Introduction to RooFit

1. Introduction and overview
2. Creation and basic use of models
3. Addition and Convolution
4. Common Fitting problems
5. Multidimensional and Conditional models
6. Fit validation and toy MC studies
7. Constructing joint model
8. Working with the Likelihood, including systematic errors
9. Interval and Limits

W. Verkerke (NIKHEF)

4 Common fitting Problems

- *Understanding MINUIT output*
- *Instabilities and correlation coefficients*

A brief description of MINUIT functionality

- MIGRAD
  - Find function minimum. Calculates function gradient, follow to (local) minimum, recalculate gradient, iterate until minimum found
    - To see what MIGRAD does, it is very instructive to do RooMinuit::setVerbose(1). It will print a line for each step through parameter space
  - Number of function calls required depends greatly on number of floating parameters, distance from function minimum and shape of function
- HESSE
  - Calculation of error matrix from 2nd derivatives at minimum
  - Gives symmetric error. Valid in assumption that likelihood is (locally parabolic)

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

  - Requires roughly N^2 likelihood evaluations (with N = number of floating parameters)

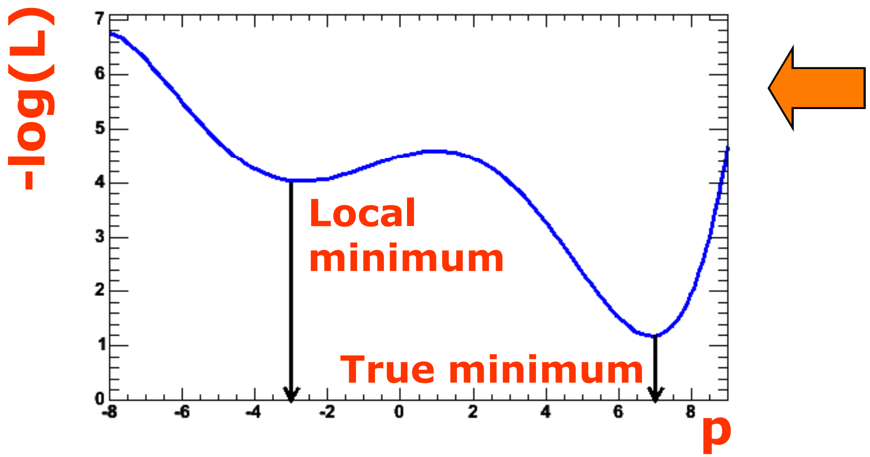
A brief description of MINUIT functionality

- MINOS
  - Calculate errors by explicit finding points (or contour for >1D) where Δ-log(L)=0.5
  - Reported errors can be asymmetric
  - Can be very expensive in with large number of floating parameters

- CONTOUR
  - Find contours of equal Δ-log(L) in two parameters and draw corresponding shape
  - Mostly an interactive analysis tool

Note of MIGRAD function minimization

- For all but the most trivial scenarios it is not possible to automatically find reasonable starting values of parameters
  - So you need to supply 'reasonable' starting values for your parameters



Reason: There may exist multiple (local) minima in the likelihood or χ^2

  - You may also need to supply 'reasonable' initial step size in parameters. (A step size 10x the range of the above plot is clearly unhelpful)
  - Using RooMinuit, the initial step size is the value of **`RooRealVar::getError()`**, so you can control this by supplying initial error values

Minuit function MIGRAD

- Purpose: find minimum

```
**********
**    13 **MIGRAD        1000           1
**********
(some output omitted)
MIGRAD MINIMIZATION HAS CONVERGED.
MIGRAD WILL VERIFY CONVERGENCE AND ERROR MATRIX.
COVARIANCE MATRIX CALCULATED SUCCESSFULLY
FCN=257.304 FROM MIGRAD    STATUS=CONVERGED      31 CALLS        32 TOTAL
                    EDM=2.36773e-06    STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                                   STEP         FIRST
 NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
  1  mean        8.84225e-02   3.23862e-01   3.58344e-04  -2.24755e-02
  2  sigma       3.20763e+00   2.39540e-01   2.78628e-04  -5.34724e-02
                                ERR DEF= 0.5
EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  2    ERR DEF=0.5
 1.049e-01  3.338e-04
 3.338e-04  5.739e-02
PARAMETER  CORRELATION COEFFICIENTS
      NO.  GLOBAL      1       2
        1  0.00430   1.000  0.004
        2  0.00430   0.004  1.000
```

