             // styl linii: 1 - ciagła, 2 - przerywana
                                 // 3 - kropkowana i bardziej skomplikowane
```

TMarker - punkt

```
TMarker *p1 = new TMarker();
p1->DrawMarker(x, y);                // współrzędne użytkownika
TMarker *p2 = new TMarker(x, y);
p2->Draw();
p2->SetNDC(kTRUE);                   // (nie ma tej funkcji dla TLine)
p2->DrawMarker(xncd, yndc);          // współrzędne (0,0) do (1,1)

l1->SetMarkerColor(kRed);            // kolor punktu: numer lub nazwa
l1->SetMarkerSize(2.5);              // rozmiar punktu (2.0 - zwykle optymalny)
l1->SetMarkerStyle(20);              // styl punktu: kółko, kwadrat, trójkąt ...
```



TText - tekst

```
TText *t1 = new TText();
t1->DrawText(x, y, ”tekst 1”);                  // współrzędne użytkownika
TText *t2 = new TText(x, y, ”tekst 2”);
t2->Draw();
t2->DrawTextNDC(xncd, yndc, ”tekst 2 - nowy”);        // współrzędne (0,0) do (1,1)
t2->SetNCD();                          // współrzędne NDC dla przyszłych operacji
                                       // Draw() i DrawText(), ale i dla już wcześniej
                                       // narysowanego przez Draw() tekstu

t1->SetTextColor(kRed);                // kolor tekstu: numer lub nazwa
t1->SetTextSize(0.1);                  // rozmiar tekstu (0.05 - zwykle nieco za mały)
t1->SetAlign(align);                   // sposób wyrównywania tekstu :
                                       // align = 10*HorizontalAlign + VerticalAlign
                                       // horizontal: 1 - do lewej, 2 - do środka, 3 - do prawej
.                                      // vertical: 1 - do dołu, 2 - do środka, 3- do góry

t1->SetTextAngle(30);                  // pochylenie tekstu w stopniach (0 - normalne)
t1->SetTextFont(code);                 // rodzaj czcionki: code = 10*fontnumber + precision
                                       // precision: 0 - niedokładne 1-3 - skalowalne
                                       // standard: 42 - helvetica-medium-r-normal  albo Arial
```

TLatex - tekst z możliwościami formatowania

```
TLatex *tl1 = new TLatex();
tl1->DrawLatex(x, y, ”X_{1}”);                  // indeks dolny:  X_1
tl1->DrawLatex(x, y, ”ax^{2}+bx+c”);            // indeks górny:  ax^2+bx+c
tl1->DrawLatex(x, y, ”{}^{40}_{20}Ca”);         // indeksy przed tekstem właściwym
tl1->DrawLatex(x, y, ”x = #frac{y+z/2}{y^{2}+1}”);                  // ułamki: x = (y+z/2)/(y^2+1)

tl1->DrawLatex(x, y, ”#alpha #leq #Phi”);       // litery greckie i symbole:  α ≤ Φ
tl1->DrawLatex(x, y, ”#ver{v}”);                // wektor
tl1->DrawLatex(x, y, ”#bar{a}”);                // kreska nad literą
tl1->DrawLatex(x, y, ”#sqrt{10}”);              // pierwiastek
tl1->DrawLatex(x, y, ”#sqrt[3]{10}”);           // pierwiastek wyższego rzędu
tl1->DrawLatex(x, y, ”#sum_{i=1}^{5}”);         // sumowanie od  i=1  do  5
tl1->DrawLatex(x, y, ”#int_{-1}^{1}cos^{2}(x)dx”);      // całka oznaczona od  -1  do  1

tl1->DrawLatex(x, y, ”#splitline{linia 1}{linia 2}”);   // wypisanie tekstu w 2 liniach
tl1->DrawLatex(x, y, ”#color[2]{Red} and #color[4]{Blue}”);      //  Red and Blue
tl1->DrawLatex(x, y, ”#bf{pogrubienie} #it{kursywa} oraz #bf{#it{pogrubiona kursywa}}”);
        //  pogrubienie kursywa oraz pogrubiona kursywa
tl1->DrawLatex(x, y, ”#scale[1.5]{Big} and #scale{0.7]{small}”);      //  Big and small
```

ROOT - pliki

TFile - plik zawierające obiekty ROOT'a

```
TFile *tf = new TFile(”plik.root”, ”READ”);          // otwarcie pliku do odczytu
TFile *tf = new TFile(”plik.root”);                  // tak jak z READ

