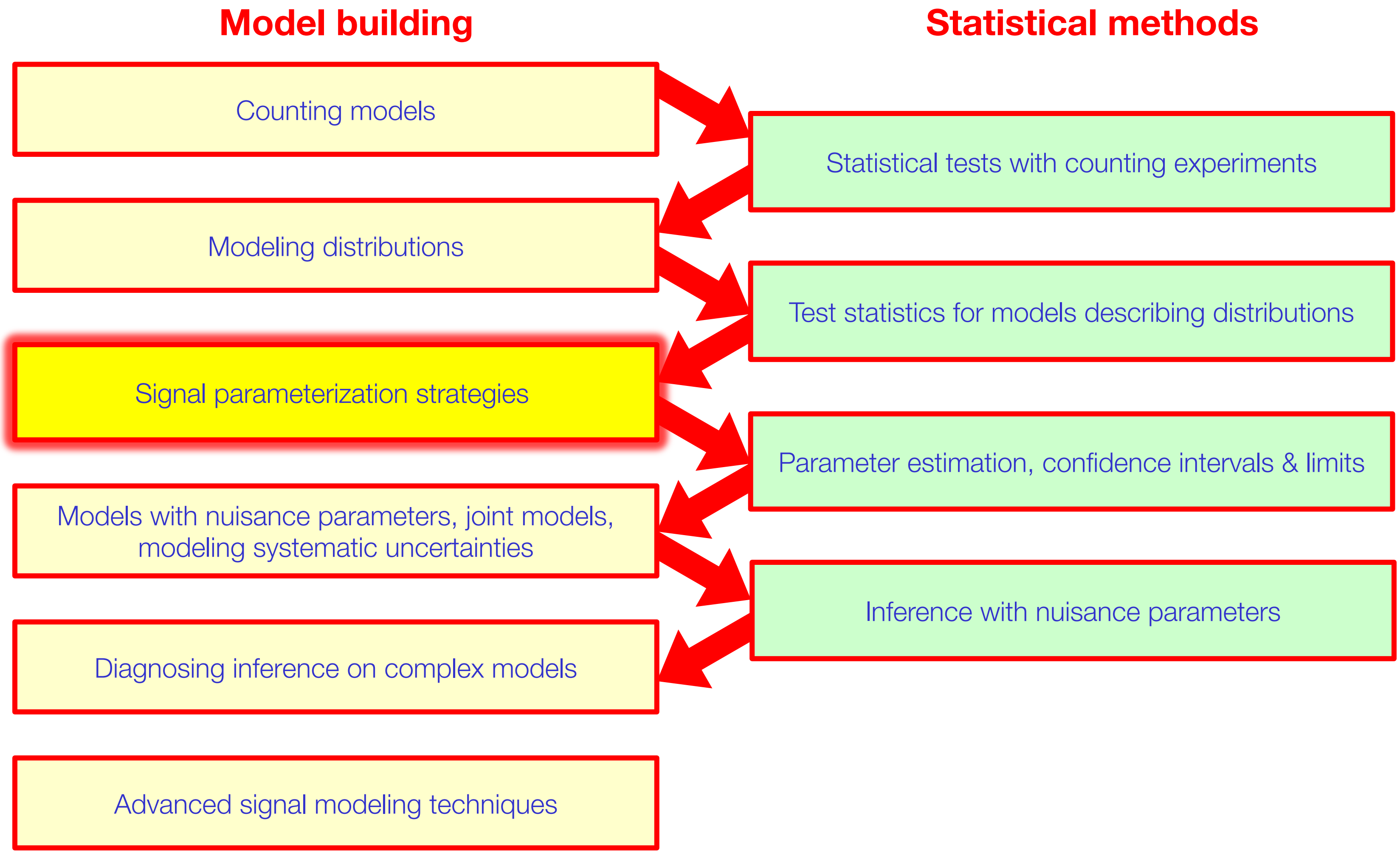
Model building 3

Models with parameters I -
analytical parametric models,
multi-dimensional models
template morphing approach for
histogram-based models

Wouter Verkerke, NIKHEF

Roadmap of this course

- Start with basics, gradually build up to complexity

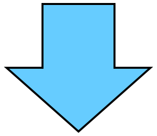


Introduce concept of composite hypotheses

- In most cases in physics, a hypothesis is not “simple”, but “composite”
- **Composite hypothesis** = Any hypothesis which does *not* specify the population distribution completely
- Example: counting experiment with signal and background, that leaves signal expectation unspecified

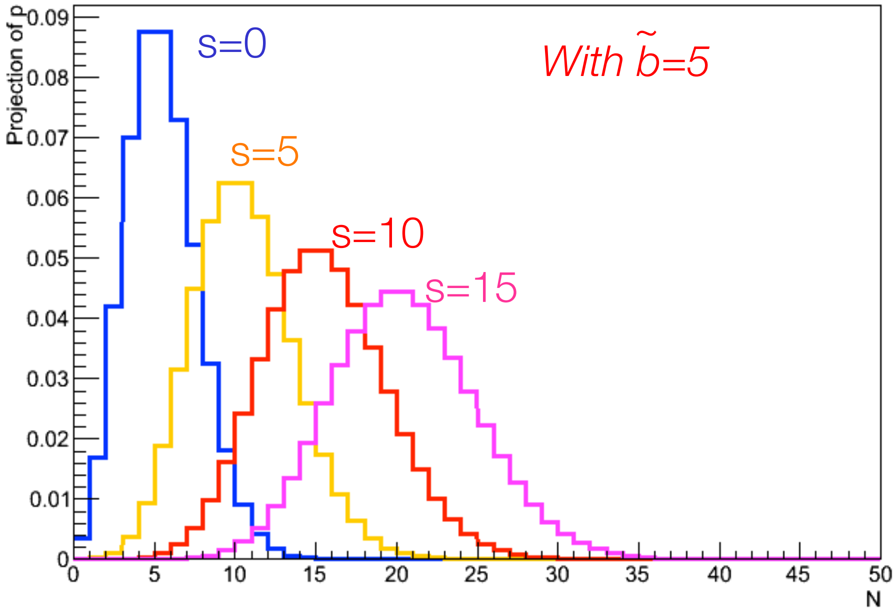
Simple hypothesis

$$L = Poisson(N \mid \tilde{s} + \tilde{b})$$



$$L(s) = Poisson(N \mid s + \tilde{b})$$

Composite hypothesis



(My) notation convention: all symbols with ~ are constants

A common convention in the meaning of model parameters

- A common convention is to recast signal rate parameters into a normalized form (e.g. w.r.t the Standard Model rate)

Simple hypothesis

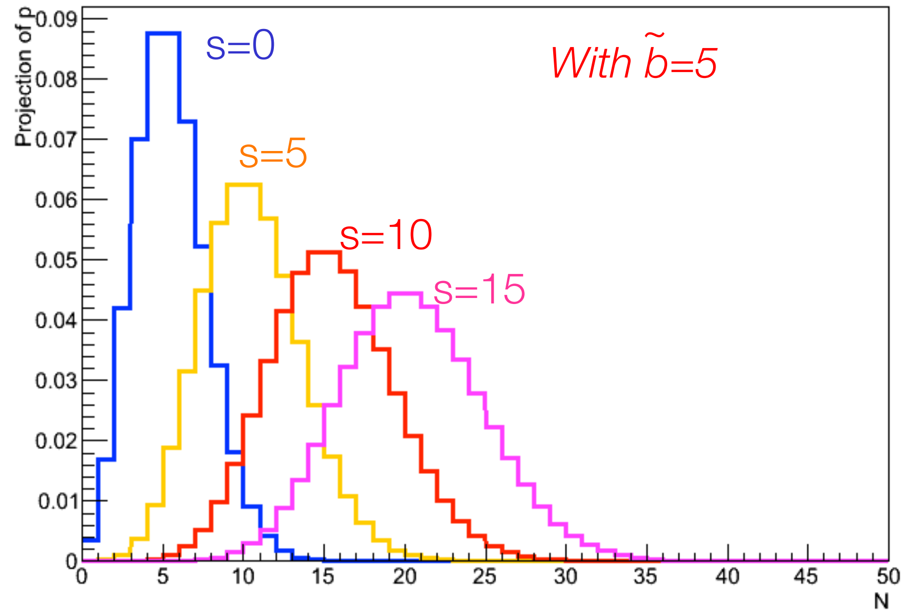
$$L = Poisson(N \mid \tilde{s} + \tilde{b})$$

$$L(s) = Poisson(N \mid s + \tilde{b})$$

Composite hypothesis

$$L(\mu) = Poisson(N \mid \mu \cdot \tilde{s} + \tilde{b})$$

Composite hypothesis
with normalized rate parameter



'Universal' parameter interpretation makes it easier to work with your models

μ=0 → no signal
μ=1 → expected signal
μ>1 → more than expected signal

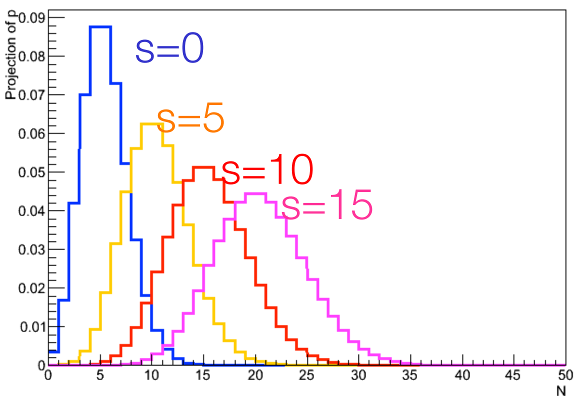
What can we do with composite hypothesis

- With simple hypotheses – inference is restricted to making statements about P(D|hypo) or P(hypo|D)
- With composite hypotheses – many more options
- 1 Parameter estimation and variance estimation
  - What is value of ***s*** for which the observed data is most probable?
  - What is the variance (std deviation squared) in the estimate of ***s?***

  } s=5.5 ± 1.3
- 2 Confidence intervals
  - Statements about model parameters using frequentist concept of probability
  - s<12.7 at 95% confidence level
  - 4.5 < s < 6.8 at 68% confidence level
- 3 Bayesian credible intervals
  - Bayesian statements about model parameters
  - s<12.7 at 95% credibility

Model building for discovery, X-section → yield parameter

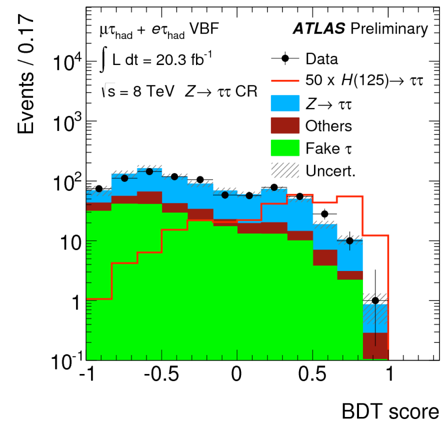
0-dimensional (counting)



Poisson(N|**S**+B)

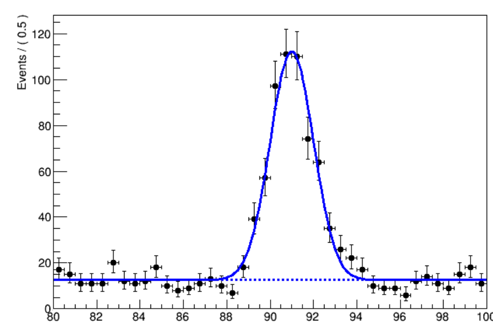
1-dimensional (discriminant)

MVA discriminant



S*sig(x)+B*bkg(x)

Physics-inspired discriminant

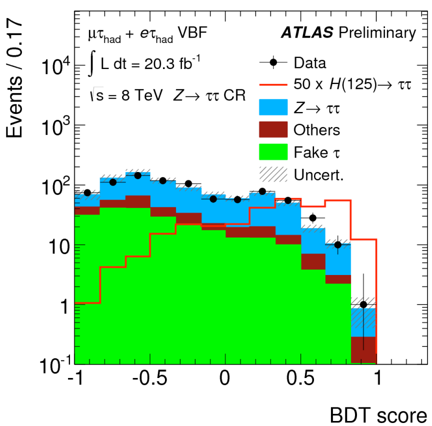


S*sig(x)+B*bkg(x)

Models for discovery, X-section → yield parameter

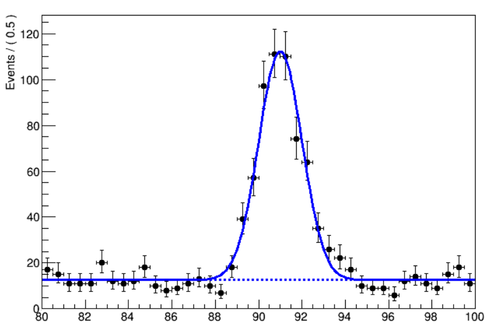
1-dimensional (discriminant)

MVA discriminant



S*sig(x)+B*bkg(x)

Physics-inspired discriminant



S*sig(x)+B*bkg(x)

2-dimensional?

Q: When is it useful to build probability models in ≥2 observables?

A1: When you have a physics model with a clear prediction for the full 2D model..

Often you don't and then you let an MVA reduce the n-Dim space to 1-dimension

But sometimes you have clear models described 2 or more observables → No point in letting an MVA approximate what you know analytically.

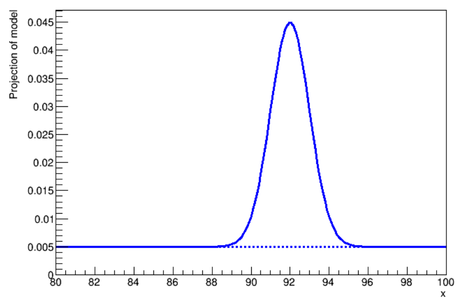
'ex04.C'

Case study – dependence of 1-D model on another observable

- A common scenario for 2D modelling is the following: You observe that the mean reconstructed mass of some particle depends on another observable

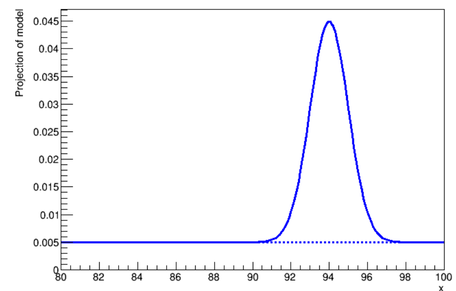
Model for mass at (y=0)

sig(*m*)=Gaussian(*m*,*92*,*1*)



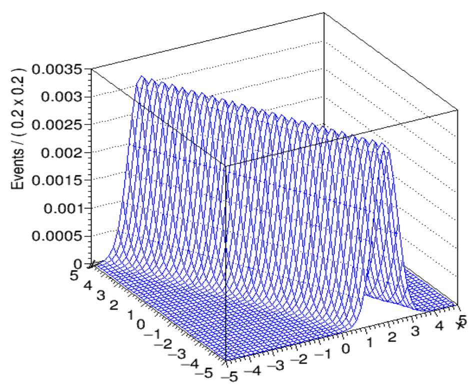
Model for mass at (y=3)

sig(*m*)=Gaussian(*m*,*94*,*1*)



sig(*m*,*y*)=Gaussian(*m*,**mean(y)**,*1*)

Solution: introduce a function **mean(y)** that describes dependence of mean of y

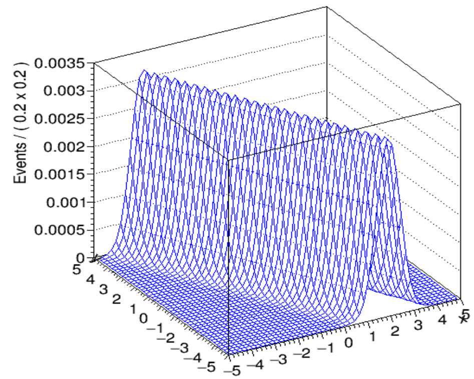


Q: Is sig(m,y) a proper 2-dimensional model?

Case study – dependence of 1-D model on another observable

sig(m,y)=Gaussian(m,**mean(y)**, 1)

Solution: introduce a function **mean(y)** that describes dependence of mean of y



Q: Is sig(m,y) a proper 2-dimensional model?

