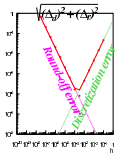
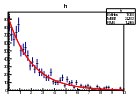
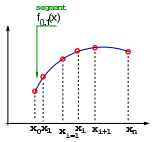
Computational Physics

numerical methods with C++ (and UNIX)

2018-19



Fernando Barao
Instituto Superior Tecnico, Dep. Fisica
email: fernando.barao@tecnico.ulisboa.pt

Computational Physics

ROOT

A data analysis graphics tool with a C++ interpreter

Fernando Barao, Phys Department IST (Lisbon)

ROOT - outline

- ✔ ROOT installation
- ✔ general concepts
- ✔ interactive use and macros
- ✔ canvas and graphics style
- ✔ histograms and other objects
- ✔ fitting
- ✔ input/ouput
- ✔ using ROOT from user programs

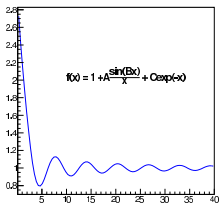
site: http://root.cern.ch

Users Guide: http://root.cern.ch/drupal/content/root-users-guide-600

ROOT - function plotter

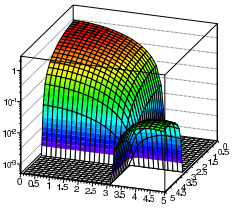
- **ROOT can be used to plot functions: classes TF1, TF2**
- **Plot function:** $f_1(x) = A\frac{sin(Bx)}{x} + Ce^x$

```
[0] gROOT->Reset();
[1] gStyle->SetOptTitle(0);
[2] TCanvas *c = new TCanvas("c", "Phys Comput canvas", 0, 0, 500, 500);
[3] TF1 *f1 = new TF1("f1","1.+ [0]*sin([1]*x)/x + [2]*exp(-x)",0.1, 40.);
[4] f1->SetParameters(1.,1.,1.);
[5] f1->SetLineColor(2);
[6] f1->GetHistogram()->GetXaxis()->SetTitle("x");
[7] f1->Draw();
[8] TLatex l(10.,2.0,''f(x) = 1 +A#frac{sin(Bx)}{x} + Cexp(-x)'');
[9] l.SetTextSize(0.04);
[10] l.Draw();
[11] c->Modified();
[12] c->SaveAs("Sfunctionplotter.eps");
```



- **Plot function:** $f_2(x,y) = \frac{sin(x)\cdot sin(y)}{x\cdot y}$

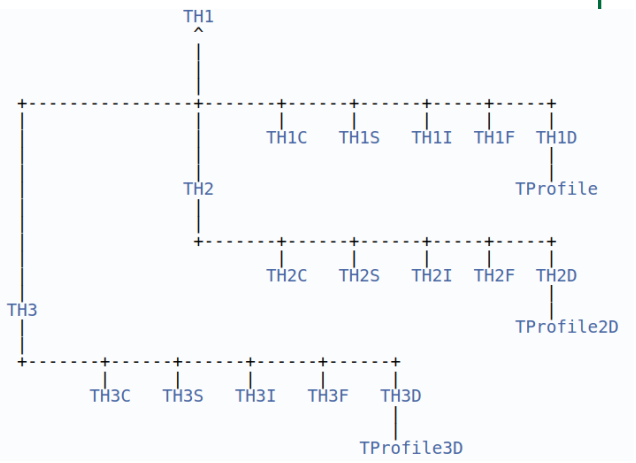
```
[13] TF2 *f2 = new TF2("f2","sin(x)*sin(y)/(x*y)",0,5,0,5);
[14] gPad->SetTheta(25);
[15] gPad->SetPhi(-110);
[16] gPad->SetLogz();
[17] f2->Draw("surf1"); //"", plot contours
```



