Alexandre A. P. Suaide

ROOT
IFT - 2010

Objetivos

- Mostrar os fundamentos do c++
  - Não é um curso de lógica de programação
  - Programação orientada a objetos
- Introdução ao ROOT
  - ROOT como ferramenta de programação e análise de dados
- 2 aulas
  - Notas e exemplos em algum lugar

Programa

- Introdução ao c++
  - Comandos básicos, referências e ponteiros
  - Conceito de classe e objeto
- Conceitos básicos do ROOT
- Gráficos e funções no ROOT
- Histogramas de 1 e 2D
- Funções e ajustes de dados
- Ajustes de funções, legendas, etc.
- Monte Carlo e simulações

Referências (http://root.cern.ch)

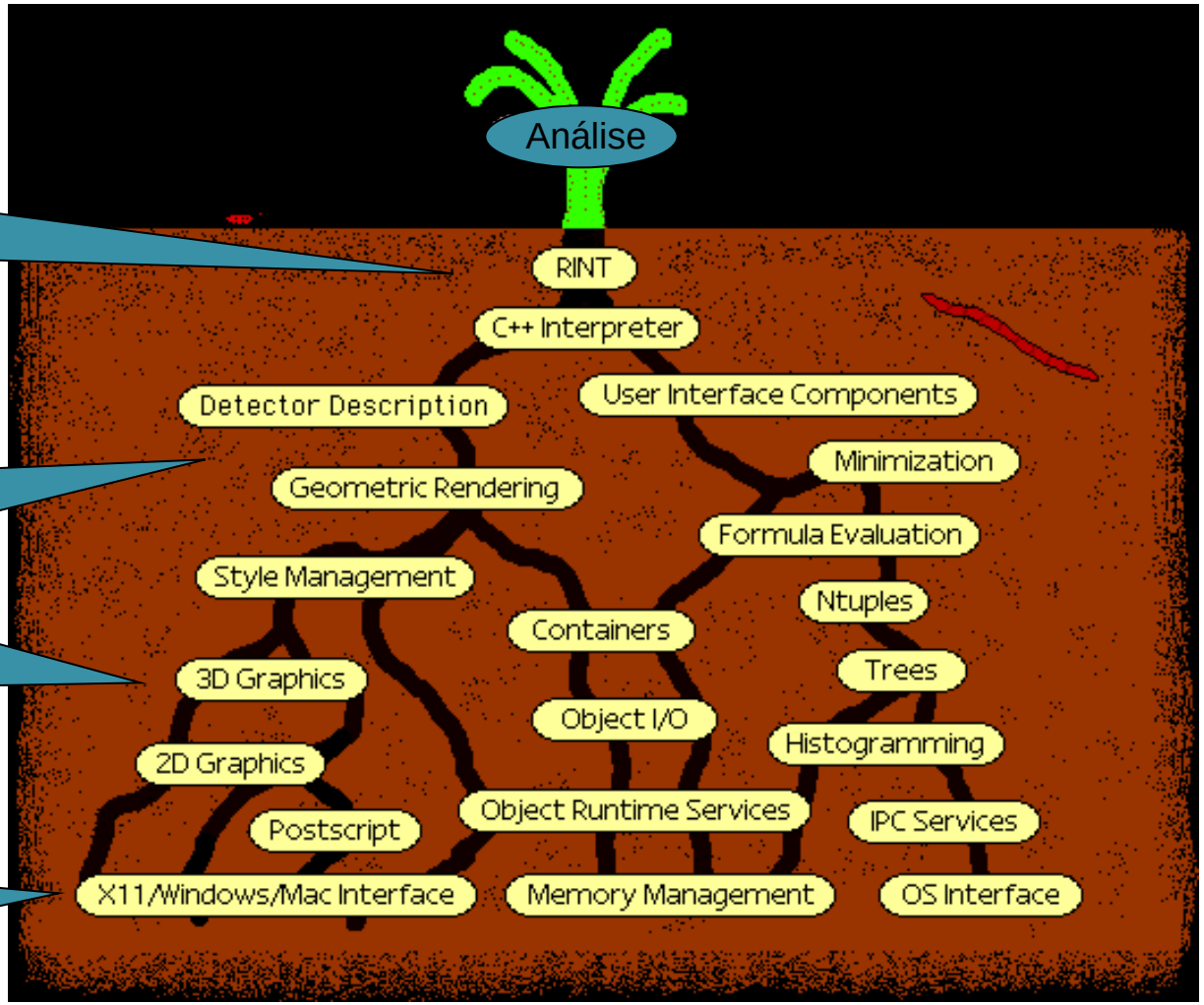
O que é o ROOT?

Interface entre o usuário e as ferramentas de análise

Ferramentas de visualização e manipulação gráfica

Ferramentas básicas I/O, ferramentas gráficas, métodos de cálculo

Nível mais fundamental. Interface com S.O.

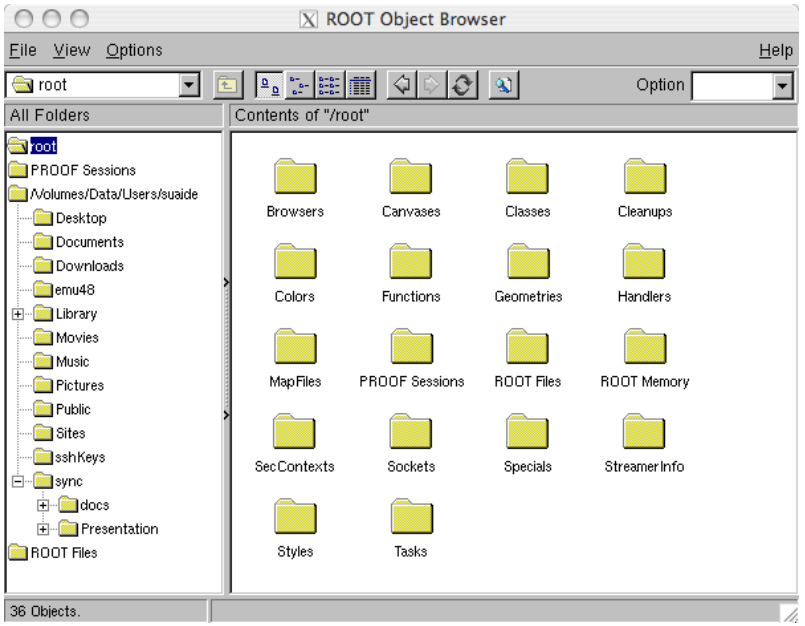


Comandos básicos

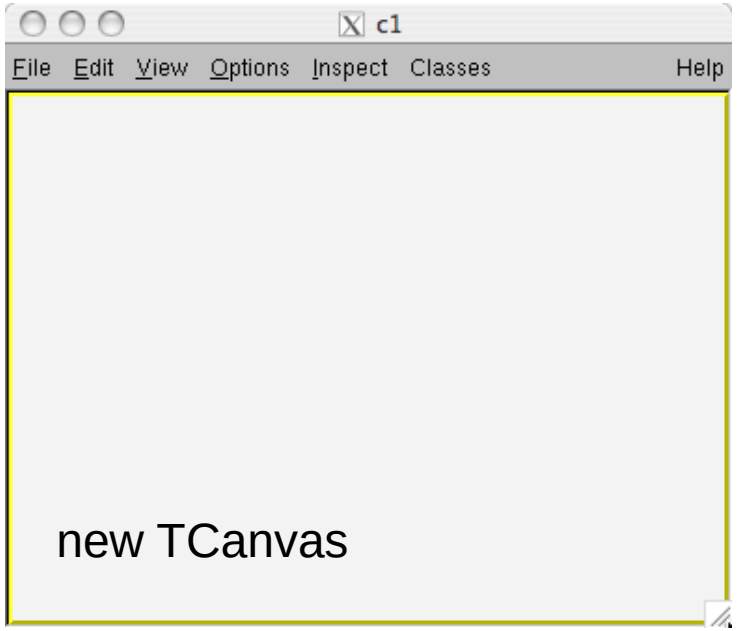
- Como iniciar o programa
  - Duplo clique no ícone
  - Linha de comando: digite root
- Como sair do ROOT
  - Digite .q
- Estranho, mas como o ROOT é um interpretador c++, os comandos internos do mesmo têm que ser diferenciados. Assim, todos os comandos do ROOT começam com “.”. Os comandos mais importantes, alem do .q são
  - .L para carregar um programa (macro) na memória
  - .x para carregar e executar um programa
  - .h para um help dos comandos disponíveis

