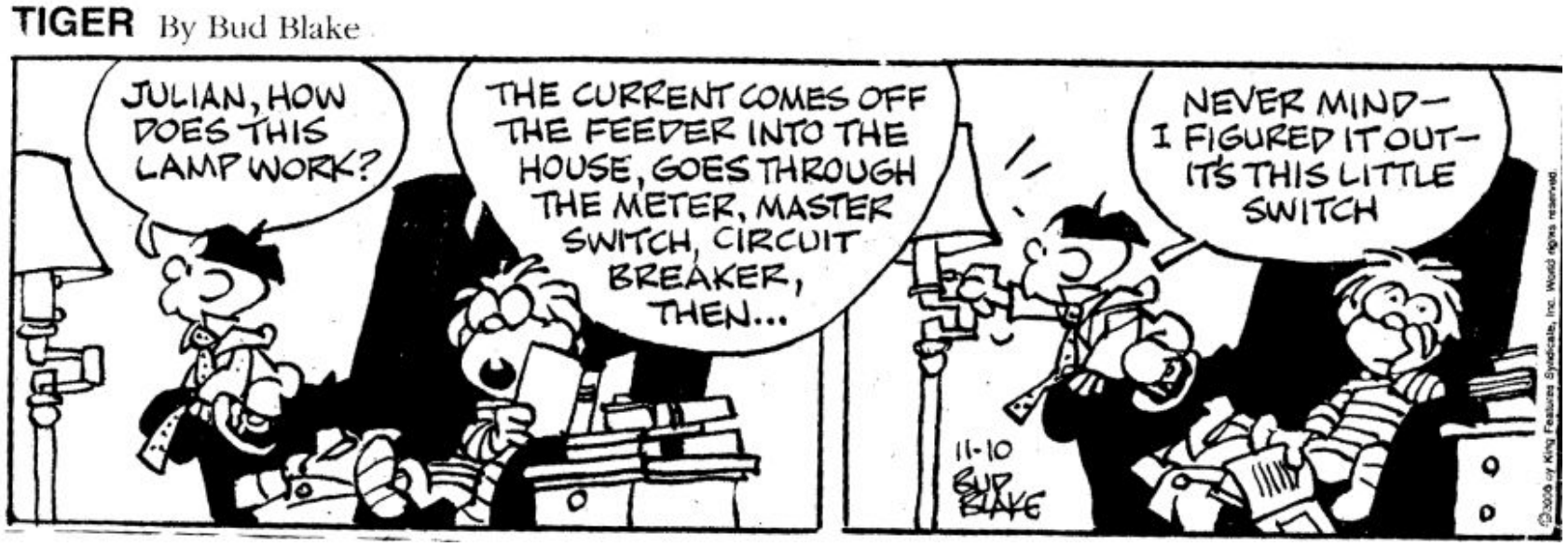
Логашенко И.Б.

Современные методы обработки экспериментальных данных

ROOT – программная оболочка для обработки данных

Что такое ROOT?



ROOT – объектно-ориентированная (C++) программная оболочка (библиотека), предоставляющая большое количество инструментов, необходимых для обработки данных.

Почему ROOT

- Бесплатная
- Открытый код
- Кросс-платформенная
- Может использоваться и как приложение, и как библиотека
- Огромное количество иструментов
  - гистограммы, функции, подгонка
  - сохранение и обработка больших объемов данных
  - научная графика
  - математическая библиотека
  - многопараметрический анализ данных (например, нейронные сети)
  - интеграция с Python, Ruby, Mathematica

Документация

- Веб-страница
  http://root.cern.ch
- Руководство пользователя
  http://root.cern.ch/drupal/content/users-guide
- Подробное описание классов
  http://root.cern.ch/drupal/content/reference-guide
- Примеры
  http://root.cern.ch/root/html/tutorials/
  http://root.cern.ch/drupal/content/howtos
- Описание пакета TMVA
  http://tmva.sourceforge.net/docu/TMVAUsersGuide.pdf

Запуск ROOT

- Настройка переменных окружения (bash)

```
export ROOTSYS=…
export PATH=${PATH}:${ROOTSYS}/bin
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${ROOTSYS}/lib
```

- Запуск ROOT из командной строки

```
Usage: root [-l] [-b] [-n] [-q] [file1.C ... fileN.C]
Options:
  -b : run in batch mode without graphics
  -n : do not execute logon and logoff macros as specified in .rootrc
  -q : exit after processing command line macro files
  -l : do not show splash screen
```

Пример: root -l

ROOT как интерпретатор C++

В качестве командного языка ROOT использует CINT – интерпретатор C++

```
root[0] TH1D *h = new TH1D("h1","Test histogram",100,0,10);
root[1] TF1 *f = new TF1("f1","x-x*x",0,1);
root[2] h->FillRandom("f1",1000);
root[3] h->Draw();
root[4] char *axis_title = "x, cm";
root[4] h->GetXaxis()->SetTitle(axis_title);
root[5] .q
```

Расширение C++ в CINT

Дополнительные возможности интерпретатора:

- <TAB> автоматически дополняет имя класса и показывает список методов

```
root[0] TBrow<TAB>
root[1] TH1D(<TAB>
```