ROOT - the histogram class

TH1.h

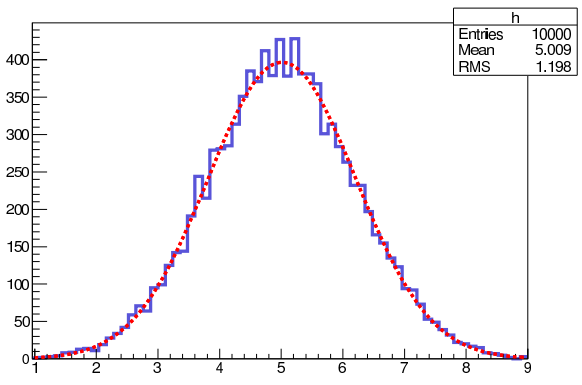
```
class TH1F : public TH1, public TArrayF {

public:
   TH1F();
   TH1F(const char *name,const char *title,Int_t nbinsx,Double_t xlow,Double_t xup);
   TH1F(const char *name,const char *title,Int_t nbinsx,const Float_t  *xbins);
   TH1F(const char *name,const char *title,Int_t nbinsx,const Double_t *xbins);
   TH1F(const TVectorF &v);
   TH1F(const TH1F &h1f);
   virtual ~TH1F();

   ...
};
```



ROOT - histograms

Let's make an histogram from random numbers generated from a gaussian of mean 5. **and standard deviation** 1.2

```
 gStyle->SetOptTitle(0); //no title
// define gaussian function
 TF1 *f1 = new TF1("f1","gaus()",0.,12.);
 f1->SetParameters(1.,5.,1.2); //set gaussian params

// histogram to store randoms

 TH1F *h = new TH1F("h", "histogram", 100, 0., 12.);
 for (int i=0; i<10000; i++) {h->Fill(f1->GetRandom());}

// cosmetics

 h->GetXaxis()->SetRangeUser(1.,9.);
 h->SetLineWidth(4);
 h->SetLineColor(9);
 h->Draw();
 h->Fit("f1");

// retrieve function used on fit and plot

 TF1 *fg = h->GetFunction("f1");
 fg->SetLineWidth(4);
 fg->SetLineStyle(2);
 fg->DrawCopy("same"); //superimpose plots
```



Simulating muon decay

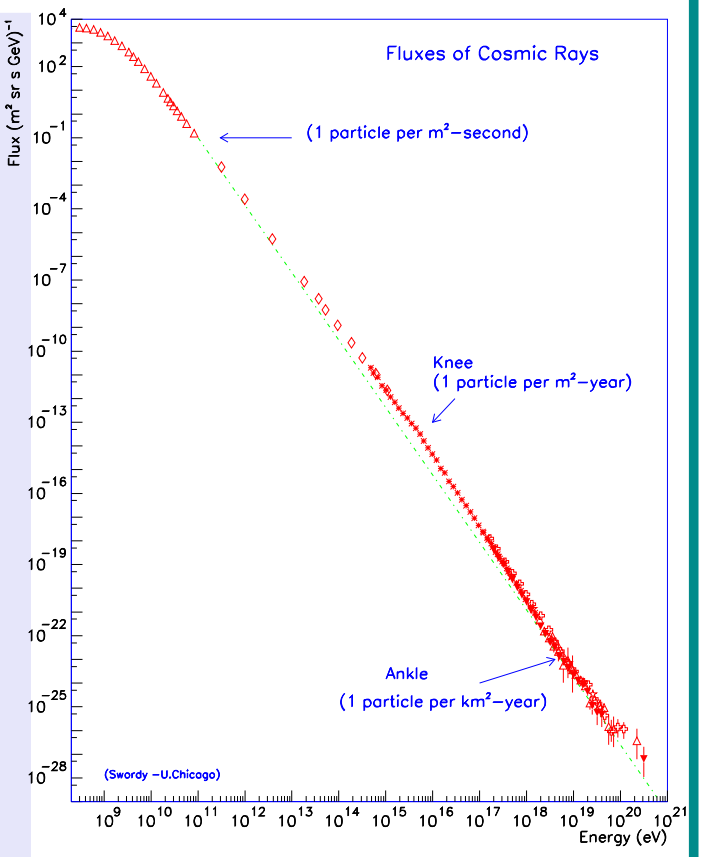
Imagine that you want to simulate a muon decay experiment; i.e., we want to get the time distribution between events

This implies to simulate "all"the physics processes (we have to start somewhere) while stepping our muons!