Interface

new TBrowser



- Use a tecla TAB para obter ajuda
- **root [0]** b = new TB <TAB>
- **root [1]** b = new TBrow<TAB>
- **root [2]** b = new TBrowser(<TAB>
- Útil para descobrir a lista de métodos
- Descobrir também lista de parâmetros



Variáveis

- **Float, int, char, etc...**
- **No ROOT TAMBÉM: Float_t, Int_t, Char_t**
  - **Para compensar diferenças entre sistemas**

- Exemplos de definição de variáveis

```
int a; Definição simples
int i,j,k; Definição múltipla
float pi=3.1415, r0=1.2; Definição c/inicialização
double a[10], b[5][20]; Definição de vetores
float c[5] = {1,2,3,4,5}; Definição c/inicialização
float d[3][2] = {{1,2,3},{4,5,6}};
char texto[20] = "testando";
```

Loops

- for (cond. inicial; condição de teste; alteração) { comandos }

```
for(int i=0; i<10; i++)
{
  int a = i*i;
  cout << i << "*" << i << " = " << a << endl;
}
```

- do { alguma coisa } while (condição de teste);

```
int i = 0;
do
{
  int a = i*i;
  cout << i << "*" << i << " = " << a << endl;
  i++;
} while (i<10);
```

- while (condição de teste) { alguma coisa }

```
int i = 0;
while (i<10)
{
  int a = i*i;
  cout << i << "*" << i << " = " << a << endl;
  i++;
}
```

if .. then … else

```
if ( condição ) comando;
else outroComando;
```

O comando **else** é opcional

```
if ( condição )
{
   comando1;
   comando2;
}
else
{
   comando1;
   comando2;
}
```

```
if (a<10) cout << "a é menor que 10" << endl;
else cout << "não é menor que 10" << endl;

if(a>10 && a<20) cout <<"a entre 10 e 20"<<endl;

if(a!=10) cout <<"a é diferente de 10"<<endl;
```

- Operadores condicionais

  == (igual), != (diferente), || (or), && (and), ! (not), etc.

funções

```
tipo nome(parametros)
{
  comandos;
  return valor;
}
```

- **Exemplo: cálculo da área de um retângulo**

```
float area(float lado1, float lado2)
{
  float a = lado1*lado2;
  return a;
}
float l1,l2;
cout << "Entre os lados do retângulo ";
cin >> l1 >> l2;
cout << "A área desse retângulo é " << area(l1,l2) <<endl;
```

Polimorfismo

- **cálculo da área de um retângulo e quadrado**

```
float area(float lado1, float lado2)
{
  float a = lado1*lado2;
  return a;
}
float area(float lado)
{
  float a = lado*lado;
  return a;
}
...
float area1 = area(10,20);
float area2 = area(10);
```

Ponteiros e referências

- Quando criamos uma variável (ou qualquer outra coisa) esta ocupa um lugar na memória do computador
  - float a = 10;
- Em alguma posição desta memória temos armazenado o valor a e em algum outro lugar da memória temos armazenado que existe a variável a e em que lugar o seu conteúdo está armazenado.
- Como acessar estas informações?
- Como acessar esta memória e modificá-la?
- Como atuar sobre a variável?

Referências (&)

- Referências são apelidos para as variáveis/objetos criados durante um programa
- Exemplos

```
float  a = 10;
float& b = a;
cout <<"a = "<<a<<"  b = "<<b<<endl;
a = 10   b = 10
a = 20;
cout <<"a = "<<a<<"  b = "<<b<<endl;
a = 20   b = 20
b = b-5;
cout <<"a = "<<a<<"  b = "<<b<<endl;
a = 15   b = 15
```

Quando usar referências?

- Uso mais comum é como parâmetros de funções
- Quando definimos uma função, por exemplo

```
float funcao(float a, int b)
```

  - Os parâmetros a e b são tratados, pela função, como variáveis locais
  - Ex:

```
void func(float a)
{
  a = a*2;
  cout << a << endl;
  return;
}
float var = 1;
func(var);
2
cout << var << endl;
1
```

Note que o valor de x é alterado localmente. A variável original, var, neste caso, permanece com o seu valor original

Quando usar referências?

- Passar variáveis para funcões

```
float funcao(float& a, int b)
```

  - Neste caso, posso manipular o valor da variável a enquanto b é tratado localmente
  - Ex:

```
void func(float& a)
{
  a = a*2;
  cout << a << endl;
  return;
}
float var = 1;
func(var);
2
cout << var << endl;
2
func(3);
Error: could not convert '3' to float&
```

Note que o valor de var foi alterado pois eu passei uma referência desta variável para a função

Apesar do ROOT aceitar esta sintaxe no prompt de comando, o compilador c++ retornará uma mensagem de erro

Ponteiros (*)

- Ponteiros são variáveis que contém endereços de memória para alguma coisa
- Ao contrário de referências estes podem mudar de localização
- Operadores
  - * → define um ponteiro
    - Ex: float* a;
      - 'a' contém um endereço de memória para que contém um número float
  - * → também pode ser utilizado para acessar o conteúdo de uma posição de memória
  - & → utilizado para obter o endereço de memória de uma variável/objeto

Ponteiros (*)

- Exemplos

```
float x = 10;
float *p = &x;
cout << x <<endl;
10
cout << p <<endl;
(float*)0x45daa58
cout << *p <<endl;
10
*p = 20;
cout << x <<endl;
20
cout << p <<endl;
(float*)0x45daa58
cout << *p <<endl;
20
```

Cria um ponteiro do tipo float cujo valor corresponde ao endereço de memória onde esta armazenada a variável x

O valor de x

O valor de p. Este corresponde ao endereço de memória onde a variável x está armazenada

O conteúdo armazenado nesta posição de memória

Mudando o conteúdo da memória

Isto reflete automaticamente na variável x

Arrays são ponteiros

- O c++ trata vetores/arrays como sendo ponteiros

```
float a[10] = {1,2,3,4,5,6,7,8,9,10};
cout << a <<endl;
0x95bf5b8
cout <<a[0]<<" "<<*a<<endl;
1  1
cout <<a[5]<<" "<<*(a+5)<<endl;
6  6
```