Progress information, watch for errors here

Parameter values and approximate errors reported by MINUIT

Error definition (in this case 0.5 for a likelihood fit)

Minuit function MIGRAD

- Purpose: find minimum

```
**********
**    13 **MIGR
**********
(some output o
MIGRAD MINIMIZ
MIGRAD WILL VERI                  ENCE AND ERROR MATRIX.
COVARIANCE MATRIX CALCULATED SUCCESSFULLY
FCN=257.304 FROM MIGRAD    STATUS=CONVERGED      31 CALLS          32 TOTAL
                     EDM=2.36773e-06    STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                                           STEP         FIRST
 NO.   NAME      VALUE            ERROR           SIZE      DERIVATIVE
  1  mean         8.84225e-02   3.23862e-01   3.58344e-04  -2.24755e-02
  2  sigma        3.20763e+00   2.39540e-01   2.78628e-04  -5.34724e-02
                                   ERR DEF= 0.5
EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  2    ERR DEF=0.5
 1.049e-01  3.338e-04
 3.338e-04  5.739e-02
PARAMETER  CORRELATION COEFFICIENTS
      NO.  GLOBAL      1       2
       1  0.00430   1.000   0.004
       2  0.00430   0.004   1.000
```

Value of $\chi 2$ or likelihood at minimum

(NB: χ^2 values are not **divided by $N_{d.o.f}$)**

Approximate Error matrix And covariance matrix

Minuit function MIGRAD

- Purpose: find minimu

Status:
Should be 'converged' but can be 'failed'

Estimated Distance to Minimum
should be small $O(10^{-6})$

Error Matrix Quality
should be 'accurate', but can be 'approximate' in case of trouble

```
**********
**   13 **MIGRAD        1000
**********
(some output omitted)
MIGRAD MINIMIZATION HAS CONVERGED
MIGRAD WILL VERIFY CONVERGENCE AND E     MATRIX.
COVARIANCE MATRIX CALCULATED SUCCESSFULLY
FCN=257.304 FROM MIGRAD    STATUS=CONVERGED       31 CALLS          32 TOTAL
                       EDM=2.36773e-06    STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                                         STEP         FIRST
 NO.   NAME      VALUE            ERROR               SIZE      DERIVATIVE
  1  mean        8.84225e-02   3.23862e-01   3.58344e-04  -2.24755e-02
  2  sigma       3.20763e+00   2.39540e-01   2.78628e-04  -5.34724e-02
                              ERR DEF= 0.5
EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  2    ERR DEF=0.5
 1.049e-01  3.338e-04
 3.338e-04  5.739e-02
PARAMETER  CORRELATION COEFFICIENTS
      NO.  GLOBAL      1       2
       1  0.00430   1.000  0.004
       2  0.00430   0.004  1.000
```

Minuit function HESSE

- Purpose: calculate error matrix from $\frac{d^2L}{dp^2}$

```
**********
**   18 **HESSE        1000
**********
COVARIANCE MATRIX CALCULATED SUCCESSFULLY
FCN=257.304 FROM HESSE     STATUS=OK                               OTAL
                    EDM=2.36534e-06    STRAT                           CURATE
 EXT PARAMETER                                      ERNAL       INTERNAL
 NO.   NAME      VALUE            ERROR          STEP SIZE       VALUE
  1  mean         8.84225e-02   3.23861e-01   7.16689e-05   8.84237e-03
  2  sigma        3.20763e+00   2.39539e-01   5.57256e-05   3.26535e-01
                                   ERR DEF= 0.5
EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  2    ERR DEF=0.5
 1.049e-01  2.780e-04
 2.780e-04  5.739e-02
PARAMETER  CORRELATION COEFFICIENTS
      NO.  GLOBAL      1      2
       1  0.00358   1.000  0.004
       2  0.00358   0.004  1.000
```

