
DFNAE
DEPARTAMENTO DE FÍSICA NUCLEAR E ALTAS ENERGIAS

# RooFit

## Aula 06

Eliza Melo

Rio de Janeiro, 21 de Setembro de 2021

# Esboço

- Introdução ao RooFit

  –Funcionalidades básicas
  –Construindo modelos usando “workspace”
  –Modelos Compostos

- Exercício com o RooFit:

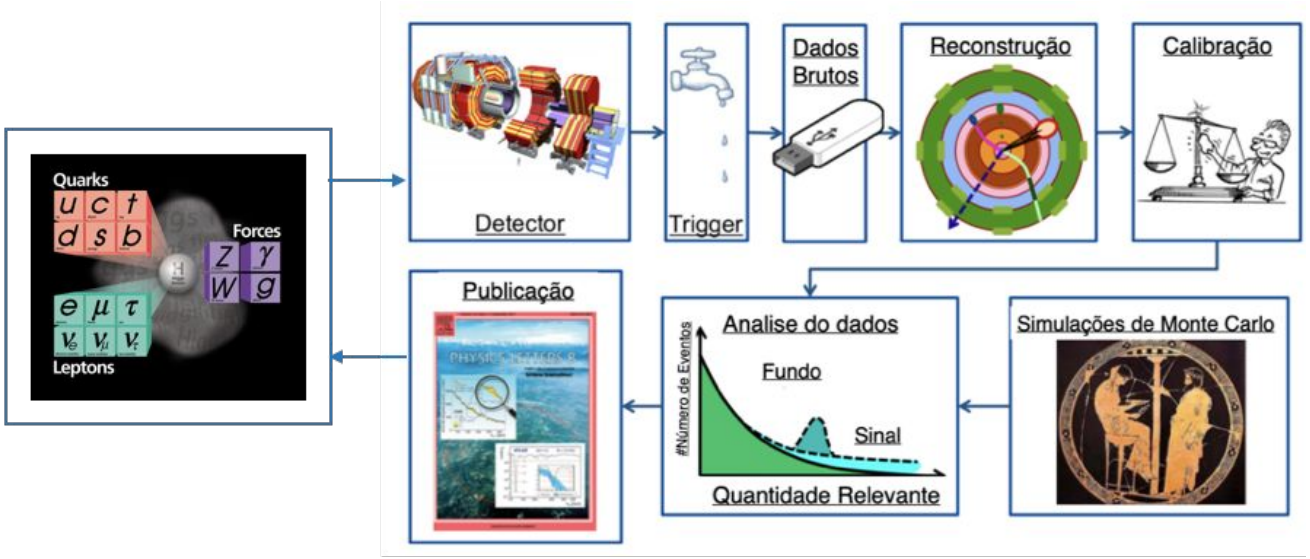
  – construindo e ajustando modelos

## Créditos

Material baseado em:

- slides do W. Verkerke (autor do RooFit)

# A jornada para a medida…

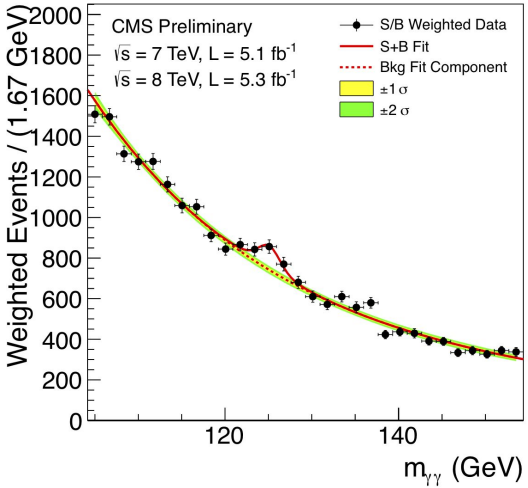



- Além da detecção e lermos os dados brutos, temos as fases de:
  - calibração;
  - reconstrução;
  - busca por sinal de física
    - Extrair os observáveis de interesse com certa precisão.

- publicação da análise estatística final.

# O que é Ajuste (Fitting) ?

- Estimar parâmetros de um uma distribuição hipotética a partir de dados observados
  - y = f ( x | θ ) é a função de modelo de ajuste de dados
- Encontre a melhor estimativa do parâmetro θ assumindo f ( x | θ )
- Os ajustes Likelihood e Chi2 são suportados no ROOT




***Exemplo***

Higgs → γγ espectro

Podemos ajustar para:

- o número esperado de eventos de Higgs
- massa do Higgs

# Estimativa de Parâmetros

• Dado um modelo para os nossos dados observados (*Probability Density Function-PDF*), nós queremos estimar o parâmetro do nosso modelo.

• O modelo dos dados observados é expresso usando a Função Densidade de Probabilidade(PDF)

–A PDF é uma probabilidade diferencial $f(\vec{x}, \theta)$

- ex.: a probabilidade de observar um evento em um bin de um histograma $P_{bin} = \int_{bin} f(\vec{x}, \theta) d\vec{x}$

–A PDF é normalizada a 1 quando integrada em todo o espaço da amostra Ω $\int_{\Omega} f(\vec{x}, \theta) d\vec{x} = 1$

• Para estimar os parâmetros usamos **Likelihood Function** $L(\vec{x}_1, .., \vec{x}_N|\theta) = \prod_{i=1}^{N} f(\vec{x}_i, \theta)$

• Por conveniência usamos o log da likelihood-function

• Usando o negativo log-likelihood function encontramos o minimum global

$$-\log L(\vec{x}_1, .., \vec{x}_N|\theta) = -\sum_i \log f(\vec{x}_i, |\theta)$$

• *Extended Likelihood*

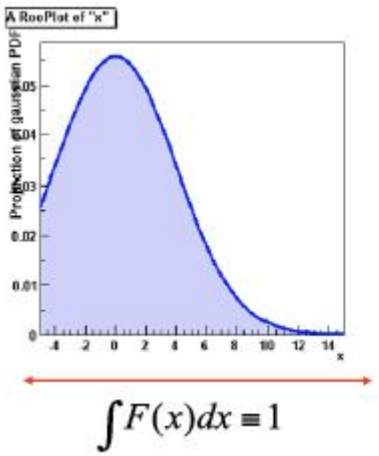
$$\log L(x|\theta) = \sum_{bin} \log e^{-\nu} \frac{\nu^N}{N!} f(x|\theta)$$

# O que é o RooFit ?

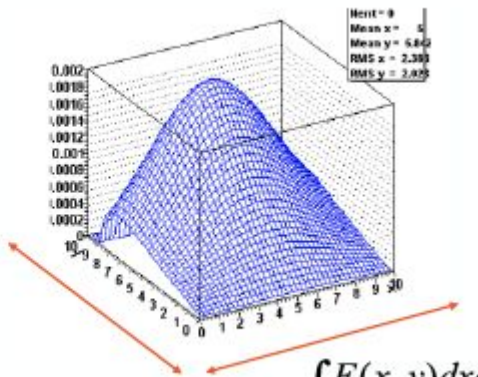
É um pacote de ferramentas distribuído com o ROOT para modelagem de dados.