Isto pode ser útil caso queiramos passar um array como argumento de função

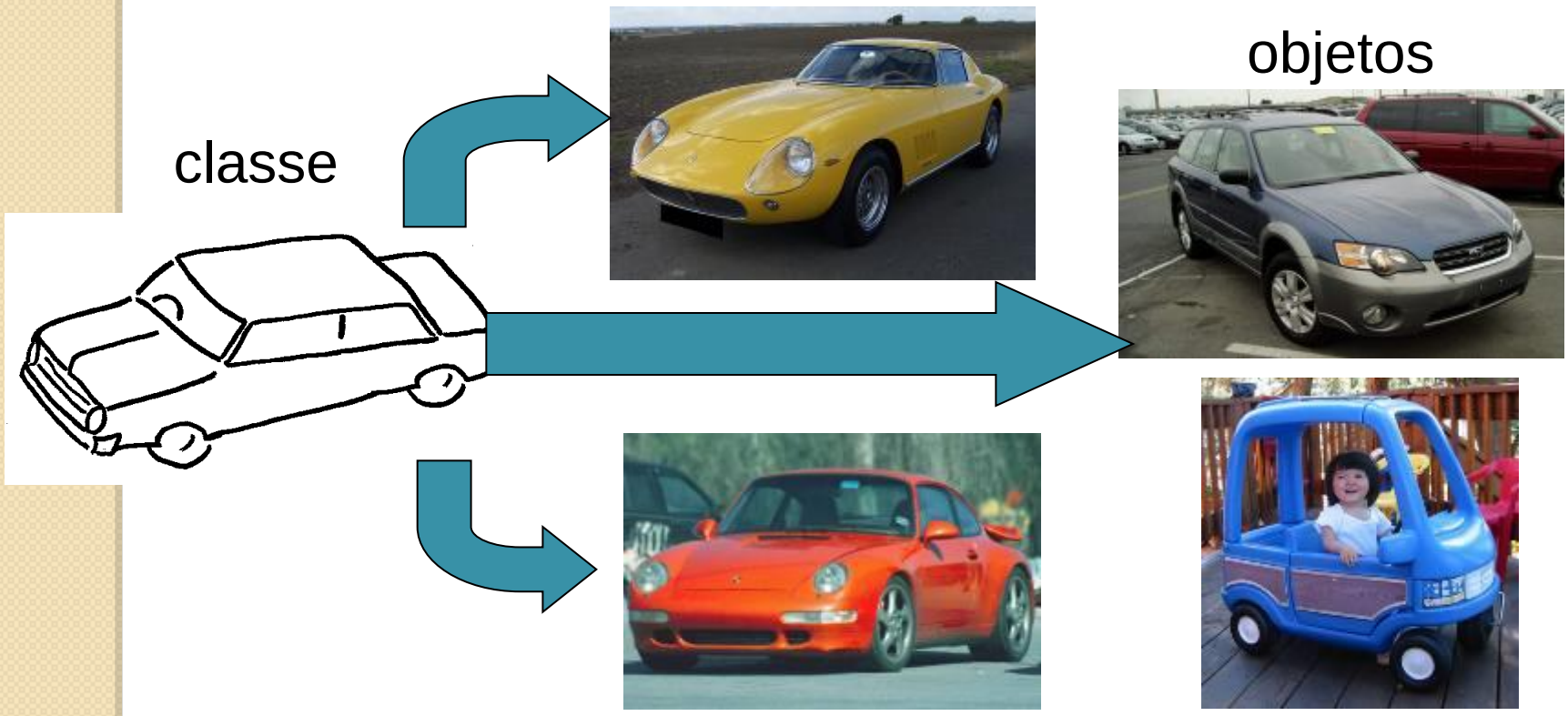
Arrays como argumentos de funções

- Caso queiramos passar um array para uma função em c++ utilizamos o seu ponteiro

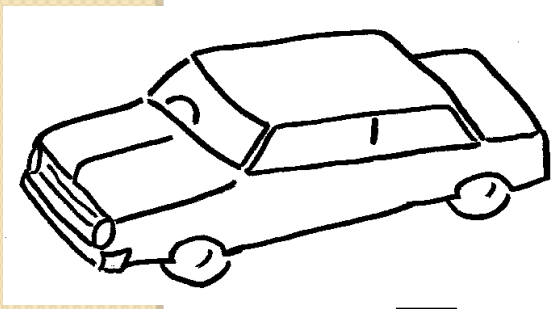
```
void massa(float* M)
{
  cout <<"M[0] = "<<M[0]<<"   "
       <<"M[1] = "<<M[1]<<"   "
       <<"M[2] = "<<M[2]<<"   "
       <<"M[3] = "<<M[3]<<endl;
  return;
}
float A[4] = {16,10,20,6};
massa(A);
M[0] = 16  M[1] = 10  M[2] = 20  M[3] = 6
```

Classes e objetos

- Classes são os moldes para criação de um objeto em c++
- Objetos são entidades concretas (espaço em memória) criados e organizados com estrutura definida pelas classes

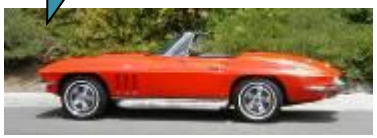


Classes derivadas



Classe genérica

Classes derivadas



Objetos

Classes em c++

Define uma classe chamada Bla

Esta classe é derivada da classe Blu (caso necessário)

```
class Bla : public Blu
{
  protected:
     <membros protegidos>;
  private:
     <membros privados>;
  public:
     <membros publicos>;

     Bla();
     virtual ~Bla();
};
```

Os membros e funções definidas serão tratados como protegidos

Os membros e funções definidas serão tratados como privados

Os membros e funções definidas serão tratados como públicos

Construtor da classe.

Destrutor (~) da classe.

As funções construtoras e destrutoras DEVEM ter o mesmo nome da classe

Tipos de membros

- Os membros de uma classe podem ser variáveis, funções, ponteiros ou até mesmo objetos de outras classes
  - Dependem da tarefa a ser executada
- Membros podem ser potected, private ou public
  - Public
    - Não há restrições no acesso a estes membros, sejam em classes derivadas ou pelo usuário
  - Protected
    - Só podem ser acessados pela própria classe ou classes derivadas
  - Private
    - Só podem ser acessados pela própria classe
- Existe um tipo de classe chamada “friend” que pode acessar todos os membros de outra classe mas o seu uso não é recomendado.

Construtores

Classes das quais esse objeto também faz parte. Métodos públicos dessas classes também são acessíveis

```
class TGraph : public TNamed, public TAttLine, public TAttFill,
   public TAttMarker
{
  public:
            TGraph();
            TGraph(Int_t n);
            TGraph(Int_t n, const Int_t* x, const Int_t* y);
            TGraph(Int_t n, const Float_t* x, const Float_t* y);
            TGraph(Int_t n, const Double_t* x, const Double_t* y);
   Double_t GetErrorX(Int_t bin);
   Double_t GetErrorY(Int_t bin);
   Double_t GetCovariance();
   void     SetTitle(const char* title = "");
...
};
```

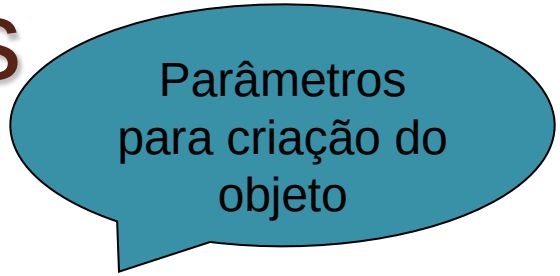
construtores

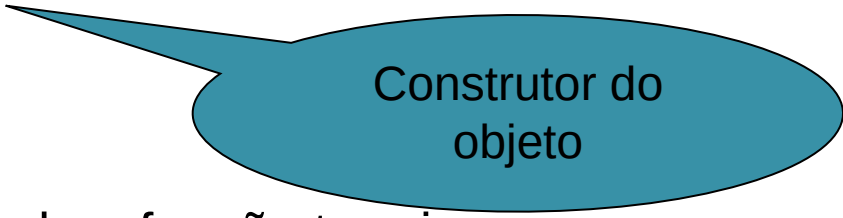
Métodos diversos

Criando objetos

- Criando objetos no stack

```
void exemplo_obj_1()
 {
   TH1F h("hist","histograma",100,0,10);
   h.SetLineColor(1);
   h.Draw();
 }
```





  - O objeto `h` deixa de existir quando a função termina
- Criando objetos no heap (`new` e `delete`)

```
void exemplo_obj_2()
 {
   TH1F* h = new TH1F("hist","histograma",100,0,10);
   h->SetLineColor(1);
   h->Draw();
 }
```