Symmetric errors calculated from 2nd derivative of –ln(L) or χ^2

Minuit function HESSE

- Purpose: calculate error matrix from $\frac{d^2L}{dp^2}$

Error matrix (Covariance Matrix) calculated from

$$V_{ij} = \left(\frac{d^2(-\ln L)}{dp_i dp_j} \right)^{-1}$$

```
**********
**
***
COV                                   UCCESSFULLY
FCN                                   TUS=OK              10 CALLS          42 TOTAL
                                      4e-06    STRATEGY= 1       ERROR MATRIX ACCURATE
 EX                                                    INTERNAL         INTERNAL
 NO                                       ERROR        STEP SIZE          VALUE
  1                                    3.23861e-01    7.16689e-05    8.84237e-03
  2   si               3.20763e+00     2.39539e-01    5.57256e-05    3.26535e-01
                                   ERR DEF= 0.5
EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  2    ERR DEF=0.5
 1.049e-01  2.780e-04
 2.780e-04  5.739e-02
PARAMETER  CORRELATION COEFFICIENTS
      NO.  GLOBAL      1      2
       1  0.00358   1.000  0.004
       2  0.00358   0.004  1.000
```

Minuit function HESSE

- Purpose: calculate error matrix from $\frac{d^2L}{dp^2}$

```
**********
**   18 **HESSE        1000
**********
COVARIANCE MATRIX CALCULATED SUCCESSFULLY
FCN=257.304 FROM HESSE     STATUS=OK              10 CALLS          42 TOTAL
                     EDM=2.36534e-06    STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                                   INTERNAL       INTERNAL
 NO.   NAME      VALUE            ERROR          STEP SIZE        VALUE
  1  mean         8.84225e-02                                   8.84237e-03
  2  sigma        3.20763e+00                                   3.26535e-01

EXTERNAL ERROR MATRIX.    NDIM                               F=0.5
 1.049e-01  2.780e-04
 2.780e-04  5.739e-02
PARAMETER  CORRELATION COEFFICIENT
      NO.  GLOBAL      1       2
        1  0.00358   1.000   0.004
        2  0.00358   0.004   1.000
```

Correlation matrix ρ_{ij} calculated from

$$V_{ij} = \sigma_i \sigma_j \rho_{ij}$$

Minuit function HESSE

- Purpose: calculate error matrix from $\frac{d^2L}{dp^2}$

```
 **********
 **   18 **HESSE        1000
 **********
 COVARIANCE MATRIX CALCULATED SUCCESSFULLY
 FCN=257.304 FROM HESSE     STATUS=OK            10 CALLS         42 TOTAL
                     EDM=2.36534e-06    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                INTERNAL      INTERNAL
  NO.   NAME      VALUE            ERROR       STEP SIZE       VALUE
   1  mean                                     7.16689e-05   8.84237e-03
   2  sigma                                    5.57256e-05   3.26535e-01

 EXTERNAL ERROR                                2    ERR DEF=0.5
  1.049e-01  2.
  2.780e-04  5.739e-
 PARAMETER  CORRELA   N COEFFICIENTS
       NO.  GLOBAL       1       2
        1  0.00358    1.000   0.004
        2  0.00358    0.004   1.000
```

Global correlation vector: correlation of each parameter with *all other* parameters

Minuit function MINOS

- Error analysis through Δnll contour finding

```
**********
**   23 **MINOS        1000
**********
FCN=257.304 FROM MINOS     STATUS=SUCCESSFUL     52 CALLS          94 TOTAL
                    EDM=2.36534e-06    STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                  PARABOLIC         MINOS ERRORS
 NO.   NAME      VALUE            ERROR      NEGATIVE      POSITIVE
  1  mean         8.84225e-02   3.23861e-01  -3.24688e-01   3.25391e-01
  2  sigma        3.20763e+00   2.39539e-01  -2.23321e-01   2.58893e-01
                               ERR DEF= 0.5
```