- Декларация переменных не обязательна

```
root[0] h = new TH1D("h1","Test histogram",10,0,10)
```

- Возможность использования как `.` , так и `->`

```
root[0] h = new TH1D("h1","Test histogram",10,0,10)
root[1] h.Draw()
```

- Обращение к объекту по имени

```
root[0] new TH1D("h1","Test histogram",10,0,10)
root[1] h1->Draw()
```

Дополнительные команды CINT

В интерпретаторе определено несколько встроенных команд, начинающихся с `.`

`root[0].?`
список всех дополнительных команд CINT

`root[1].X [filename]`
загрузить [filename] и выполнить функцию [filename]

`root[2].L [filename]`
загрузить [filename]

`root[3] .! ls`
выполнить команду shell

`root[4] .files`
список загруженных файлов

`root[5] .class [name]`
определение класса [name]

`root[6] .g`
список всех глобальных объектов

`root[7] .ls`
список объектов в текущей директории

`root[8] .pwd`
имя текущей директории, графического окна и графического стиля

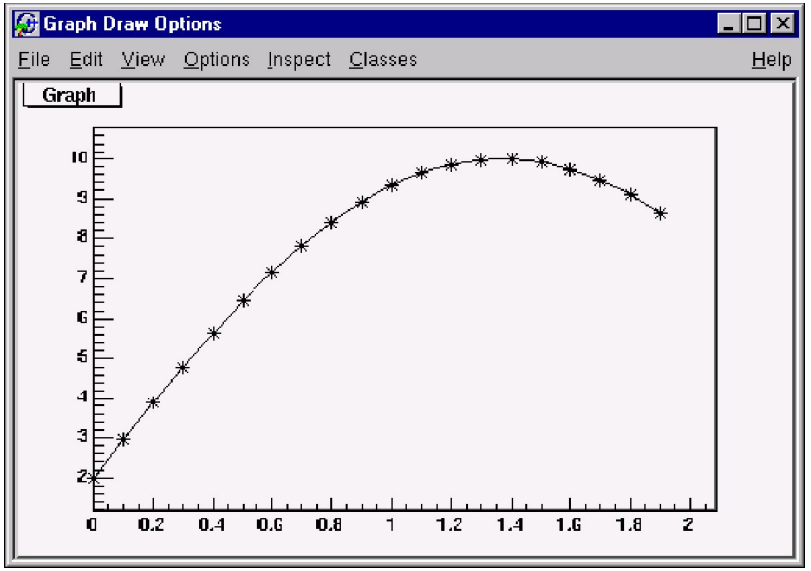
Глобальные переменные

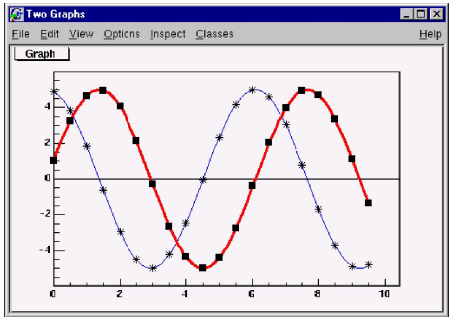
В CINT определено несколько глобальных переменных:

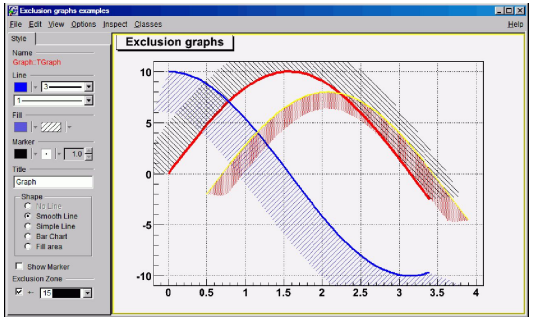
- gRandom - генератор случайных чисел
```
gRandom->Gaus(1,2)
gRandom->Rndm()
gRandom->Poisson(4)
```
- gFile – указатель на текущий рабочий файл
```
gFile->GetName()
```
- gDirectory – указатель на текущую рабочую директорию
```
gDirectory->GetName()
```
- gSystem – системная информация о текущей сессии ROOT
```
gSystem->HostName()
```
- gROOT – доступ к внутренней информации текущей сессии ROOT
```
gROOT->GetListOf<Files>()
gROOT->ProcessLine(".! ls")
gROOT->LoadMacro("myutil.cc")
TH1D *h1 = (TH1D*)gROOT->FindObject("h1")
```

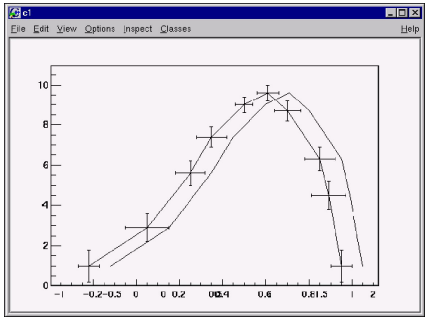
TGraph

```
Int_t n = 20;
Double_t x[n], y[n];
for (Int_t i=0; i<n; i++) {
   x[i] = i*0.1;
   y[i] = 10*sin(x[i]+0.2);
}

TGraph *gr1 = new TGraph (n, x,
y);

gr1->Draw("ALP");
```