- É usado para modelar distribuições que são usadas para ajustes e análise estatística de dados.
  - distribuição do modelo da observável **x** em termos do parâmetro **p**
- Função Densidade de Probabilidade (P.D.F.): P(x;p)
- PDF são normalizada sobre o intervalo permitido de observáveis **x** em relação aos parâmetros **p**

$$\int_{\Omega} P(\vec{x};\vec{p})d\vec{x} = 1$$




$$\int F(x)dx \equiv 1$$




$$\int F(x,y)dxdy \equiv 1$$

# Por que o RooFit ?

–ROOT pode lidar com funções complicadas, porém pode exigir a escrita de uma grande quantidade de código

  –Normalização de PDF nem sempre trivial

  •RooFit faz isso automaticamente

–Em ajuste complexo, o desempenho de computação é importante

  • necessidade de otimizar o código para um desempenho aceitável

  • otimização integrada disponível no RooFit

    –avaliação de partes do modelo apenas quando necessário

–Ajuste simultâneo para diferentes amostras de dados

–Fornece uma descrição completa do modelo para uso posterior

# RooFit

- RooFit fornece funcionalidades para construir as  PDFs
  - –construção de modelo complexo a partir de componente padrão
  - –composição com produto de adição e convolução
- Todos os modelos fornecem a funcionalidade para:
  - –ajuste *maximum likelihood*
  - –gerador de toy MC
  - –visualização

# Funções vs Funções Densidades de Probabilidades

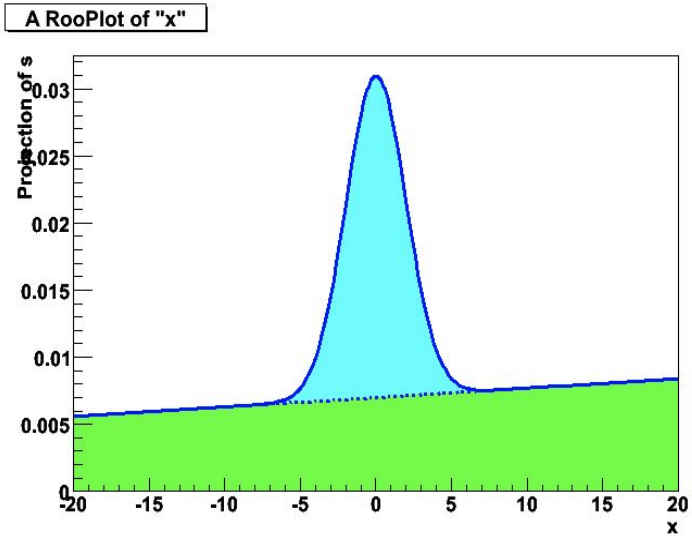
• Por que usar PDFs em vez de funções “simples” para modelar os dados?

–Fácil de interpretar os modelos.

-Se tivermos a garantia que as pdfs em Azul e em Verde sejam normalizadas a 1

-Então as frações de Azul e Verde podem claramente ser interpretadas como #eventos

–Muitas técnicas estatísticas só funcionam corretamente com p.d.f. (ex.: maximum likelihood fits).




• O que é difícil com p.d.f ?

–A normalização pode ser difícil de calcular

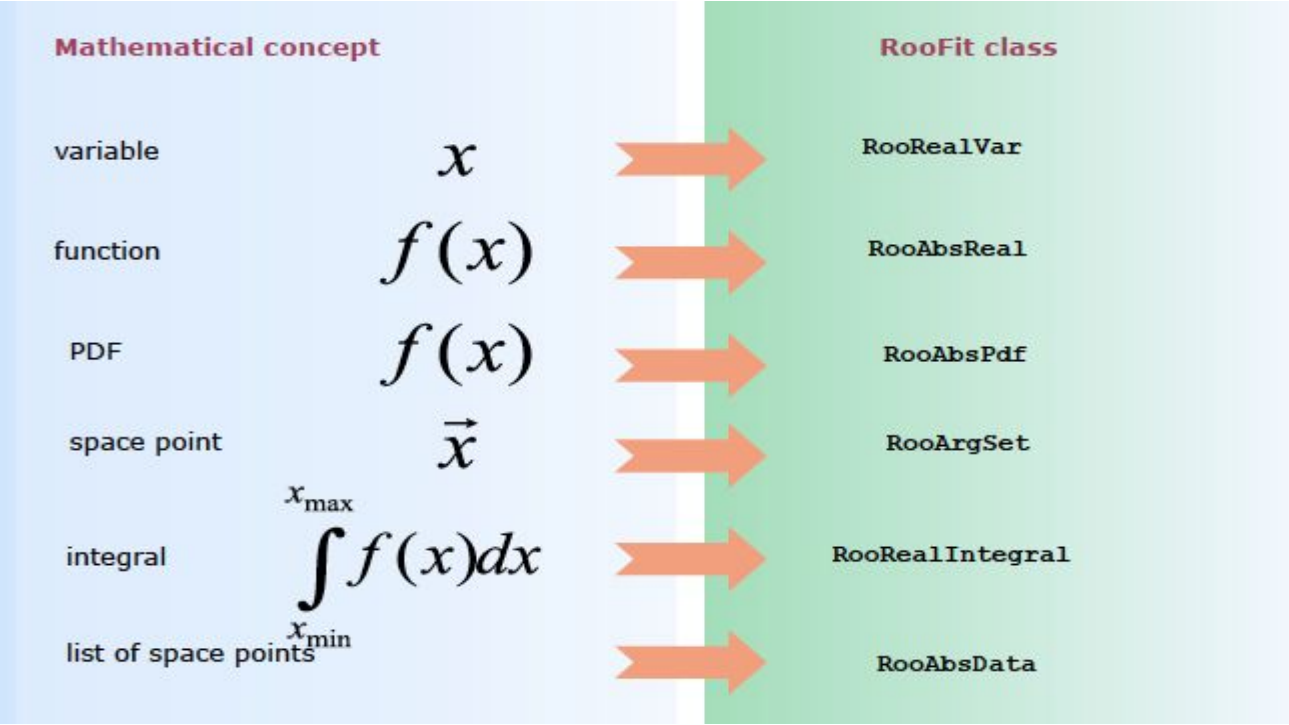
(ex.: isso pode ser diferente para cada conjunto de valores de parametro p)

• Para dimensões >1 a integração é difícil

–O RooFit visa simplificar essas tarefas

# Filosofia de design central do RooFit

Conceitos matemáticos são representados como objetos em C++



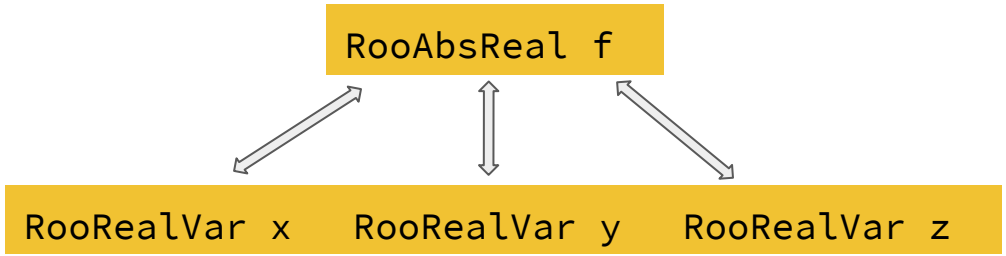
# Filosofia de design central do RooFit

Relações entre variáveis e funções como links de clientes/servidores entre objetos