A: No! Distribution in y is unlikely to be flat…

- Challenge for 2D models: distributions in x,y and all correlations must all be correct! Seems intractable, but solutions exists
- Instead of immediately defining a 2D model **f(x,y)**, define first the *conditional* probability density function **f(x|y)**

f(x,y) = 2D model for both x and y

$$\int f(x,y)\,dx\,dy \equiv 1$$

f(x|y) = 1D model for x at a given value of y

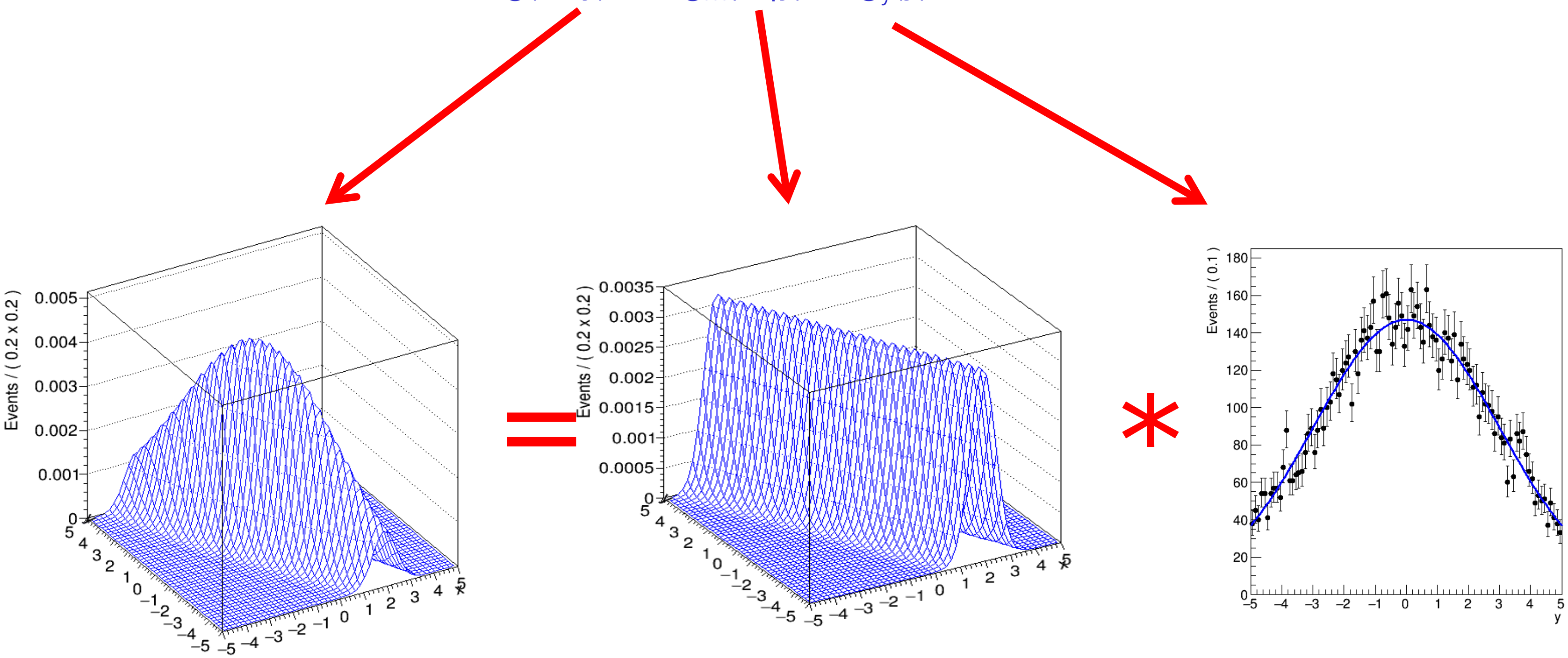
$$\int f(x,y)\,dx \equiv 1 \quad \forall y$$

This is really what we meant when we formulated this:
Gaussian(m,**mean(y)**, 1)

Case study – dependence of 1-D model on another observable

- Given a conditional model f(x|y) can build full 2D model by multiplying with a model g(y)

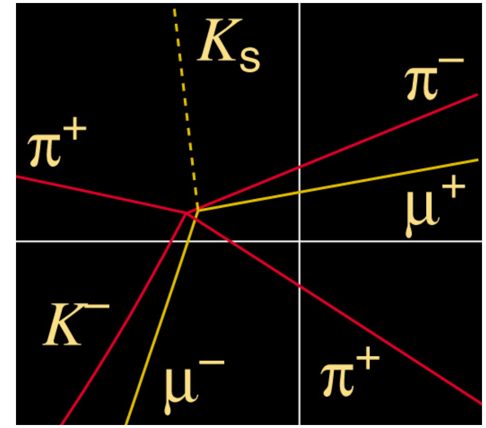
$\text{sig}(m,y) = \text{sig}_m(m|y) * \text{sig}_y(y)$



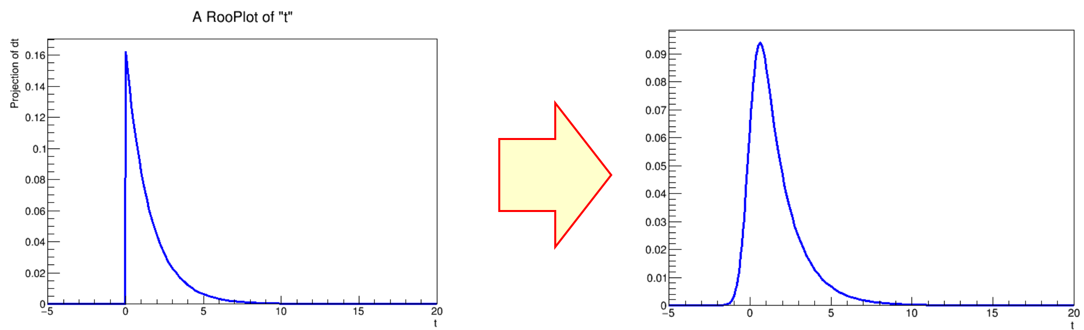
Gaussian(*m*, **mean(y)**, *1*)

Gaussian(*y*)

Case study – per-event errors



- Another common variant of this type of modeling problem is the so-called 'per-event' error
- Example: observable = decay time distribution, measured from reconstructed vertex.
  - In absence of a detector resolution, exponential decay distribution
  - In real life, distribution is convoluted with (Gaussian) reconstruction resolution



- But vertex reconstruction gives also estimate of uncertainty for every reconstructed vertex → the 'per-event error'
  - Can take this into account: well-reconstructed events carry more information
- How? Scale assumed resolution with per-event error

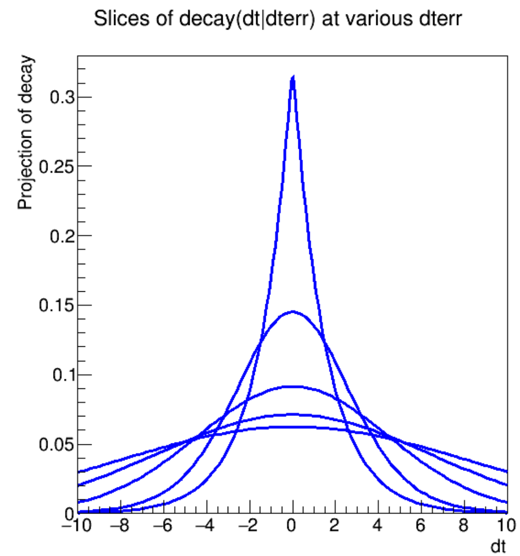
$$f(t\,|\,\delta t) = Decay(t) \otimes Gaussian(t, 0, \boxed{\sigma \cdot \delta t})$$

Case study – per-event errors

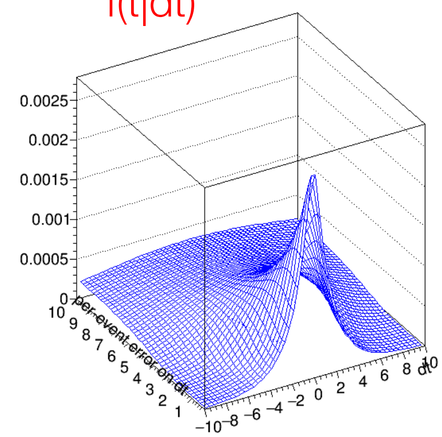
- Visualization of decay function with variable resolution

Decay function (symmetrized)
convoluted with Gaussian resolution
at 4 different values of per-event error

$$f(t\,|\,\delta t) = Decay(t) \otimes Gaussian(t, 0, \sigma \cdot \delta t)$$



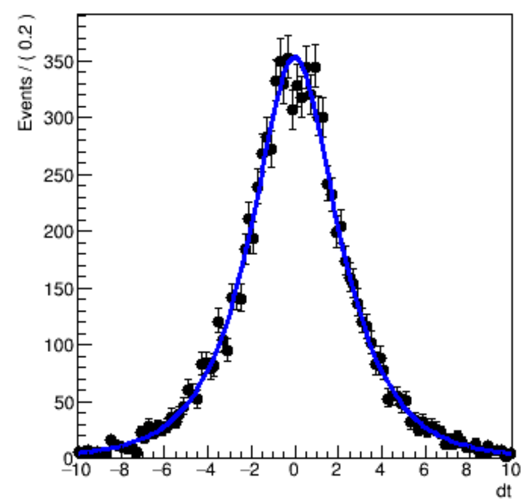
f(t|dt)



Gain: high-resolution events carrymore weight in likelihood → better estimate of model parameters

Full 2D-model:
F(t,dt) = F_1(t|dt)*F_2(dt)

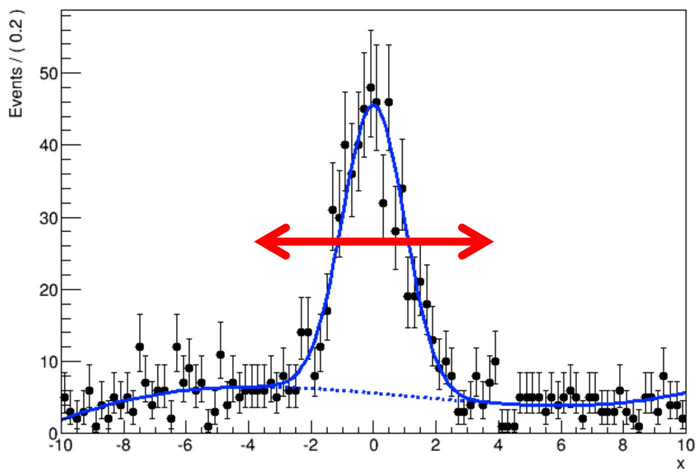
Shown here: *projection* on t
F(t) = Int [F_1(t|dt)*F_2(dt)] dt



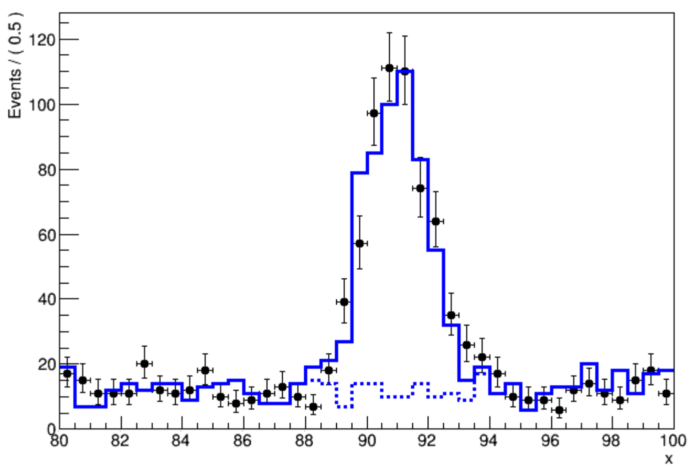
Model building for measurements → shape parameter

- Beyond discovery/rate measurements, can also build models to measure properties of particles (e.g mass)
  → introduce shape parameters
- Often trivial for analytical models,
  less so for simulation-based models

F(x|**m**) = Gaussian(x,**m**,σ)+bkg

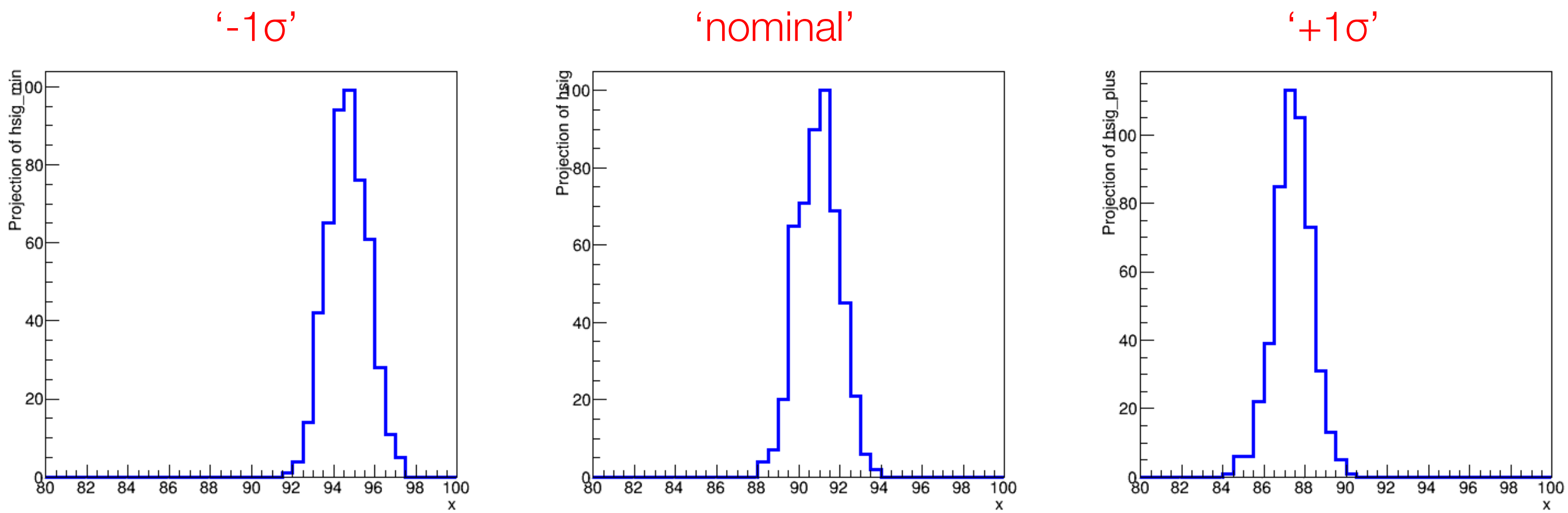


F(x|**m**) = ??



Modeling of shape variations in the likelihood

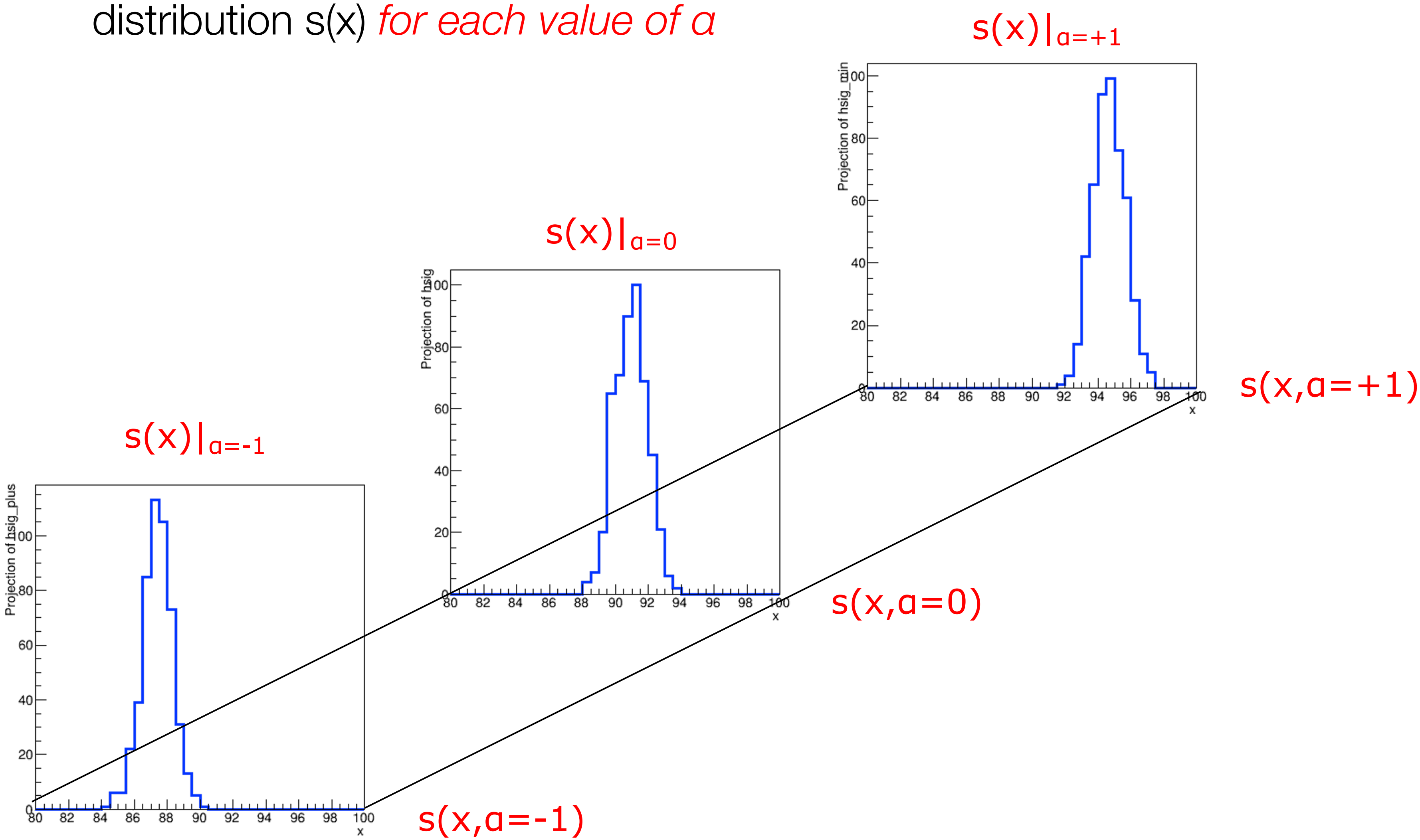
- If underlying simulation has free parameter θ, can assess impact on reconstructed shapes by rerunning simulation at different values
  - Obtain histogram templates for distributions at '+1σ' and '-1σ' settings of systematic effect



- Challenge: **construct an empirical response function based on the interpolation of the shapes of these three templates**.