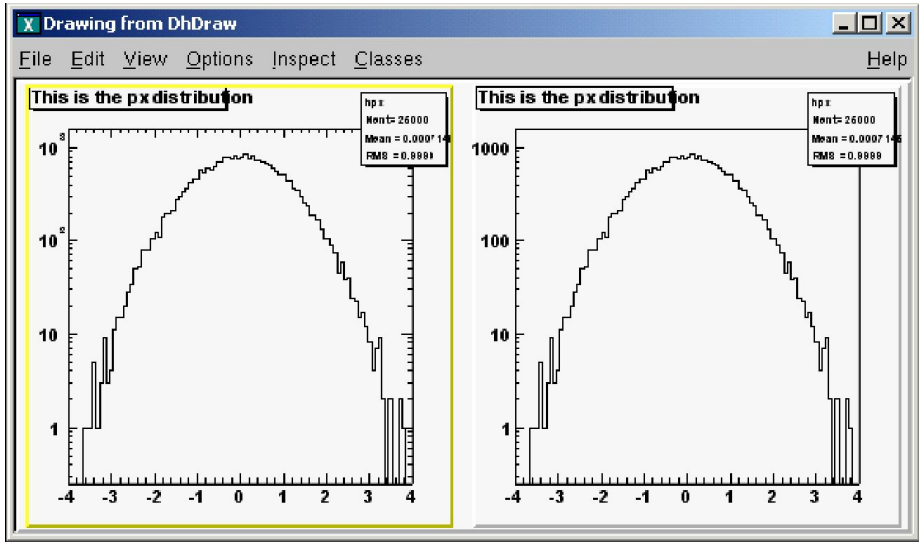




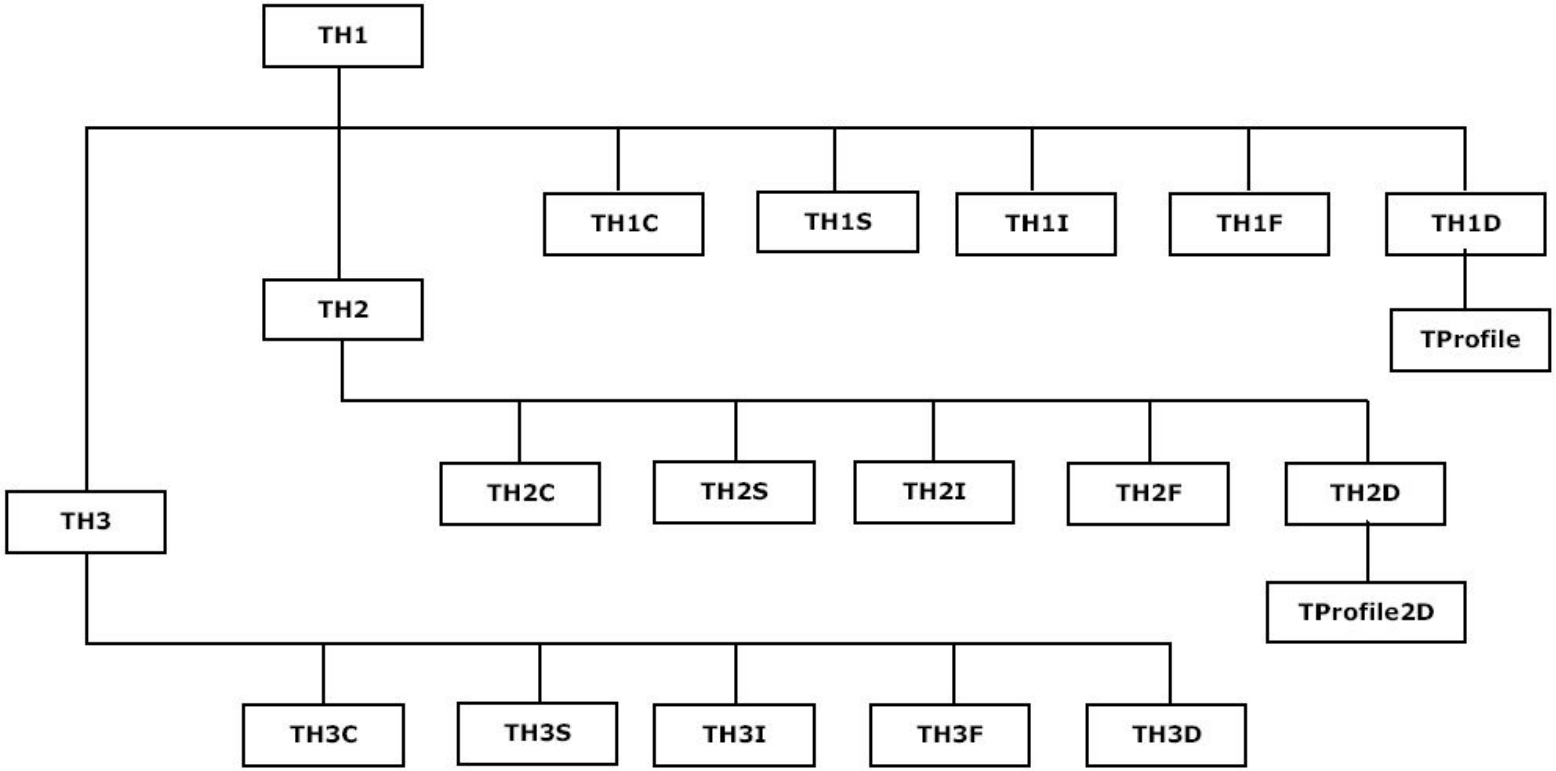
TCanvas и TPad

Для рисования графиков создается графическое окно – TCanvas. Собственно отрисовка производится в специальных областях – TPad. Внутри одного TCanvas может быть несколько TPad.

```
TCanvas *c1 = new TCanvas("c1","Example",10,10,800,400);
c1->Divide(2,1);
c1->cd(1);
h1->Draw();
c1->cd(2);
h2->Draw();
c1->Update();
```