- ✔ For instance, we can start from the muon flux arriving at earth
  of course, people needing to study it, need to simulated all the atmospheric interactions...
  in terms of numerical simulation, we need to randomly generate muon momenta and direction according to known spectra
- ✔ we need to simulate the interactions of the muons being detected by our apparatus
  basically, the muons can loose energy and decay
- ✔ Once muons decay, we can simulate the decay time distribution
  in fact, what is detected is somewhat more complicated because apart muon decay signals, we also detect:
  - time difference between cosmic muons crossing detector
  - negative muons can be captured by the nucleus giving a neutrino plus a neutron

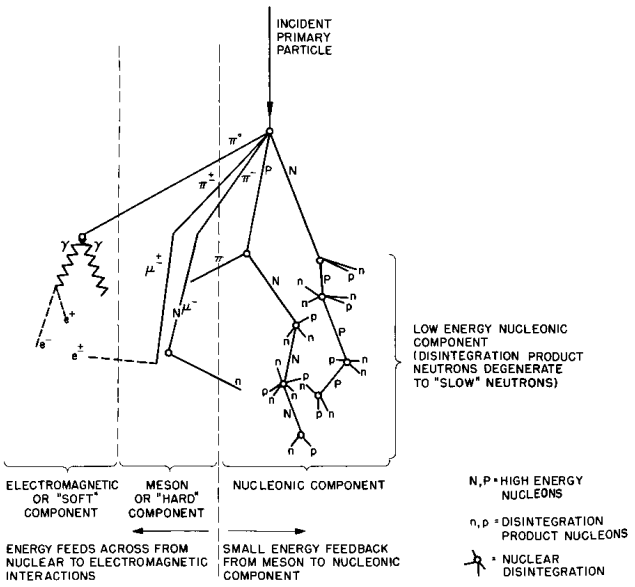
Cosmic Rays

- ✔ Essencialmente de origem galáctica
- ✔ Maioritariamente ($> 90\%$) protões
- ✔ Raios Cósmicos altamente energéticos
  $E \sim 10^{21} eV$
- ✔ Espectro de energia
  $\frac{dN}{dE} \propto E^{-\alpha} \quad (\alpha \sim 2.7)$
- ✔ Região de baixa energia
  modulada pelo vento solar e campo geomagnético



Produção de muões cósmicos

- ✔ interacção do raio cósmico primário induz um air shower
- ✔ o chuveiro possui uma forte componente hadrónica
  induz a produção de múltiplos chuveiros electromagnéticos ($\pi^0 \to \gamma\gamma$)
- ✔ Os piões carregados ($\pi^\pm$) podem interagir com os núcleos ou desintegrar-se
  $\tau_\pi \simeq 2.6 \times 10^{-8}\ s$
- ✔ Os muões surgem da desintegração dos piões
  $\pi^\pm \to \mu^\pm + \nu_\mu$
- ✔ fotões,electrões e positrões: dominam no chuveiro
- ✔ muões: uma ordem de grandeza abaixo