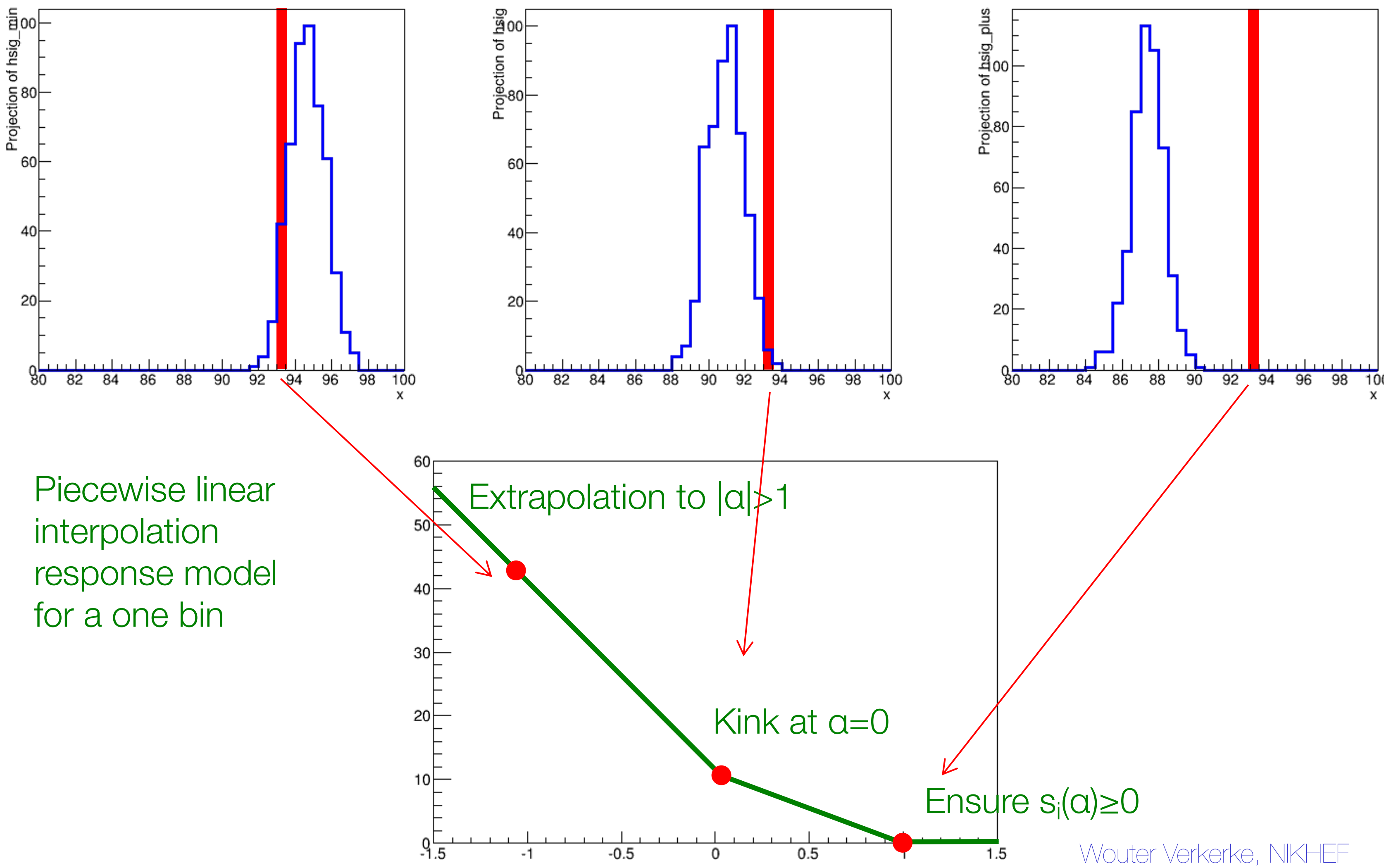
Need to interpolate between template models

- Need to define 'morphing' algorithm to define distribution s(x) *for each value of α*



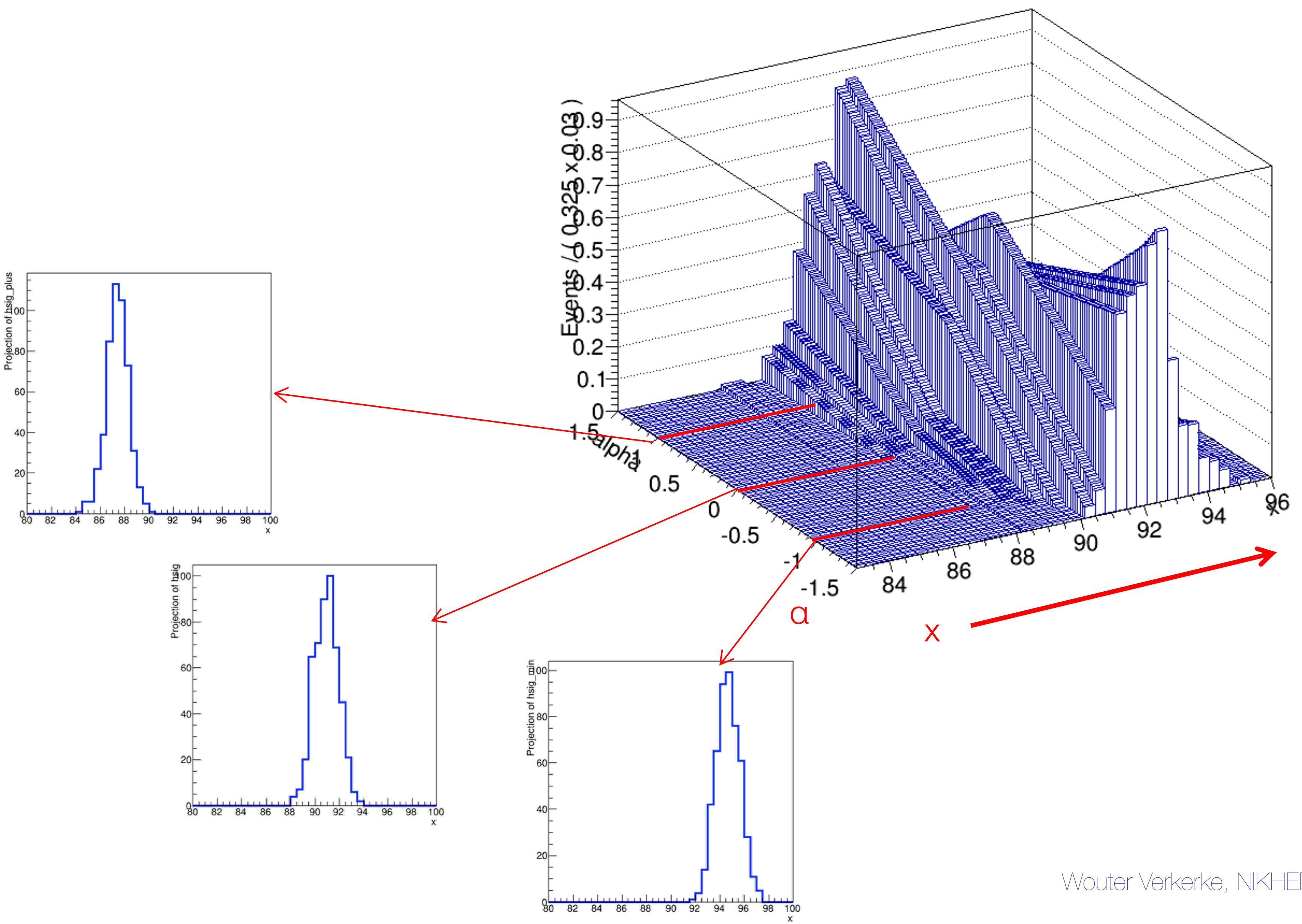
Piecewise linear interpolation

- Simplest solution is piece-wise linear interpolation for each bin



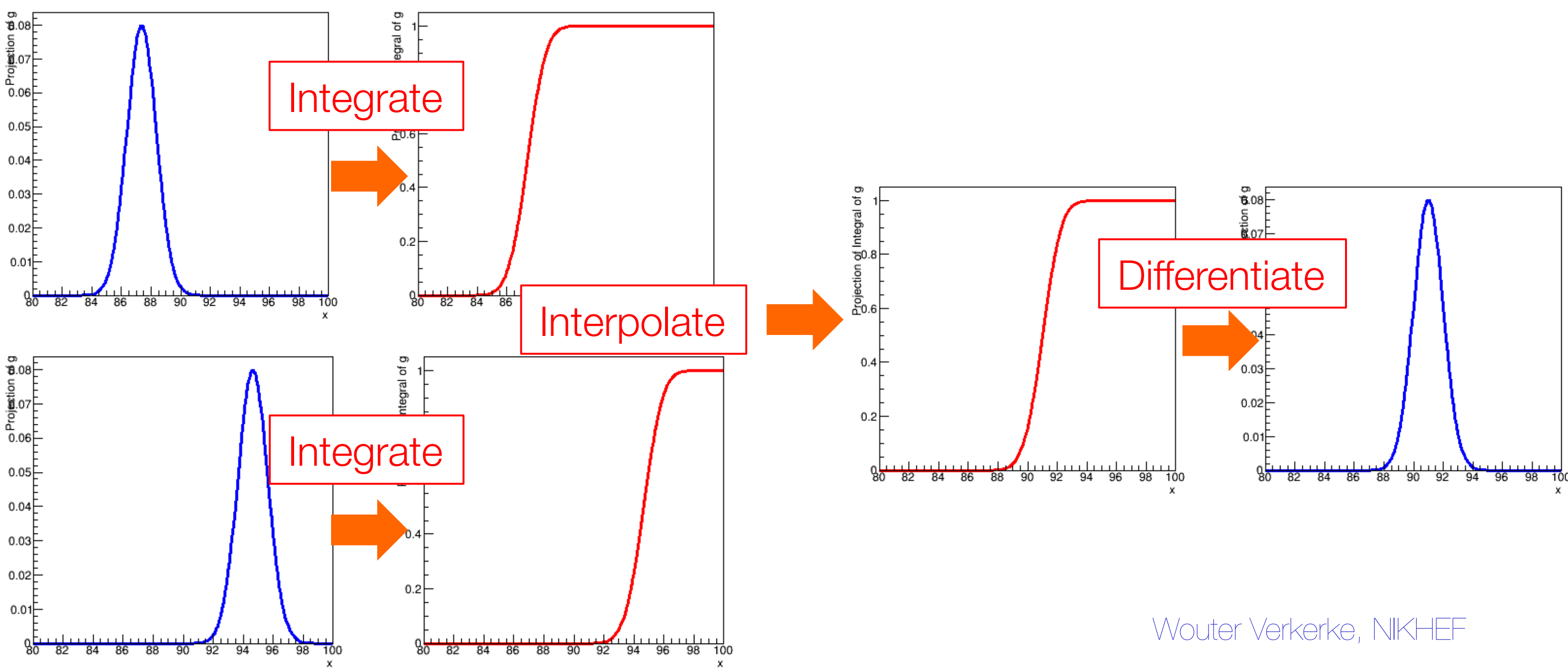
Piecewise linear interpolation response model for a one bin ‘ex15_build_binned_morphing.C’

Visualization of bin-by-bin linear interpolation of distribution



Other morphing strategies – ‘horizontal morphing’

- Other template morphing strategies exist that are less prone to unintended side effects
- A ‘horizontal morphing’ strategy was invented by Alex Read.
  - Interpolates the cumulative distribution function instead of the distribution
  - Especially suitable for shifting distributions
  - Here shown on a continuous distribution, but also works on histograms
  - Drawback: computationally expensive, algorithm only worked out for 1 NP



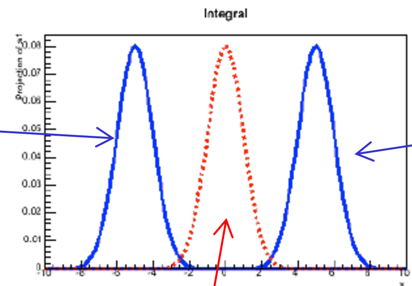
Yet another morphing strategy – 'Moment morphing'

M. Baak & S. Gadatsch

- Given two template model $f_-(x)$ and $f_+(x)$ the strategy of moment morphing considers first two moment of template models (mean and variance)

$$\mu_- = \int x \cdot f_-(x)dx$$

$$V_- = \int (x-\mu_-)^2 \cdot f_-(x)dx$$



$$\mu_+ = \int x \cdot f_+(x)dx$$

$$V_+ = \int (x-\mu_+)^2 \cdot f_+(x)dx$$

- The goal of moment morphing is to construct an interpolated function that has linearly interpolated moments

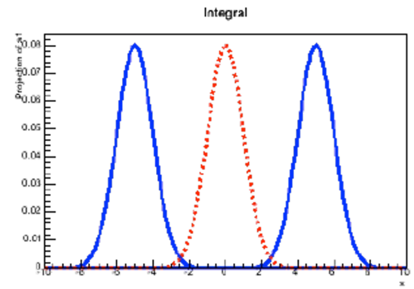
$$\mu(\alpha) = \alpha\mu_- + (1-\alpha)\mu_+$$
$$V(\alpha) = \alpha V_- + (1-\alpha)V_+ \quad [1]$$

- It constructs this morphed function as combination of linearly transformed input models

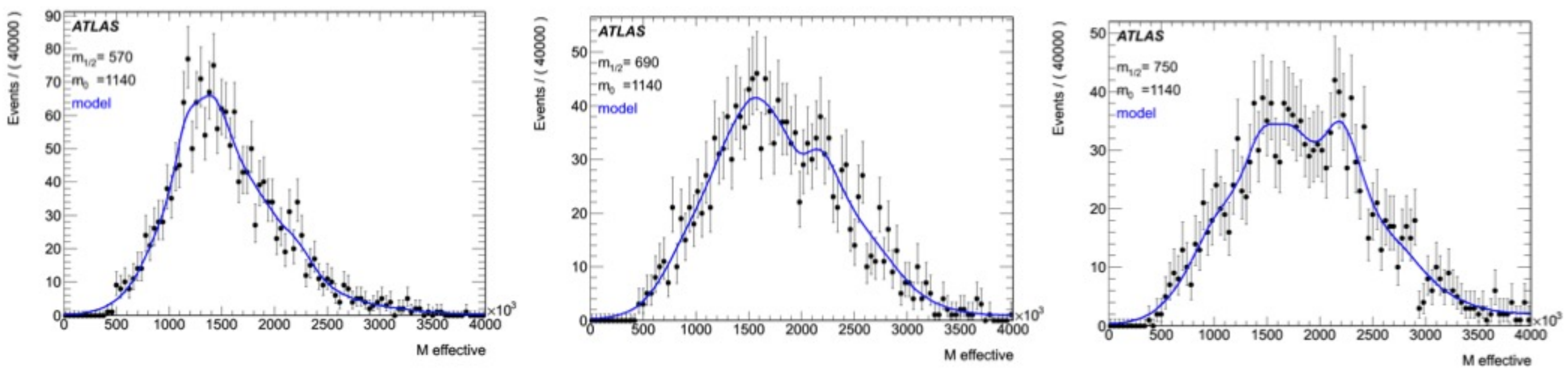
$$f(x,\alpha) \rightarrow \alpha f_-(ax+b) + (1-\alpha)f_+(cx-d)$$

  – Where constants a,b,c,d are chosen such so that f(x,α) satisfies conditions [1]

Yet another morphing strategy – 'Moment morphing'



- For a Gaussian probability model with linearly changing mean and width, moment morphing of two Gaussian templates is the exact solution
- But also works well on 'difficult' distributions