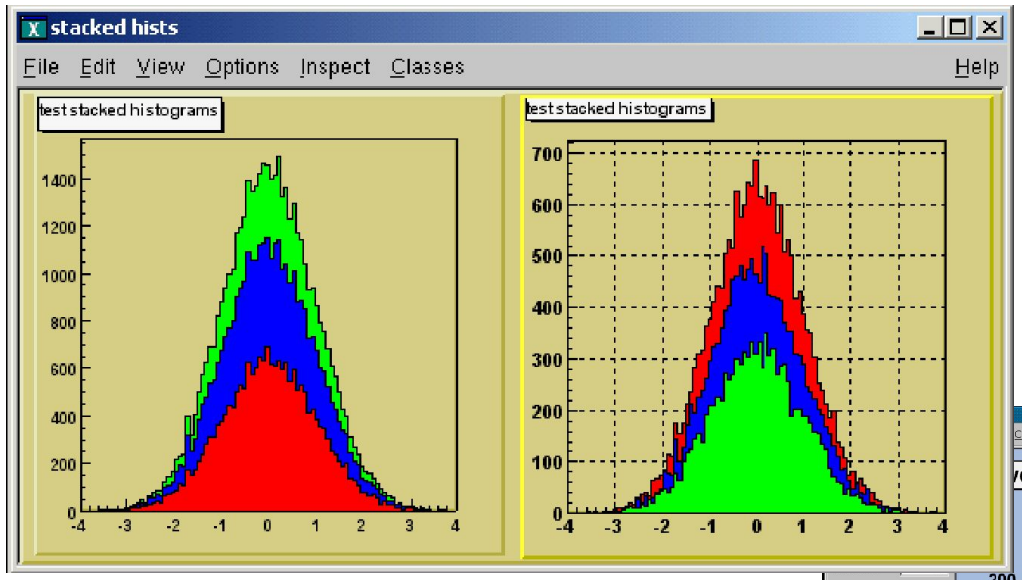
Гистограммы - классы

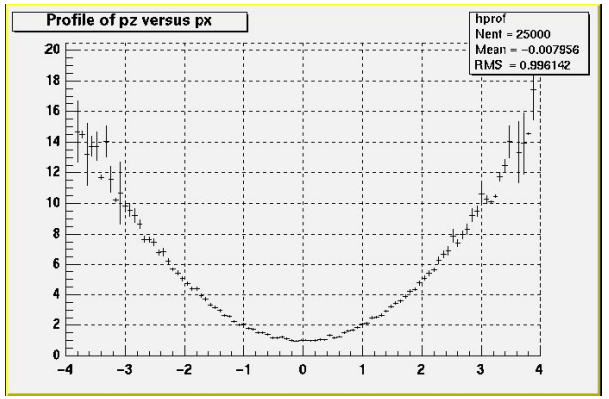


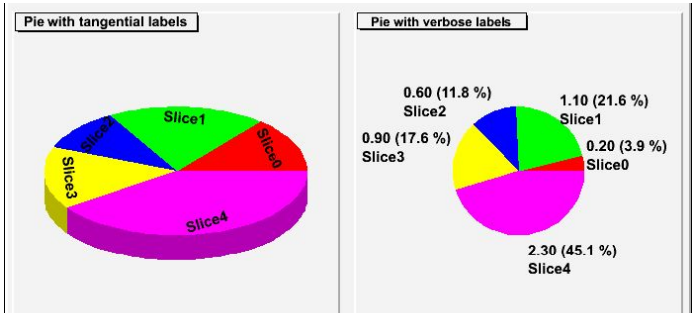
Гистограммы – пример C++

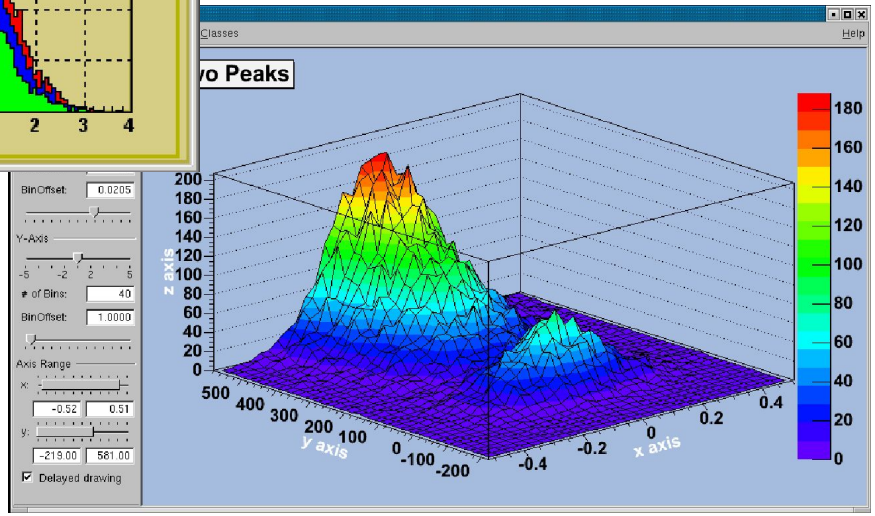
```
// Создание гистограмм
TH1D *h1 = new TH1D(“h1”,“Test 1-D histo”,40,0,1);
TH1D *h2 = new TH2D(“h2”,“Test 2-D histo”,40,0,1,40,0,1);
TH1D *hp = new TProfileD(“h1”,“Test profile
histo”,40,0,1,0,1);

// Заполнение гистограмм
for( int i=0; i<100; i++ ) {
   double x = gRandom->Rndm();
   h1->Fill(x*x);
   for( int j=0; j<100; j++ ) {
