RooFit

Aula 04

Parte 02

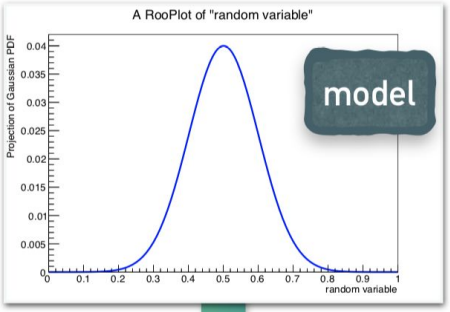
Expressões genéricas de PDFs

Se a sua PDF favorita não estiver lá :

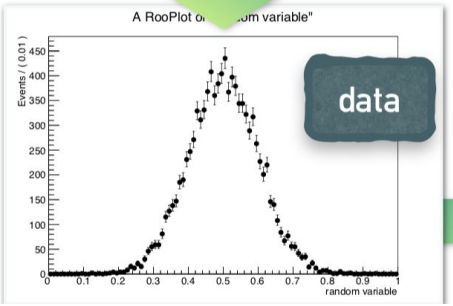
Use a classe **RooGenericPdf.**

```
// PDF variables
RooRealVar x("x","x",-10.,10.);
RooRealVar y("y","y",0,5);
RooRealVar a("a","a",3.0);
RooRealVar b("b","b",-2.0);
// Generic PDF
RooGenericPdf model("model","GenericPDF","exp(x*y+a)-b",RooArgSet(x,y,a,b));
```

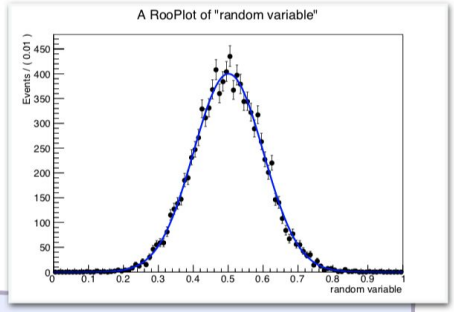
Geração e Ajuste




```
model.generate(x,10000);
```




```
RooPlot* fm = x.frame();
data->plotOn(fm);
model.plotOn(fm);
fm->Draw();
```




```
COVARIANCE MATRIX CALCULATED SUCCESSFULLY
FCN=-8863.01 FROM HESSE STATUS=OK 10 CALLS 34 TOTAL
EDM=1.57332e-06  STRATEGY= 1  ERROR MATRIX ACCURATE
EXT PARAMETER
NO.   NAME      VALUE            ERROR
 1  mu           5.00665e-01   9.97372e-04
 2  sigma        9.97366e-02   7.05314e-04
```

```
RooFitResult *res = model.fitTo(*data, ...);
```

Essa classe armazena os resultados de um ajuste de modelo, incluindo os parâmetros ajustados, suas incertezas, a matriz de covariância, entre outros.

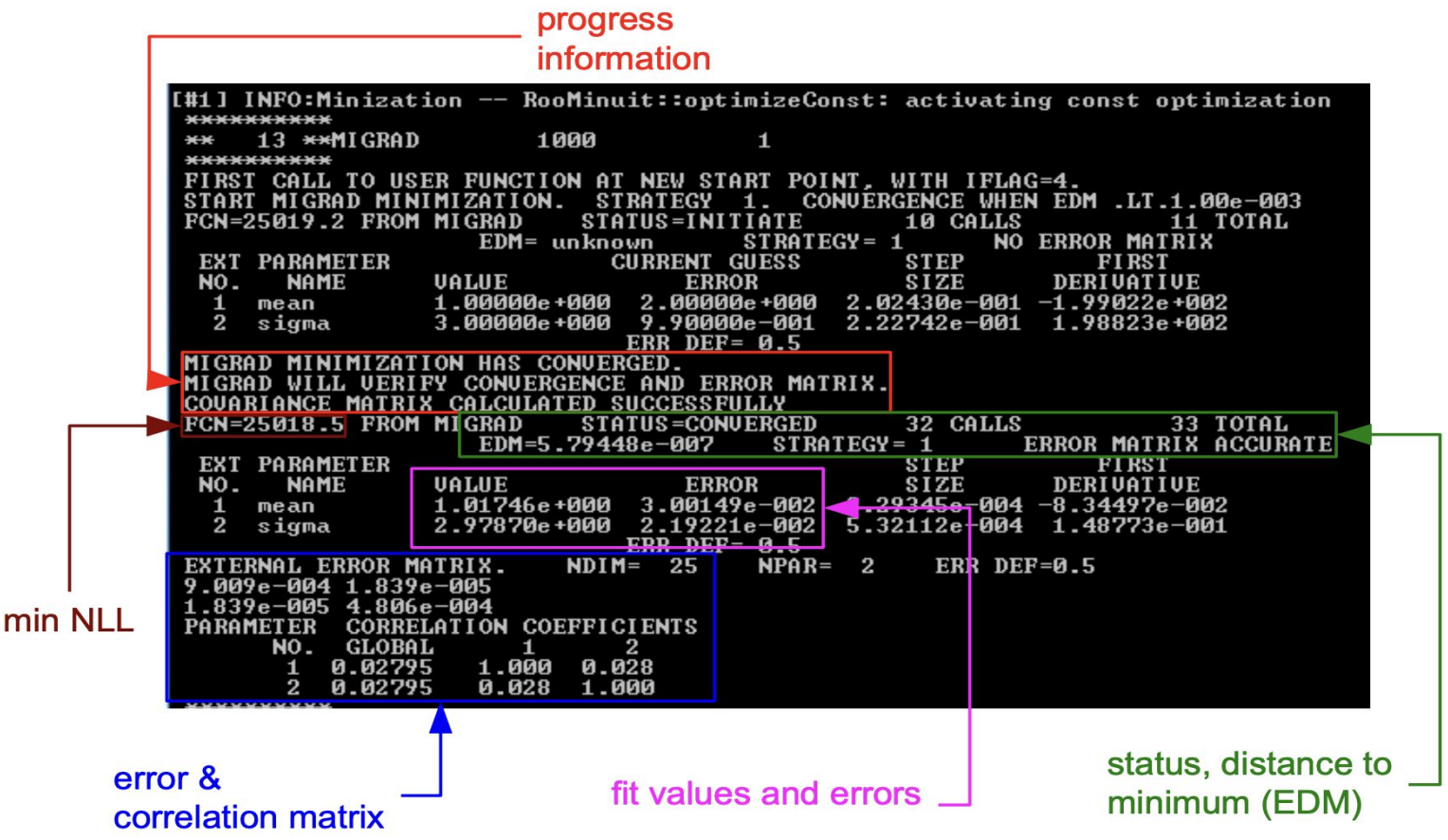
Geração e Ajuste

Algumas opções úteis podem ser adicionadas para “gerar” e “ajustar” à parte “...”:

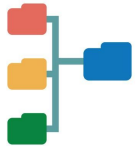
```
RooFitResult* res = model.fitTo(*data, ...);
```

Save( )	Se o RooFitResult é produzido . RooFit::Save().
Extended( )	Adiciona o termo de likelihood estendida. N_obs.
Verbose( )	Informações adicionais são impressas sobre como o cálculo de probabilidade é configurado.
NumCPU(N)	Paralelizar o cálculo de probabilidade sobre N processos.