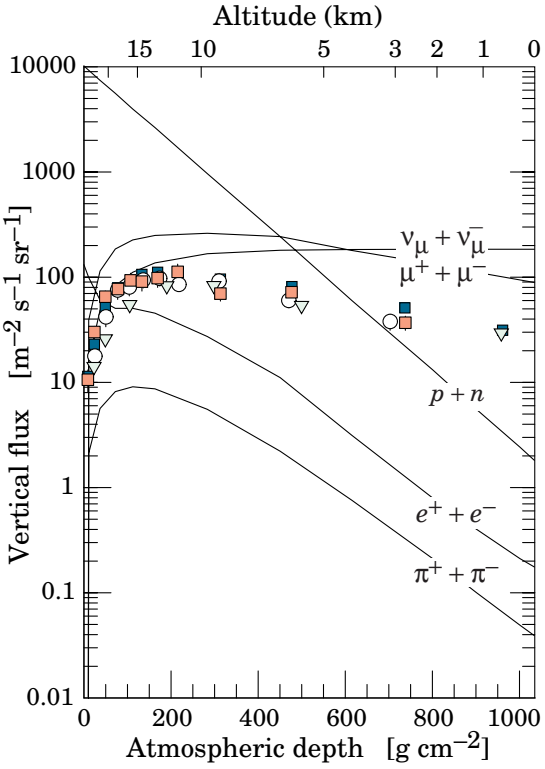


Schematic Diagram of Cosmic Ray Shower

Fluxo de partículas cósmicas

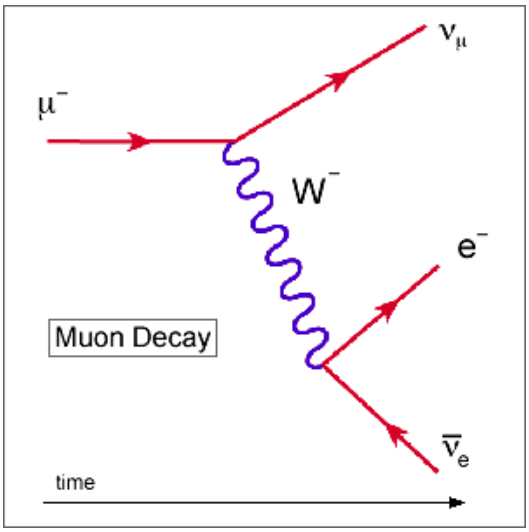
Medido ao nível do mar

- ✔ À superfície da Terra, os raios cósmicos observados são de natureza secundária
- ✔ São essencialmente constituídos por muões e cerca de 1% de protões (partículas carregadas... obviamente os neutrinos estão presentes!)
- ✔ O fluxo de representado é o chamado espectro *inclusive* ou descorrelado: inclui partículas de múltiplos chuveiros que atravessam um dado detector



Desintegração do muão

$\mu \rightarrow e + \nu_e + \nu_\mu$



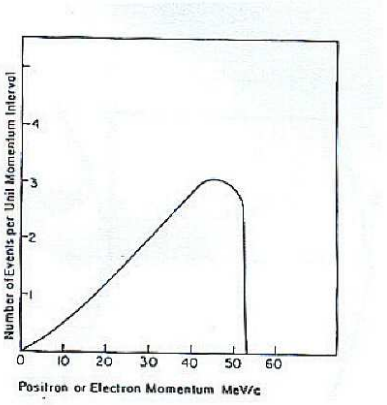
no decaimento do muão existe a troca de um bosão W (um dos bosões mediadores da força fraca)

- Espectro de energia do electrão (μ frame):

$$\frac{dN}{dE_e} = \frac{G^2}{12\pi^3} m_\mu^2 E_e^2 \left(3 - 4\frac{E_e}{m_\mu}\right)$$

onde G é a constante de Fermi

- energia máxima do electrão: $53\ MeV$



Desintegração do muão

A probabilidade diferencial de decaimento do muão que resulta num electrão (positrão) com energia reduzida entre x e $x + dx$, emitido numa direcção que faz um ângulo entre θ e $\theta + d\theta$ com respeito à linha de vôo do muão:

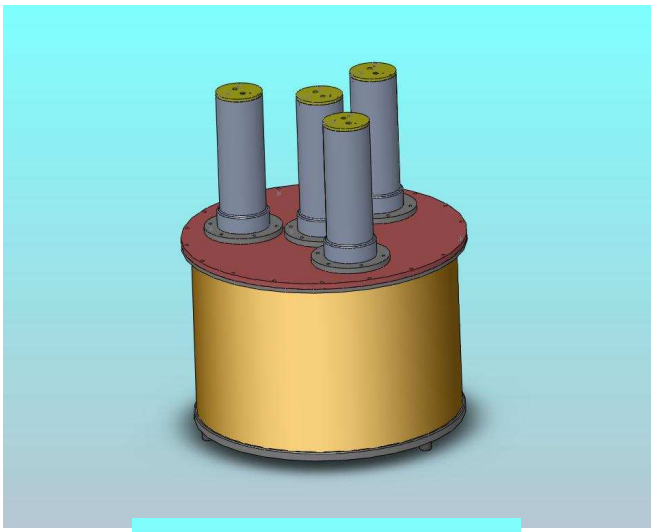
$$\frac{d^2\Gamma}{dx\, d\cos\theta} = \frac{G_F^2 m_\mu^5}{192\pi^3} x^2 \left[3 - 2x + \cos\theta(2x - 1)\right]$$

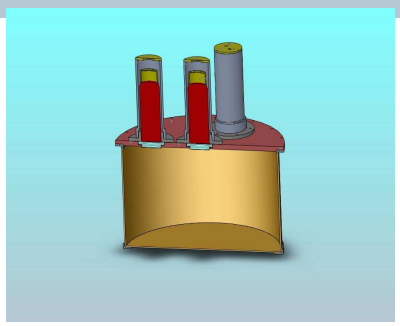
onde:

$x = 2E_e/m_\mu$ energia reduzida do electrão

E_e energia do electrão no referencial do muão

Decaimento do muão: Cerenkov tank

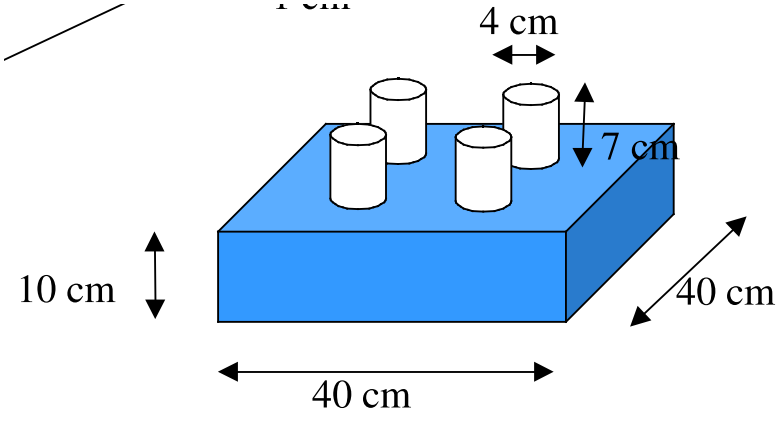


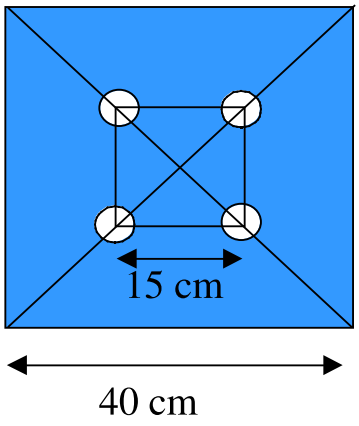


Um reservatório preenchido com água desmineralizada onde se explora o efeito de Cerenkov na detecção de partículas

muon decay: scintillator det

- ✔ bloco cintilador de $40 \times 40 \times 10\ cm^3$
- ✔ leitura do sinal por (até) 4 fotomultiplicadores





BC408 - Cintilador plástico
light output: 64% Anthracene
λ_{max}: 425 nm
http://www.detectors.saint-gobain.com/Data