- Good computational performance
  - Calculation of moments of templates is expensive, but just needs to be done once, otherwise very fast (just linear algebra)

$$f(x,\alpha) \to \alpha f_{-}(ax+b) + (1-\alpha) f_{+}(cx-d)$$

- Multi-dimensional interpolation strategies exist

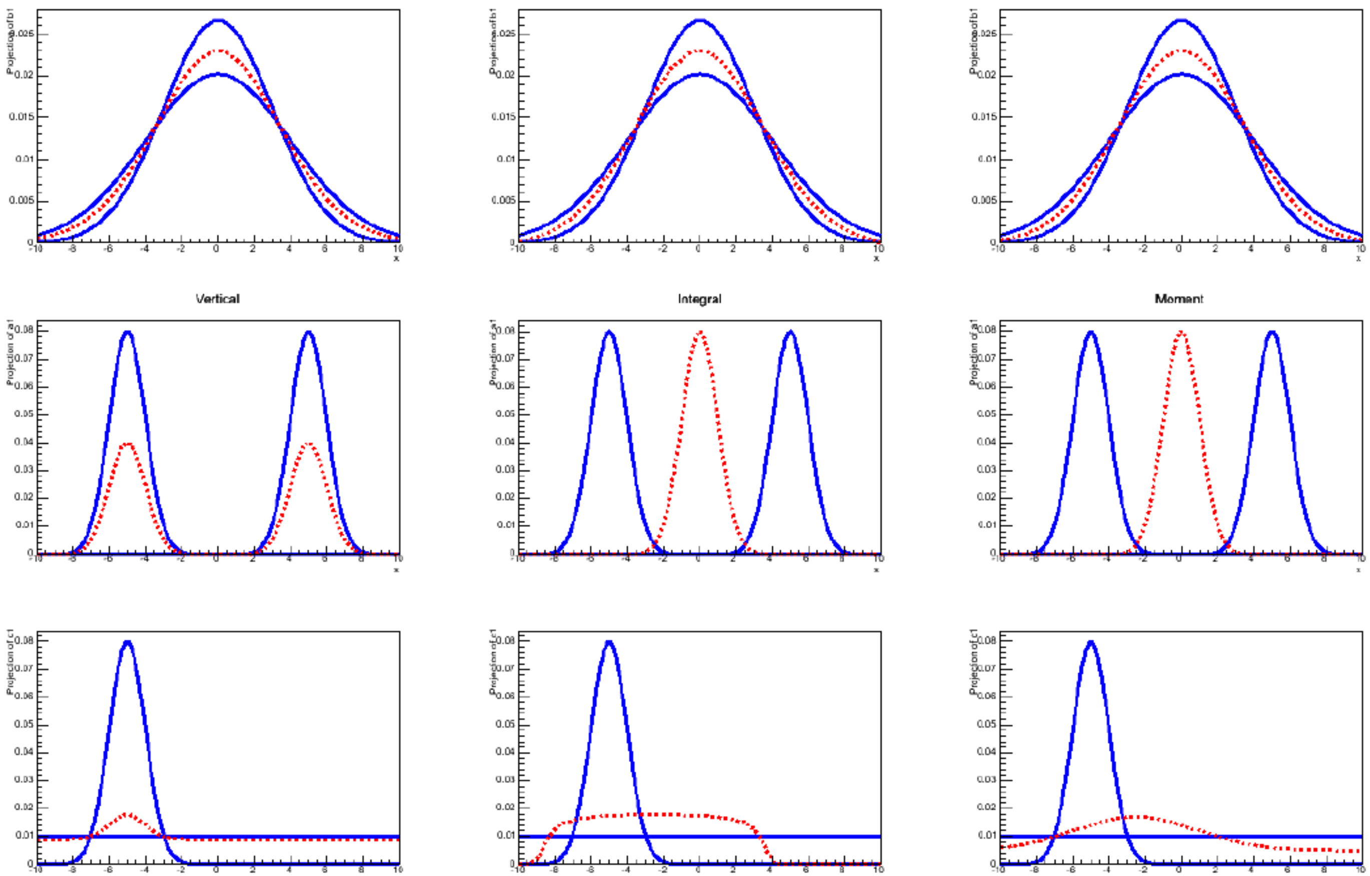
There are other morphing algorithms to choose from

Vertical Morphing | Horizontal Morphing | Moment Morphing

Gaussian varying width

Gaussian varying mean

Gaussian to Uniform (this is conceptually ambigous!)



n-dimensional morphing?

Vertical Morphing: ✓ | Horizontal Morphing: X | Moment Morphing: ✓

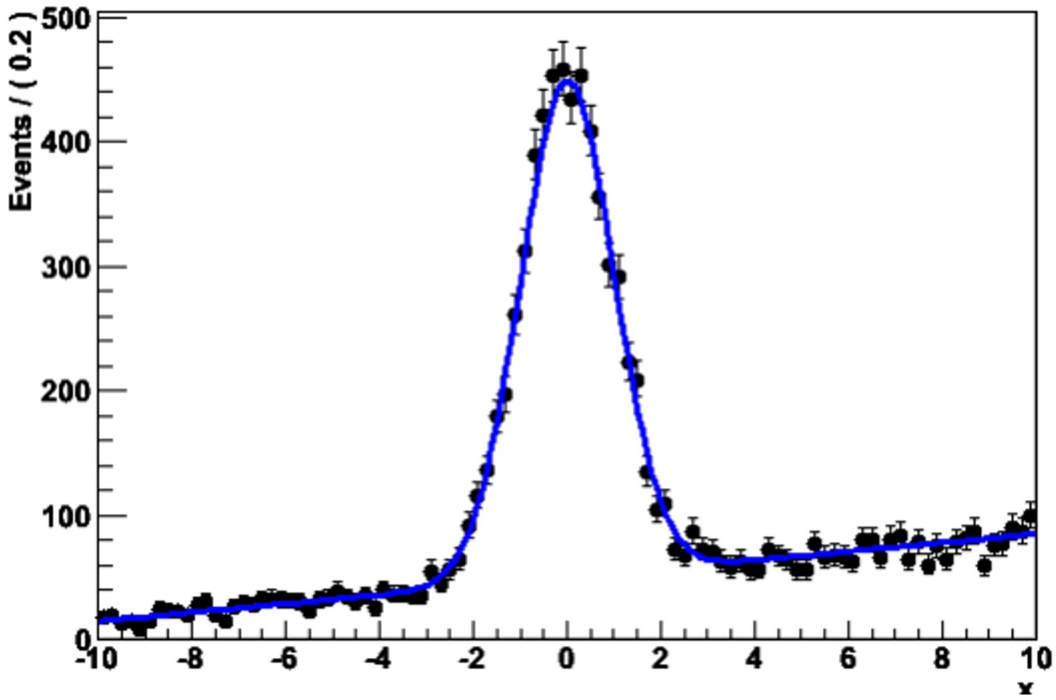
Software tools 1

Basic RooFit modeling

Wouter Verkerke, NIKHEF

RooFit – Focus: coding likelihood functions

- Focus on one practical aspect of many data analysis in HEP: How do you formulate your likelihood functions in ROOT
  - For 'simple' problems (gauss, polynomial) this is easy



  - But if you want to do unbinned ML fits, use non-trivial functions, or work with multidimensional functions you quickly find that you need some tools to help you

RooFit core design philosophy

- Mathematical objects are represented as C++ objects

Mathematical concept			RooFit class
variable	x	→	`RooRealVar`
function	$f(x)$	→	`RooAbsReal`
PDF	$f(x)$	→	`RooAbsPdf`
space point	$\vec{x}$	→	`RooArgSet`
integral	$\int_{x_{\text{min}}}^{x_{\text{max}}} f(x)dx$	→	`RooRealIntegral`
list of space points		→	`RooAbsData`

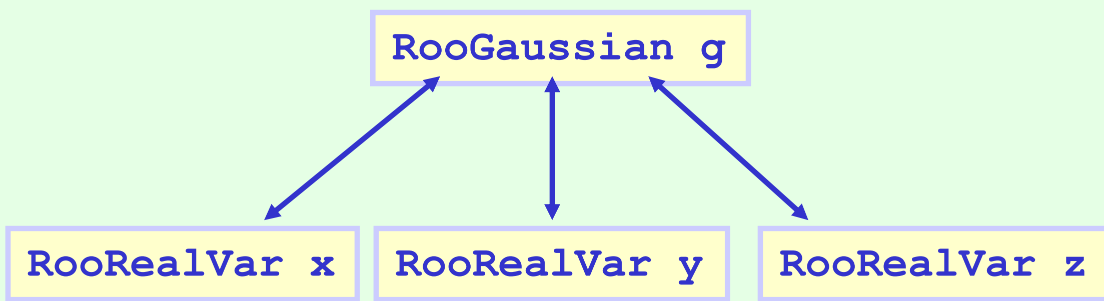
RooFit core design philosophy - Workspace

- Instead of **`'double Likelihood(double paramVec[])'`**, a flexible modular structure of 'programmed' functions

Math

$$\mathrm{Gauss}(x,\mu,\sigma)$$

RooFit diagram



RooFit code

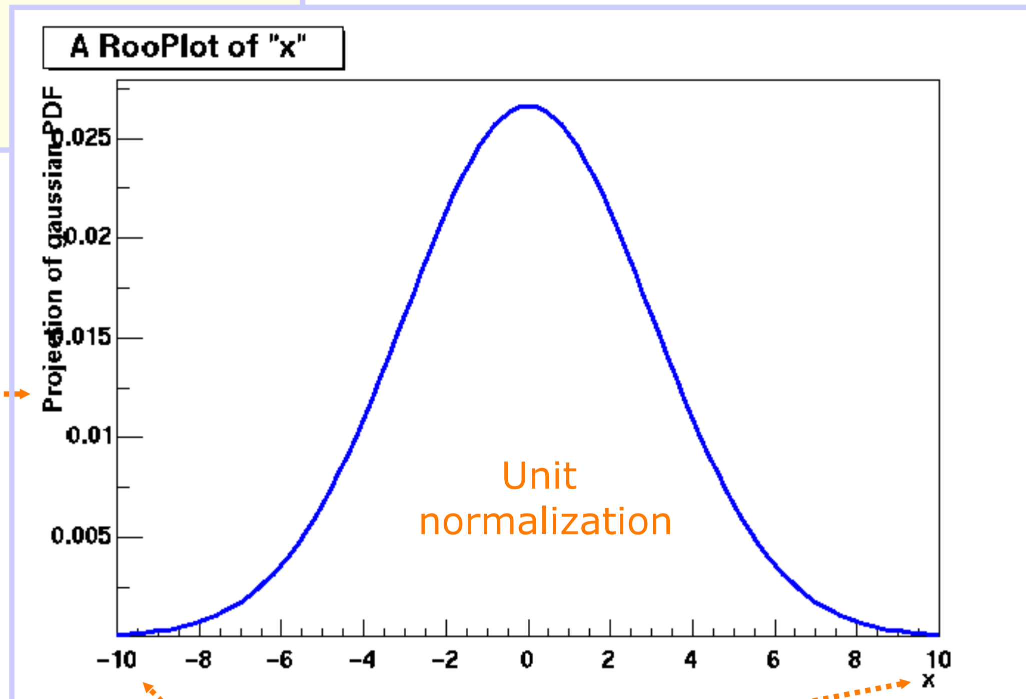
```
RooRealVar x("x","x",-10,10) ;
RooRealVar m("m","y",0,-10,10) ;
RooRealVar s("s","z",3,0.1,10) ;
RooGaussian g("g","g",x,m,s) ;
```

'ex02.C'

Basics – Creating and plotting a Gaussian p.d.f

Setup gaussian PDF and plot

```
// Create an empty plot frame
RooPlot* xframe = w::x.frame() ;

// Plot model on frame
model.plotOn(xframe) ;

// Draw frame on canvas
xframe->Draw() ;
```



Axis label from `gauss` title

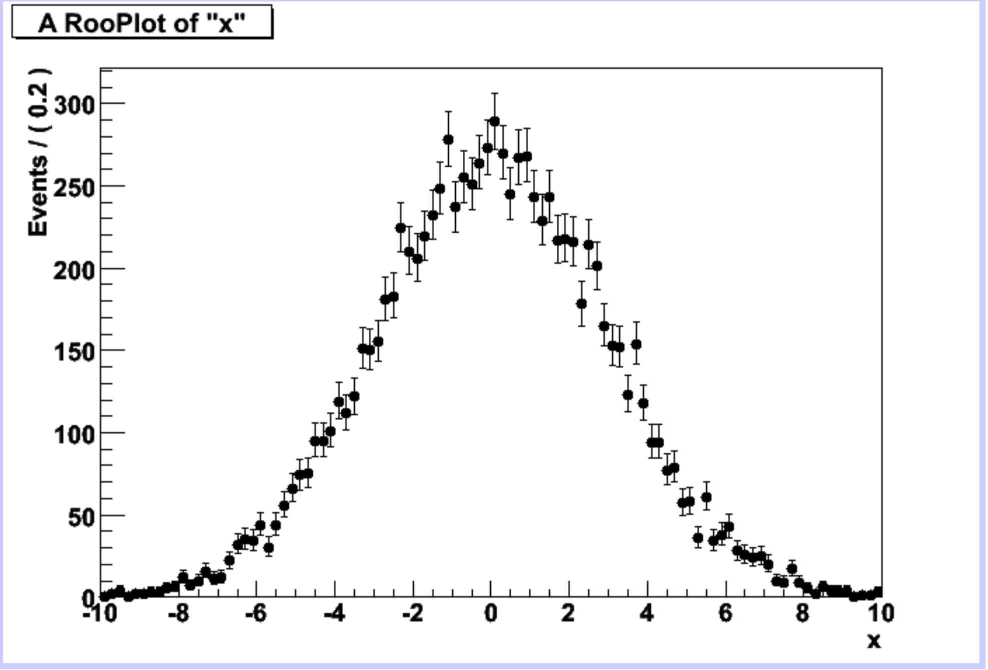
A `RooPlot` is an empty frame capable of holding anything plotted versus it variable

Plot range taken from limits of `x`

Basics – Generating toy MC events

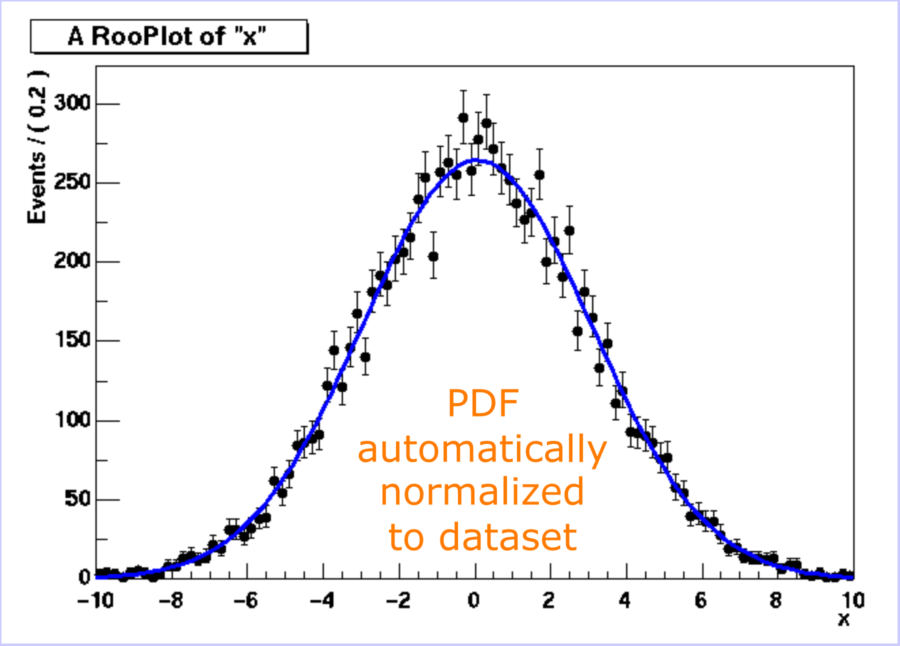
Generate 10000 events from Gaussian p.d.f and show distribution

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

// Generate an binned toy MC set
RooDataHist* data = w::gauss.generateBinned(w::x,10000) ;

// Plot PDF
RooPlot* xframe = w::x.frame() ;
data->plotOn(xframe) ;
xframe->Draw() ;
```



Can generate both binned and unbinned datasets

Basics – ML fit of p.d.f to *unbinned* data




```
// ML fit of gauss to data
w::gauss.fitTo(*data) ;
(MINUIT printout omitted)

// Parameters if gauss now
// reflect fitted values
w::mean.Print()
RooRealVar::mean = 0.0172335 +/- 0.0299542
w::sigma.Print()
RooRealVar::sigma = 2.98094  +/- 0.0217306