      double y = gRandom->Gaus(0.5,0.2);
      h2->Fill(x,y);
      hp->Fill(x,y);
   }
}

// Отрисовка гистограмм
h1->Draw()
h2->Draw(“lego”)
```

Гистограммы - графика





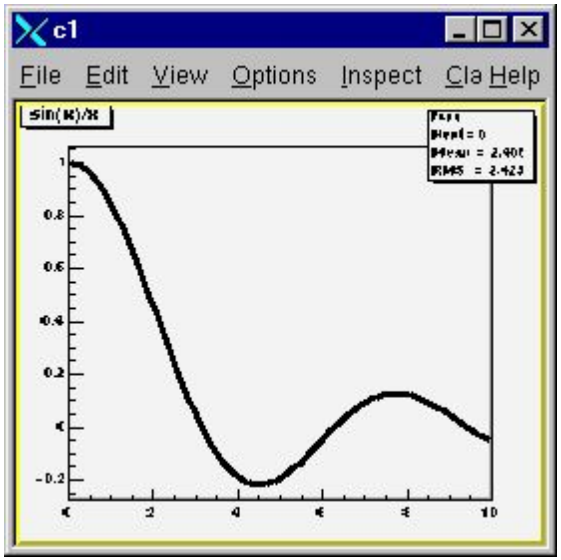


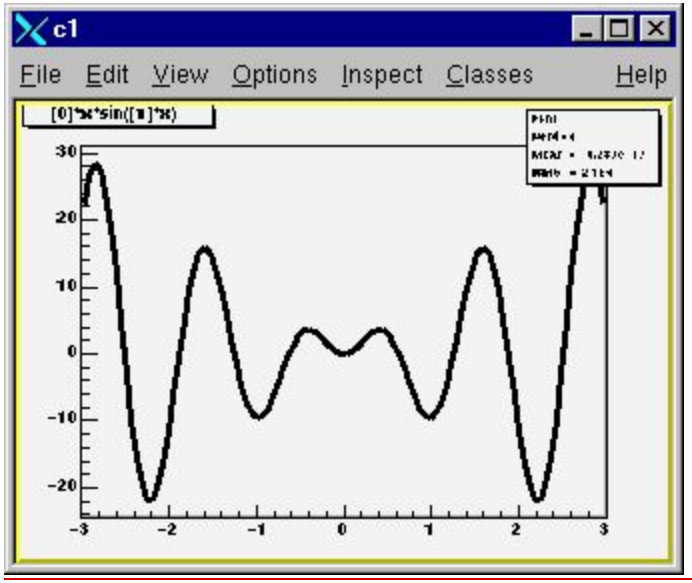


Функции

- Функции без параметров

```
TF1 *f1 = new TF1("f1","sin(x)/x",0,10);
f1->Draw();
```





- Функции с параметрами

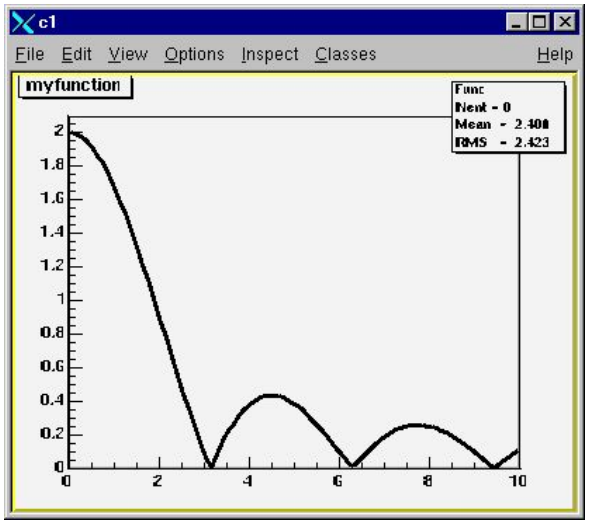
```
TF1 *f1 = new TF1("f1",
                  "[0]*x*sin([1]*x)",
                  -3,3);
f1->SetParameter(0,10);
f1->SetParameter(1,5);
f1->Draw();
```