Exemplo da saída do ajuste



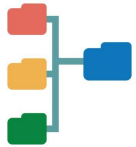
RooFit Workspace



- A classe **`RooWorkspace`**: container para todos os objetos do RooFit que foram criados:
  - –configuração completa do modelo
    - descrição dos parâmetros/observáveis e PDF
    - incertezas
  - –(múltiplos) conjuntos de dados
- Mantém uma completa descrição de todo o modelo
  - –possibilidade de salvar o modelo por completo em um ROOT file
  - –toda informação estará disponível para uma análise aprofundada
- Combinação dos resultados juntando workspaces em um só
  - –formato padrão para combinar e compartilhar resultados de física

```
RooWorkspace workspace("w");
workspace.import(*data); workspace.import(*pdf);
workspace.writeToFile("myWorkspace.root")
```

Usando o workspace



- Workspace
  - Um classe de contêiner genérica para todos os objetos RooFit do seu projeto
  - Ajuda a organizar projetos de análise

- Criação de um workspace

```
RooWorkspace w("w");
```

- Colocando as variáveis e funções em um workspace
  - Ao importar uma função, todas as suas componentes(variáveis) também são importadas automaticamente

```
RooWorkspace w("w");//criando o workspace
RooRealVar x("x","x",-10,10);
RooRealVar mean("mean","mean",5);
RooRealVar sigma("sigma","sigma",3);
RooGaussian f("f","f",x,mean,sigma);
w.import(f); // importando f,x,mean e sigma
```

Usando o workspace

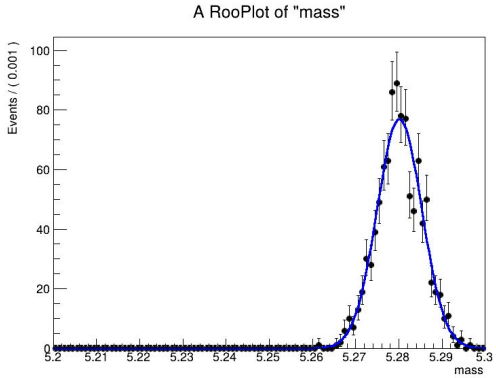
- Dentro de um workspace

```
w.Print() ;
variables
---------
(mean,sigma,x)
p.d.f.s
-------
RooGaussian::f[ x=x mean=mean sigma=sigma ] = 0.249352
```

- Acessando variáveis e funções fora de um workspace

```
RooPlot* frame = w.var("x")->frame();
w.pdf("f")->plotOn(frame);
```

```
#include "TH1.h"
#include "RooRealVar.h"
#include "RooDataSet.h"
#include "RooArgSet.h"
#include "RooPlot.h"
#include "RooFit.h"
#include "TCanvas.h"
#include <iostream>
using namespace RooFit ;

void exWorkspace()
{
  //pdf
  RooRealVar mass("mass","mass",5.20,5.30) ;
  RooRealVar mean("mean","mean",5.28,5.20,5.30) ;
  RooRealVar sigma("sigma","sigma",0.005,0.001,0.05) ;
  RooGaussian model("model","model",mass,mean,sigma) ;

  // Importar a  pdf no  workspace
  RooWorkspace w("w") ;
  w.import(model) ;

  // Gerar dados
  RooDataSet* dado = w.pdf("model")->generate(*w.var("mass"),1000) ;

  // Executar o ajuste aos dados
  w.pdf("model")->fitTo(*dado) ;

  RooPlot* frame = w.var("mass")->frame() ;

  TCanvas *c = new TCanvas("c", "c", 800, 600);
  dado->plotOn(frame) ;
  w.pdf("model")->plotOn(frame) ;
  frame->Draw() ;
  c->Draw();
  c->SaveAs("w.png");
  w.writeToFile("wspacecpp.root");
}
```



Usando o workspace

- O Workspace pode ser gravado em um arquivo com todo o seu conteúdo
  - –Escrever o workspace e o conteúdo no arquivo:

```
w.writeToFile("wspacecpp.root");
```

```
eliza@93f99f4d0fb7:~/Aula_IntroducaoAnalaiseDados_2024_02/testes/RooFit$ root -l
root [0] TFile *f = TFile::Open("wspacecpp.root");
root [1] .ls
TFile**         wspacecpp.root
 TFile*         wspacecpp.root
  KEY: RooWorkspace     w;1     w
  KEY: TProcessID       ProcessID0;1    7ba93ac6-818f-11ef-a482-050013acbeef
root [2] f->Get("w");
root [3] w->Print();

RooWorkspace(w) w contents

variables
---------
(mass,mean,sigma)

p.d.f.s
-------
RooGaussian::model[ x=mass mean=mean sigma=sigma ] = 4.83288e-08

root [4]
```

RooFit Factory - gerador de objetos

```
RooRealVar x(“x”,”x”,2,-10,10)
RooRealVar s(“s”,”s”,3) ;
RooRealVar m(“m”,”m”,0) ;
RooGaussian g(“g”,”g”,x,m,s)
```

4 linhas

Fornece uma “fábrica(*factory*)” para geração automática de objetos de uma linguagem semelhante à matemática.

```
RooWorkspace w;
w.factory(“Gaussian::g(x[2,-10,10],m[0],s[3])”)
```

1 linha

Agora, você pode usar o workspace factory para construir modelos.

Factoring Sintaxe

- Regra #1 – Crie uma variável (w.factory("x[0, 10]");)

```
x[-10,10] // Create variable with given range
x[5,-10,10] // Create variable with initial value and range
x[5] // Created initially constant variable
```

- Regra #2 – Crie uma função ou o objeto pdf

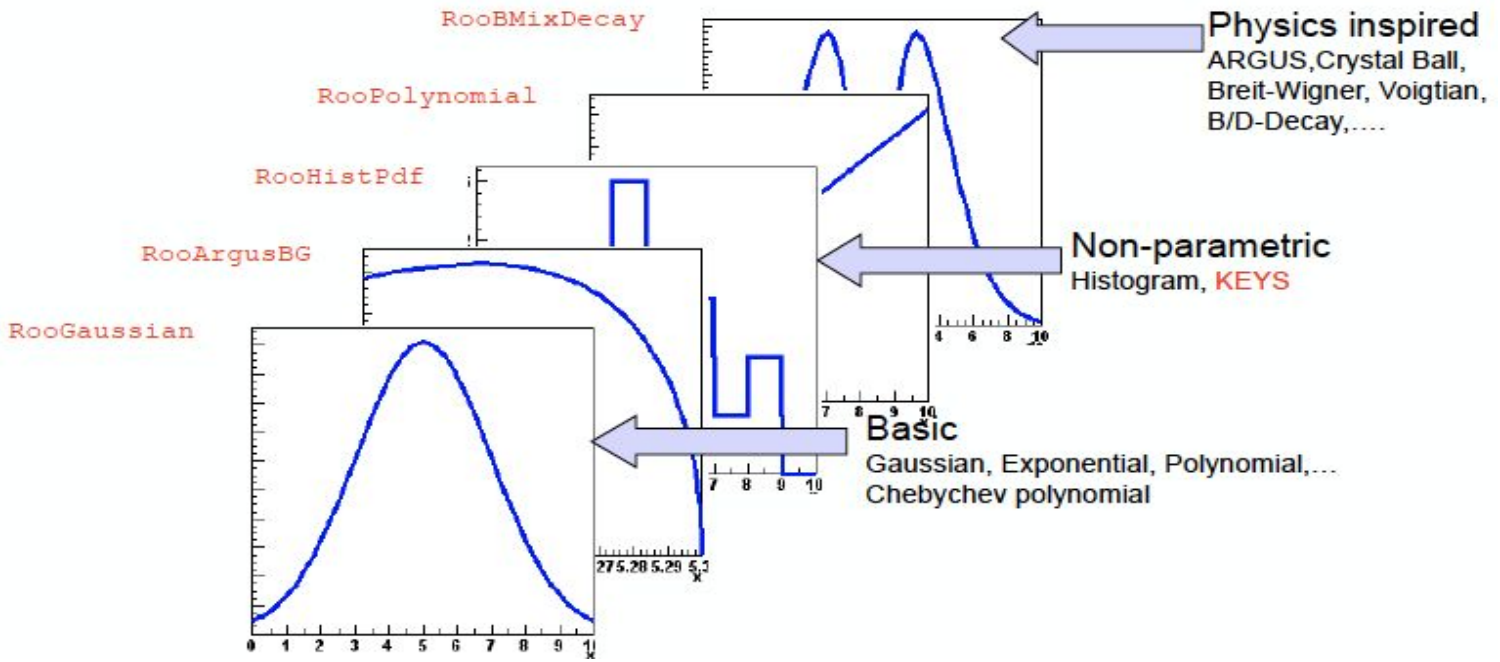
```
ClassName::Objectname(arg1,[arg2],...)
```