  - Objetos no heap são acessados com ponteiros
  - O objeto `h` só deixa de existir com o `delete h;`

O Básico do ROOT

- Objetos do ambiente
  - gROOT, gStyle, gMinuit, etc.
    - Servem para controlar aspectos globais do ROOT
- Biblioteca de classes
  - ~1200 classes (19 módulos)
    - http://root.cern.ch/root/html/ClassIndex.html
- Documentação ampla na web
  - http://root.cern.ch/drupal/content/documentation

Para começar

- Classe básica para display gráfico
  - TCanvas
    - Um TCanvas é dividido em pads (TPad)
      - Corresponde a uma área na tela gráfica padrão
    - gPad → variável global que aponta para o Pad padrão

```
TCanvas *c = new TCanvas();   // cria uma janela padrão

TCanvas *c = new Tcanvas("nome","titulo", size-x, size-y);

c->Divide(nx,ny); // divide a tela em nx,ny pads

c->cd(2); //vai para o pad numero 2

gPad->SetLogy(); // muda a escala y para log do pad padrão
```

Gráficos e histograms

- O ROOT possui uma quantidade enorme de classes para tratar objetos gráficos
- Gráficos
  - TGraph – Gráficos de X e Y simples
  - TGraphErrors – Gráficos com barras de erro
- Funções
  - TF1 – Função de 1 variável F=F(x)
  - TF2 – Função em 2 variáveis F=F(x,y)
- Histogramas
  - TH1 – Histogramas de 1 dimensão
    - TH1I, TH1S, TH1F, TH1D, ... (estabelece a precisão do eixo)
  - TH2 – Histogramas de 2 dimensões
  - TH3 – Histogramas de 3 dimensões

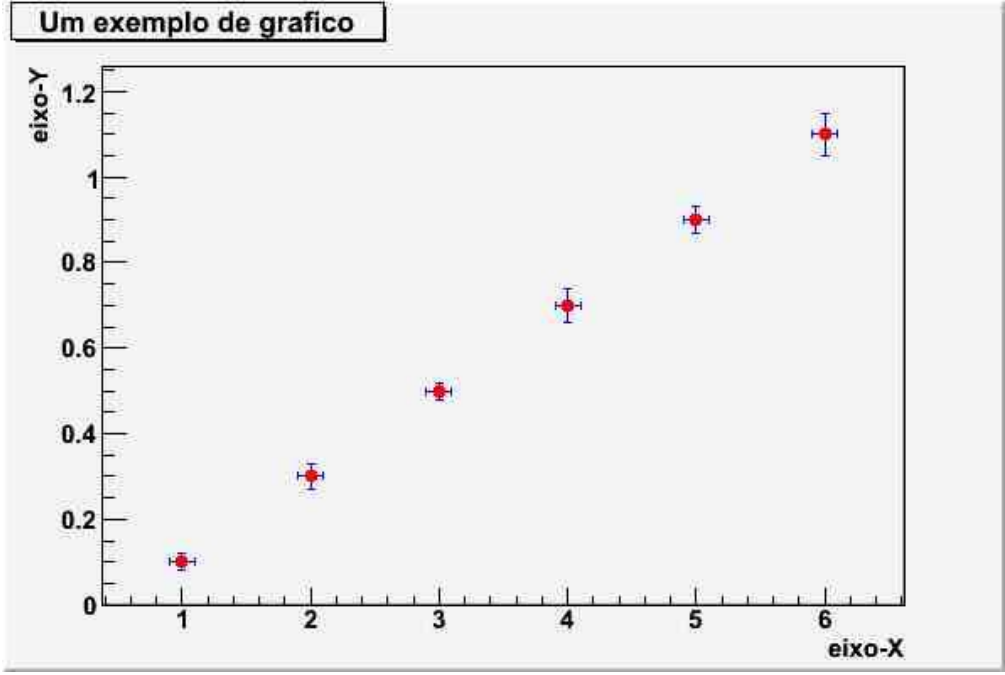
Gráficos x-y

- TGraph e TGraphErrors

```
TGraph *g = new TGraph(N, x, y);

TGraphErrors *g = new TGraphErrors(N, x, y, ex, ey);

TGraphErrors *g = new TGraphErrors("file","formato");

g->Draw("opcoes de desenho");
```

- Onde
  - N = número de pontos
  - x, y são ponteiros para os vetores com os dados
  - ex, ey são ponteiros para os vetores com os erros

Exemplo




```
void exemplo_Tgraph_2()
{
 float x[]  = {1,2,3,4,5,6};
 float y[]  = {0.1,0.3,0.5,0.7,0.9,1.1};
 float ex[] = {0.1,0.1,0.1,0.1,0.1,0.1};
 float ey[] = {0.02,0.03,0.02,0.04,0.03,0.05};
 TGraphErrors *g = new TGraphErrors(6,x,y,ex,ey);
 g->Draw("AP"); // A desenha os eixos, P desenha pontos
 g->SetMarkerStyle(20); // estilo do ponto 20 = circulo
 g->SetMarkerColor(2);
 g->SetLineColor(4);
 g->SetTitle("Um exemplo de grafico");
 g->GetXaxis()->SetTitle("eixo-X");
 g->GetYaxis()->SetTitle("eixo-Y");
 return;
}
```

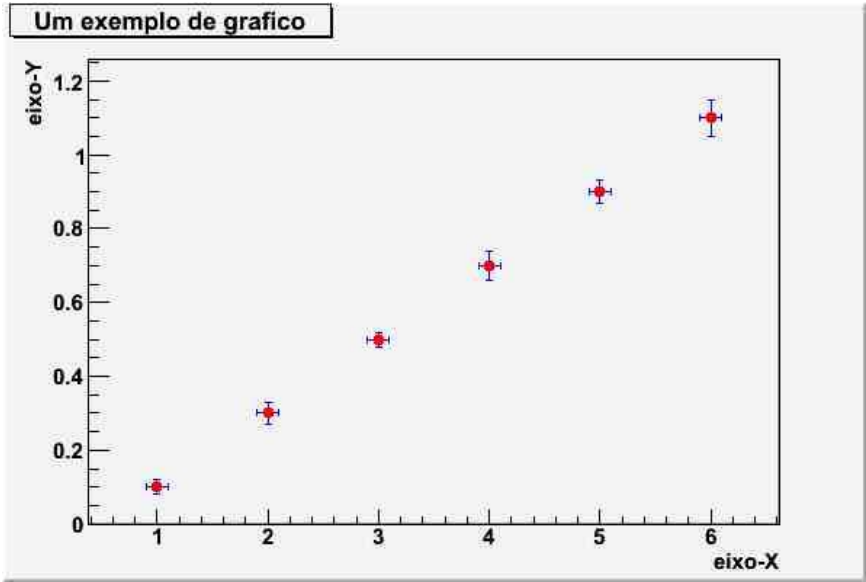
Exemplo

dados.txt (modified) –

File Edit Search
Preferences Shell Macro
Windows Help

```
1  0.1  0.1  0.02
2  0.3  0.1  0.03
3  0.5  0.1  0.02
4  0.7  0.1  0.04
5  0.9  0.1  0.03
6  1.1  0.1  0.05
```

```
void exemplo_TGraph_3()
{
 TGraphErrors *g = new TGraphErrors("dados.txt", "%lg %lg %lg %lg");
 g->Draw("AP"); // A desenha os eixos, P desenha pontos
 g->SetMarkerStyle(20); // estilo do ponto 20 = circulo
 g->SetMarkerColor(2);
 g->SetLineColor(4);
 g->SetTitle("Um exemplo de grafico");
 g->GetXaxis()->SetTitle("eixo-X");
 g->GetYaxis()->SetTitle("eixo-Y");
 return;
}
```