XP2020 - Photomultiplicador

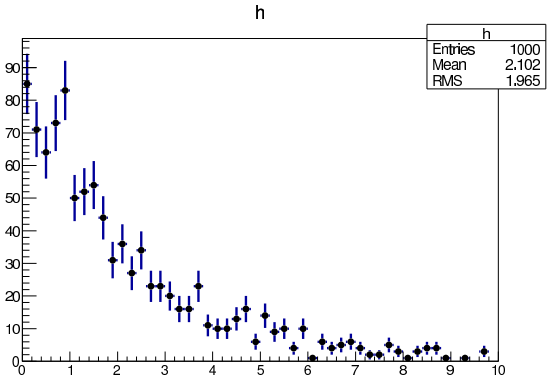
ROOT - graphics and muons

muons particles arrive all time to earth from cosmic rays showers induced by primary protons coming from the galaxy

their lifetime is short ($2.2\ \mu$sec) but Lorentz boost dilates their time and we can "see"them at earth surface!!!

let's simulate decay time that follows and exponential law with a time constant of $2.2\ \mu$sec

```
TCanvas *c = new TCanvas();

//create histogram object
//0 to 10 microseconds (object name = "h")
TH1F h("h","h",50, 0.,10.);

//get exponential random
for (int i=0; i<1000; i++) {
  h.Fill(gRandom->Exp(2.2));
}
h.SetLineWidth(3);
h.SetMarkerStyle(20);
//DrawCopy provide us with a copy of the histogram
//here is mandatory because h goes out of scope
h.DrawCopy("E");

c->Update();
```



ROOT - muons (cont.)

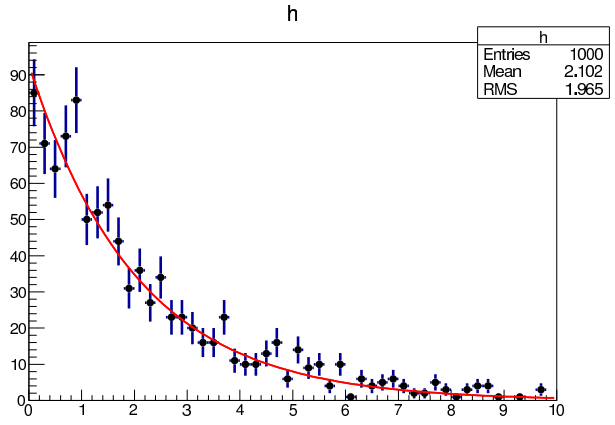
let's fit now the exponential law to the data

$N(t) = N_0 e^{-t/\tau}$ (2-parameters law)

```
//make canvas
TCanvas *c = new TCanvas();

//fit distribution with exponential
h.Fit("expo");

// draw hist data with error bars
h.DrawCopy("E");

// show fit function
TF1 *f = h.GetFunction("expo");
f->DrawCopy("same");

// refresh canvas
c->Update();
```

TFormula

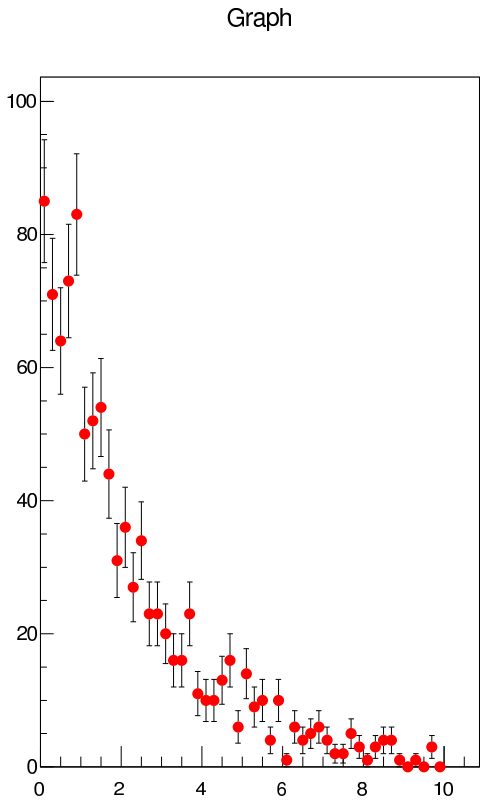
```
Example of valid expressions:
 sin(x)/x
 [0]*sin(x) + [1]*exp(-[2]*x)
 x + y**2
 x^2 + y^2
 [0]*pow([1],4)
 2*pi*sqrt(x/y)
 gaus(0)*expo(3)  + ypol3(5)*x
 gausn(0)*expo(3) + ypol3(5)*x
```