  - O 'Roo' no nome da classe pode ser omitido
  - Argumentos são nomes de objetos que já existem em um workspace
  - Os objetos nomeados devem ser do tipo correto, se não a factory imprime erro
  - Os argumentos definidos e listados podem ser construídos entre chaves {}

```
Gaussian::g(x,mean,sigma)
// equivalent to RooGaussian("g","g",x,mean,sigma)
Polynomial::p(x,{a0,a1})
// equivalent to RooPolynomial("p","p",x",RooArgList(a0,a1));
```

Construindo Modelos

- RooFit fornece uma coleção de classes de PDF



É fácil estender a biblioteca: cada p.d.f. é uma classe C++ separada

(Re)usando componentes padrões

```
Gaussian::g(x,mean,sigma)
BreitWigner::bw(x,mean,gamma)
Landau::l(x,mean,sigma)
Exponential::e(x,alpha)
Polynomial::p(x,{a0,a1,a2})
Chebychev::p(x,{a0,a1,a2})
KeysPdf::k(x,dataSet)
Poisson::p(x,mu)
Voigtian::v(x,mean,gamma,sigma)
```

Factory - usando expressões

- PDF customizada a partir de expressões interpretadas

```
w.factory("EXPR::mypdf('sqrt(a*x)+b',x,a,b)");
```

- re-parametrização de variáveis (fazendo funções)

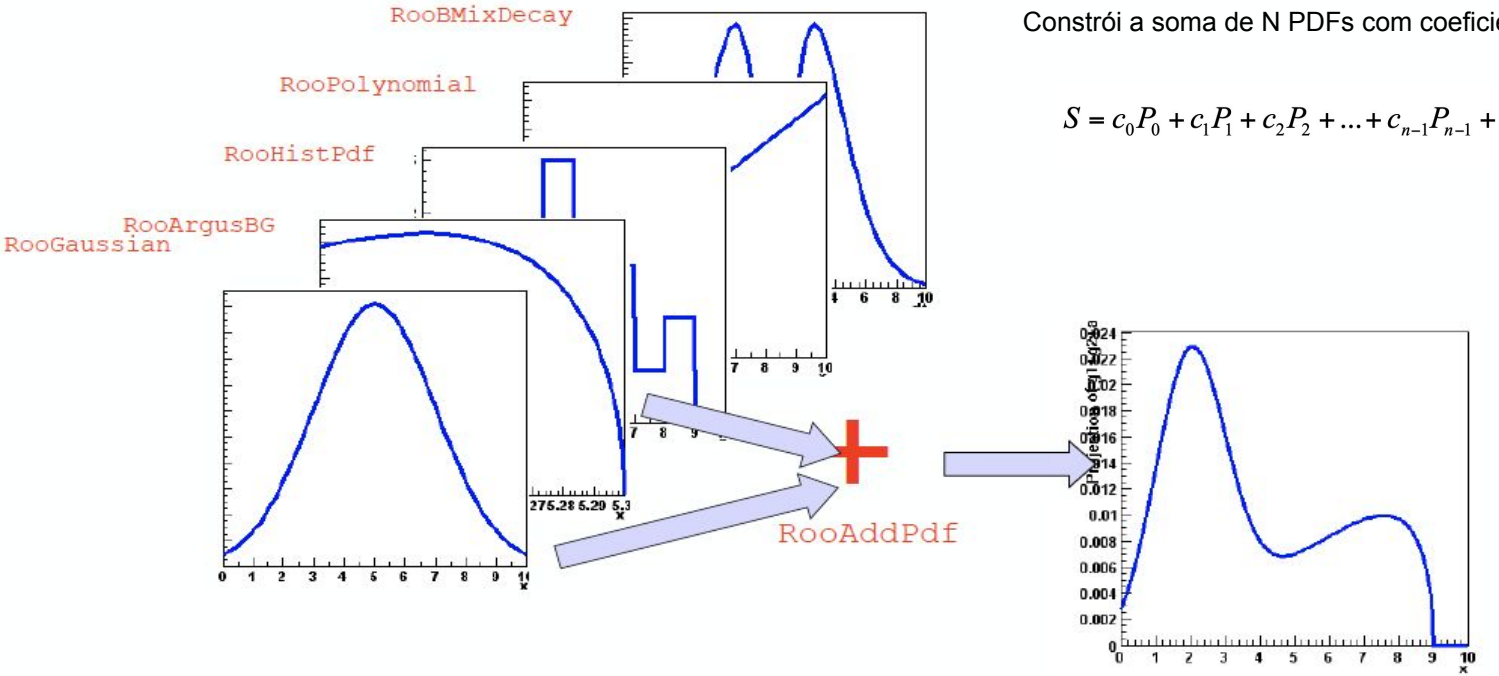
```
w.factory("expr::w('(1-D)/2',D[0,1])");
```

– nota: usando **expr** (cria-se uma função, uma RooAbsReal)
usando **EXPR** (cria-se uma PDF, uma RooAbsPdf)

O uso de maiúscula e minúscula também se aplica a outros comandos da factory (SUM, PROD,....)