Histogramas

- O ROOT possui uma quantidade enorme de classes para tratar objetos gráficos
- Histogramas
  - TH1 – Histogramas de 1 dimensão
    - TH1I, TH1S, TH1F, TH1D, ... (estabelece a precisão do eixo)
  - TH2 – Histogramas de 2 dimensões
  - TH3 – Histogramas de 3 dimensões

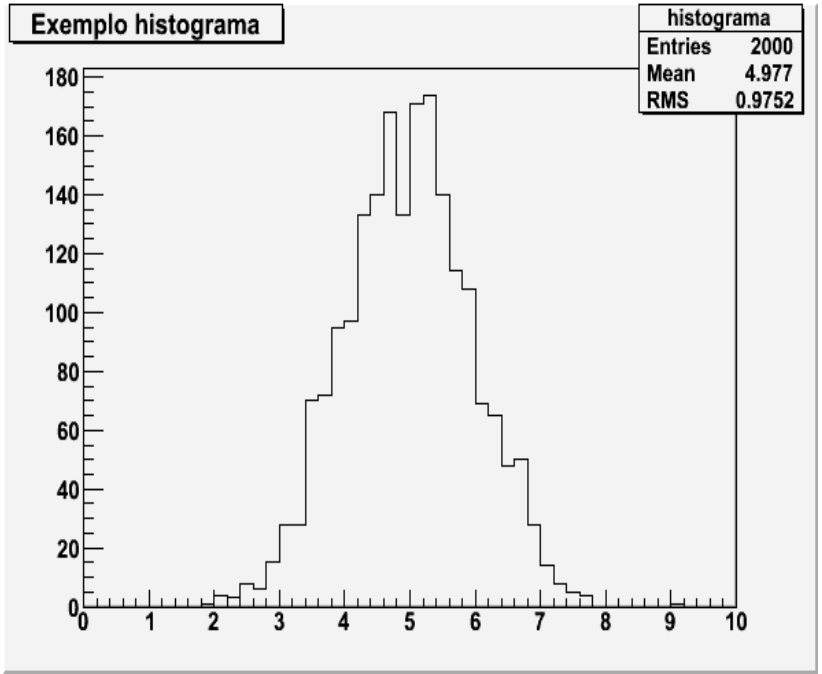
Monte Carlo no ROOT

- Sorteio de números aleatórios simples
  - TRandom, TRandom1, TRandom2, TRandom3
    - A única diferença é o algorítimo de geração
  - Geradores básicos
    - Exp(tau)
    - Integer(imax)
    - Gaus(mean,sigma)
    - Rndm()
    - Uniform(x1)
    - Landau(mpv,sigma)
    - Poisson(mean)
    - Binomial(ntot,prob)
- Ou usar funções (TF1,etc.) para outras distribuições de probabilidade

Histogramas de 1 D

- Criando um Histograma de 1 dimensão
  - TH1F *h = new TH1F(“nome”,”título”, Nbins, Xmin, Xmax);
  - TH1F h (“nome”,”título”, Nbins, Xmin, Xmax);

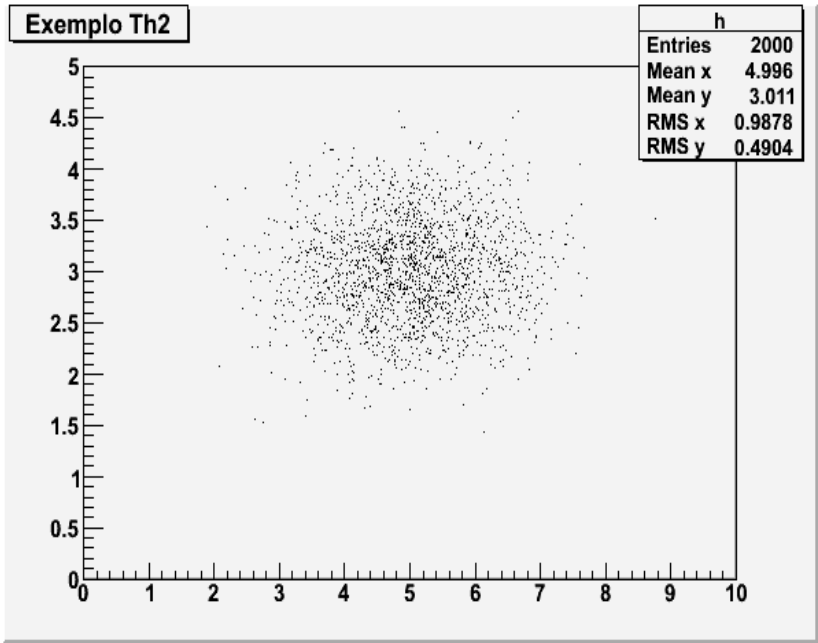
```
void exemplo_TH1()
{
  TRandom *r = new TRandom();
  TH1F *h1 = new TH1F("histograma","Exemplo histograma",50,0,10);
  for(int i = 0;i<2000;i++)
  {
    float x = r->Gaus(5,1);
    h1->Fill(x);
  }
  h1->Draw();
}
```



Histogramas de 2 D

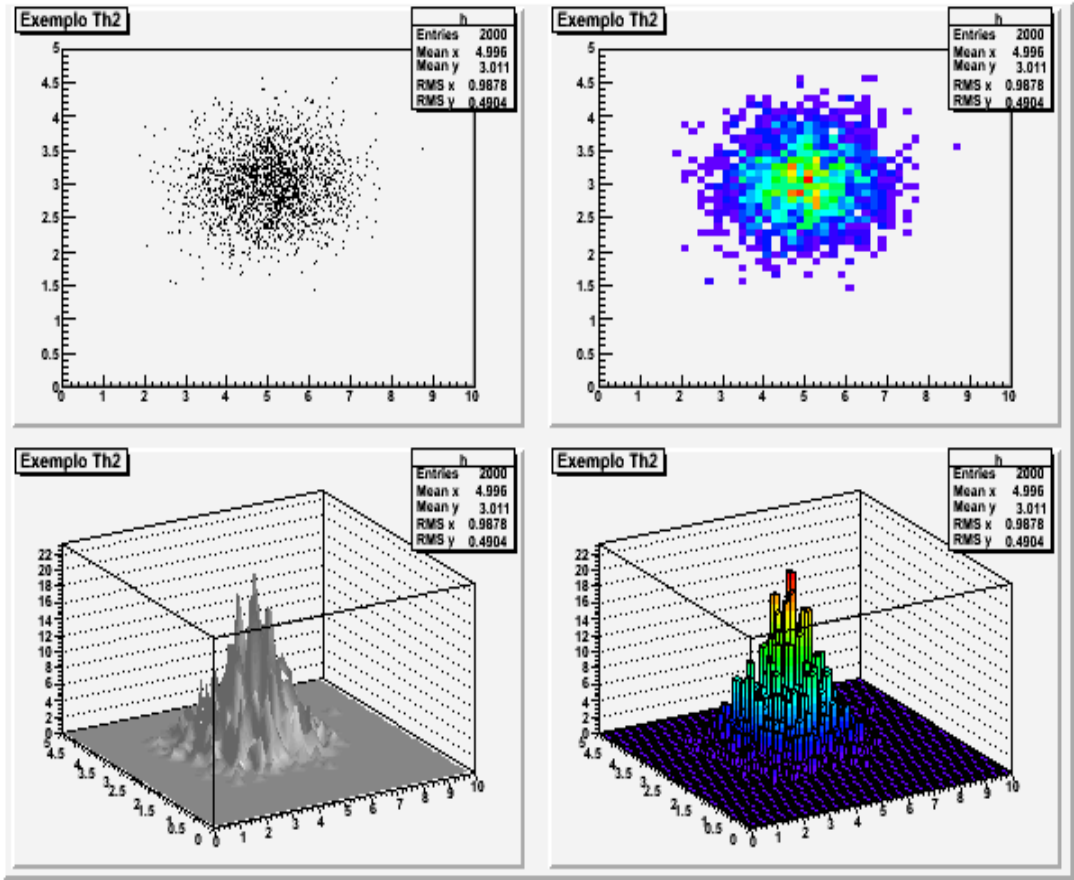
- Muito similar ao TH1
  - TH2F *h = new TH2F(“nome”,”título”, NbinsX, Xmin, Xmax, NBinsY, Ymin, Ymax);
  - TH2F h (“nome”,”título”, NbinsX, Xmin, Xmax, NbinsY, Ymin,Ymax);

```
void exemplo_TH2()
{
  TRandom *r = new TRandom();
  TH2F *h2 = new TH2F("h","Exemplo Th2",50,0,10,50,0,5);
  for(int i = 0;i<2000;i++)
  {
    float x = r->Gaus(5,1);
    float y = r->Gaus(3,0.5);
    h2->Fill(x,y);
  }
  h2->Draw();
}
```