ROOT - retrieving histogram data

let's retrieve the histogram data and store them in a data file

```
////////////////////////// output measurements
ofstream F("muon-data.txt");
float *xbin = new float[h.GetNbinsX()];
float *vbin = new float[h.GetNbinsX()];
float *ey = new float[h.GetNbinsX()];
cout << "histo: nb bins = " << h.GetNbinsX() << endl;
for (Int_t i=0;i<h.GetNbinsX();i++) {
  xbin[i] = h.GetBinCenter(i+1);
  vbin[i] = h.GetBinContent(i+1);
  ey[i] = h.GetBinError(i+1);
  F << xbin[i] << " " << vbin[i] << " "
  << ey[i] << endl;
}
F.close();
////////////////////////// make graph
TGraphErrors g("muon-data.txt",''%lg %lg %lg'');
g.SetMarkerColor(2);
g.SetMarkerStyle(20);
g.SetMarkerSize(1);
g.DrawClone(''AP'');
c1->Update();
```



ROOT classes for Physics

There are many classes in ROOT useful to be used in Physics

- ✔ **TVector3**
  Suppose you are following a particle for studying its interactions in materials.
  In fact, a *particle track* can be defined with three-dimensional points
- ✔ **TLorentzVector** a four-dimensional vector used for relativistic and kinematics computations, both in spacetime (x, y, z, t) and in momentum space (px, py, pz, E). It is implemented as a TVector3 and a *Double_t* variable.

```
// using constructors:
TLorentzVector A(1.,2.,3.,4.);
TLorentzVector A(TVector3(5.,6.,7.),8.);

// setting elements
A.SetVect(TVector3(1,2,3));
A.SetXYZT(x,y,z,t);

//accessing elements
A.X() //
A.T() //
A.Px() //
A.E() //
```

ROOT classes for Physics (cont.)

There are many classes in ROOT useful to be used in Physics

- ✔ **TPhaseSpace**
  It generates kinematics events for decays of a particle into **N** particles with provided masses.
- ✔ **TParticlePDG**
  Interesting class to access the PDG database for retrieving information on particles properties, quantum numbers, decay channels and branching ratios. The all Particle Data Group booklet is provided in a ASCII file called "pdg_table.txt".

ROOT - user interface to ROOT

- ✔ User class methods of general interest can be placed available in a *Computational Physics library* - collaborative work! The user has to link its C++ program with this library to get access to the class methods there implemented
- ✔ In order to provide an easy user interface to graphics displays using ROOT I made a starting class interface
- ✔ *cFCgraphics class* that provides access to ROOT in a easy way

cFCgraphics.h

```
#ifndef __CFCG__
#define __CFCG__

#include <string>
using namespace std;

class TList;
class TApplication;

class cFCgraphics {
  public:
  cFCgraphics();
  ~cFCgraphics();
  //create Pad and add objects to it
  TPad* CreatePad(string PadName);
  void AddObject(TObject *obj, string PadName);
  void AddObject(TObject *obj, string PadName,
                               string sopt);
  void DrawPad(string PadName);
  void DumpPad(string PadName);
  (...)
```

cFCgraphics.h

```
  (...)
  //add objects to Canvas
  void AddObject(TObject *obj=NULL );
  void ListObjects();
  void Draw(string sopt="all");
  int GetNumberOfListEntries();
  void Print(string); //save to file

  // clear objects
  void Clear();

  private:
  TApplication *gMyRootApp;
  TList *L, *Ltmp; //list of ROOT objects
  map<string,TPad*> Mpad;
  map<string,TPad*>::iterator itMpad;
  TCanvas* fCanvas;
  static int Ncanvas;
};
#endif
```

ROOT - user interface to ROOT (cont.)