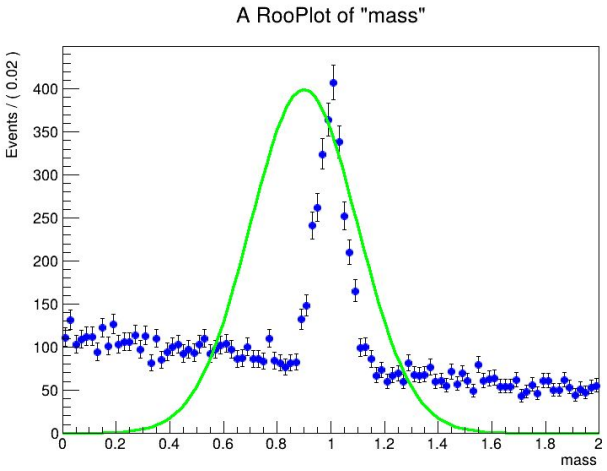
Construindo Modelo - usando componentes padrões

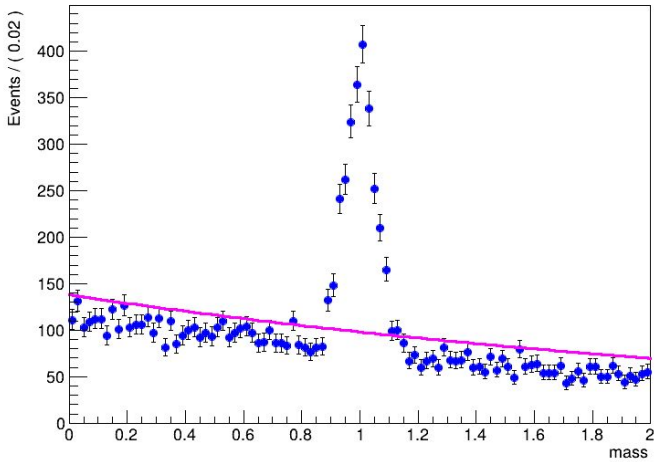
- Os modelos mais realistas são construídos como a soma de uma ou mais p.d.f.s (ex.: sinal e fundo (*background*))
- Facilitado por meio de classes `operador p.d.f RooAddPdf`

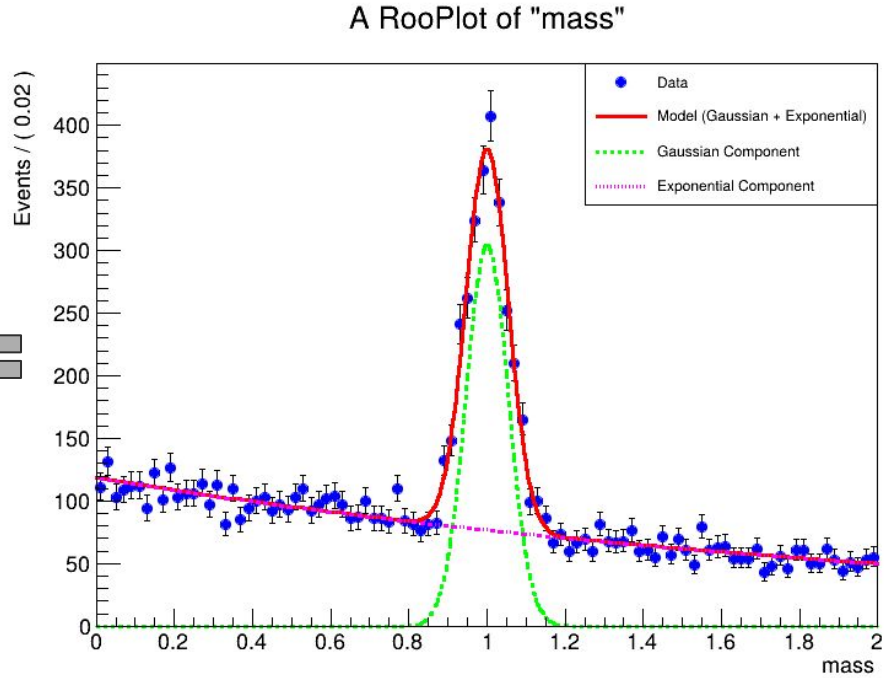


Constrói a soma de N PDFs com coeficientes N-1:

$$S = c_0 P_0 + c_1 P_1 + c_2 P_2 + \ldots + c_{n-1} P_{n-1} + \left(1 - \sum_{i=0,n-1} c_i\right) P_n$$


A RooPlot of "mass"
Events / (0.02)
mass


A RooPlot of "mass"
Events / (0.02)
mass

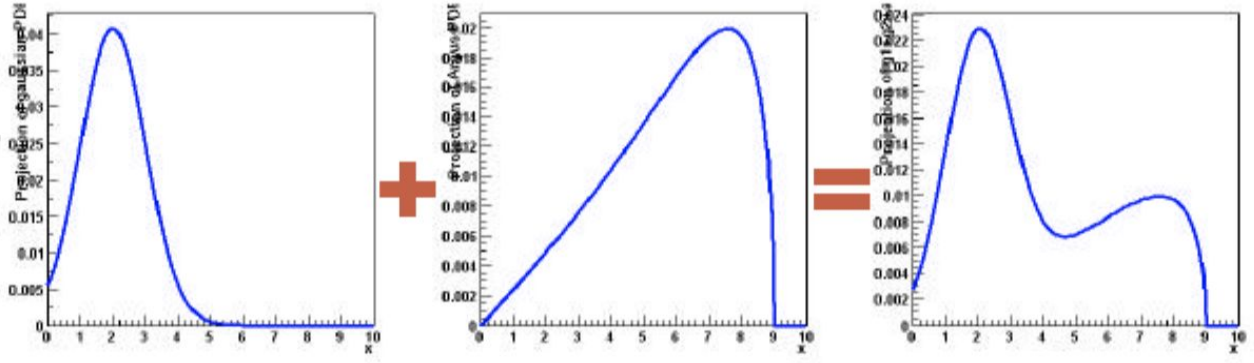

A RooPlot of "mass"
Data
Model (Gaussian + Exponential)
Gaussian Component
Exponential Component
Events / (0.02)
mass

```
    // Definir a função gaussiana (sinal)
    RooRealVar mean("mean", "Mean of Gaussian", 1.0, 0.9, 1.1);
    RooRealVar sigma("sigma", "Width of Gaussian", 0.1, 0.01, 0.2);
    RooGaussian gauss("gauss", "Gaussian PDF", x, mean, sigma);
    // Definir os parâmetros da exponencial (fundo)
    RooRealVar expParam("expParam", "Exponential decay", -1.0, -5.0, 0.0);
    RooExponential expo("expo", "Exponential PDF", x, expParam);

    // Ajustar a função aos dados
    //gauss.fitTo(dataUnbinned);
    //expo.fitTo(dataUnbinned);

    // Combinar os dois modelos
    RooRealVar frac("frac", "Fraction of Gaussian", 0.5, 0.0, 1.0);
    RooAddPdf model("model", "gauss+expo", RooArgList(gauss, expo), RooArgList(frac));
    model.fitTo(dataUnbinned);

    TCanvas *c = new TCanvas("c", "Imported Data", 800, 600);

    RooPlot* frame = x.frame();

    dataUnbinned.plotOn(frame, RooFit::MarkerColor(kBlue), RooFit::Name("Unbinned"));
    //Plotar a Gaussiana+expo ajustada no mesmo frame
    //gauss.plotOn(frame, RooFit::LineColor(kGreen), RooFit::Name("Gaussian Fit"));
    //expo.plotOn(frame, RooFit::LineColor(kMagenta), RooFit::Name("Expo Fit"));
    model.plotOn(frame, RooFit::LineColor(kRed), RooFit::Name("Model Fit"));
    model.plotOn(frame, RooFit::Components("gauss"), RooFit::LineStyle(kDashed),
RooFit::LineColor(kGreen), RooFit::Name("Gaussian"));
    model.plotOn(frame, RooFit::Components("expo"), RooFit::LineStyle(kDotted),
RooFit::LineColor(kMagenta), RooFit::Name("Exponential"));
    frame->Print();
    frame->Draw();
    // Adicionar legenda
    TLegend *legend = new TLegend(0.6, 0.7, 0.9, 0.9);  // Posição da legenda
    legend->AddEntry(frame->findObject("Unbinned"), "Data", "p");
    legend->AddEntry(frame->findObject("Model Fit"), "Model (Gaussian + Exponential)", "l");
    legend->AddEntry(frame->findObject("Gaussian"), "Gaussian Component", "l");
```