**matemática:**

f(x,y,z)

**Diagrama do RooFit:**

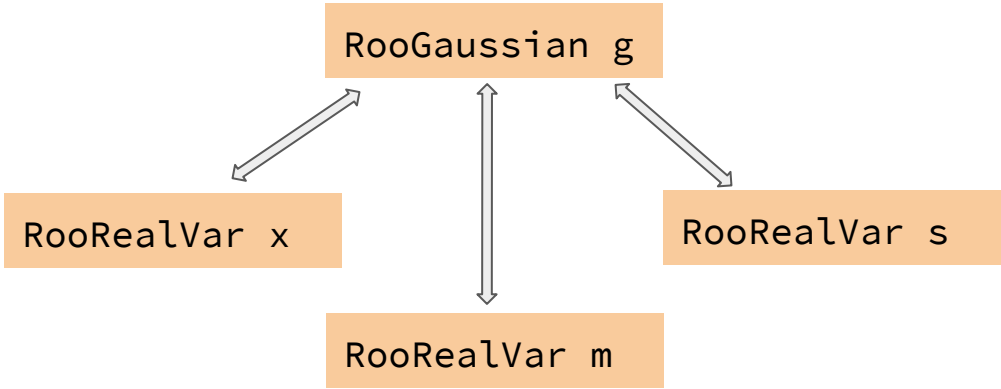



**código RooFit:**

```
RooRealVar x(“x”, “x”, 2);
RooRealVar y(“y”, “y”, 3);
RooRealVar z(“z”, “z”, 4);
RooBogusFunction f(“f”, “f”, x,y,z);
```

# Modelagem com o RooFit

**Exemplo: pdf Gaussianna** Gaus(x,m,s)

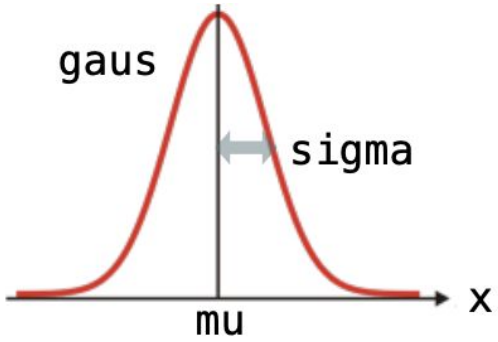



**RooFit code:**

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

# O exemplo mais simples possível (PDF Gaussiana)

**o notebook "RooFitBasics"**




Nome do objeto (no seu código)

Nome do objeto

Título do objeto

range inicial(min, máx)

objetos representando um valor "real"

```
RooRealVar x("x", "Observable",-10,10);
RooRealVar mean("mean", "B0 mass", 0.00027);
RooRealVar sigma("sigma", "B0 mass width", 5.2794, "GeV");
```

initial range

unidade opcional

Objeto PDF

```
RooGaussian model("model", "signal pdf", x,mean,sigma);
```

Referências as variáveis(objetos)

# Gerando eventos com toy MC

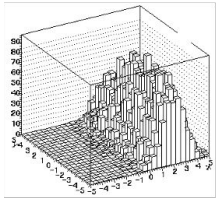
Gera 10000 eventos a partir de uma p.d.f Gaussiana e mostra a distribuição

```
// Generate an unbinned toy MC set
RooDataSet* toyData = gauss.generate(x,10000);

// Generate an binned toy MC set
RooDataHist* toyData = gauss.generateBinned(x,10000);
```

Unbinned

| x | y | z |
|---|---|---|
| 1 | 3 | 5 |
| 2 | 4 | 6 |
| 1 | 3 | 5 |
| 2 | 4 | 6 |

Binned



RooDataSet RooDataHist

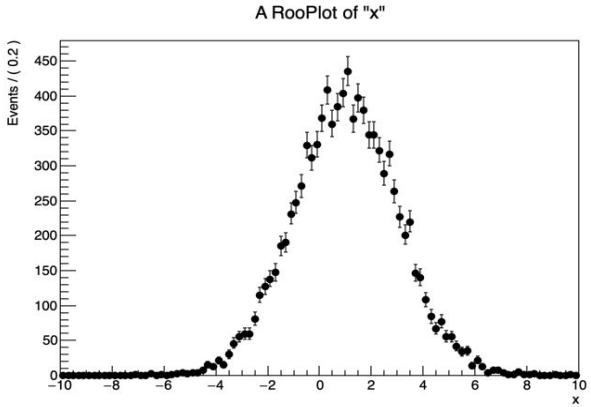
RooAbsData

Pode-se gerar ambos, binned and unbinned datasets

## Visualização dos dados

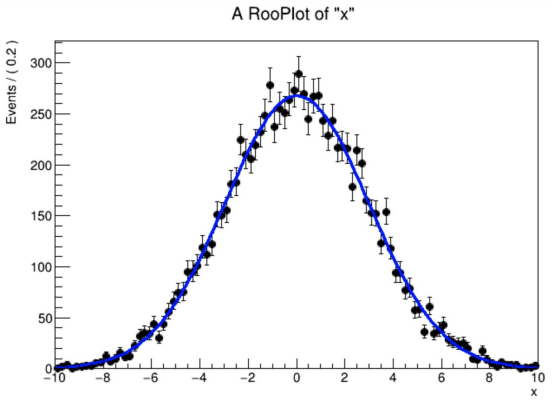
```
// Plot PDF
RooPlot * xframe = x->frame();
toyData->plotOn(xframe);
xframe->Draw();
```

o notebook "RooFitBasics"



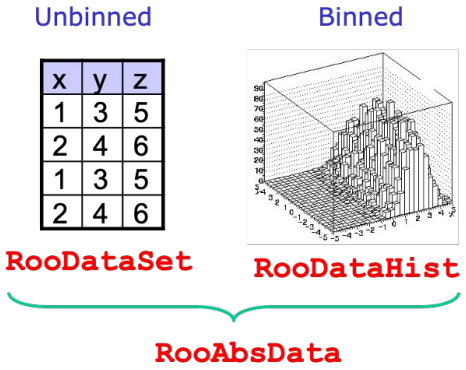
# Gerando eventos com toy MC

```
// generate unbnined dataset of 10k events
RooDataSet* toyData = g.generate(x,10000) ;

// Perform unbinned ML fit to toy data
g.fitTo(*toyData) ;

// Plot toy data and pdf in observable x
RooPlot* frame = x.frame() ;
toyData->plotOn(frame) ;
g.plotOn(frame) ;
frame->Draw() ;
```




PDF automaticamente
normalizada ao dataset

# Importando dados




- Importar dados unbinned de ROOT **TTrees**

```
// Import unbinned data
RooDataSet data("data","data",x,Import(*myTree));
```

  – Importa o **TTree** branch chamado "**x**".

  – Tipos possíveis:`Double_t, Float_t, Int_t` or `UInt_t`.

  Todo dado é convertido internamente em `Double_t`

  – Especifique um `RooArgSet` de vários observáveis para importar múltiplos observáveis

- Importar dados de histogramas ROOT THx

```
// Import binned data
RooDataHist data("data","data",x,Import(*myTH1));
```

  – Importa valores, define binning e erros (se definido)

  – Especifica uma lista de observáveis RooArgList quando importa um TH2/3.