// Plot fitted PDF and toy data overlaid
RooPlot* xframe = w::x.frame() ;
data->plotOn(xframe) ;
w::gauss.plotOn(xframe) ;
```

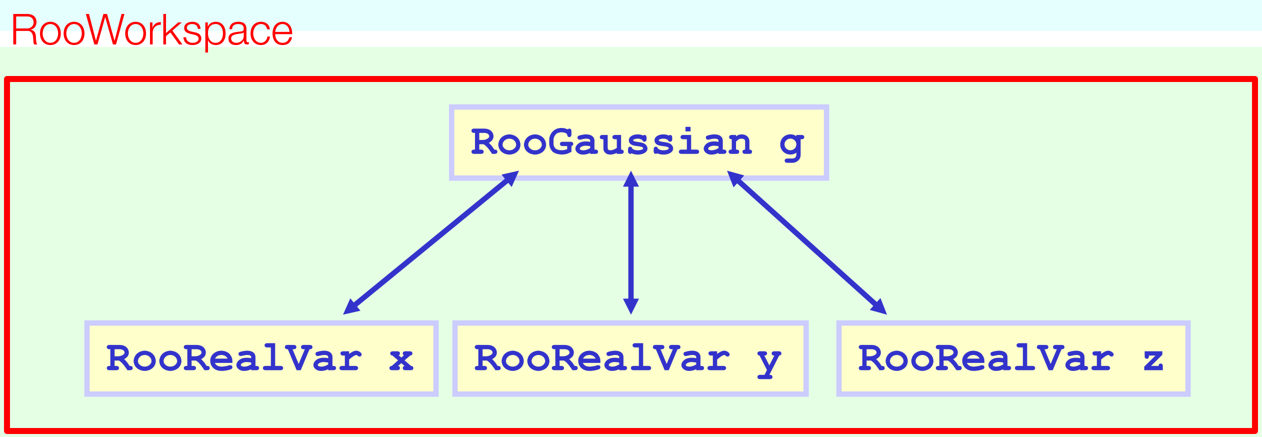
RooFit core design philosophy - Workspace

- The workspace serves a container class for all objects created

Math

$$\mathrm{Gauss}(x,\mu,\sigma)$$

RooFit diagram

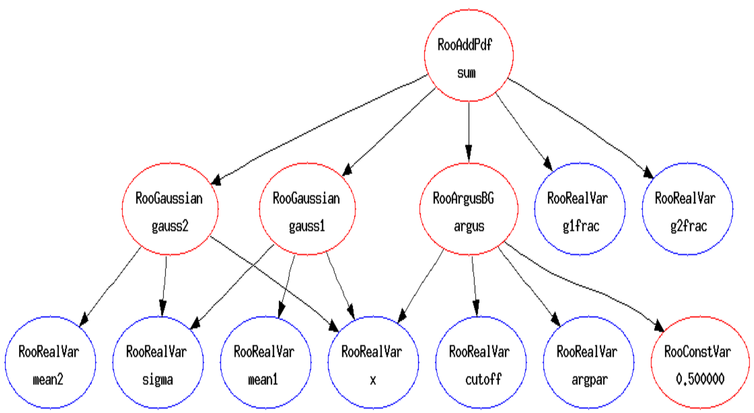


RooFit code

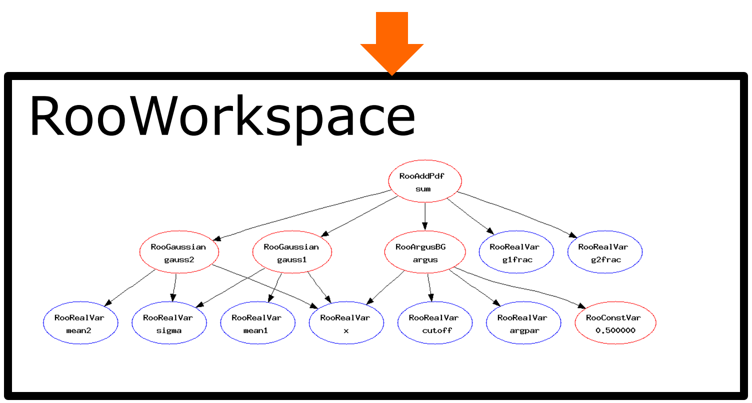
```
RooRealVar x("x","x",-10,10) ;
RooRealVar m("m","y",0,-10,10) ;
RooRealVar s("s","z",3,0.1,10) ;
RooGaussian g("g","g",x,m,s) ;
RooWorkspace w("w") ;
w.import(g) ;
```

The workspace

- The workspace concept has revolutionized the way people share and combine analysis
  - Completely factorizes process of building and using likelihood functions
  - You can give somebody an analytical likelihood of a (potentially very complex) physics analysis in a way to the easy-to-use, provides introspection, and is easy to modify.

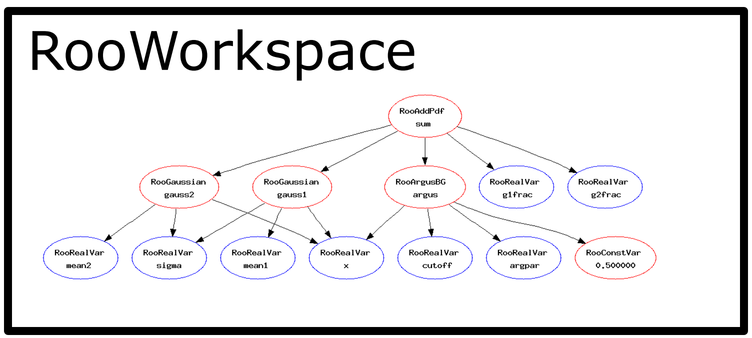


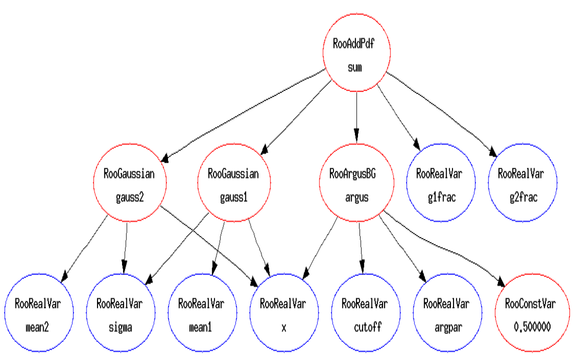

```
RooWorkspace w("w") ;
w.import(sum) ;
w.writeToFile("model.root") ;
```



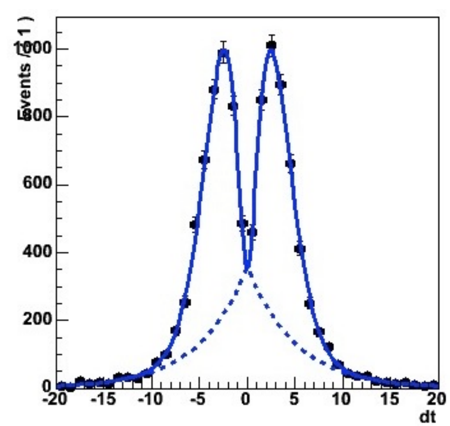
model.root

Using a workspace






```
// Resurrect model and data
TFile f("model.root") ;
RooWorkspace* w = f.Get("w") ;
RooAbsPdf* model = w->pdf("sum") ;
RooAbsData* data = w->data("xxx") ;

// Use model and data
model->fitTo(*data) ;

RooPlot* frame =
         w->var("dt")->frame() ;
data->plotOn(frame) ;
model->plotOn(frame) ;
```



Factory and Workspace

- *One C++ object per math symbol* provides ultimate level of control over each objects functionality, but results in lengthy user code for even simple macros
- Solution: add factory that auto-generates objects from a math-like language. Accessed through factory() method of workspace
- Example: reduce construction of Gaussian pdf and its parameters from 4 to 1 line of code

```
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) ;
```

```
w.factory("Gaussian::f(x[-10,10],mean[5],sigma[3])") ;
```

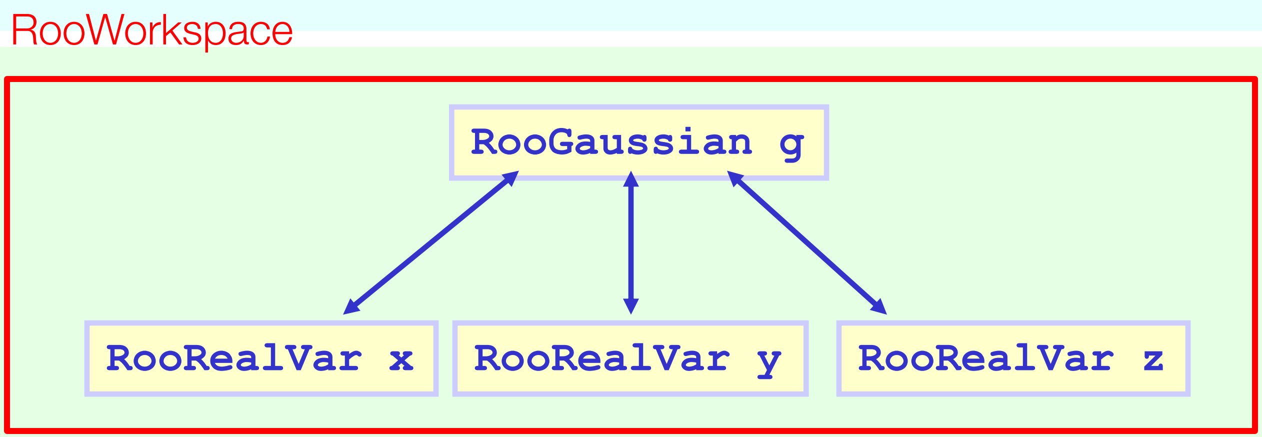
RooFit core design philosophy - Workspace

- The workspace serves a container class for all objects created

Math

$$\text{Gauss}(x,\mu,\sigma)$$

RooFit diagram



RooFit code

```
RooRealVar x("x","x",-10,10) ;
RooRealVar m("m","y",0,-10,10) ;
RooRealVar s("s","z",3,0.1,10) ;
RooGaussian g("g","g",x,m,s) ;
RooWorkspace w("w") ;
w.import(g) ;
```

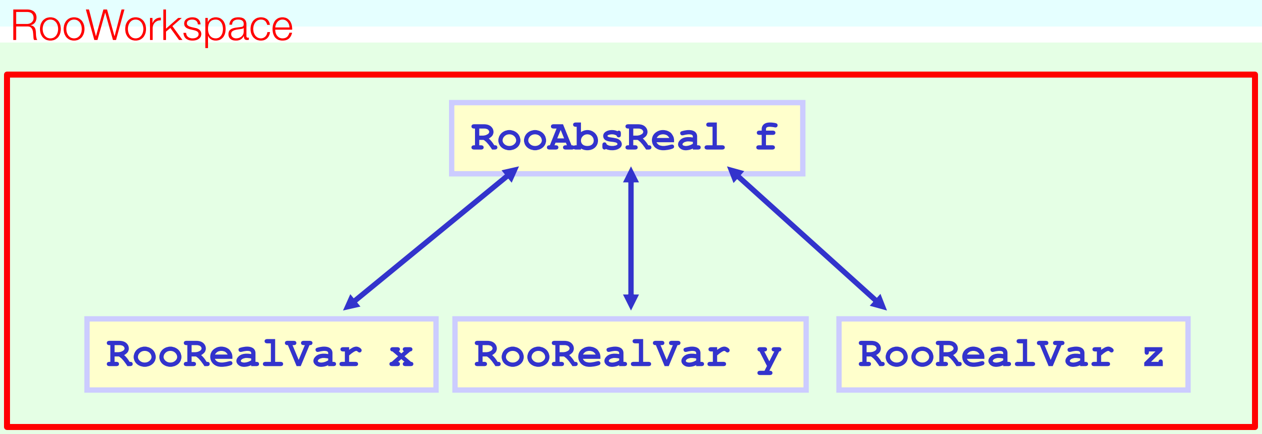
Populating a workspace the easy way – "the factory"

- The factory allows to fill a workspace with pdfs and variables using a simplified scripting language

Math

$\mathrm{Gauss}(x,\mu,\sigma)$

RooFit diagram

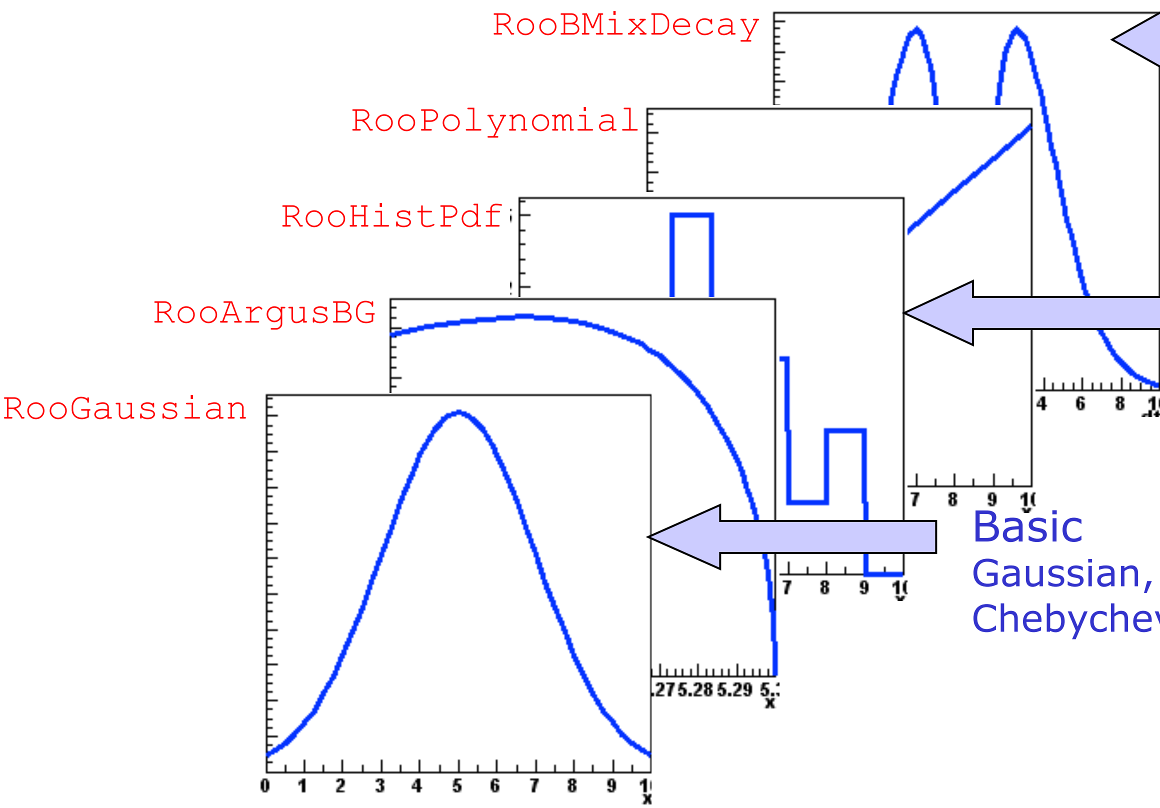


RooFit code

```
RooWorkspace w("w") ;
w.factory("RooGaussian::g(x[-10,10],m[-10,10],z[3,0.1,10])");
```

Model building – (Re)using standard components

- RooFit provides a collection of compiled standard PDF classes



Physics inspired
ARGUS,Crystal Ball,
Breit-Wigner, Voigtian,
B/D-Decay,….

Non-parametric
Histogram, KEYS

Basic
Gaussian, Exponential, Polynomial,…
Chebychev polynomial

Easy to extend the library: each p.d.f. is a separate C++ class

Model building – (Re)using standard components

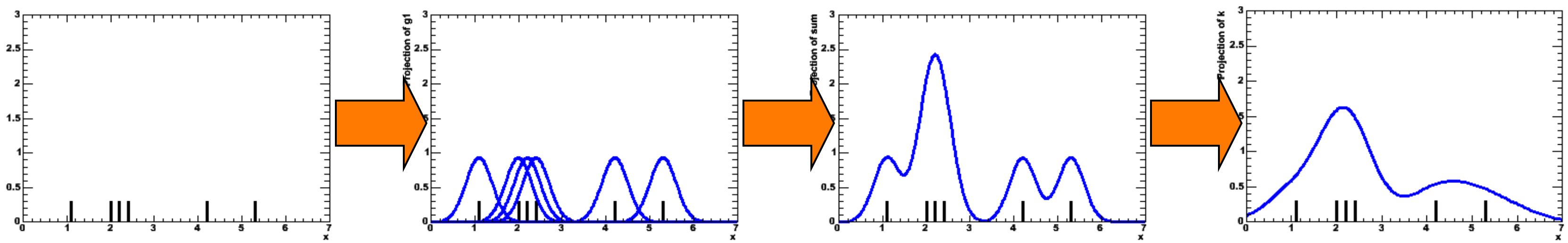
- List of most frequently used pdfs and their factory spec

Gaussian	`Gaussian::g(x,mean,sigma)`
Breit-Wigner	`BreitWigner::bw(x,mean,gamma)`
Landau	`Landau::l(x,mean,sigma)`
Exponential	`Exponental::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 (=BW⊗**G**)	`Voigtian::v(x,mean,gamma,sigma)`

The power of pdf as building blocks – Advanced algorithms

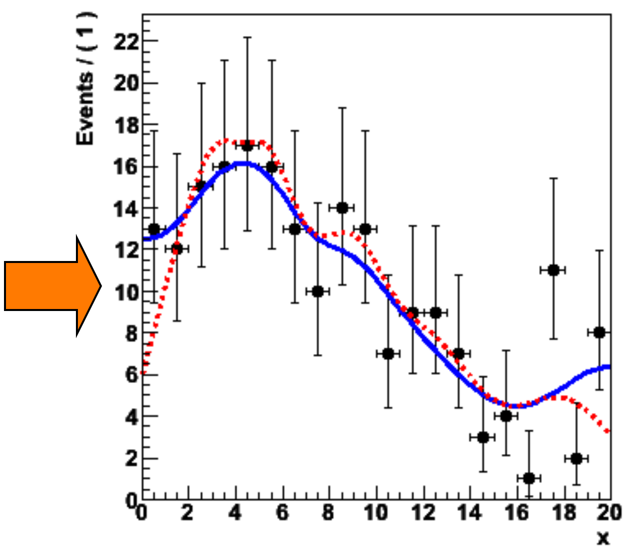
- Example: a ‘kernel estimation probability model’
  - Construct smooth pdf from unbinned data, using kernel estimation technique

Sample of events | Gaussian pdf for each event | Summed pdf for all events | Adaptive Kernel: width of Gaussian depends on local event density



- Example