Функции

- Используя пользовательскую С-функцию

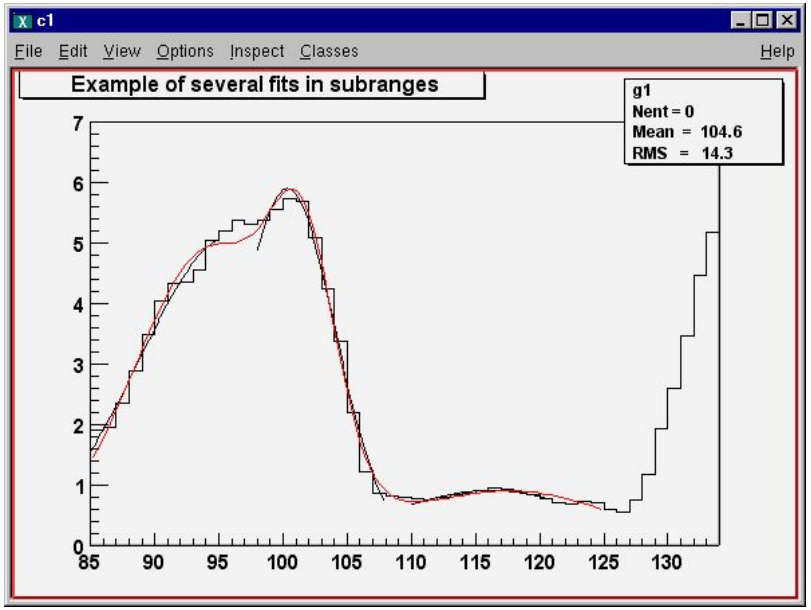
```
Double_t MyFunction(Double_t *x, Double_t *par){
     Float_t  xx  =   x[0];
     Double_t val =
          TMath::Abs(par[0]*sin(par[1]*xx)/xx);
     return val;
}

TF1 *f1 = new TF1("f1",MyFunction,0,10,2);

f1->SetParameters(2,1);

f1->Draw();
```

количество параметров



Подгонка гистограммы

```
Double_t par[9];
Double_t err[9];

TF1 *total = new TF1("total","gaus(0)+gaus(3)+gaus(6)",85,125);
total->SetLineColor(2);

h->Fit(total);

for( int i=0; i<9; i++ ) {
   par[i] = total->GetParameter(i);
   err[i] = total->GetParError(i);
}
```



ROOT scripts

File script1.c:

```
{
#include <iostream.h>
cout << " Hello" << endl;
float x = 3.0;
int i = 101;
cout <<" x=‟ << x
     <<" i="<< I << endl;
}
```

root[0] .x script1.c

File script2.c:

```
#include <iostream.h>
void test() {
   cout << " Hello" << endl;
   float x = 3.0;
   int i = 101;
   cout <<" x=‟ << x
        <<" i=‟ << i << endl;
}
```

root[0] .L script2.c

root[1] test()

TObject

Все объекты ROOT наследуют от общего базового класса – TObject.

Функции TObject:

- Object I/O (Read(), Write())
- Error handling (Warning(), Error(), SysError(), Fatal())
- Sorting (IsSortable(), Compare(), IsEqual(), Hash())
- Inspection (Dump(), Inspect())
- Printing (Print()) Drawing (Draw(), Paint(), ExecuteEvent())
- Bit handling (SetBit(), TestBit())
- Memory allocation (operator new and delete, IsOnHeap())
- Access to meta information (IsA(), InheritsFrom())
- Object browsing (Browse(), IsFolder())

ROOT I/O

ROOT сохраняет данные в файлах специального формата (обычно, бинарных файлах прямого доступа). Любой объект, отнаследованный от TObject, можно сохранить в файле ROOT с помощью object->Write(). Внутренняя структура файлов ROOT напоминает UNIX-подобную структуру директорий, в которой отдельные объекты играют роль файлов.

```
TFile f(“demo.root”, “recreate”);
TH1F *h = new TH1F(“h1”,“Demo histo”,100,-4,4);
h->FillRandom("gaus",1000);
h->Write();
f.Close();
```

ROOT trees

Специальный класс TTree предназначен для сохранения и анализа большого количества однородных объектов.