# Importando dados

```
RooRealVar x("x", "x", -10.0, 10.0);
RooRealVar c("c", "c", 0.0, 30.0);
// Import unbinned data
RooDataSet data(“data”,”data”,Import(*myTree),RooArgSet(x,c));
```

- Remoção automática de entradas fora do intervalo da variável.

```
// Importando um arquivo ASCII
RooDataSet* data = RooDataSet::read(“ascii.file”,RooArglist(x,c));
```

- Uma linha por entrada; ordem de variável dada pela lista de argumentos.

**o notebook “exemplo3”**

# Expressões genéricas de PDFs

Se a sua PDF favorita não estiver lá :

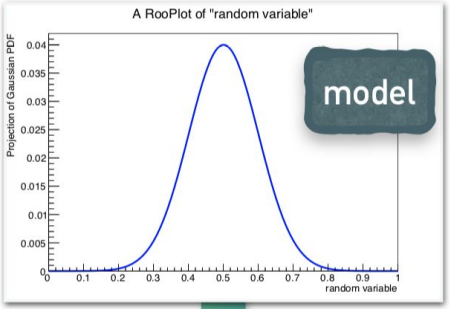
- usar o **RooGenericPdf**

```
// 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","Generic PDF",
"exp(x*y+a)-b*x",RooArgSet(x,y,a,b));
```

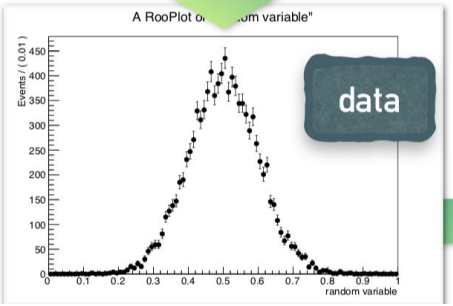
o notebook
“exemplo2”

# Geração e Ajuste

o notebook
"exemplo4"

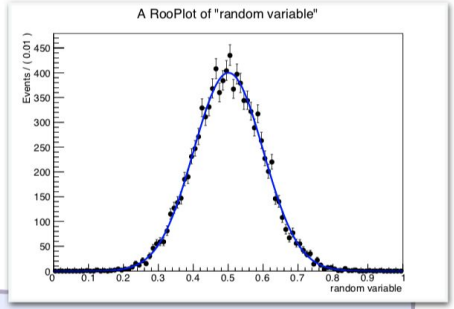



```
model.generate(x,10000);
```




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

```
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
```

# Geração e Ajuste

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

```
RooDataSet* data = model.generate(x, ...);
```

| | |
|---|---|
| Extended(flag) | O número real de eventos gerados será mostrado a partir de uma distribuição de Poisson com μ = Nevt. |

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

| | |
|---|---|
| Save(flag) | Se o RooFitResult é produzido |
| Extended(flag) | Adiciona o termo de likelihood estendida |

# Trabalhando com o RooFit

## Exercício 1

- Crie uma p.d.f Crystall Ball, gere um toy data e fit
- Extra:
  - –Brinque com outra p.d.f
    - ex.: Exponencial
- ou outra p.d.f do seu interesse.
- Você pode encontrar diversas pdfs em RooFit no link abaixo:
  - –https://root.cern/download/doc/RooFit_Users_Manual_2.91-33.pdf
  - –(todas os nomes das classes em RooFit começa com “Roo”)

https://github.com/sandrofonseca/rootFitTutorial/blob/master/roofitUERJ/GausModelRooFit.ipynb

# RooFit Workspace

- A classe **`RooWorkspace`**: container para todos os objetos 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 está disponível para uma análise aprofundada
- Combinação dos resultados juntando workspaces em um só
  - –formato comum para combinar e compartilhar resultados de física

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

# RooFit Factory

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

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])”)
```

Trabalharemos com  exemplos usando o workspace factory para construir modelos

# 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

```
RooRealVar x(“x”,”x”,-10,10);
RooRealVar mean(“mean”,”mean”,5);
RooRealVar sigma(“sigma”,”sigma”,3);
RooGaussian f(“f”,”f”,x,mean,sigma);
// imports f,x,mean and sigma
w.import(f);
```

# 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

```
//Variety of accessories available
RooPlot* frame = w.var("x")->frame() ;
w.pdf("f")->plotOn(frame) ;
```

# 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(“wspace.root”);
```

- Organizando seu código – Separa a construção e o uso de modelos

```
void driver() {
RooWorkspace w(“w”) ;
makeModel(w) ;
useModel(w) ;
}
void makeModel(RooWorkspace& w) {
// Construct model here
}
void useModel(RooWorkspace& w) {
// Make fit, plots etc here
}
```

# Factoring Syntax

- Regra #1 – Crie uma variável

```
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 object 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));
```

# Factoring Syntax

• Regra #3 – Cada expressão criada retorna o nome do objeto criado
– Permite criar argumentos de entrada para funções “no local” em vez de antecipadamente

```
Gaussian::g(x[-10,10],mean[-10,10],sigma[3])
//--> x[-10,10]
// mean[-10,10]
// sigma[3]
// Gaussian::g(x,mean,sigma)
```

• Pontos diversos
– Você pode sempre usar valores numéricos onde funções ou valores são esperados