```
w.import(myData,Rename("myData")) ;
w.factory("KeysPdf::k(x,myData)") ;
```



- Also available for n-D data

The power of pdf as building blocks – adaptability

- RooFit pdf classes do not require their parameter arguments to be variables, one can plug in functions as well
- Allows trivial customization, extension of probability models

class RooGaussian

$$Gauss(x \mid \mu, \sigma)$$

also class RooGaussian!

$$Gauss(x \mid \underbrace{\mu \cdot (1+2\alpha)}, \sigma)$$

Introduce a response function for a systematic uncertainty

```
// Original Gaussian
w.factory("Gaussian::g1(x[80,100],m[91,80,100],s[1])")


// Gaussian with response model in mean
w.factory("expr::m_response("m*(1+2alpha)",m,alpha[-5,5])") ;
w.factory("Gaussian::g1(x,m_response,s[1])")
```

NB: "expr" operates builds an intepreted function expression on the fly

The power of building blocks – operator expressions

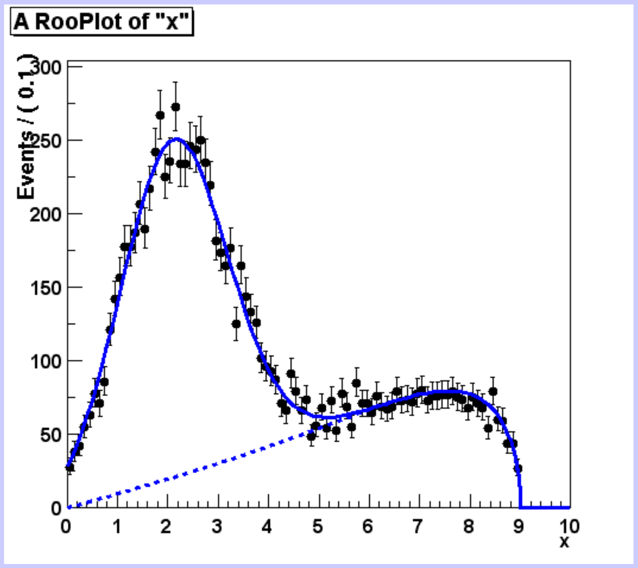
'ex05.C'

- Create a SUM expression to represent a sum of probability models

```
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)")
```

- In composite model visualization components can be accessed by name

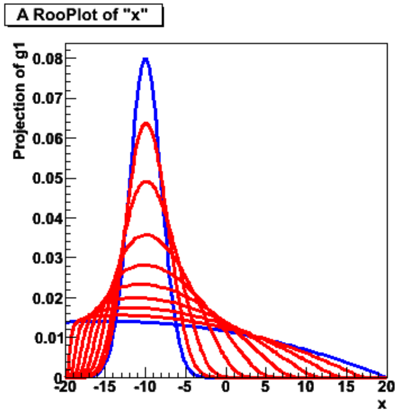
```
// Plot only argus components
w::sum.plotOn(frame,Components("argus"),
              LineStyle(kDashed)) ;
```

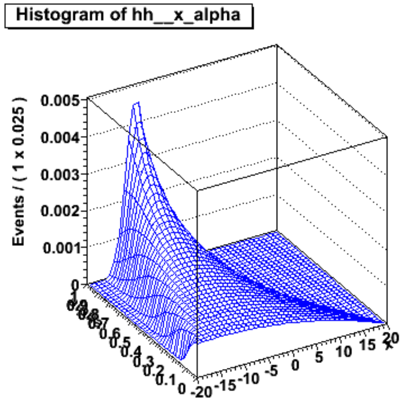


Powerful operators – Morphing interpolation

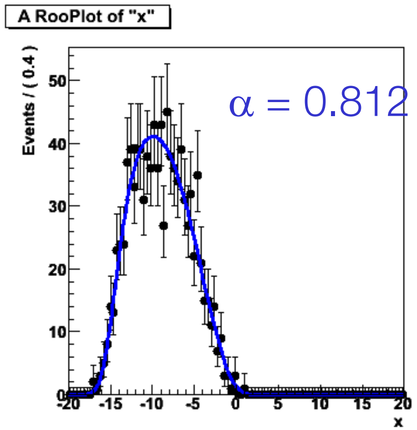
- Special operator pdfs can interpolate existing pdf shapes
  - Ex: interpolation between Gaussian and Polynomial

```
w.factory("Gaussian::g(x[-20,20],-10,2)") ;
w.factory("Polynomial::p(x,{-0.03,-0.001})") ;
w.factory("IntegralMorph::gp(g,p,x,alpha[0,1])") ;
```





Fit to data



$\alpha = 0.812 \pm 0.008$

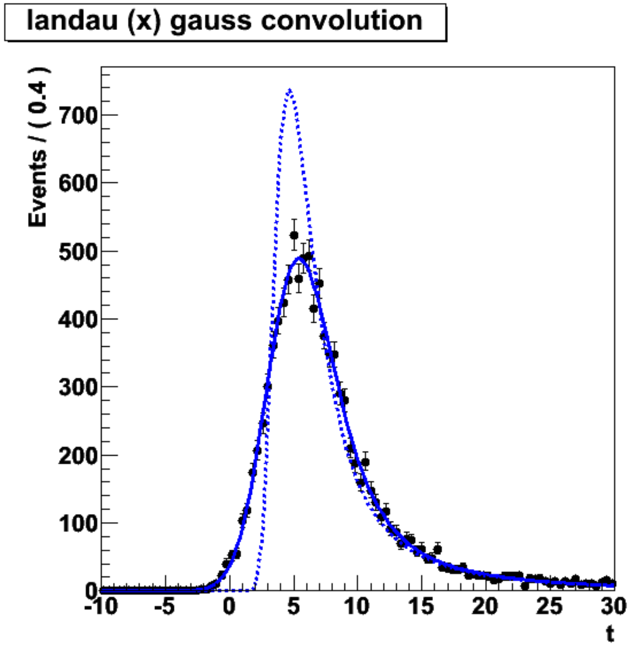
- Three morphing operator classes available
  - IntegralMorph (Alex Read).
  - MomentMorph (Max Baak).
  - PiecewiseInterpolation (via HistFactory)

Powerful operators – Fourier convolution

- Convolve any two arbitrary pdfs with a 1-line expression

```
w.factory("Landau::L(x[-10,30],5,1)") :
w.factory("Gaussian::G(x,0,2)") ;

w::x.setBins("cache",10000) ; // FFT sampling density
w.factory("FCONV::LGf(x,L,G)") ; // FFT convolution
```

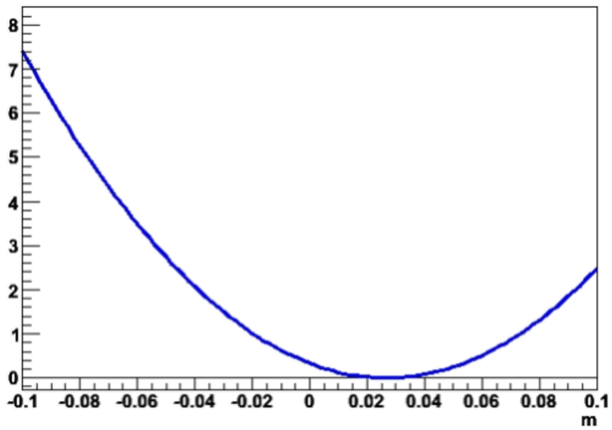
- Exploits power of FFTW package available via ROOT
  - Hand-tuned assembler code for time-critical parts
  - Amazingly fast: unbinned ML fit to 10.000 events take ~5 seconds!



Working with the likelihood function

- Plot the likelihood function versus a parameter

```
RooAbsReal* nll = w::model.createNLL(data) ;

RooPlot* frame = w::param.frame() ;
nll->plotOn(frame,ShiftToZero()) ;
```

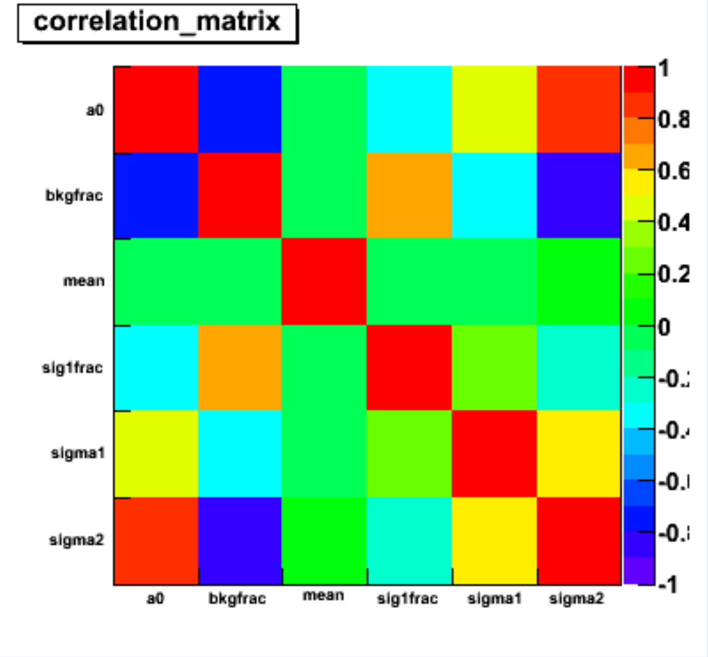


- Maximum Likelihood estimation of parameters and variance

```
RooMinimizer m(*nll) ;

// ML Parameter estimation
m.minimize("Minuit2","migrad") ;

// Variance estimation
m.hesse() ;

// Alternatively - all this in one line
pdf->fitTo(*data) ;
```

Working with covariance and correlation matrices

- Detailed information on parameter and covariance estimates can be saved for detailed information




```
RooMinimizer m(*nll) ;
m.minimize("Minuit2","migrad") ;
m.hesse() ;
RooFitResult* r = m.save() ;

// Visualize correlation matrix
r->correlationHist->Draw("colz") ;

// Extract correlation,covariance matrix
TMatrixDSym cov = fr->covarianceMatrix() ;
TMatrixDSym cov = fr->covarianceMatrix(a,b) ;
```

Use covariance matrices for correlated error propagation

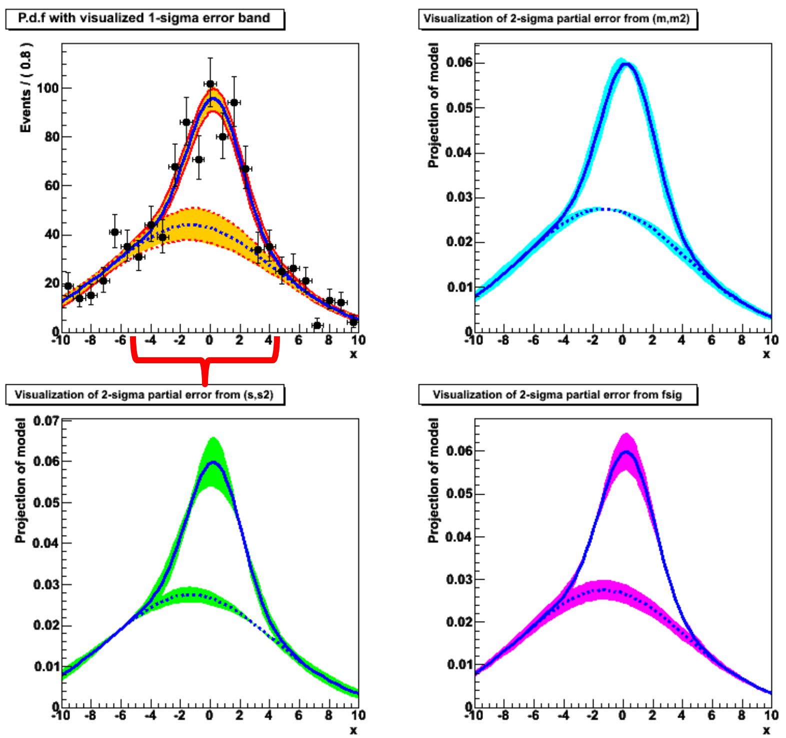
- Can (as visual aid) propagate errors in covariance matrix of a fit result to a pdf projection

```
w::model.plotOn(frame,VisualizeError(*fitresult)) ;
w::model.plotOn(frame,VisualizeError(*fitresult,fsig)) ;
```

  - Linear propagation on pdf projection $\Delta = \vec{E} V^{-1} \vec{E}$

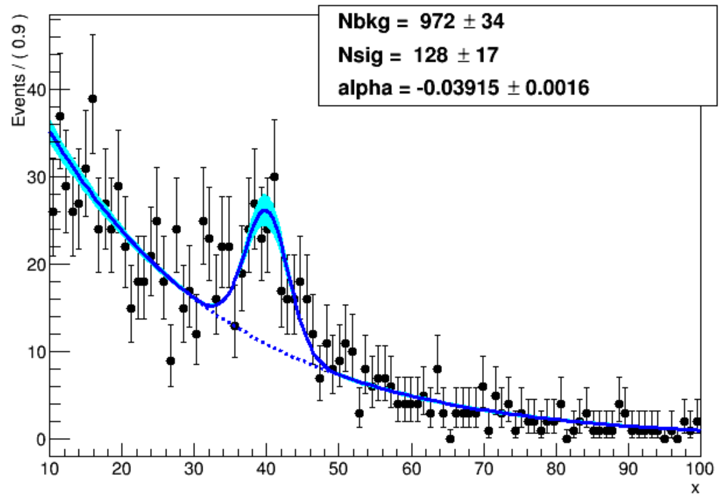
- Propagated error can be calculated on arbitrary function
  - E.g fraction of events in signal range

```
RooAbsReal* fracSigRange =
     w::model.createIntegral(x,x,"sig") ;

Double_t err =
    fracSigRange.getPropagatedError(*fr);
```



Some RooFit practical examples – from start to end

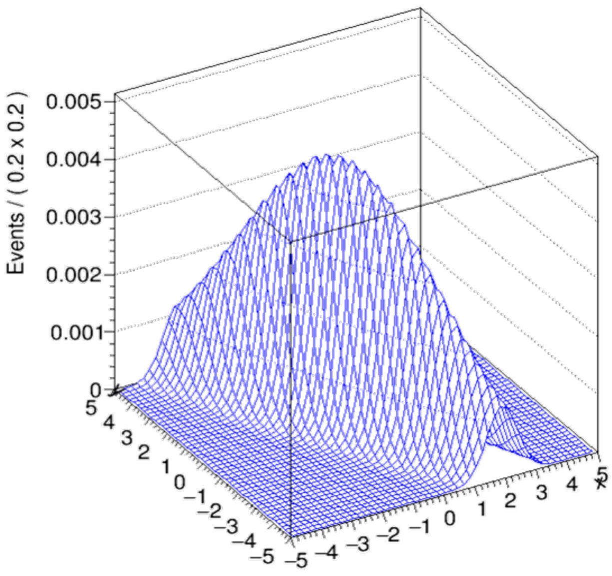
- Signal + Background (analytical)




```
RooWorkspace w("w") ;

// Construct exponential background model
w.factory("Exponential::bkg(x[10,100],alpha[-0.04,-0.1,-0])") ;

// Construct Gaussian signal model
w.factory("Gaussian::sig(x,mean[40],width[3])") ;

// Construct extended ML model of sum of signal and background
w.factory("SUM::modelsum(Nsig[100,0,200]*sig,Nbkg[1000,0,2000]*bkg)") ;