Возможности TTree:

- сохранять объекты сложной структуры – “древесная” организация данных
- встроенные механизмы компрессии данных
- встроенные механизмы обратной совместимости файлов при изменении структуры объектов
- возможность селективного доступа к данным
- возможность прозрачного объединения деревьев из разных файлов

TNtuple – упрощенная версия TTree (электронная таблица с float)

ROOT tree - создание

```
void tree1w() {
   TFile f("tree1.root","recreate");
   TTree t1("t1","a simple Tree with simple variables");
   Float_t px, py, pz;
   Double_t random;
   Int_t ev;
   t1.Branch("px",&px,"px/F");
   t1.Branch("py",&py,"py/F");
   t1.Branch("pz",&pz,"pz/F");
   t1.Branch("ev",&ev,"ev/I");
   for (Int_t i=0; i<10000; i++) {
      gRandom->Rannor(px,py);
      pz = px*px + py*py;
      random = gRandom->Rndm();
      ev = i;
      t1.Fill();
   }
   t1.Write();
}
```

ROOT tree – чтение в C++

```
void tree1r() {
   TFile *f = new TFile("tree1.root");
   TTree *t1 = (TTree*)f->Get("t1");
   Float_t px, py, pz; Double_t random; Int_t ev;
   t1->SetBranchAddress("px",&px);
   t1->SetBranchAddress("py",&py);
   t1->SetBranchAddress("pz",&pz);
   t1->SetBranchAddress("random",&random);
   t1->SetBranchAddress("ev",&ev);
   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);
      hpxpy->Fill(px,py);
   }
}
```

ROOT tree – получение информации

`tree->Print();`

Распечатать структура дерева (“ветви”)

`tree->Show(10);`

Распечатать значение всех переменных (“листьев”) для записи номер 10 ва

`tree->Scan(“px:py”);`

Распечатать таблицу значений px и py для всех записей в таблице

`tree->Scan(“px:py”,“abs(pz)>0.1”);`

Распечатать таблицу значений px и py для тех записей в таблице, для которых выполняется условие `abs(pz)>0.1`

ROOT tree – анализ данных

`tree->Draw("px");`

Отобразить распределение переменной `px`

`tree->Draw("px:py","abs(pz)>0.1");`

Отобразить двумерное распределение переменных px и py при условии, что `abs(pz)>0.1`

`tree->Draw("(px*px) >> h_px");`

Отобразить распределение переменной px^2 и сохранить его в гистограмму с именем `h_px`

`tree->Draw("px:py","","lego");`

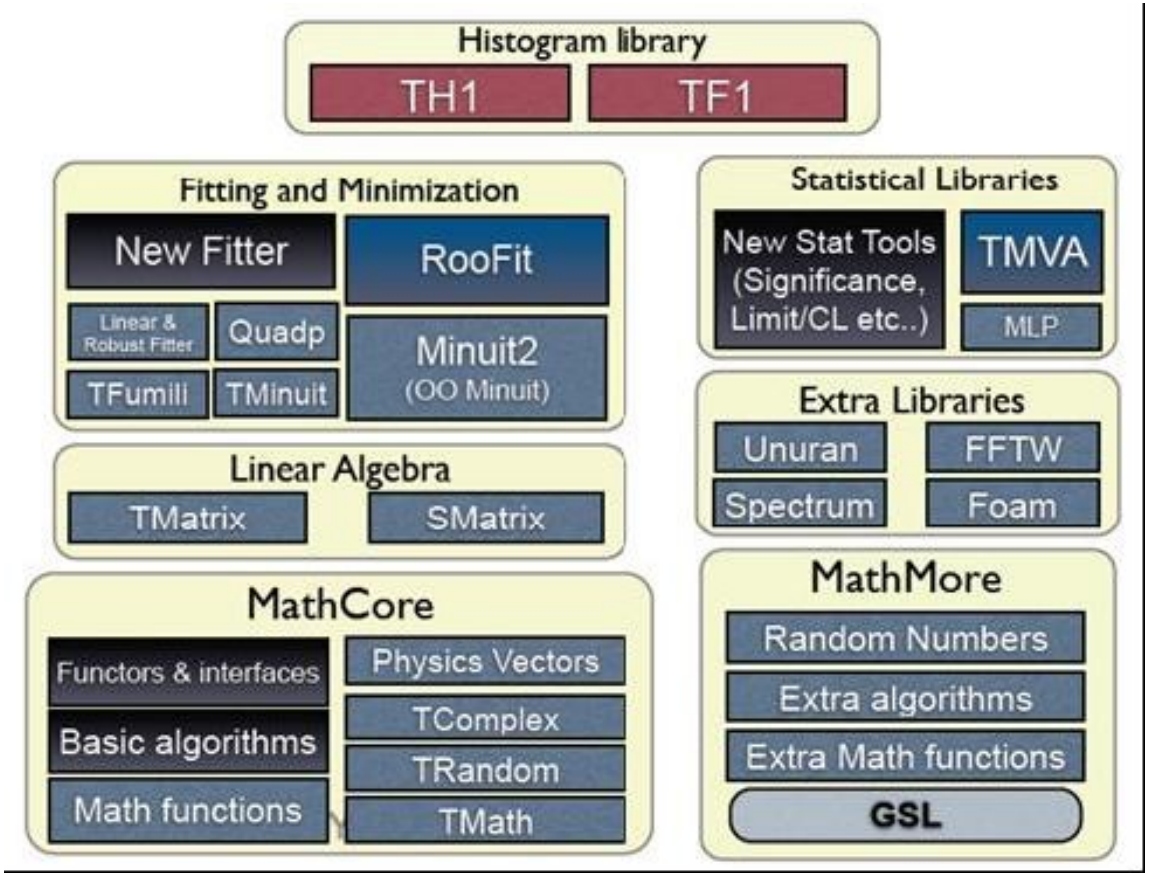
Отобразить двумерное распределение переменных px и py, полученное по всем событиям, записанным в дереве. Использовать графическое представление в виде LEGO-plot.