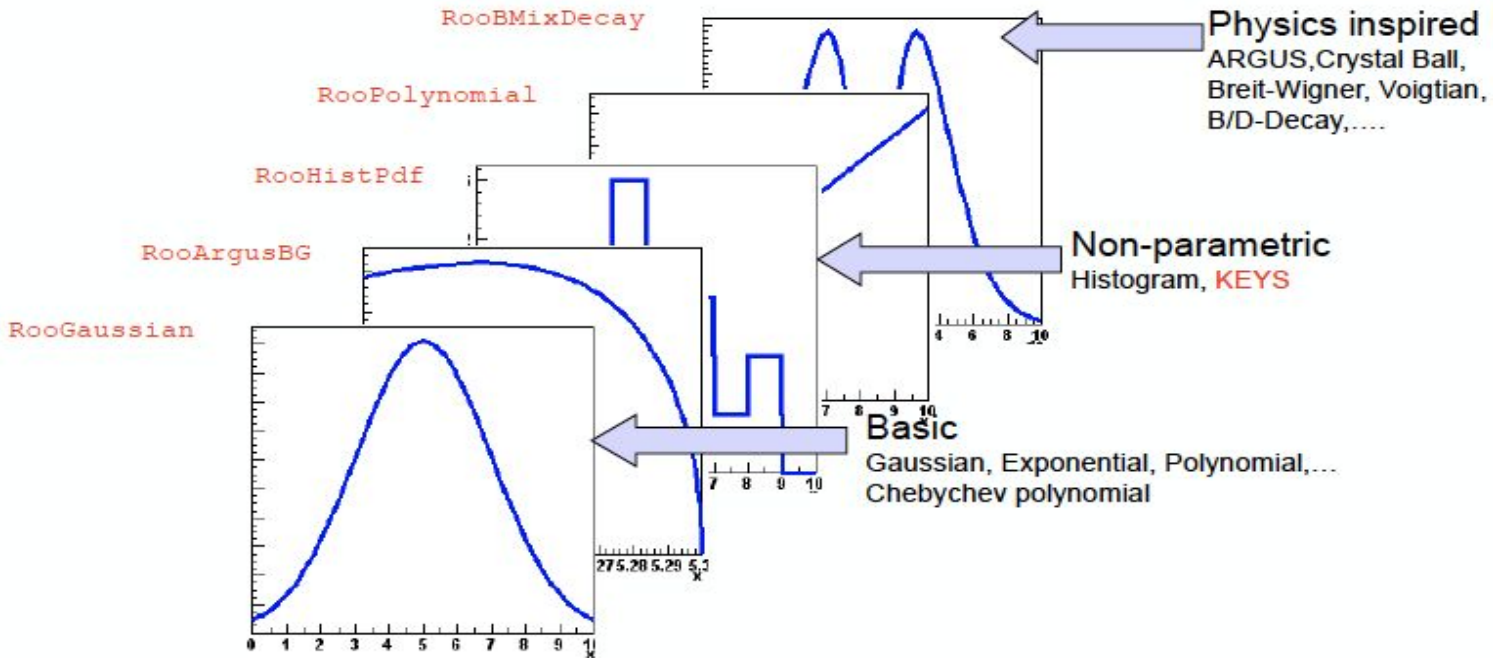
```
Gaussian::g(x[-10,10],0,3)
```

– Não é necessário dar um nome aos objetos da componente :

```
SUM::model(0.5*Gaussian(x[-10,10],0,3),Uniform(x));
```

# 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

- Lista das pdfs mais usadas e suas Factories

  Gaussian `Gaussian::g(x,mean,sigma)`

  Breit-Wigner `BreitWigner::bw(x,mean,gamma)`

  Landau `Landau::l(x,mean,sigma)`

  Exponential `Exponential::e(x,alpha)`

  Polynomial `Polynomial::p(x,{a0,a1,a2})`

  Chebychev `Chebychev::p(x,{a0,a1,a2})`

  Kernel Estimation `KeysPdf::k(x,dataSet)`

  Poisson `Poisson::p(x,mu)`

  Voigtian `Voigtian::v(x,mean,gamma,sigma)`

# Factory syntax - 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])");
```

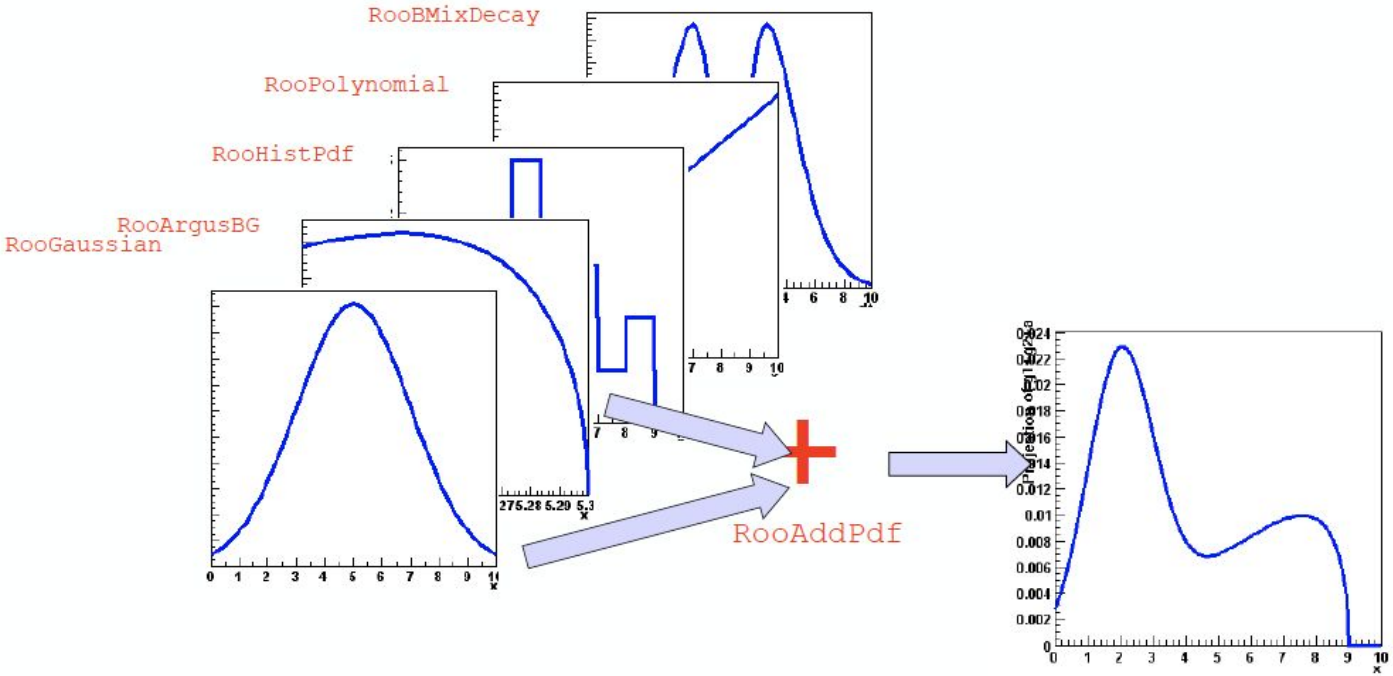
– nota: usando **expr** (cria-se uma função, uma RooAbsReal)
usando **EXPR** (cria-se uma PDF, uma RooAbsPdf)

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

# Construindo Modelo - (Re)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)*)

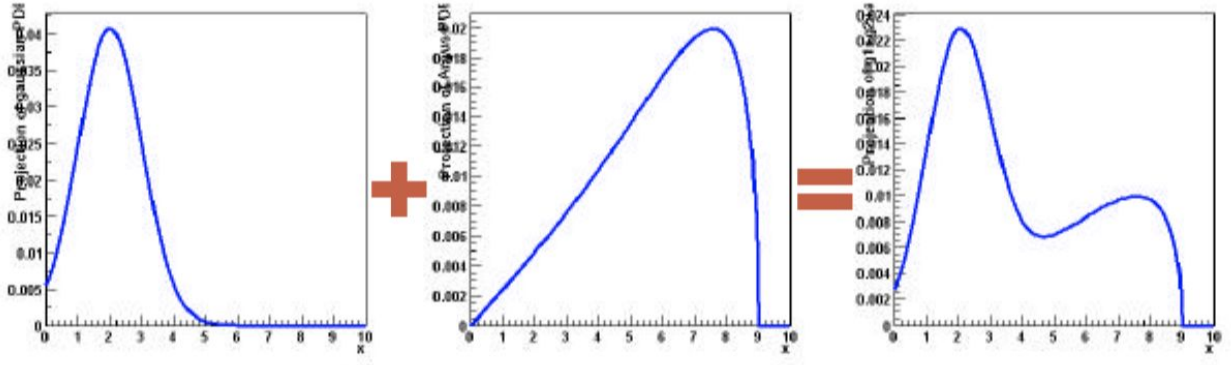
• Facilitado por meio de classes `operador p.d.f RooAddPdf`



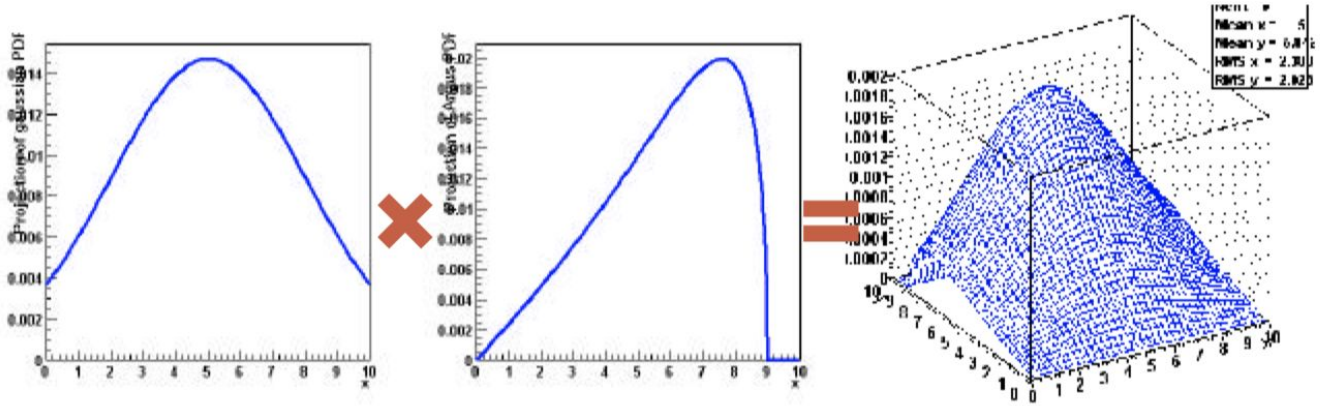
# Construindo Modelo - (Re)usando componentes padrões

RooAddPdf

o notebook "exemploAddPdf"



RooProdPdf



# Adicionando p.d.f.s - Factory syntax

• Adições criadas através de uma expressão SUM usando frações