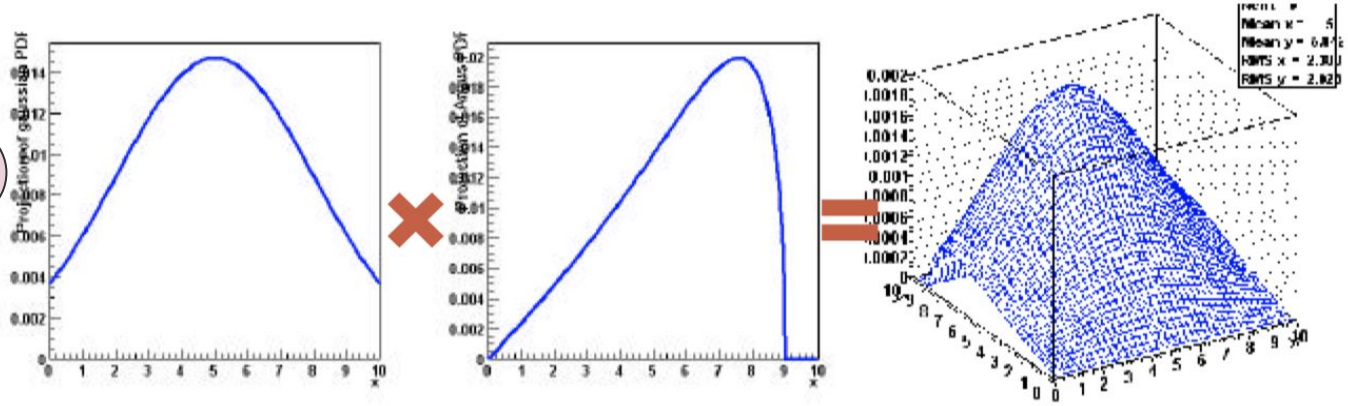
```
    // Garantir que a legenda é desenhada
    legend->SetBorderSize(1);  // Borda visível para verificar a presença da legenda
    legend->Draw();

    // Atualizar o canvas para garantir que a legenda seja mostrada
    c->Update();

    //c->Draw();

    c->SaveAs("ImportarDadosFitModel.png");

    file->Close();
}
```

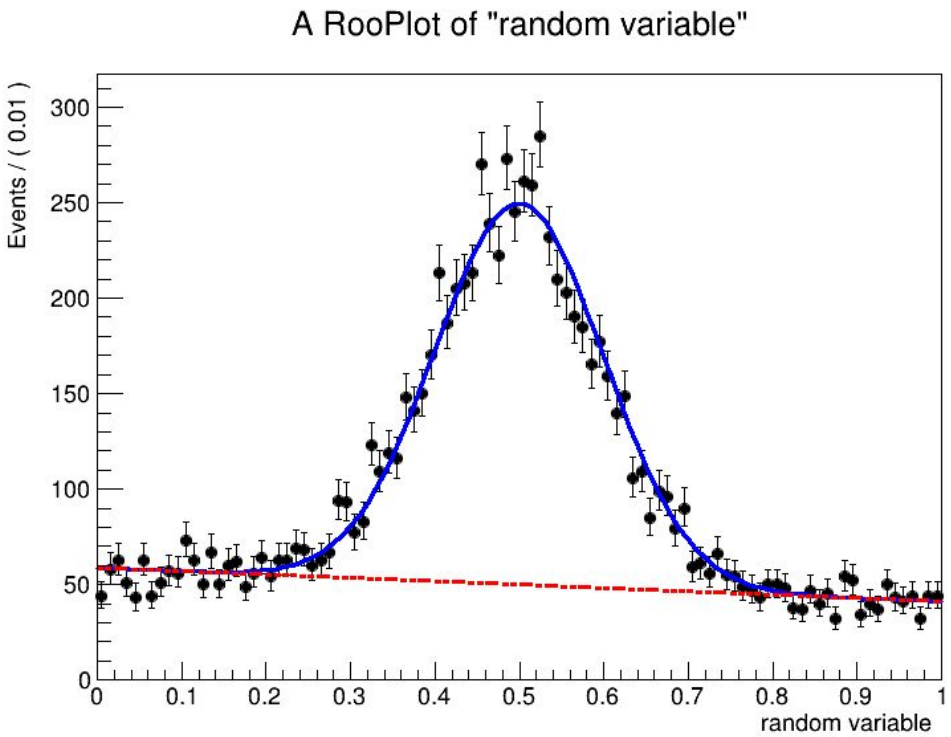
Construindo Modelo - usando componentes padrões