TFile *tf = new TFile(”plik.root”, ”NEW”);           // utworzenie nowego pliku
                                                     // błąd, gdy już istnieje
TFile *tf = new TFile(”plik.root”, ”RECREATE”);      // utworzenie nowego pliku
                                                     // z usunięciem starego, gdy już istniał

TObject *obj = tf->Get(”objectname”);    // odczytanie obiektu o nazwie ”objectname”
TH1F *h = (TH1F *) tf->Get(”hist”);      // odczytanie histogramu o nazwie  hist
                                         // (TH1F *) przekształca TObject na TH1F
                                         // błąd gdy ”hist” jest innego typu
tf->Close();                             // zamknięcie pliku
```

Przeglądanie zawartości pliku ROOT'a

```
TFile *tf = new TFile(”plik.root”);
TBrowser *br = new TBrowser();     // otwiera okno, w którym można przeglądać strukturę
                                   // pliku, wyświetlać histogramy, sprawdzać zawartość
                                   // dowolnych zapisanych w pliku obiektów
```

Zapis do pliku ROOT'a

```
TFile *tf1 = new TFile(”plik1.root”, ”RECREATE”);
TFile *tf2 = new TFile(”plik2.root”, ”RECREATE”);       // utworzenie 2 plików

mamy np histogramy: ha, hb, hc, hd, he
tf1->cd();
ha->Write();
hb->Write();
tf2->cd();
hc->Write();
hd->Write();
tf1->cd();
he->Write()
tf1->Close();
tf2->Close();

W  plik1.root  zapisane zostały: ha, hb, he
W  plik2.root  zapisane zostały  hc, hd
```

Organizacja danych

Najwygodniejszą do analizy formą organizacji danych są tablice indeksowane numerem przypadku, zawierające informacje o takiej samej strukturze. Dla efektywnej analizy danych bardzo istotne jest, by można było używać danych wybranego typu, bez wczytywania i analizowania tych, które w danym momencie są nieistotne.
ROOT pozwala przechowywać dane w drzewach **TTree** zawierajacych obiekty o prawie dowolnej postaci, pogrupowane w niezależnych gałęziach (**TBranch**).
W wersji uproszczonej (klasa **TNtuple**) informacjami tymi są wyłącznie liczby float,

Przykład

- numer przypadku // jedna liczba
- liczba cząstek // jedna liczba (zawiera wartość N)
- px cząstek // N liczb
- py cząstek // N liczb
- pz cząstek // N liczb
- energia cząstek // N liczb
- sygnał w det A // jedna liczba
- sygnały w det B // dwie liczby

Dla dowolnej liczby cząstek N konieczne jest TTree, jeśli maksymalne N jest małe (np. 4) można skorzystać z TNtuple

TNtuple

```
TNtuple *nt = new TNtuple(”nazwa”, ”tytul”,
                                        ”nr:np:px1:py1:pz1:e1:px2:py2:pz2:e2:da:db1:db2”);
```

Wprowadzanie informacji do nt:

```
nt->Fill(nr, 2, px[0], py[0], pz[0], e[0], px[1], py[1], pz[1], e[1], sda, sdb[0], sdb[1]);
```

podajemy wartości wszystkich elementów (ale maksymalnie 15!)
alternatywnie:

```
Float_t vartab[20];
var[0] = nr;
var[1] = 2;
var[2] = px[0];
var[3] = py[0];
var[4] = pz[0];
var[5] = e[0];
  ...
nt->Fill(vartab);
```

TTree - struktura z gałęziami o różnych możliwych formatach

```
TTree *tr = new TTree( ”parttree”, ”tytul”);
Int_t nr, np;
std::vector<float> px;
std::vector<float> py;
std::vector<float> pz;
std::vector<float> e;
Float_t sda, sdb[2];
tr->SetBranch(”nr”, &nr, ”nr/I”);        // gałąź z numerem przypadku (liczba całkowita)
tr->SetBranch(”npart”, &np, ”npart/I”);     // gałąź z liczbą cząstek (liczba całkowita)
tr->SetBranch(”px”, &px);                   // gałąź z pędami (px) N cząstek
tr->SetBranch(”py”, &py);
tr->SetBranch(”pz”, &pz);
tr->SetBranch(”e”, &e);
tr->SetBranch(”detA”, &sda, "signalA/F”);                  // gałąź z sygnałem z detektora A
tr->SetBranch(”detB”, &sdb, ”signalB1/F:signalB2/F); // gałąź z sygnałami z detektora B
```

Wprowadzanie informacji do tr:

- wypełniamy wszystkie zmienne
- tr->Fill();

TTree - wypełnianie