Symmetric error

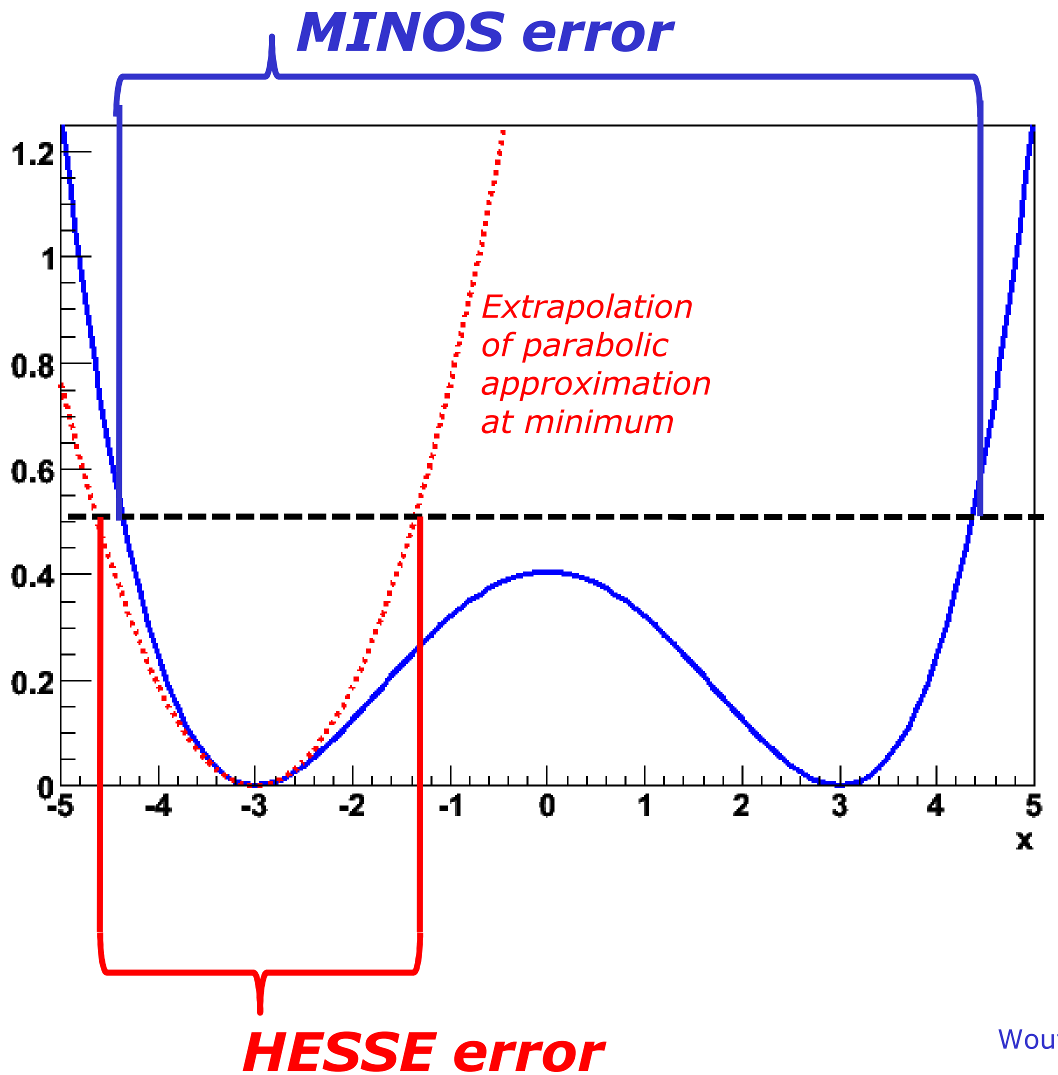
(repeated result from HESSE)

MINOS error
Can be asymmetric

(in this example the 'sigma' error is slightly asymmetric)

Illustration of difference between HESSE and MINOS errors

- ‘Pathological’ example likelihood with multiple minima and non-parabolic behavior



Practical estimation – Fit converge problems

- Sometimes fits don't converge because, e.g.
  - MIGRAD unable to find minimum
  - HESSE finds negative second derivatives (which would imply negative errors)
- Reason is usually numerical precision and stability problems, but
  - The underlying cause of fit stability problems is usually by **highly correlated parameters** in fit
- HESSE correlation matrix in primary investigative tool

```
PARAMETER  CORRELATION COEFFICIENTS
      NO.  GLOBAL        1        2
        1  0.99835   1.000    0.998
        2  0.99835   0.998    1.000
```