Анализ нескольких файлов

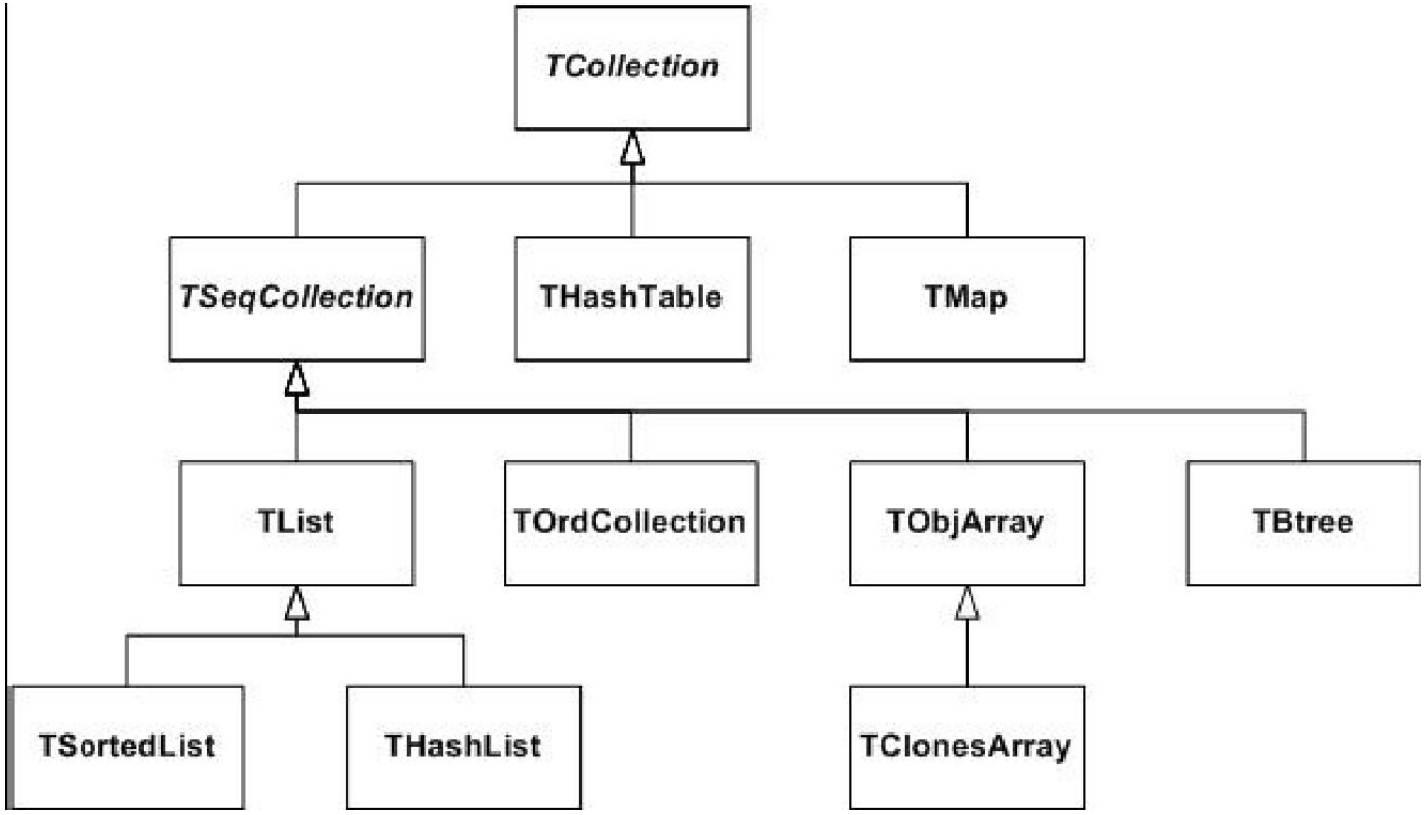
При обработки больших массивов данных часто возникает необходимость обработать большое количество файлов, в которых сохранены деревья одинаковой структуры. Для прозрачного доступа к цепочке файлов в ROOT предусмотрен класс TChain.

```
root[0] TChain chain("T");
root[1] chain.Add("Event.root")
root[2] chain.Add("Event50.root")
root[3] chain.Draw("fTracks.fPx")
```

Математическая библиотека



Коллекции объектов



PyROOT

В стандартный дистрибутив ROOT входит PyROOT – модуль, позволяющий использовать объекты ROOT из интерпретатора python. Используя PyROOT, можно быстро интегрировать ROOT с другими программными продуктами, для которых существует модуль для python. Примеры: графические приложения с использованием Qt, генерация динамических веб-страниц и т.п.

```
from ROOT import gRandom,TCanvas,TH1F
c1 = TCanvas('c1','Example',200,10,700,500)
hpx = TH1F('hpx','px',100,-4,4)
for i in xrange(25000):
   px = gRandom.Gaus()
   hpx.Fill(px)
hpx.Draw()
c1.Update()
```