RooAddPdf



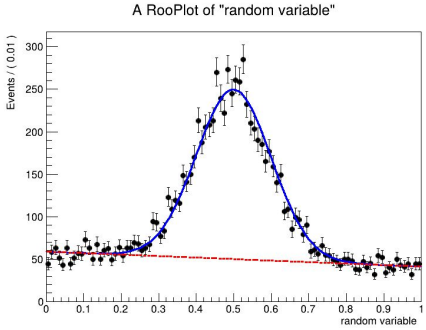
RooProdPdf



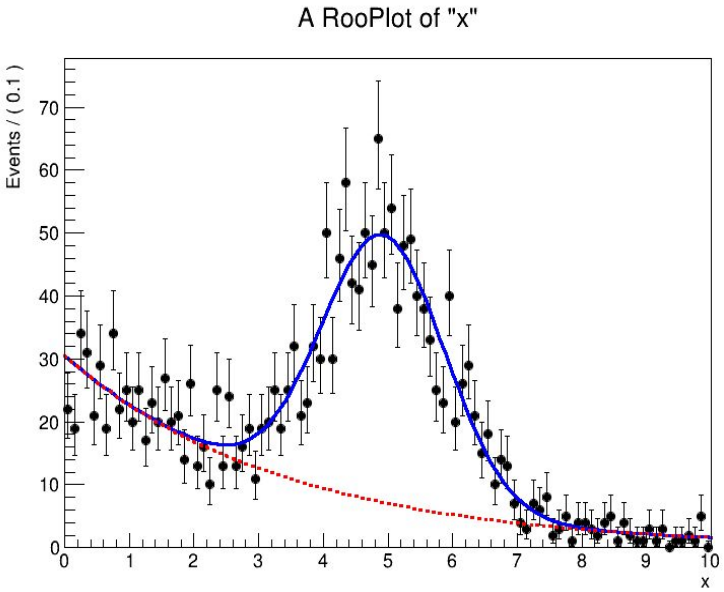


um modelo composto por uma distribuição **Gaussiana** e uma função **linear,** gera dados aleatórios com base nesse modelo e, em seguida, plota os dados e o modelo no mesmo gráfico.

```
#include "TCanvas.h"
#include <iostream>

using namespace RooFit;

void exemploAddPdf(){

// observable
RooRealVar x("x", "random variable", 0.0, 1.0);

// Gaussian model
RooRealVar mu("mu", "mean parameter", 0.5, 0.0, 1.0);
RooRealVar sigma("sigma", "width parameter", 0.1, 0.0, 0.3);
RooGaussian gaus("gaus", "Gaussian PDF", x, mu, sigma);

// Linear function: 1 + slope*x
RooRealVar slope("slope", "slope parameter", -0.3, -10., 10.);
RooPolynomial linear("linear", "Linear function", x, RooArgSet(slope));

// add up: Gaussian + linear
RooRealVar fraction("fraction","fraction of Gaussian",0.5,0.,1.);
RooAddPdf model("model","PDF model", RooArgList(gaus,linear), RooArgList(fraction));

// generate random data, plot
RooDataSet *dataset = model.generate(x, 10000);
RooFitResult* fit_result = model.fitTo(*dataset, RooFit::Save());
fit_result->Print("v");

TCanvas *c1 = new TCanvas("c1", "c1", 800, 600);
RooPlot* frame = x.frame();
dataset->plotOn(frame);
model.plotOn(frame);
model.plotOn(frame,Components(linear),LineStyle(7),LineColor(kRed));
frame->Draw() ;
c1->Draw() ;
c1->SaveAs("exemploSomaPDF.png");
}
```

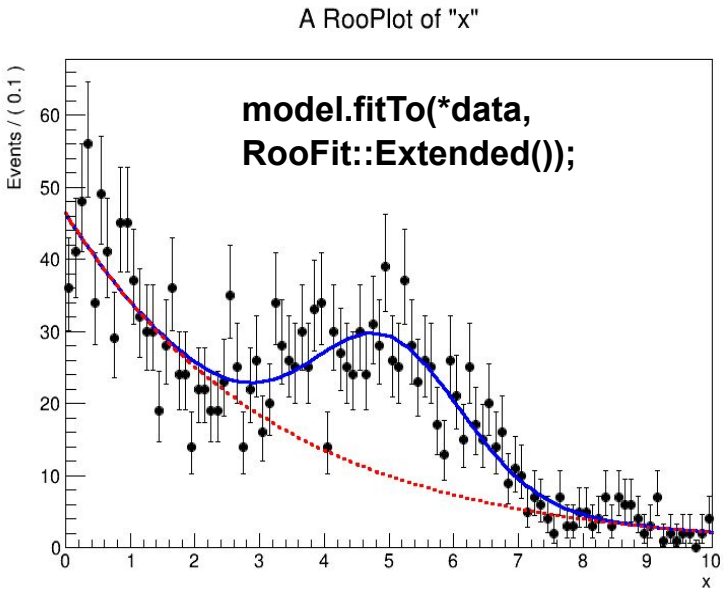


Ajuste não estendido vs Ajuste estendido:



$$L_i = f_s \cdot P_s(x_i;\mu,\sigma) + (1-f_s)\cdot P_b(x_i;c_1)$$

Floating Parameter	FinalValue +/- Error
bkgc	-2.9312e-01 +/- 1.81e-02
fraction	5.0838e-01 +/- 2.01e-02
sigmean	4.9237e+00 +/- 4.07e-02
sigwidth	9.5401e-01 +/- 4.02e-02



$$L_i = n_s \cdot P_s(x_i;\mu,\sigma) + n_b \cdot P_b(x_i;c_1)$$

Floating Parameter	FinalValue +/- Error
bkgc	-3.0682e-01 +/- 1.68e-02
nbkg	1.4397e+03 +/- 6.05e+01
nsig	5.6009e+02 +/- 5.27e+01
sigmean	4.9749e+00 +/- 8.75e-02
sigwidth	1.1567e+00 +/- 8.77e-02

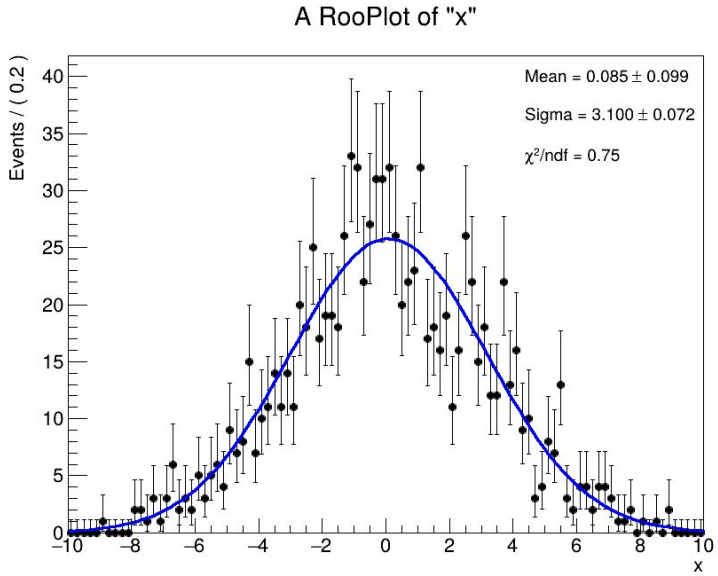
```
#include "RooPlot.h"
#include "TCanvas.h"

void extended_likelihood() {

    RooRealVar x("x", "x", -10, 10);

    // Definir o PDF do sinal (Gaussiana)
    RooRealVar mean("mean", "mean of gaussian", 0, -10, 10);
    RooRealVar sigma("sigma", "width of gaussian", 2, 0.1, 5);
    RooGaussian gauss("gauss", "Gaussian Signal PDF", x, mean, sigma);

    // Definir o PDF do fundo (Exponencial)
    RooRealVar tau("tau", "slope of exponential", -0.5, -5.0, 0.0);
    RooExponential expo("expo", "Exponential Background PDF", x, tau);

    // Número esperado de eventos de sinal e fundo
    RooRealVar nsig("nsig", "number of signal events", 500, 0, 1000);
    RooRealVar nbkg("nbkg", "number of background events", 500, 0, 1000);

    // Modelo estendido combinando sinal e fundo
    RooAddPdf model("model", "Signal + Background", RooArgList(gauss, expo), RooArgList(nsig, nbkg));

    // Gerar um conjunto de dados simulados
    RooDataSet* data = model.generate(x, 1000);

    // Ajustar o modelo aos dados (verossimilhança estendida)
    //model.fitTo(*data, RooFit::Extended());
    RooFitResult* fit_result = model.fitTo(*data, RooFit::Save(), RooFit::Extended());
    // Acessar os parâmetros ajustados e seus erros
    fit_result->Print("v");  // Imprime os detalhes do ajuste

    // Acessar os valores ajustados e incertezas
    double nsig_val = nsig.getVal();
    double nsig_err = nsig.getError();
    double nbkg_val = nbkg.getVal();
    double nbkg_err = nbkg.getError();

    std::cout << "Número ajustado de eventos de sinal: " << nsig_val << " ± " << nsig_err << std::endl;
    std::cout << "Número ajustado de eventos de fundo: " << nbkg_val << " ± " << nbkg_err << std::endl;

    // Criar um gráfico para a variável x e plotar os dados
    RooPlot* frame = x.frame();
    data->plotOn(frame);
```

Como você sabe se o seu fit está “bom”?