```
SUM::name(frac1*PDF1,PDFN)
```

$$S(x) = fF(x) + (1-f)G(x)$$

```
SUM::name(frac1*PDF1,frac2*PDF2,...,PDFN)
```

–Observe que a última PDF não tem uma fração associada no caso de normalização geral flutuante.

• quando a normalização é ajustada a partir dos eventos observados

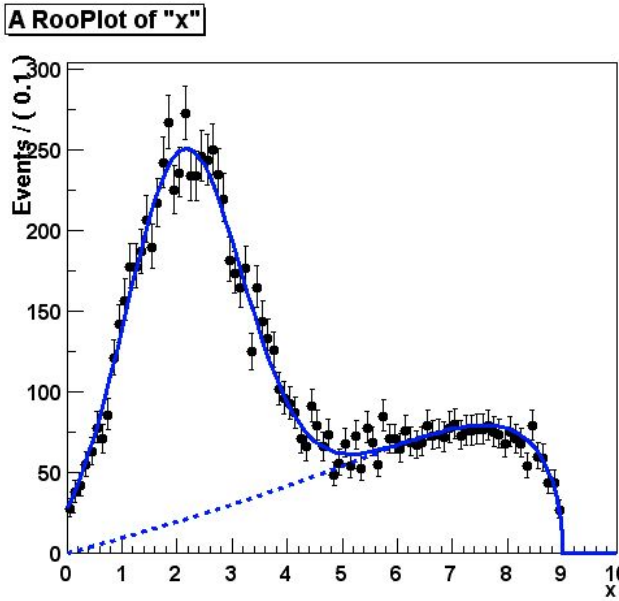
• Exemplo completo:

```
w.factory("Gaussian::gauss1(x[0,10],mean1[2],sigma[1]") ;
w.factory("Gaussian::gauss2(x,mean2[3],sigma)") ;
w.factory("ArgusBG::argus(x,k[-1],9.0)") ;
w.factory("SUM::sum(g1frac[0.5]*gauss1, g2frac[0.1]*gauss2,argus)");
```

# Plotando as Componentes de uma p.d.f

- Plotando, geração de eventos toy e aplicando o ajuste funciona identicamente para p.d.f.s compostas.
  – Diversas otimizações aplicadas nos bastidores que são específicas para modelos compostos (ex.: delegar geração de eventos as componentes)
- Funcionalidade extra específica para plotar p.d.f.s compostas
  – Plotando a componente