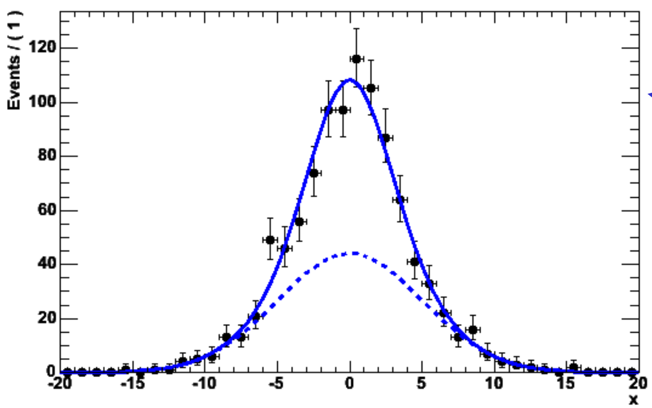
Signs of trouble...

  - In limit of 100% correlation, the usual point solution becomes a line solution (or surface solution) in parameter space. Minimization problem is no longer well defined

Mitigating fit stability problems

- Strategy I – More orthogonal choice of parameters
  - Example: fitting sum of 2 Gaussians of similar width

$$F(x; f, m, s_1, s_2) = fG_1(x; s_1, m) + (1 - f)G_2(x; s_2, m)$$



HESSE correlation matrix

PARAMETER NO.	GLOBAL	[f]	[m]	[s1]	[s2]
[f]	0.96973	1.000	-0.135	0.918	0.915
[m]	0.14407	-0.135	1.000	-0.144	-0.114
[s1]	0.92762	0.918	-0.144	1.000	0.786
[s2]	0.92486	0.915	-0.114	0.786	1.000

PARAMETER CORRELATION COEFFICIENTS

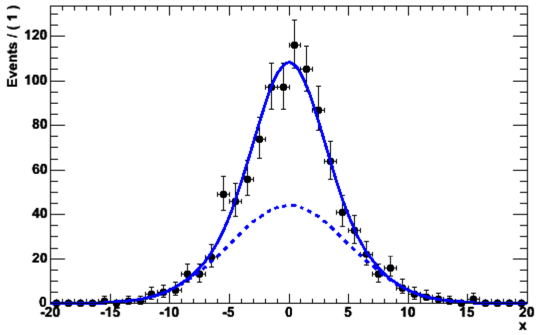
Widths s_1,s_2 strongly correlated fraction f

Mitigating fit stability problems

- Different parameterization:

$$fG_1(x;s_1,m_1)+(1-f)G_2(x;s_1\cdot s_2,m_2)$$

```
PARAMETER  CORRELATION COEFFICIENTS
      NO.  GLOBAL      [f]     [m]    [s1]    [s2]
     [ f]  0.96951   1.000  -0.134   0.917  -0.681
     [ m]  0.14312  -0.134   1.000  -0.143   0.127
     [s1]  0.98879   0.917  -0.143   1.000  -0.895
     [s2]  0.96156  -0.681   0.127  -0.895   1.000
```



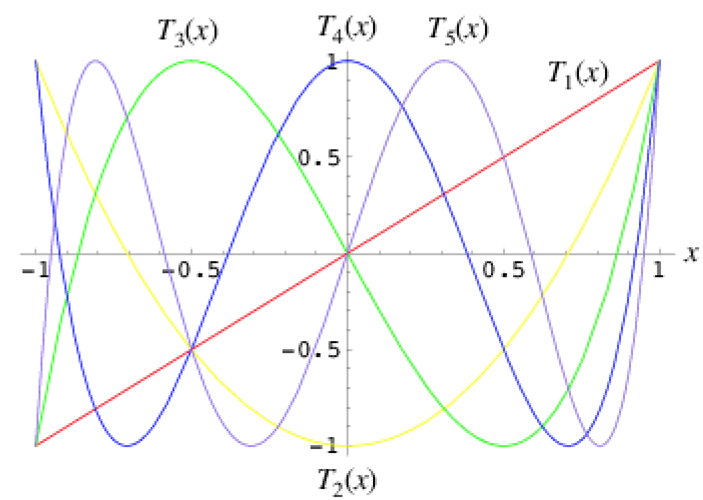
- Correlation of width s2 and fraction f reduced from 0.92 to 0.68
- Choice of parameterization matters!