- Para 1-D fit, χ^2 é usualmente a melhor coisa a fazer

- Algumas ferramentas implementadas no RooPlot é capaz de calcular o

```
double chi2 = frame->chisquare(nFloatParam);
```




```
pdf.paramOn(frame, RooFit::Layout(0.6, 0.9,0.9));
```

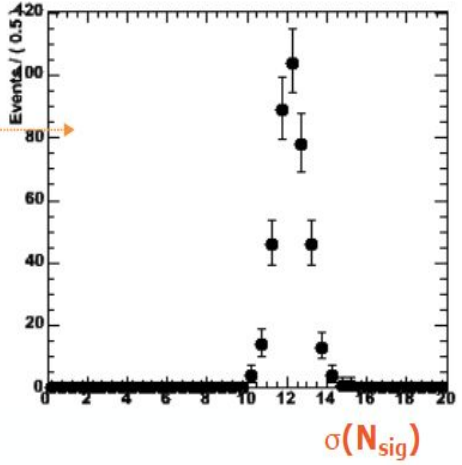
```
// Adicionar um TLegend para exibir informações dos parâmetros e chi2
TLegend *leg = new TLegend(0.6, 0.7, 0.9, 0.9);
leg->SetTextSize(0.03);
leg->SetBorderSize(0);
leg->SetFillStyle(0);
leg->AddEntry((TObject*)0, Form("Mean = %.3f #pm %.3f", mean.getVal(), mean.getError()), "");
leg->AddEntry((TObject*)0, Form("Sigma = %.3f #pm %.3f", sigma.getVal(), sigma.getError()), "");
leg->AddEntry((TObject*)0, Form("#chi^{2}/ndf = %.2f", chi2), "");
leg->Draw();
```

Estudo de Validação do Ajuste - A distribuição pull

E quanto à validade do erro?

–Distribuição de erro de experimentos simulados é difícil de interpretar

–Não temos o equivalente de N_{sig}(gerado) para o erro



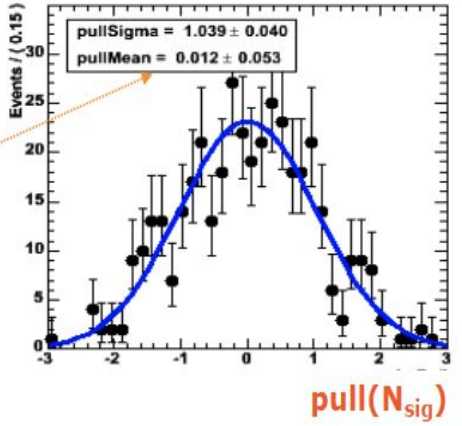
• Solução: verificar a ***pull distribution***

Definição:

$$pull(N_{sig}) = \frac{N_{sig}^{fit} - N_{sig}^{true}}{\sigma_N^{fit}}$$

– Propriedades da pull:

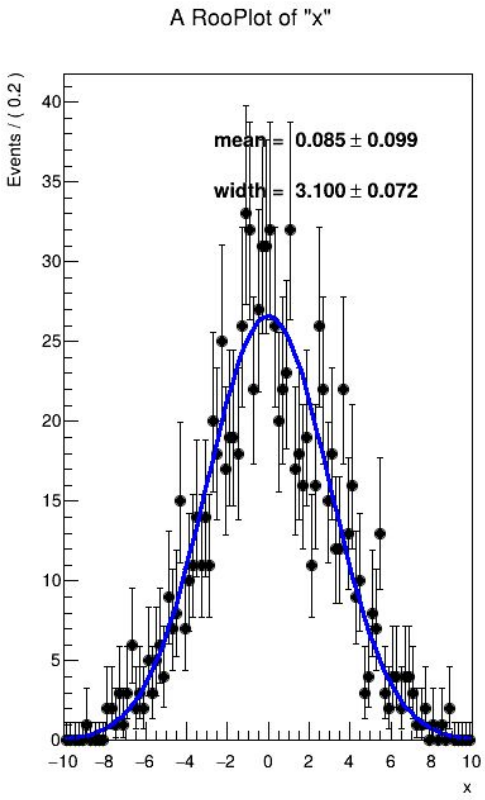
```
frame->pullHist();
```

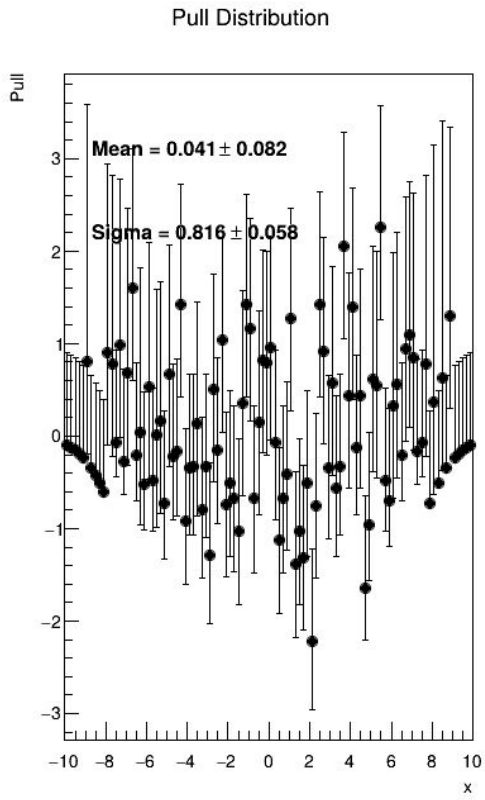


• Mean é 0 se não há bias

• Width é 1 se o erro está correto

– Nesse exemplo: sem bias, erro correto dentro da precisão estatística de estudo





- Pull : http://physics.rockefeller.edu/luc/technical_reports/cdf5776_pulls.pdf

Trabalhando com o RooFit

Exercício 1

• Crie uma p.d.f Crystall Ball, gere uma amostra de dados a partir dessa pdf e ajuste aos dados, adicione a caixa de informação estatistica dos dados e do modelo. Quais foram os valores ajustados para os parâmetros dessa pdf?

• Você pode encontrar diversas pdfs no RooFit no link abaixo:

  –https://root.cern/download/doc/RooFit_Users_Manual_2.91-33.pdf

  –(todas os nomes das classes em RooFit começam com “Roo”)

Exercício 2

Defina uma variável contínua x no intervalo de 0 a 10.
Defina uma função exponencial decrescente (exp(-lambda * x)), onde lambda é um parâmetro inicial com valor de 1 e limites de ajuste entre 0.1 e 2.
Gere 1500 eventos simulados a partir dessa distribuição exponencial.
Realize um ajuste **estendido** da função exponencial, ajustando tanto o parâmetro lambda quanto o número total de eventos observados.
Visualize o ajuste e exiba os resultados ajustados para lambda e o rendimento total (número de eventos).

Responda:
Qual é o valor ajustado para o parâmetro lambda?
Qual é o número total de eventos de eventos ajustados?
Compare os valores ajustados com os valores gerados. Eles estão dentro das expectativas?