Dividindo uma tela

```
void exemplo_TH2_2()
 {

      gStyle->SetPalette(1,0);
      TRandom *r = new TRandom();
      TH2F *h2 = new TH2F("h","Exemplo Th2",50,0,10,50,0,5);
      for(int i = 0;i<2000;i++)
      {
        float x = r->Gaus(5,1);
        float y = r->Gaus(3,0.5);
        h2->Fill(x,y);
      }
      TCanvas *c = new TCanvas();
      c->Divide(2,2);
      c->cd(1);
      h2->Draw();
      c->cd(2);
      h2->Draw("col");
      c->cd(3);
      h2->Draw("surf4");
      c->cd(4);
      h2->Draw("lego2");

 }
```



Funções

- O ROOT possui classes para definir funções.
  - TF1, TF2 e TF3
- Uso

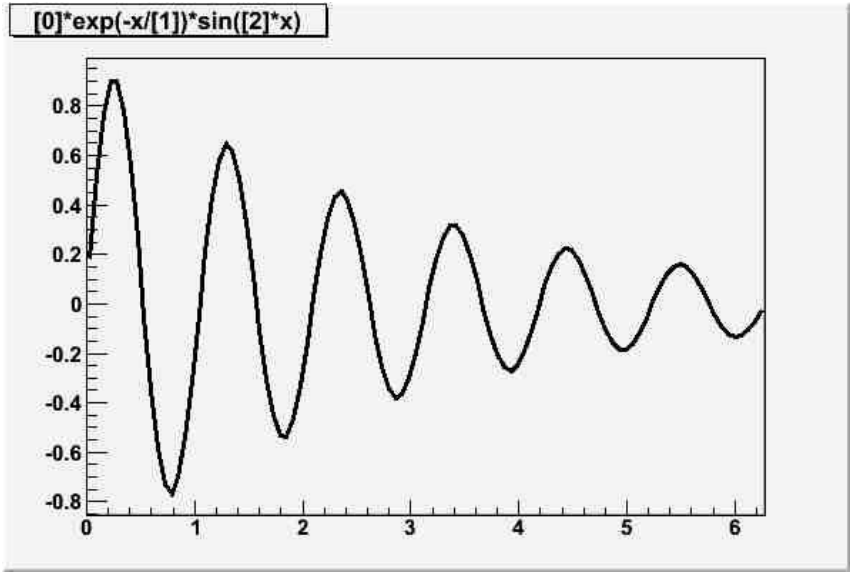
```
TF1 *f = new TF1(“nome”,”formula”,xmin, xmax);
```

- A fórmula deve ser escrita usando a sintaxe padrão de c++.
  - Parâmetros variáveis devem vir entre brackets
    - [0], [1], etc
  - As variáveis são x, y e z
- Alguns métodos interessantes
  - SetParameter(), GetParameter(), GetParError(), GetChisquare(), Eval(), etc.

Exemplo

O fim de um comando só ocorre quando se coloca o ;

```
void exemplo_func()
{
   TF1 *f1 = new TF1("func",
                     "[0]*exp(-x/[1])*sin([2]*x)",
                     0,6.28);
   f1->SetParameter(0,1);
   f1->SetParameter(1,3);
   f1->SetParameter(2,6);
   f1->Draw();
}
```



TLegend

- A classe TLegend cria legenda de gráficos

```
TLegend *L = new TLegend(xi,yi,xf,yf);

L->AddEntry(objeto, “titulo”, “opcoes”);
```

  - Onde xi, yi, xf, yf são as coordenadas na tela.
    - Números reais (float) entre 0 e 1
  - Ex:
    - TLegend *L = new TLegend(0.25, 0.30, 0.70, 0.50);
- Adicionando ítem na legenda

“Hands on” com o ROOT

Algumas atividades práticas para exercitar e aprender ROOT

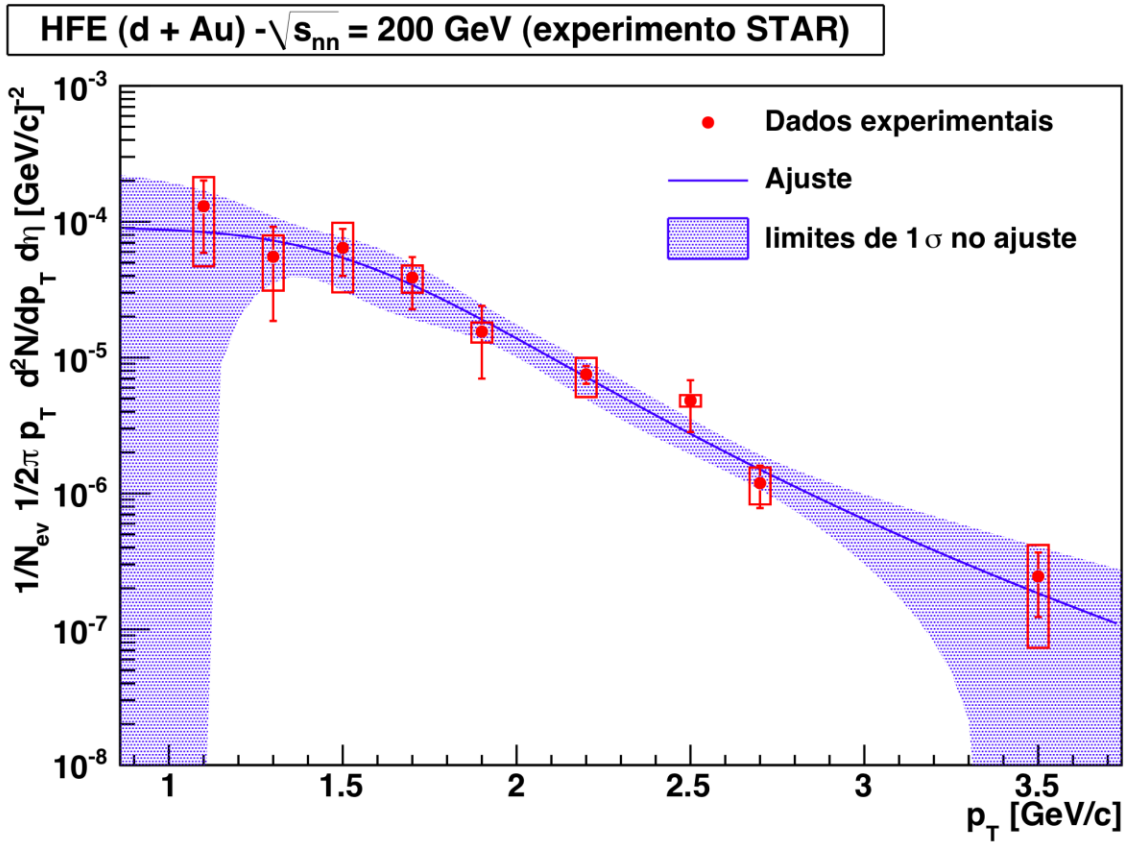
Vários outros tutoriais em:

http://root.cern.ch/root/html/tutorials/

http://root.cern.ch/drupal/content/howtos

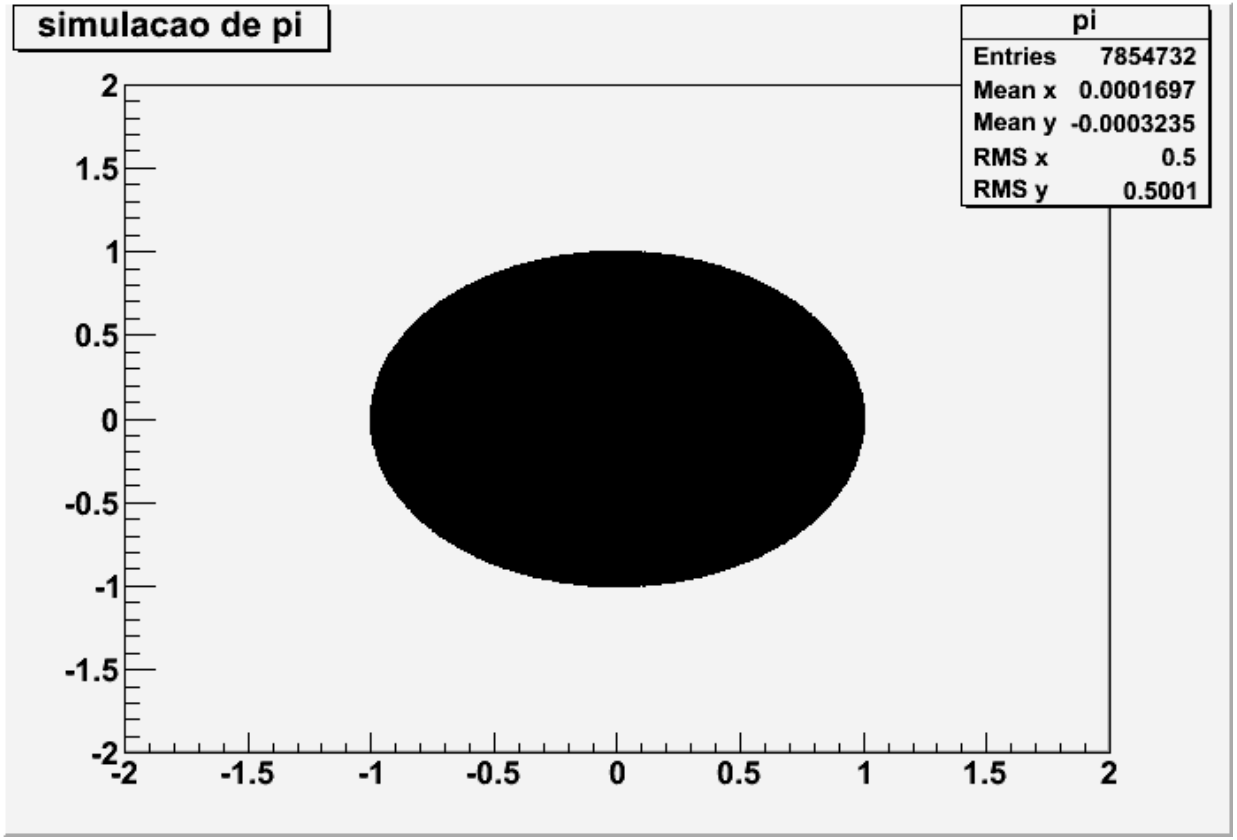
Ex01.C

- Gráficos
  - Incertezas estatísticas e sistemáticas
- Legendas
- Ajuste de funções
  - Incerteza de um ajuste
- Manipulação simples de dados



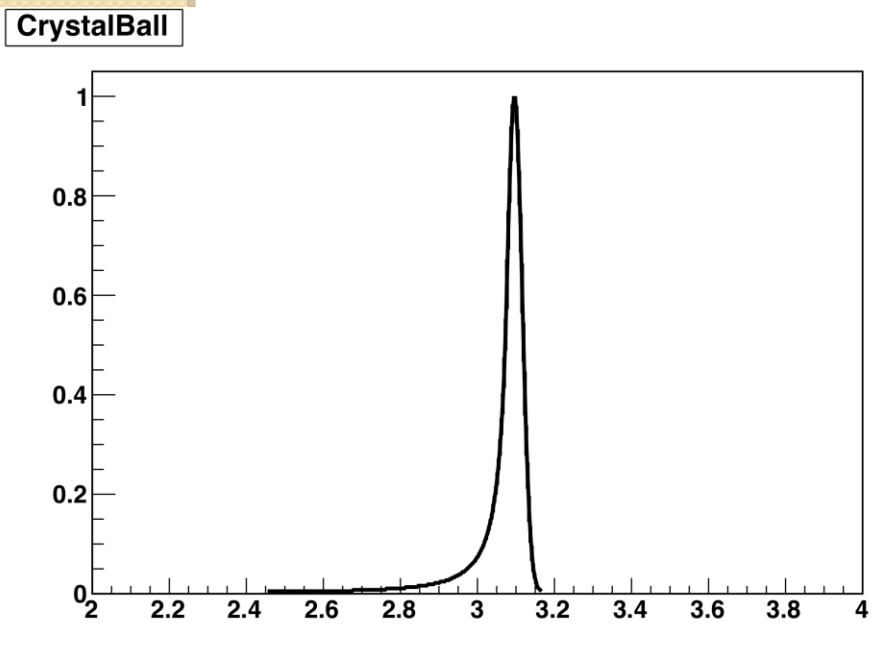
Ex02.C

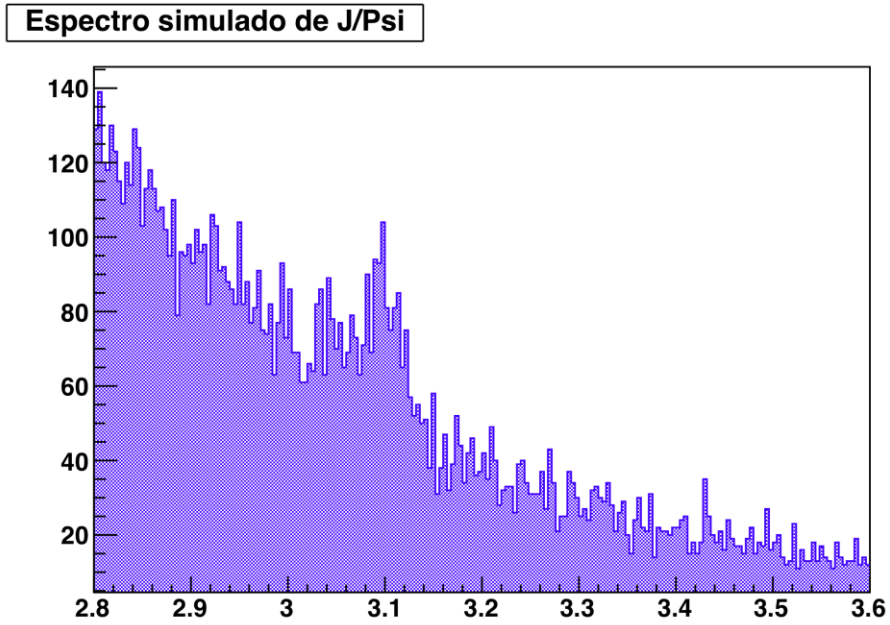
- Monte Carlo simples
- Histogramas
  - Obtendo informações de histogramas