// Generate a toy dataset (unbinned) from model, data sample size obtained from expected event count
RooDataSet* d = w.pdf("modelsum")->generate(*w.var("x")) ;

// Fit model to toy data
RooFitResult* r3 = w.pdf("modelsum")->fitTo(*d,Save()) ;

// Plot data
RooPlot* frame = w.var("x")->frame() ;
d->plotOn(frame) ;

// Plot model (background component separately) and visualization of uncertainties from fit
w.pdf("modelsum")->plotOn(frame,VisualizeError(*r3)) ;
w.pdf("modelsum")->plotOn(frame) ;
w.pdf("modelsum")->plotOn(frame,Components("bkg"),LineStyle(kDashed)) ;
w.pdf("modelsum")->paramOn(frame) ;
frame->Draw() ;
```

Some RooFit practical examples – from start to end

- Two-dimensional signal: f(x|y)*g(y)

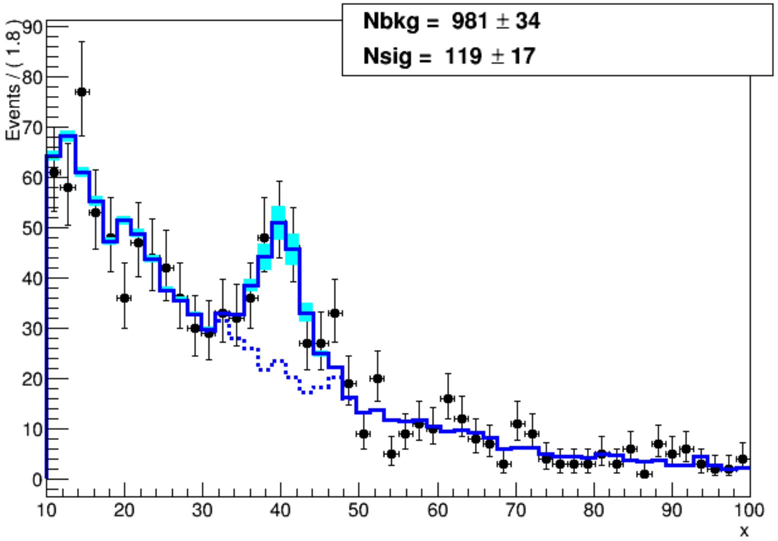



```
RooWorkspace w("w") ;

// Construct g(x|fy,0.5) where the mean of the gaussian
// is a polynomial fy=a0+a1*y
w.factory("PolyVar::fy(y[-5,5],{a0[-0.5,-5,5],a1[-0.5,-1,1]})") ;
w.factory("Gaussian::gx(x[-5,5],fy,sigmax[0.5])") ;

// Construct g(y)
w.factory("Gaussian::gy(y,0,3)") ;

// Construct the conditional product g(x|y)*g(y)
w.factory("PROD::model(gx|y,gy)") ;

// Generate 1000 events in x and y from model
RooDataSet *data = w.pdf("model")->generate(RooArgSet(*w.var("x"),*w.var("y")),10000) ;

// Plot x distribution of data and projection of model on x = Int(dy) model(x,y)
RooPlot* xframe = w.var("x")->frame() ;
data->plotOn(xframe) ;
w.pdf("model")->plotOn(xframe) ;

// Make two-dimensional plot in x vs y
TH1* hh_model = w.pdf("model")->createHistogram("hh_model",*w.var("x"),Binning(50),
                                                YVar(*w.var("y"),Binning(50))) ;
hh_model->SetLineColor(kBlue) ;
```

Some RooFit practical examples – from start to end

- Signal + Background (templates)

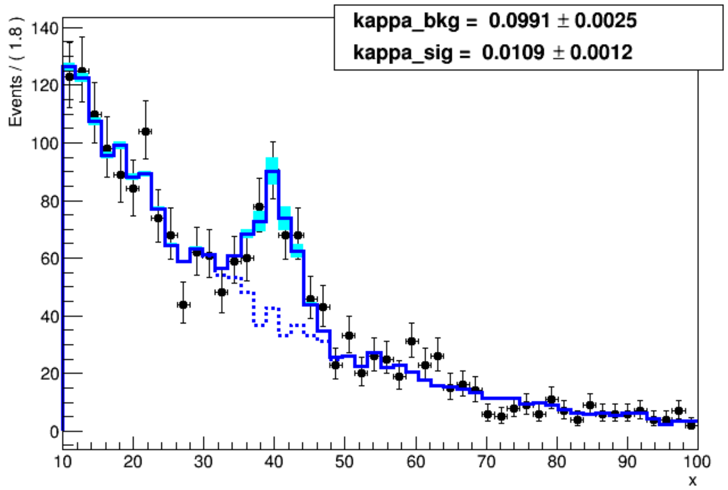
  Method 1: Construct unit-normalized pdf from histograms
  Model parameters are absolute event counts




```
RooWorkspace w("w") ;

// Import template histograms into workspace
w.import(*h_bkg,Rename("histo_bkg")) ;
w.import(*h_sig,Rename("histo_sig")) ;

// Construct sum of histogram-shaped templates
w.factory("SUM::modelsum(Nsig[100,0,200]*HistPdf::sig(x[10,100],histo_sig),
                      Nbkg[1000,0,2000]*HistPdf::bkg(x,histo_bkg))") ;

// Generate a toy dataset (unbinned) from model, data sample size obtained from expected event count
RooDataSet* d = w.pdf("modelsum")->generate(*w.var("x")) ;

// Fit model to toy data
RooFitResult* r3 = w.pdf("modelsum")->fitTo(*d,Save()) ;

// Plot data
RooPlot* frame = w.var("x")->frame() ;
d->plotOn(frame) ;

// Plot model (background component separately) and visualization of uncertainties from fit
w.pdf("modelsum")->plotOn(frame,VisualizeError(*r3)) ;
w.pdf("modelsum")->plotOn(frame) ;
w.pdf("modelsum")->plotOn(frame,Components("bkg"),LineStyle(kDashed)) ;
w.pdf("modelsum")->paramOn(frame) ;

frame->Draw() ;
```

Some RooFit practical examples – from start to end

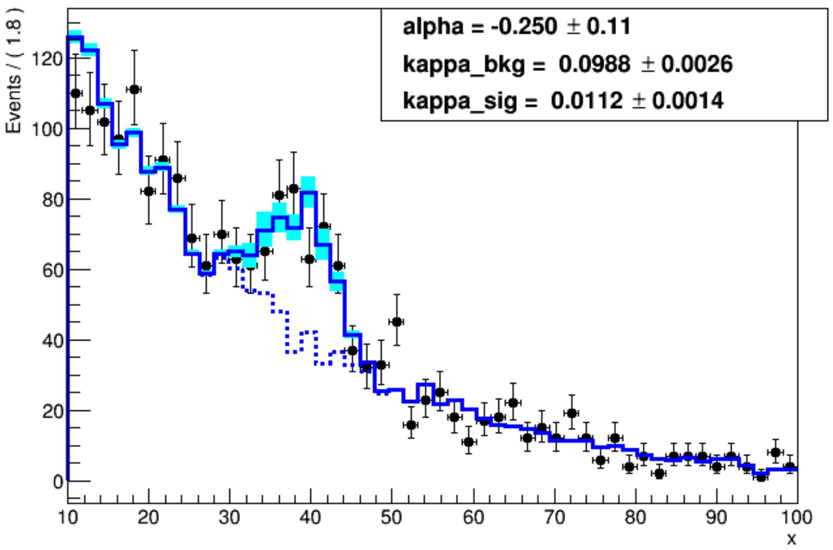
- Signal + Background (templates)
  Method 2: Construct event-count scaled pdf from histograms
  Model parameters are scale factors relative histogram counts




```
RooWorkspace w("w") ;

// Import template histograms into workspace
w.import(*h_bkg,Rename("histo_bkg")) ;
w.import(*h_sig,Rename("histo_sig")) ;

// Construct sum of histogram-shaped templates
w.factory("ASUM::modelsum(kappa_sig[0.01,-0.1,1]*HistFunc::sig(x[10,100],histo_sig),
                          kappa_bkg[0.1,-0.1,1]*HistFunc::bkg(x,histo_bkg))") ;

// Generate a toy dataset (unbinned) from model, data sample size obtained from expected event count
RooDataSet* d = w.pdf("modelsum")->generate(*w.var("x")) ;

// Fit model to toy data
RooFitResult* r3 = w.pdf("modelsum")->fitTo(*d,Save()) ;

// Plot data
RooPlot* frame = w.var("x")->frame() ;
d->plotOn(frame) ;

// Plot model (background component separately) and visualization of uncertainties from fit
w.pdf("modelsum")->plotOn(frame,VisualizeError(*r3)) ;
w.pdf("modelsum")->plotOn(frame) ;
w.pdf("modelsum")->plotOn(frame,Components("bkg"),LineStyle(kDashed)) ;
w.pdf("modelsum")->paramOn(frame) ;

frame->Draw() ;
```

Some RooFit practical examples – from start to end

- Signal + Background (templates)
  With morphing shape parameter




```
RooWorkspace w("w") ;

// Import template histograms into workspace
w.import(*h_bkg,Rename("histo_bkg")) ;
w.import(*h_sig_lo,Rename("histo_sig_lo")) ;
w.import(*h_sig_nom,Rename("histo_sig_nom")) ;
w.import(*h_sig_hi,Rename("histo_sig_hi")) ;

w.factory("PiecewiseInterpolation::sig_morph(HistFunc::sig_nom(x,histo_sig_nom),
                                             HistFunc::sig_lo(x,histo_sig_lo),
                                             HistFunc::sig_hi(x,histo_sig_hi),alpha[-5,5])") ;

// Construct sum of histogram-shaped templates
w.factory("ASUM::modelsum(kappa_sig[0.01,-0.1,1]*sig_morph,
                          kappa_bkg[0.1,-0.1,1]*HistFunc::bkg(x,histo_bkg))") ;

// Generate a toy dataset (unbinned) from model, data sample size obtained from expected event count
RooDataSet* d = w.pdf("modelsum")->generate(*w.var("x")) ;

// Fit model to toy data
RooFitResult* r3 = w.pdf("modelsum")->fitTo(*d,Save()) ;