Exercício 3

Construa um modelo (sinal + background) e ajuste aos dados para a distribuição de massa da ressonância J/ψ . Utilize o arquivo que se encontra no link: https://cernbox.cern.ch/index.php/s/DInqlmV9W52WPvY

Faça o teste estatistico do seu ajuste, calculando o χ^2 / ndf.
Use o método paramOn() para adicionar os parâmetros ajustados ao gráfico.

Informação: pico da massa do J/Psi ~ 3,096916 GeV/c²

- **a PDF do sinal no pico do J/ψ com uma função Crystal Ball**
- **a PDF do Background com uma polinomial**

Sumário sobre o RooFit

- **Overview das funcionalidades do RooFit**
  - –nem tudo foi coberto
  - –não foi discutido como isso funciona internamente (optimização, dedução analitica,etc..)
- **Capaz de lidar com modelos complexos**
  - –modelos com grande número de parâmetros
  - –sendo usado em muitas análises do LHC
- **Workspace:**
  - –fácil de criar modelos usando a sintaxe factory
  - –ferramenta para armazenar e compartilhar modelos (combinação de análise)

Backup

Teste do χ^2

Os dados observados são consistentes com o modelo proposto?

- O teste do χ^2 é uma ferramenta estatística usada para avaliar a adequação de um modelo aos dados observados.
- Ele mede a discrepância entre os dados observados e os valores esperados de um modelo, ajudando a verificar a qualidade do ajuste.
- É calculado por:

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

- onde O_i são os valores observados e E_i são os valores esperados.

Teste do χ^2

- O melhor ajuste é obtido quando o valor de χ^2 dividido pelo número de graus de liberdade (ndf) se aproxima de 1.

- Graus de liberdade referem-se ao número de valores independentes que podem variar em uma análise.

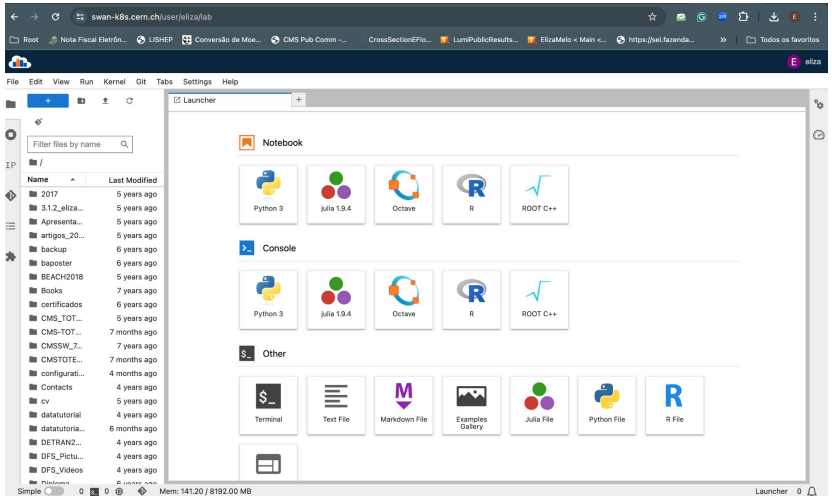
- Em um ajuste de curva, os graus de liberdade são geralmente dados pelo número de pontos de dados menos o número de parâmetros ajustados.

Terminologias

Here we give pragmatic definitions for a few basic concepts that we will use.

- **observable** - something you measure in an experiment, for example a particle's momentum. Often, a function of measured quantities, for example an invariant mass of several particles
- **global observable** or **auxiliary observable** - an observable from another measurement, for example, the integrated luminosity
- **model** - a set of probability density functions (PDFs), which describe distributions of the observables or functions of observables
- **model parameter** - any variable in your PDF expression, which is not an observable
- **parameter of interest (POI)** - a model parameter that you study, for example a cross section of your signal process
- **nuisance parameter** - every other model parameter, which is not your parameter of interest
- **data** or **dataset** - a set of values of observables, either measured in an experiment, or simulated
- **likelihood** - a model computed for a particular dataset
- **hypothesis** - a particular model, with specified observables, POI, nuisance parameters, and prior PDFs (in case of Bayesian inference)
- **prior PDF** - a probability density for an observable or a model parameter, which is known *a priori*, i.e. before a measurement is considered. This is a Bayesian concept exclusively. Prior has no meaning or place in a frequentist type of inference
- **Bayesian** - a type of statistical inference that usually produces probability of the hypothesis given the data. Requires a prior.
- **frequentist** - a type of statistical inference that usually produces probability of the data given the hypothesis.

O **Jupyter Notebook** é uma aplicação web de código aberto que permite criar e compartilhar documentos que contêm código executável, visualizações, texto explicativo e outros elementos multimídia.
Ele é amplamente utilizado em ciência de dados, análise estatística, aprendizado de máquina, simulações numéricas e em muitos outros campos onde a exploração interativa e a comunicação de resultados são importantes.



Exemplo 02 em C++

```
#include "RooRealVar.h"
#include "RooGaussian.h"
#include "RooDataSet.h"
#include "RooPlot.h"
#include "TCanvas.h"

void generate_events() {
    // Definir variável e PDF
    RooRealVar x("x", "x", -10, 10);
    RooRealVar mean("mean", "mean", 0, -10, 10);
    RooRealVar sigma("sigma", "sigma", 1, 0.1, 10);
    RooGaussian gauss("gauss", "Gaussian PDF", x, mean, sigma);

    // Gerar dataset
    RooDataSet* data = gauss.generate(RooArgSet(x), 10000);

    // Criar frame para plotagem
    RooPlot* xframe2 = x.frame(ROOT.RooFit.Title("Gaussian p.d.f. with data"));

    // Plotar dados e PDF no frame
    data->plotOn(xframe2);
    gauss.plotOn(xframe2);

    // Criar canvas e desenhar frame
    TCanvas* canvas = new TCanvas("canvas", "Gaussian PDF with Data", 800, 600);
    xframe2->Draw();
    canvas->Draw();
}

generate_events();
```

Gerando um Conjunto de Dados Unbinned:

```
#include "RooRealVar.h"
#include "RooGaussian.h"
#include "RooDataSet.h"

void generate_unbinned_data() {
    RooRealVar x("x", "x", -10, 10);
    RooRealVar mean("mean", "mean", 0, -10, 10);
    RooRealVar sigma("sigma", "sigma", 1, 0.1,
10);
    RooGaussian gauss("gauss", "Gaussian PDF",
x, mean, sigma);

    // Gerar dados unbinned
    RooDataSet* unbinned_data =
gauss.generate(RooArgSet(x), 10000);

    // Usar o conjunto de dados para análise ou
ajuste
}
```

Gerando um Conjunto de Dados binned:

```
#include "RooRealVar.h"
#include "RooGaussian.h"
#include "RooDataSet.h"
#include "RooDataHist.h"

void generate_binned_data() {
    RooRealVar x("x", "x", -10, 10);
    RooRealVar mean("mean", "mean", 0, -10, 10);
    RooRealVar sigma("sigma", "sigma", 1, 0.1, 10);
    RooGaussian gauss("gauss", "Gaussian PDF", x, mean,
sigma);

    // Gerar dados unbinned
    RooDataSet* unbinned_data =
gauss.generate(RooArgSet(x), 10000);

    // Criar histograma a partir dos dados unbinned
    RooDataHist* binned_data =
unbinned_data->binnedClone();

    // Usar o conjunto de dados binned para análise ou ajuste
}
```