Ex03.C

- Funções mais complexas
- Monte Carlo a partir de funções
  - Gerador simples de eventos
- I/O no ROOT

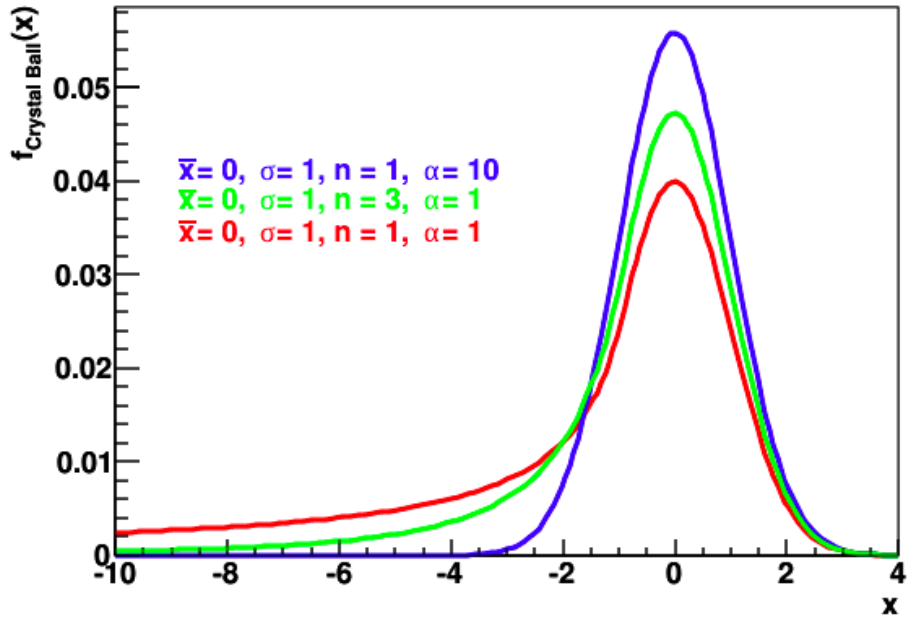




Função Crystal ball

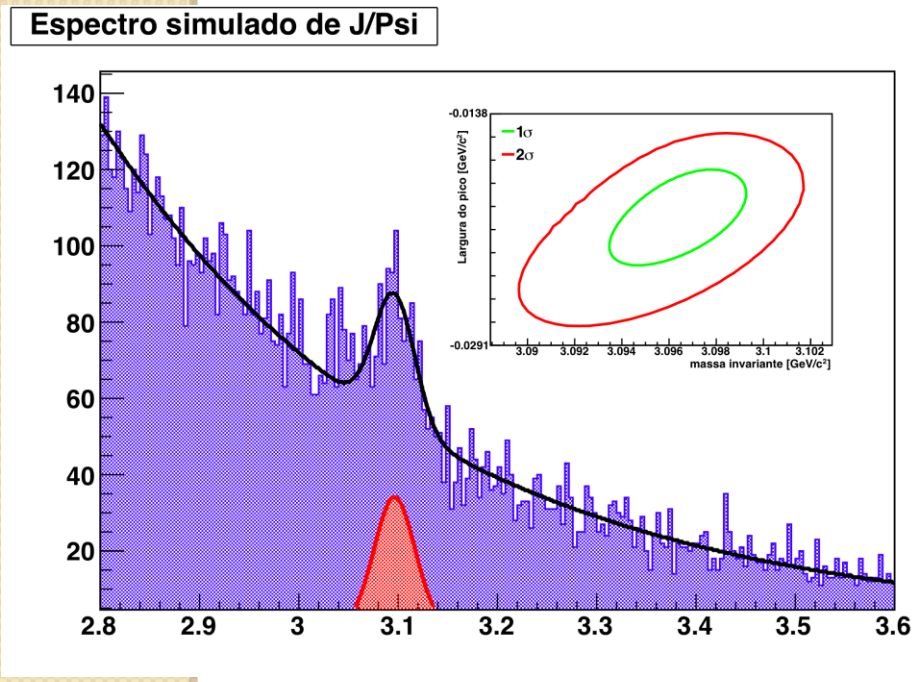
$$f(x;\alpha,n,\bar{x},\sigma) = N \cdot \begin{cases} \exp(-\frac{(x-\bar{x})^2}{2\sigma^2}), & \text{for } \frac{x-\bar{x}}{\sigma} > -\alpha \\ A \cdot (B - \frac{x-\bar{x}}{\sigma})^{-n}, & \text{for } \frac{x-\bar{x}}{\sigma} \leqslant -\alpha \end{cases}$$

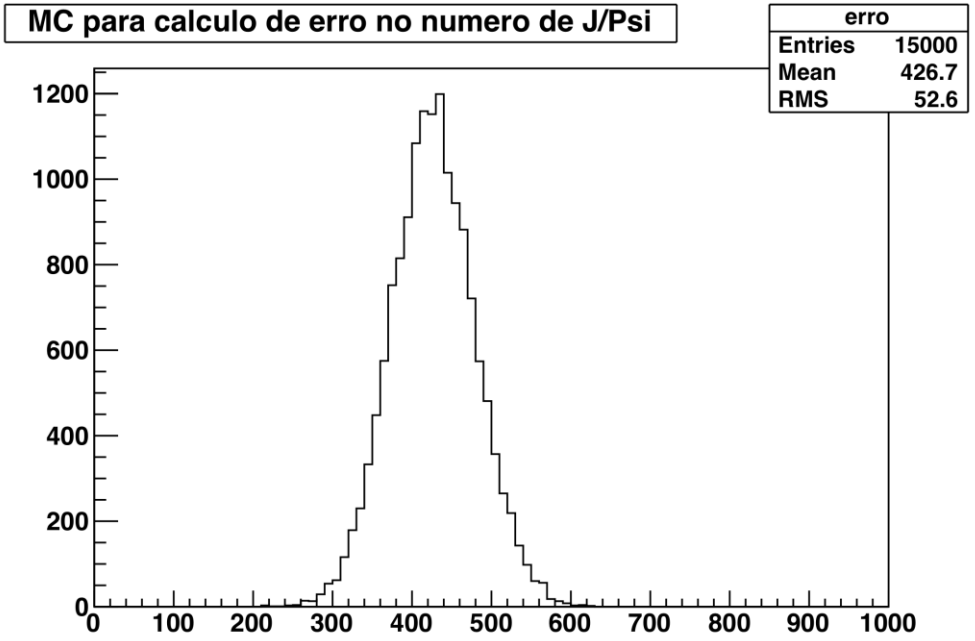
$$A = \left(\frac{n}{|\alpha|}\right)^n \cdot \exp\left(-\frac{|\alpha|^2}{2}\right) \qquad B = \frac{n}{|\alpha|} - |\alpha|$$



Ex04.C

- I/O no ROOT
- Ajustes complexos
  - Curvas de intervalo de confiança
- Desmembrando uma função
  - Integral de uma função
- Incerteza com Monte Carlo e matriz de covariância





Sorteando parâmetros com matriz de covariância

- Cholesky matrix

$$COV = LL^{*}$$

- Sorteio de parâmetros

$$\begin{pmatrix} p_1 \\ p_2 \\ \mathrm{M} \\ p_n \end{pmatrix} = L \begin{pmatrix} Gaus(0,1) \\ Gaus(0,1) \\ \mathrm{M} \\ Gaus(0,1) \end{pmatrix} + \begin{pmatrix} p_1^{fit} \\ p_2^{fit} \\ \mathrm{M} \\ p_n^{fit} \end{pmatrix}$$

Ex05.C

- I/O no ROOT
- Ajustes complexos
  - Ajustando histogramas a histogramas
- Algumas ferramentas de funções
  - Chi2, intervalo de ajuste, etc.
- Dividindo uma tela