```
// Plot only argus components
w::sum.plotOn(frame,Components(“argus”),LineStyle(kDashed));
// Wildcards allowed
w::sum.plotOn(frame,Components(“gauss*”),LineStyle(kDashed));
```



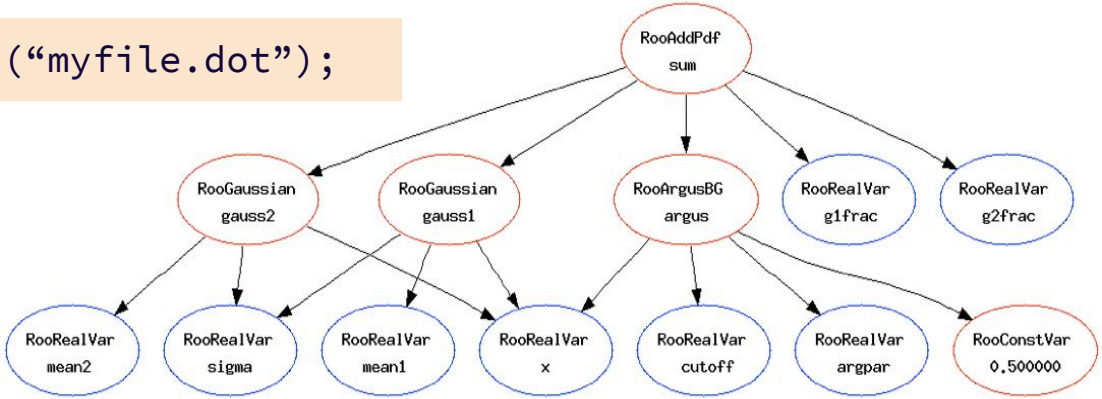
# Operações específicas para pdfs compostas

- O modo de impressão em árvore do workspace revela a estrutura da componente

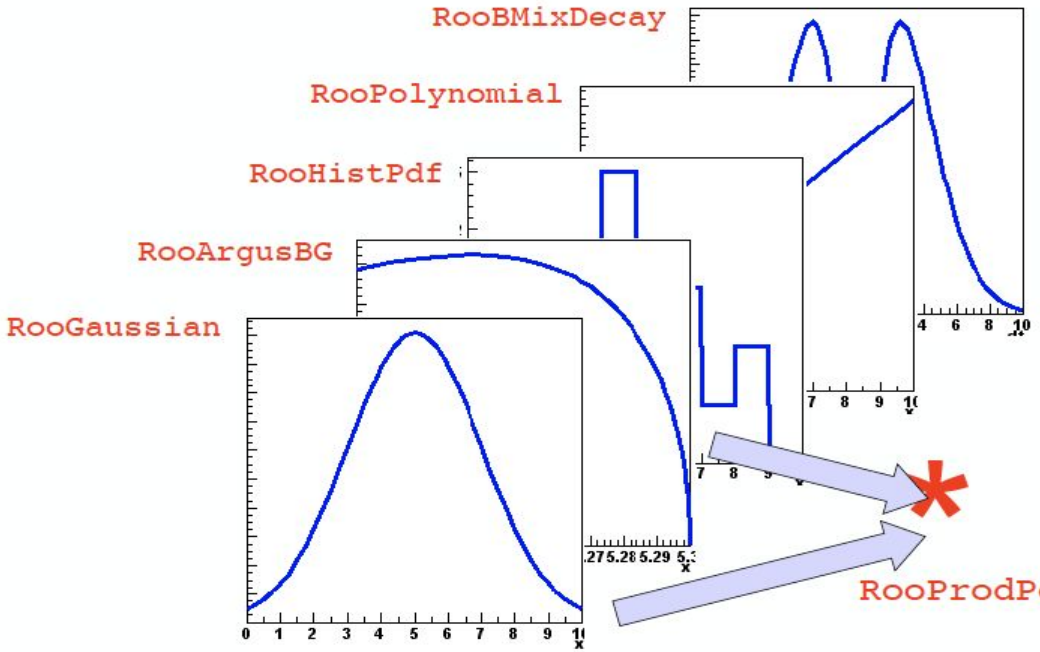
```
w.pdf("sum")->Print("t");
  RooAddPdf::sum[ g1frac * g1 + g2frac * g2 + [%] * argus ] = 0.0687785
    RooGaussian::g1[ x=x mean=mean1 sigma=sigma ] = 0.135335
    RooGaussian::g2[ x=x mean=mean2 sigma=sigma ] = 0.011109
    RooArgusBG::argus[ m=x m0=k c=9 p=0.5 ] = 0
```

– Pode-se também criar um arquivo de entrada para visualização - **GraphViz**

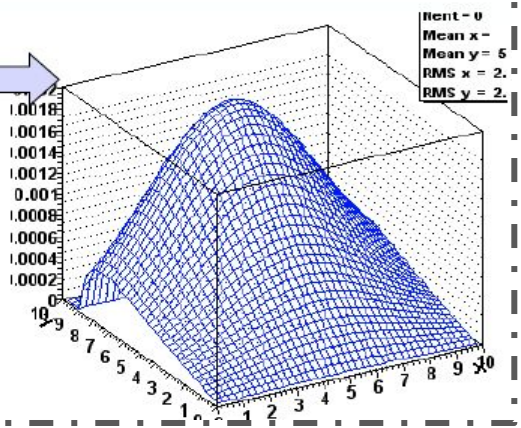
```
w.pdf("sum")->graphVizTree("myfile.dot");
```



# Produtos de p.d.f.s não correlacionadas




$$H(x,y) = F(x) \times G(y)$$



# Produtos não correlacionados - Matemática e Construtores

**2D**

$$H(x,y) = F(x) \times G(y)$$

**nD**

$$H(x^{\{i\}}) = \prod_i F^{\{i\}}(x^{\{i\}})$$

– Nenhuma normalização explicita necessária → Se as p.d.f.s de entrada são normalizadas à unidade, o produto também é normalizado à unidade

– A integração parcial e a geração de toy MC usam automaticamente as propriedades de fatoração, ex.: $\int H(x,y)dx \equiv G(y)$ é deduzida da estrutura.

- O operador correspondente na factory é o PROD

```
w.factory("Gaussian::gx(x[-5,5],mx[2],sx[1])");
w.factory("Gaussian::gy(y[-5,5],my[-2],sy[3])");
w.factory("PROD::gxy(gx,gy)");
```

# Construindo p.d.f.s conjuntas (RooSimultaneous)

- O operador class SIMUL constrói modelos conjuntos no nível pdf
  - precisa de um observável discreto (categoria) para rotular os canais

```
// Pdfs for channels 'A' and 'B'
w.factory("Gaussian::pdfA(x[-10,10],mean[-10,10],sigma[3])");
w.factory("Uniform::pdfB(x)");
// Create discrete observable to label channels
w.factory("index[A,B]");
// Create joint pdf (RooSimultaneous)
w.factory("SIMUL::joint(index,A=pdfA,B=pdfB)");
```

- Construir conjuntos de dados
  - contém observáveis ("x") e categoria ("index")

```
RooDataSet *dataA, *dataB ;
RooDataSet dataAB("dataAB","dataAB",
                  RooArgSet(*w.var("x"),*w.cat("index")),
                  Index(*w.cat("index")),
                  Import("A",*dataA),Import("B",*dataB));
```

# Construindo a likelihood

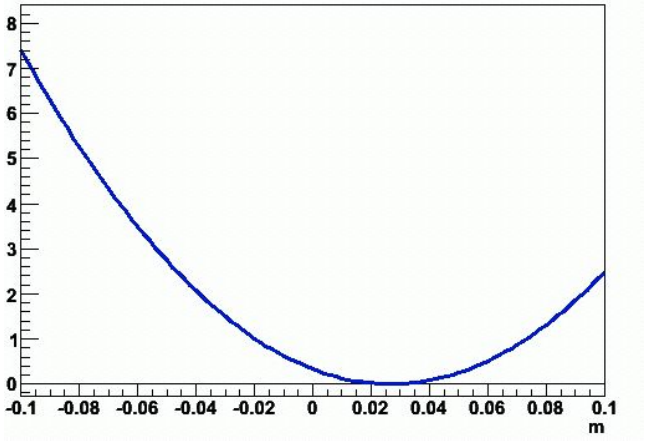
• Até agora o foco foi na construção de pdfs e no uso básico de ajuste e geração de eventos de toy MC

- Quando é feito o pdf->fitTo(*data), construímos um objeto que representa -log(L) da likelihood que é minimizada usando um algoritmo (ex.: MINUIT)

• Pode-se também construir explicitamente a função likelihood

– Pode usar (plotar, integrar) likelihood como qualquer objeto de função RooFit

```
RooAbsReal* nll = pdf->createNLL(data) ;
RooPlot* frame = parameter->frame() ;
nll->plotOn(frame,ShiftToZero()) ;
```



# Construindo a likelihood

- Exemplo – MINIMIZAÇÃO manual usando MINUIT
  – O resultado da minimização é imediatamente propagada para variável de objetos RooFit (valores e erros)

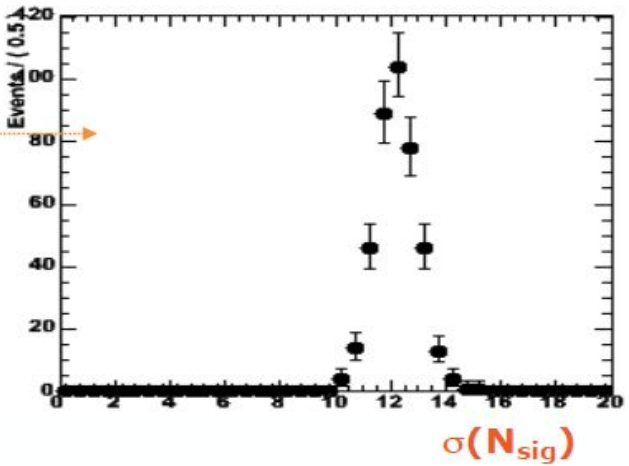
```
// Create likelihood (calculation parallelized on 8 cores)
RooAbsReal* nll = w::model.createNLL(data,NumCPU(8)) ;
RooMinimizer m(*nll) ; // create Minimizer class
m.minimize("Minuit2","Migrad"); // minimize using Minuit2
m.hesse() ; // Call HESSE
m.minos(w::param) ; // Call MINOS for 'param'
RooFitResult* r = m.save() ; // Save status (cov matrix etc)
```

  – Outros minimizadores suportados (Minuit, GSL, etc)
  – obs.: Minimizador diferente pode ser usado com `RooAbsPdf::fitTo`