```
nr++;                       // kolejny numer przypadku
np = nfound;                // znaleziona w odpowiedni sposób liczba cząstek
px.clear();  py.clear();  pz.clear();  e.clear();   // wstępne wyzerowanie wektorów
for(int n=0; n<np; n++)  {
      px.pushback(momentum[n][0]);      // pęd przechowywany w dwuwymiarowej tablicy
      py.pushback(momentum[n][1]);      // momentum[100][3]
      pz.pushback(momentum[n][2]);
      e.pushback(energy[n]);
      sda = signal[0];                  // sygnały z detektora A i B w tablicy signal
      sdb[0] = signal[1];               // odpowiednio w signal[0] oraz signal[1] i signal[2]
      sdb[1] = signal[2];
}
tr->Fill();
```

Zapisanie drzewa do pliku

```
TFile *tf = new TFile("plikzdrzewem.root", "RECREATE");
tf->cd();
tr->Write();
```

TTree - odczyt

```
TFile *tf = new TFile(”plikzdrzewem.root”);          // tylko do odczytu
TTree *tr = (TTree *) tf->Get(”parttree”);           // wydobycie wskaźnika do drzewa
                                                     // (w całości znajdującego się w pliku)
std::vector<float> e;
TBranch *be = 0;              // deklaracja gałęzi z energią
t->SetBranchAddress("e", &e, &be);
Float_t sdb[2];
TBranch *bsdb = 0;                // deklaracja gałęzi z sygnałem z detektora B
t->SetBranchAddress("detB", &sdb, &bsdb);

TH1F *hdiff = new TH1F(”hdiff”, ”hdiff”, 100, -10.0, 10.0);
ne = 0;
while(1)  {           // pętla nieskończona (wymaga przerwania wewnątrz)
      int s = tr->GetEntry(ne++);       // pobranie przypadku ne, a potem zwiększenie ne
      if(s < 1)  break;                 // brak kolejnego przypadku w drzewie - koniec pętli
      float esum = 0.0;
      for(int n=0; n<e.size(); n++)  {
            esum += e.at(n);            // suma energii
      }
      hdiff->Fill(esum-sdb[1]);         // tworzymy histogram różnicy
}                                       // między sygnałem a sumą energii
```

TTree - funkcje

```
TTree *tr = (TTree *) tf->Get(”parttree”);
TBranch *br1 = tr->Branch(”punkty”, &punkt, ”x/F:y/F:z/F”);      // dodawanie gałęzi
TBranch *br2 = tr->Branch(”vertex”, &vertex);

TBranch *b = tr->GetBranch(”vertex”);   // wydobycie wskaźnika do gałęzi
b->SetBranchStatus(0);                  // gałąź nie ma być wczytywana
 alternatywnie
tr->SetBranchStatus(”vertex”, 0);       // dezaktywowanie gałęzi
 albo:
tr->SetBranchStatus(”*”, 0);            // dezaktywowanie wszystkich
tr->SetBranchStatus(”vertex”, 1);       // aktywowanie wybranych

nev = tr->GetEntries();                 // liczba elementów w drzewie
nbytes = tr->GetEntry(jentry, getall);  // wczytanie informacji o elemencie  jentry
                                        // dla getall==0 - tylko z aktywnych gałęzi
                                        // dla getall==1 - ze wszystkich gałęzi
                                        // nbytes - ilość informacji (powinna być > 0)

tr->MakeClass();                        // utworzenie plików parttree.h i parttree.C
                                        // z kodem ułatwiającym odczyt i analizę drzewa
```

TTree - prezentacja graficzna informacji

```
tr->Draw(”px”);                          // rysuje histogram px (wszystkie przypadki)
tr->Draw(”px*px”, ””, ””, 500);           // rysuje histogram px^2 dla 500 pierwszych przypadków

tr->Draw(”px/py”, ”abs(py)>0.001”); // rysuje histogram px/py, ale dla py nie za bliskich 0

TH1F *hpx = new TH1F("hpx", "hpx", 100, 0.0, 2.0);
tr->Draw(”px>>hpx");                 // wypełnia histogram hpx
podobnie działa:
tr->Draw(”px>>hpx(100,0.0, 2.0)”);  // wypełnia histogram hpx o podanym zakresie i rysuje
                                     // standardowo wypełniany jest histogram htemp
                                     // granice wyznaczając automatycznie
TH1* h = (TH1F *) gDirectory->FindObject("htemp");  // wydobycie histogramu htemp


tr->Draw(”px:py”);                       // rysuje dwuwymiarowy histogram px i py
tr->Draw(”px:py”, ””, ”colz”);           // rysuje dwuwymiarowy histogram px i py
                                         // stosując styl colz
```