// Plot data
RooPlot* frame = w.var("x")->frame() ;
d->plotOn(frame) ;
```

Statistical methods 3

Inference with parameters:
maximum likelihood, confidence intervals, upper limits, likelihood ratio and asymptotic formulae

Wouter Verkerke, NIKHEF

Roadmap of this course

- Start with basics, gradually build up to complexity

Model building

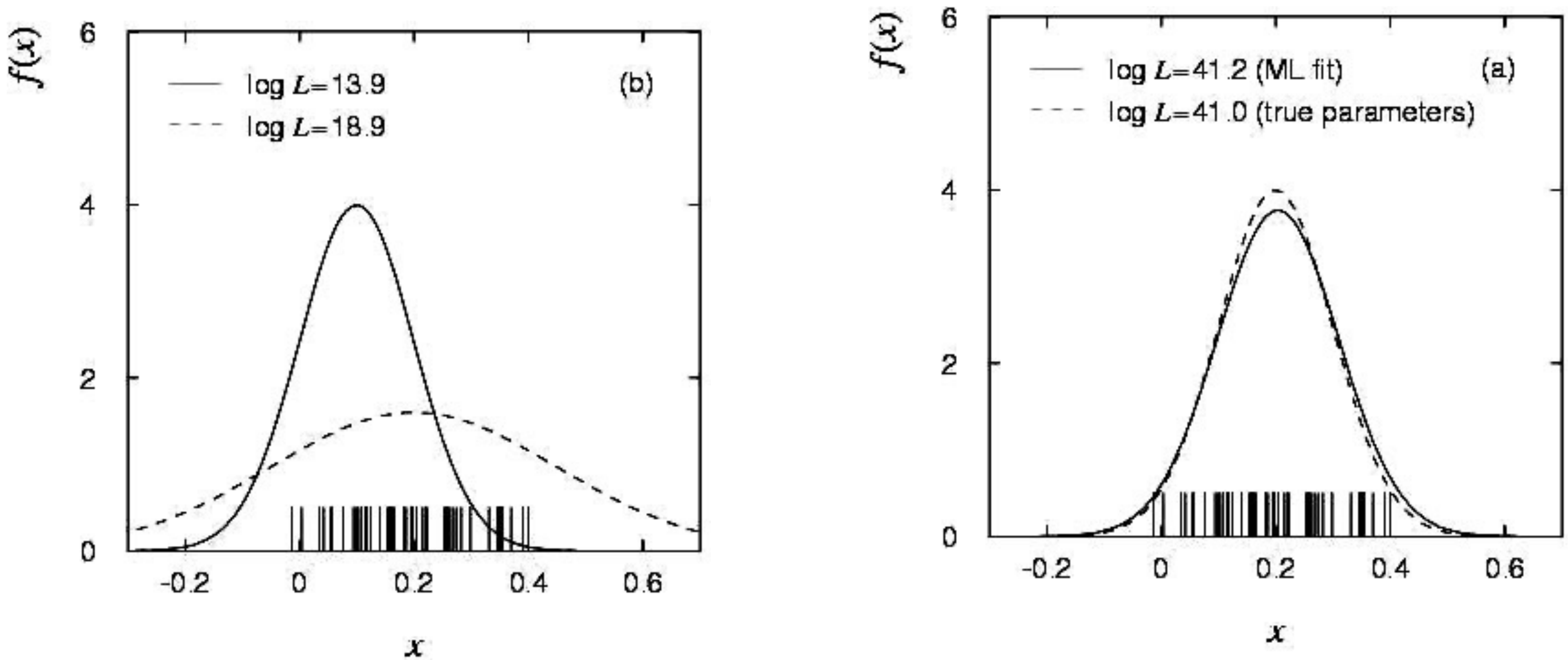
- Counting models
- Modeling distributions
- Signal parameterization strategies
- Models with nuisance parameters, joint models, modeling systematic uncertainties
- Diagnosing inference on complex models
- Advanced signal modeling techniques

Statistical methods

- Statistical tests with counting experiments
- Test statistics for models describing distributions
- Parameter estimation, confidence intervals & limits
- Inference with nuisance parameters

Parameter estimation using Maximum Likelihood

- Likelihood is high for values of p that result in distribution similar to data



- Define the maximum likelihood (ML) estimator to be the procedure that finds the parameter value for which the likelihood is maximal.

'ex06.C'

Parameter estimation – Maximum likelihood

- Practical estimation of maximum likelihood performed by minimizing the negative log-Likelihood

$$L(\vec{p}) = \prod_i f(\vec{x}_i; \vec{p})$$

$$-\ln L(\vec{p}) = -\sum_i \ln F(\vec{x}_i; \vec{p})$$

  - Advantage of log-Likelihood is that contributions from events can be summed, rather than multiplied (computationally easier)

- In practice, find point where derivative of –logL is zero

$$\left.\frac{d \ln L(\vec{p})}{d\vec{p}}\right|_{p_i = \hat{p}_i} = 0$$

- Standard notation for ML estimation of p is $\hat{p}$

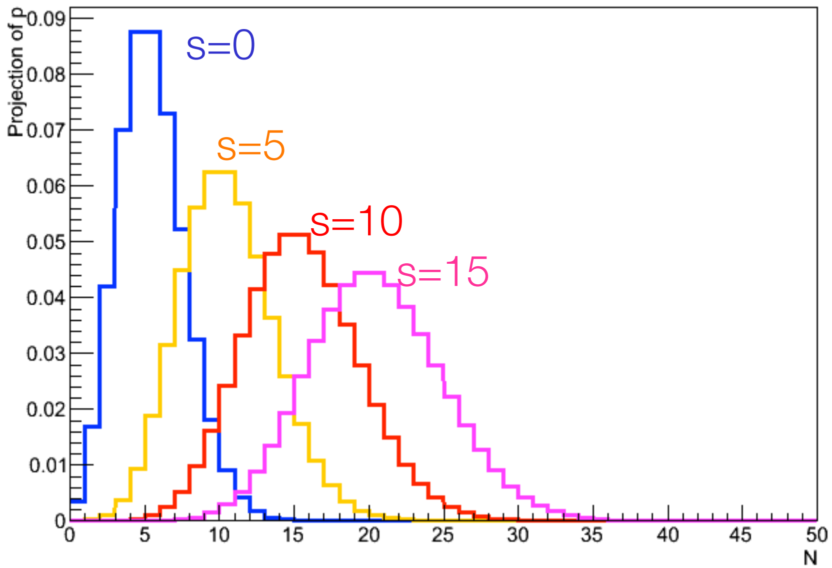
Example of Maximum Likelihood estimation

'ex09.C'

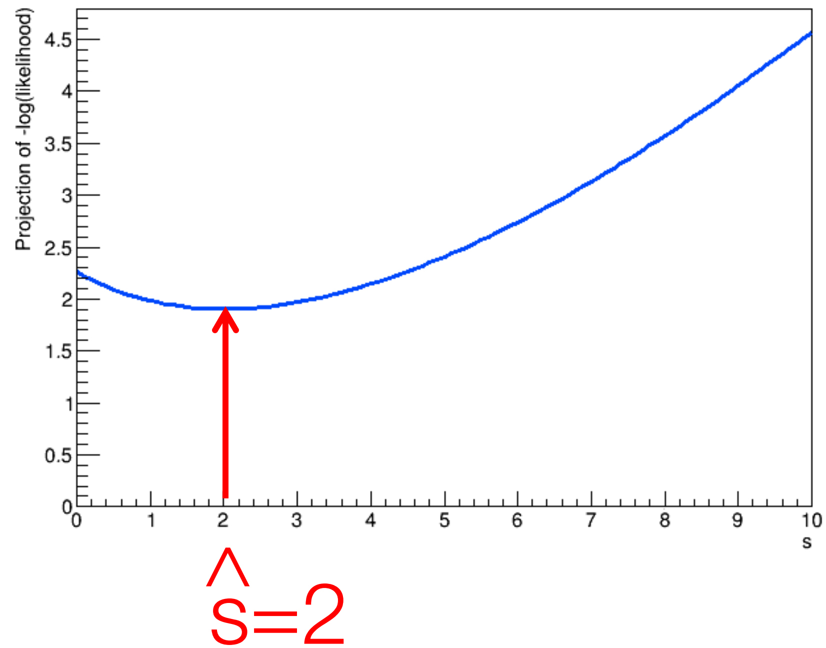
- Illustration of ML estimate on Poisson counting model

$$L(N \mid s) = Poisson(N \mid s + \tilde{b})$$

-log $L(N|s)$ versus N [s=0,5,10,15]



-log $L(N|s)$ versus s [N=7]



- Note that Poisson model is discrete in N, *but continuous in s!*

Properties of Maximum Likelihood estimators

- In general, Maximum Likelihood estimators are
  - Consistent (gives right answer for $N \rightarrow \infty$)
  - Mostly unbiased (bias $\propto 1/N$, may need to worry at small N) `'ex05.C'`
  - Efficient for large N (you get the smallest possible error)
  - Invariant: (a transformation of parameters will Not change your answer, e.g $(\hat{p})^2 = \widehat{(p^2)}$

- MLE efficiency theorem: the MLE will be *unbiased* and *efficient* if an unbiased efficient estimator exists
  - Proof not discussed here
  - Of course this does not guarantee that any MLE is unbiased and efficient for any given problem

Relation between Likelihood and χ^2 estimators

- Properties of χ^2 estimator follow from properties of ML estimator using *Gaussian probability density functions*

$$F(x_i, y_i, \sigma_i; \vec{p}) = \prod_i \exp\left[-\left(\frac{y_i - f(x_i;\vec{p})}{\sigma_i}\right)^2\right]$$

Gaussian Probability Density Function in p for single measurement y±σ from a predictive function f(x|p)

Take log,
Sum over all points (***x_i* , *y_i* , *σ_i***)

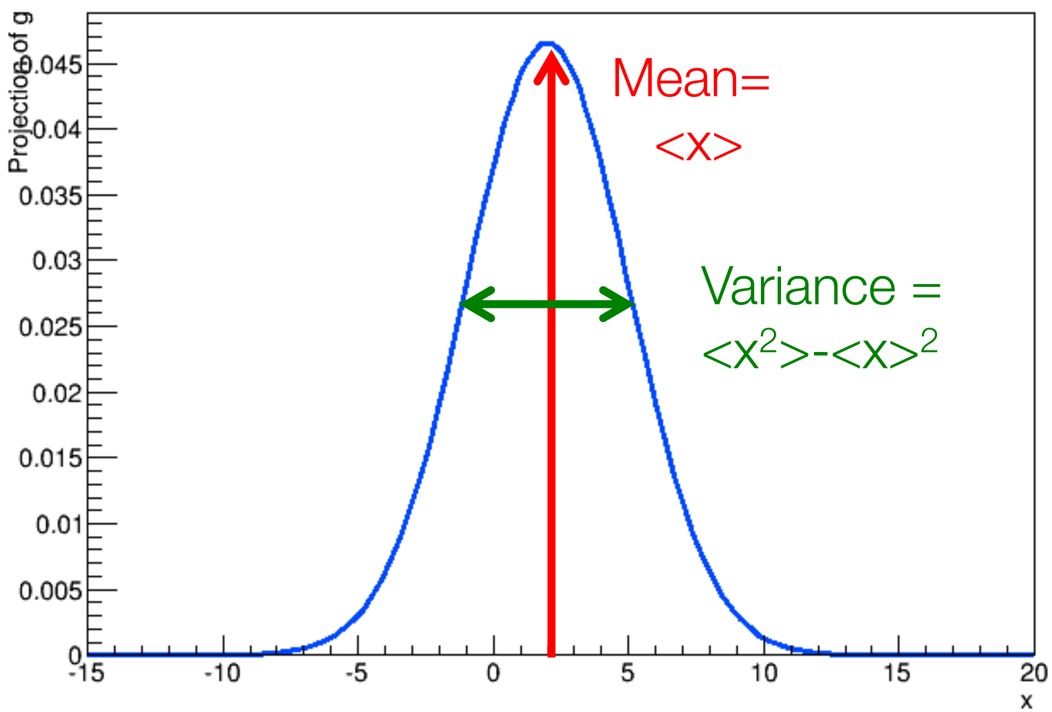
$$-\ln L(\vec{p}) = \tfrac{1}{2}\sum_i \left(\frac{y_i - f(x_i;\vec{p})}{\sigma_i}\right) = \tfrac{1}{2}\chi^2$$

The Likelihood function in p for given points $x_i(s_i)$ and function $f(x_i;p)$

- The χ^2 estimator follows from ML estimator, i.e it is
  - Efficient, consistent, bias 1/N, invariant,
  - But only in the limit that the error **on *x_i*** is truly Gaussian

Estimating parameter variance

- Note that 'uncertainty' on a parameter estimate is an ambiguous statement
- Can either mean an interval with a stated confidence or credible, level (e.g. 68%), or simply assume it is the square-root of the variance of a distribution



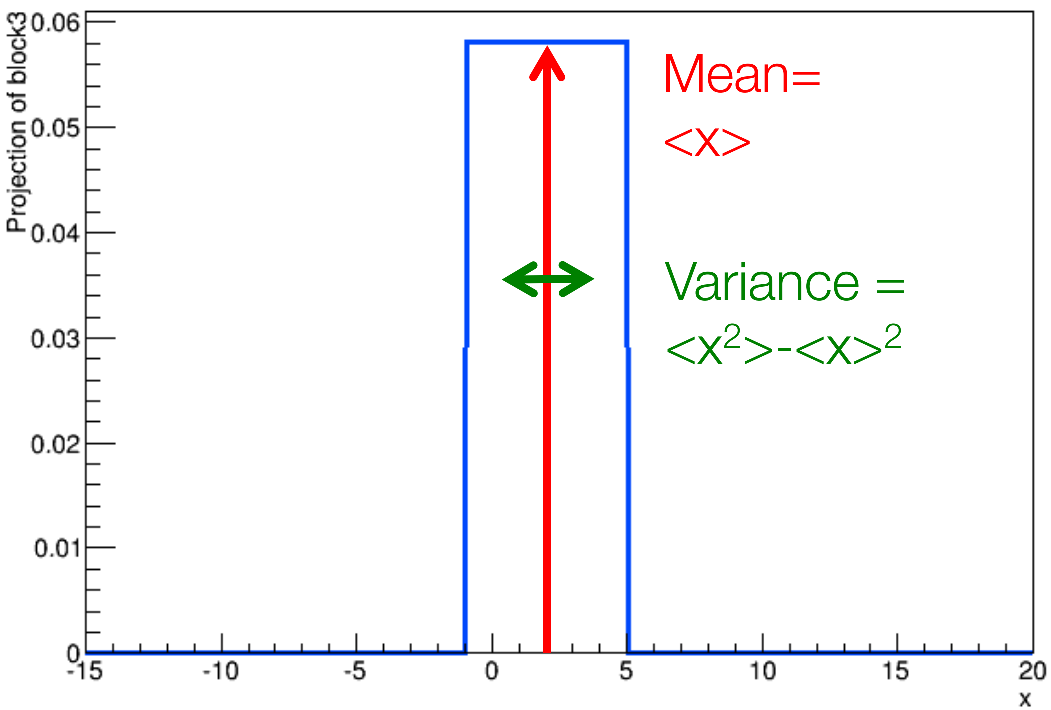
For a Gaussian distribution <u>mean</u> and <u>variance</u> map to parameters for *mean* and *sigma*^{2}

and interval defined by $\sqrt{V}$ contains 68% of the distribution (='1 sigma' by definition)

Thus for Gaussian distributions all common definitions of 'error' work out to the same numeric value

Estimating parameter variance

- Note that ‘error’ or ‘uncertainty’ on a parameter estimate is an ambiguous statement
- Can either mean an interval with a stated confidence or credible, level (e.g. 68%), or simply assume it is the square-root of the variance of a distribution



For other distributions intervals by $\sqrt{V}$ do not necessarily contain 68% of the distribution

Estimating variance on parameters

- Variance on of parameter can also be estimated from Likelihood using the variance estimator

$$\hat{\sigma}(p)^2 = \hat{V}(p) = \left(\frac{d^2 \ln L}{d^2 p} \right)^{-1}$$

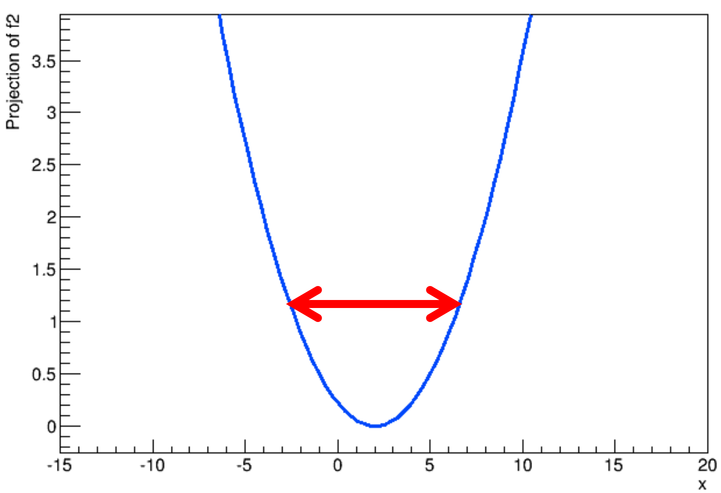
From Rao-Cramer-Frechet inequality

$$V(\hat{p}) \geq 1 + \frac{db}{dp} \Big/ \left(\frac{d^2 \ln L}{d^2 p} \right)$$

b = bias as function of p, inequality becomes equality in limit of efficient estimator

- Valid if estimator is efficient and unbiased!

- Illustration of Likelihood Variance estimate on a Gaussian distribution



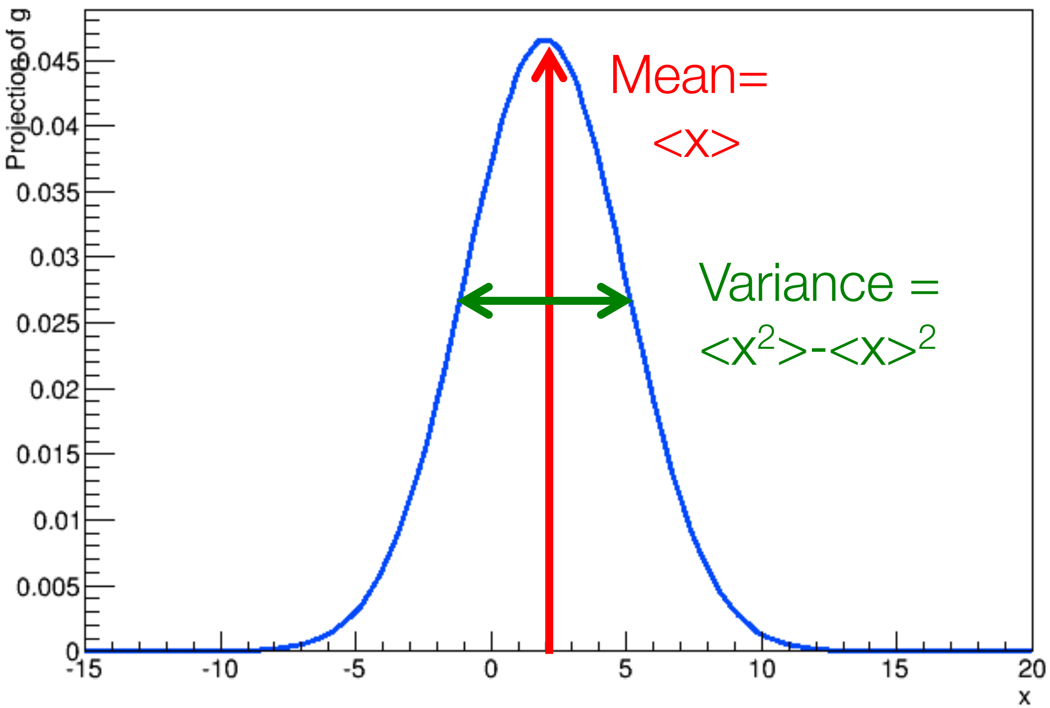
$$f(x|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2 \right]$$

$$\ln f(x|\mu,\sigma) = -\ln\sigma - \ln\sqrt{2\pi} + \frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2$$

$$\left.\frac{d\ln f}{d\sigma}\right|_{x=\mu} = \frac{-1}{\sigma} \Rightarrow \left.\frac{d^2 \ln f}{d^2\sigma}\right|_{x=\mu} = \frac{1}{\sigma^2}$$

Bayesian parameter estimation

- Bayesian parameter estimate is the posterior mean
- Bayesian variance is the posterior variance

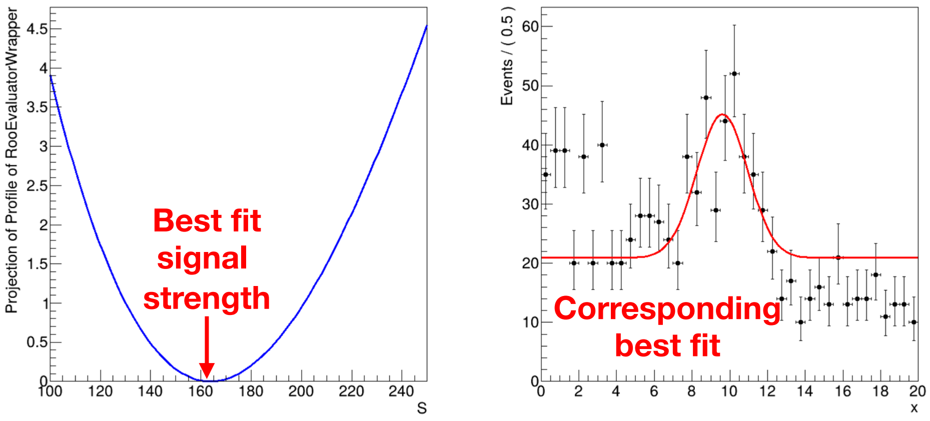


$$\hat{\mu} = \int \mu P(\mu \mid N) d\mu$$

$$\hat{V} = \int (\hat{\mu} - \mu)^2 P(\mu \mid N) d\mu$$

Goodness-of-fit

- An important question that arises in all statistical modeling:
is "is the best-fit" actually a "good fit"?



  - Fit only considers degrees of freedom expressed in the likelihood → might not capture observed data
  - In other words, the 'alternative hypotheses' considered is the ensemble of parameter values of the model

- Can you quantify goodness-of-fit 'abstractly',
without an explicit alternative hypothesis defining the whole space of possibilities?
  - Generally a hard problem, *no approach is assumption-free*

- Commonly used: reduced χ2.
  - Effectively it is p-value to obtain observed data under hypothesis of best-fit.
  - Implicit alternative hypothesis: independent Gaussian fluctuations in each bin
  - Not always a realistic assumption for deviations (ignores systematic effects)

- Much more on goodness-of-fit tomorrow → Lecture by Lydia

What can we do with composite hypothesis

- With simple hypotheses – inference is restricted to making statements about P(D|hypo) or P(hypo|D)
- With composite hypotheses – many more options
- 1 Parameter estimation and variance estimation
  - What is value of ***s*** for which the observed data is most probable?
  - What is the variance (std deviation squared) in the estimate of ***s?***

  } s=5.5 ± 1.3
- 2 Confidence intervals
  - Statements about model parameters using frequentist concept of probability
  - s<12.7 at 95% confidence level
  - 4.5 < s < 6.8 at 68% confidence level
- 3 Bayesian credible intervals
  - Bayesian statements about model parameters
  - s<12.7 at 95% credibility