cFCgraphics.C

```
#include <TROOT.h>
#include <TGFrame.h> //gClient
#include <TCanvas.h>
#include <TPad.h>
#include <TSystem.h>
#include <TList.h>
#include <TApplication.h>
#include <TVirtualX.h>
#include <TF1.h>

#include <iostream>
using namespace std;
#include "cFCgraphics.h"

cFCgraphics::cFCgraphics () {
  L = new TList(); //create TList
  //create application
  int argc = 0;
  char **argv = NULL;
  gMyRootApp = new TApplication("Comput Phys Application", &argc, argv);
}

cFCgraphics::~cFCgraphics () {
  delete gMyRootApp;
  delete L;
}
```


ROOT - user interface to ROOT (cont.)

cFCgraphics.C (cont.)

```
void cFCgraphics::AddObject(TObject *obj) {
  if (obj) {
    L->Add(obj);
    cout << Form(''[cFCgraphics::AddObject] %s added.. .(Total nb objects=%d)\n'', obj->GetName(), GetNumbe
  }
}

void cFCgraphics::ListObjects() {
  TListIter next(L);
  TObject *to;
  cout << Form(''[cFCgraphics::ListObjects] (Total nb objects=%d)\n'', GetNumberOfListEntries());
  while ((to=next())) {
    cout<< Form('' ---> %s \n'',to->GetName());
  }
  cout<< endl;
}
```

ROOT - user interface to ROOT (cont.)

cFCgraphics.C (cont.)

```
void cFCgraphics::Draw(string sopt) {
  cout<< Form(``[cFCgraphics::Draw] Drawing objects...[%s]\n'',sopt.c_str());
  //new canvas
  TCanvas *fCanvas = new TCanvas(Form(``FC_canvas_%s'',sopt.c_str()), Form(``Canvas - %s'', sopt.c_str()), 900, 700);
  //all or only one object
  if (sopt == ``all'') {
    // ... get number of list entries
    int count = GetNumberOfListEntries();
    int nlines = count/3;
    int ncolus = count%3;
    if (nlines==0) {
      fCanvas->Divide(ncolus,nlines+1);
    } else {
      fCanvas->Divide(3,nlines+1);
    }
    TListIter next(L);
    TObject *to;
    int n = 0;
    while ((to=next())) {
      n++;
      fCanvas->cd(n);
      to->Draw();
    }
  } else {
    TObject *o = L->FindObject(sopt.c_str());
    if (o) o->Draw();
  }
  fCanvas->Update();
  fCanvas->SaveAs(Form("FC_canvas_%s",sopt.c_str()));
  gMyRootApp->Run(kTRUE);
}
```


ROOT - user interface to ROOT (cont.)

cFCgraphics.C (cont.)

```
int cFCgraphics::GetNumberOfListEntries() {
  TListIter next(L);
  TObject *to;
  int count=0;
  while ((to=next())) {count++;}
  return count;
}
```

The following C++ program adds four ROOT graphics objects to the *cFCgraphics class* to display them.

main program: Rgraphics.C — **using class cFCgraphics**

```
// F Barao (FC, Jul 2014)
// g++ -o Rgraphics.exe \
// Rgraphics.C cFCgraphics.C

#include <cmath> //exp
#include <cstdlib> // for atof(3)
#include <iostream>
using namespace std;
#include <TF1.h> //ROOT TF1
#include "cFCgraphics.h" //my class

int main(int argc, const char* argv[]) {
  cFCgraphics gr;
  TF1 *f1 = new TF1("FCsinx","sin(x)", 0., 10.);
  gr.AddObject(f1);
  gr.AddObject(new TF1("FCexp","exp(x)", 0., 10.));
  gr.AddObject(new TF1("FCsinxx","sin(x)/x", 0., 10.));
  gr.AddObject(new TF1("FClogx","log(x)", 0., 10.));
  gr.ListObjects(); // list user stored objects
  gr.Draw(); // draw user objects
  gr.Print("Rgraphics.eps"); // save objects
  return 0;
}
```