```
//fit a pdf to a data set using Minuit2 as minimizer
pdf.fitTo(*data, RooFit::Minimizer("Minuit2","Migrad")) ;
```

# 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***

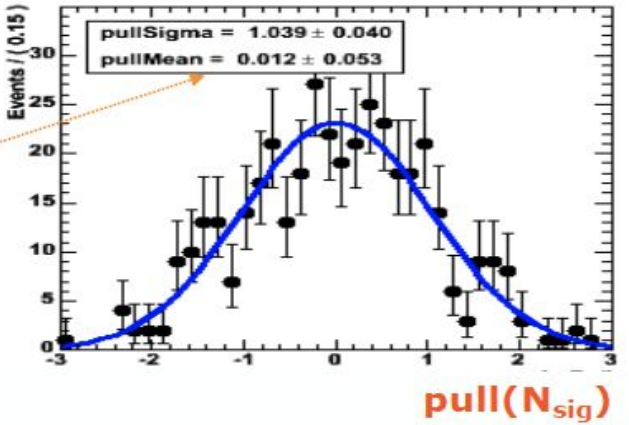
Definição:

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

– Propriedades da pull: `frame->makePullHist();`

- 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



# Exercício 2

- Construa uma PDF para as ressonâncias J/ψ and ψ(2S) + background
  - J/ψ com uma função Crystal Ball
  - ψ(2S) com uma similar(spoiler!) Crystal Ball
  - Background com uma polinomial

- obs.: A ψ(2S) envolverá uma pequena quantidade de eventos de sinal
- Fit, plot e salve

O arquivo de entrada está aqui:
https://cernbox.cern.ch/index.php/s/DInqlmV9W52WPvY

# 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**
  - –scale para 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)

# Documentação do RooFit

–ponto inicial: http://root.cern.ch/drupal/content/roofit
–Users manual (134 pages ~ 1 year old)
–Quick Start Guide (20 pages, recent)
–Link to 84 tutorial macros (also in $ROOTSYS/tutorials/roofit)
–More than 200 slides from W. Verkerke documenting all features are available at the French School of Statistics 2008

- http://indico.in2p3.fr/getFile.py/access?contribId=15&resId=0&materialId=slides&confId=750

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

https://github.com/sandrofonseca/rootFitTutorial/tree/master/roofitUERJ

# Backup

# Composition of p.d.f.s

RooFit pdf building blocks do not require variables as input, just real-valued functions

– Can substitute any variable with a function expression in parameters and/or observables

$$f(x;p) \Rightarrow f(x,p(y,q)) = f(x,y;q)$$

– Example: Gaussian with shifting mean

```
w.factory("expr::mean('a*y+b',y[-10,10],a[0.7],b[0.3])") ;
w.factory("Gaussian::g(x[-10,10],mean,sigma[3])") ;
```

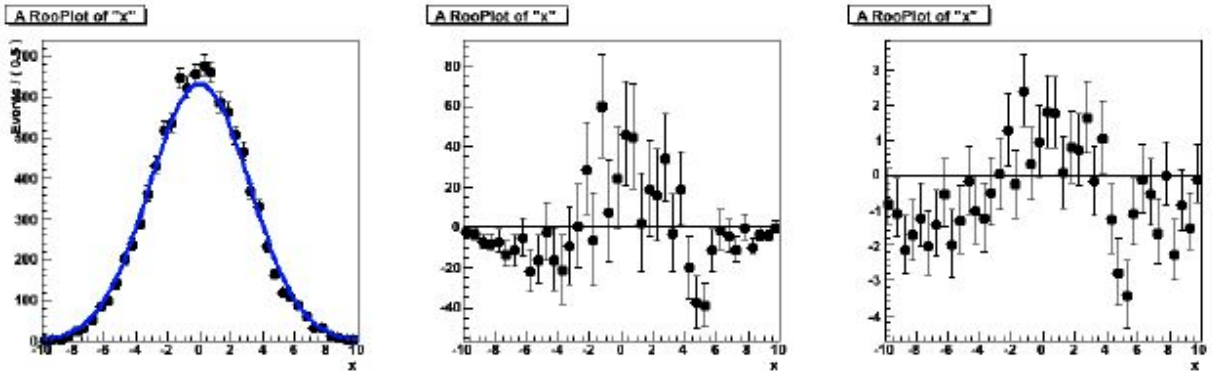
– No assumption made in function on a,b,x,y being observables or parameters, any combination will work

# How do you know if your fit was “good”?

- Goodness-of-fit broad issue in statistics (we will see maybe later)

- For one-dimensional fits, a $\chi^2$ is usually the right thing to do
  - –Some tools implemented in RooPlot to be able to calculate $\chi^2$/ndf of curve w.r.t data

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




  - –Also tools exists to plot residual and pull distributions from curve and histogram in a RooPlot

```
frame->makePullHist();
frame->makeResidHist();
```

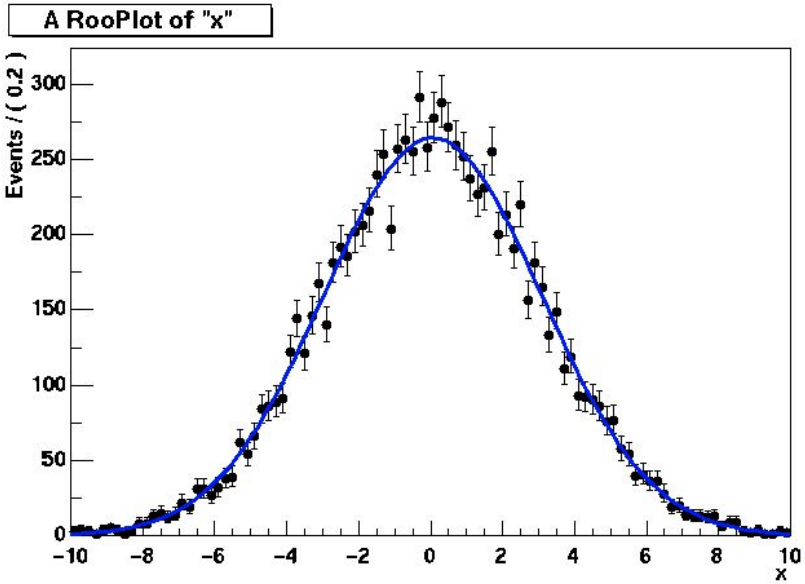
# Fitting os dados

- Ajuste do modelo aos dados
  - –ex.: unbinned maximum likelihood fit

```
pdf = pdf->fitTo(data);
```

- visualização dos dados e pdf após o fit

```
RooPlot * xframe = x->frame();
data->plotOn(xframe);
pdf->plotOn(xframe);
xframe->Draw();
```




PDF automaticamente
normalizada ao dataset