- Strategy II – Fix all but one of the correlated parameters
  - If floating parameters are highly correlated, some of them may be redundant and not contribute to additional degrees of freedom in your model

Mitigating fit stability problems -- Polynomials

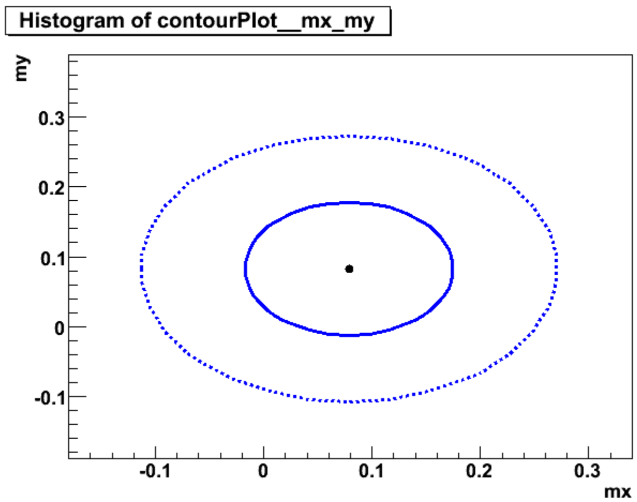
- **Warning:** Regular parameterization of polynomials $a_0+a_1x+a_2x^2+a_3x^3$ nearly always results in strong correlations between the coefficients a_i.
  - *Fit stability problems, inability to find right solution common at higher orders*
- **Solution:** Use existing parameterizations of polynomials that have (mostly) uncorrelated variables
  - ***Example: Chebychev polynomials***

$$
\begin{aligned}
T_0(x) &= 1 \\
T_1(x) &= x \\
T_2(x) &= 2x^2 - 1 \\
T_3(x) &= 4x^3 - 3x \\
T_4(x) &= 8x^4 - 8x^2 + 1 \\
T_5(x) &= 16x^5 - 20x^3 + 5x \\
T_6(x) &= 32x^6 - 48x^4 + 18x^2 - 1.
\end{aligned}
$$

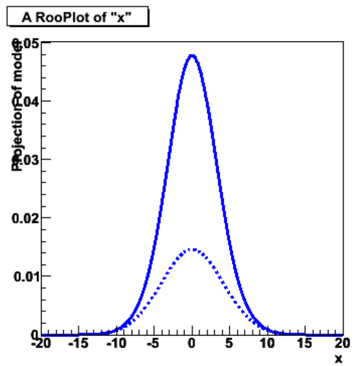


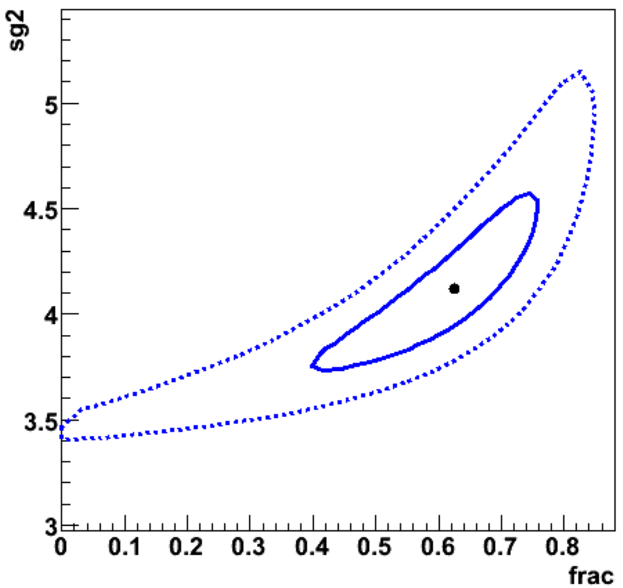
Minuit CONTOUR tool also useful to examine 'bad' correlations

- Example of 1,2 sigma contour of two uncorrelated variables
  - Elliptical shape. In this example parameters are uncorrelation



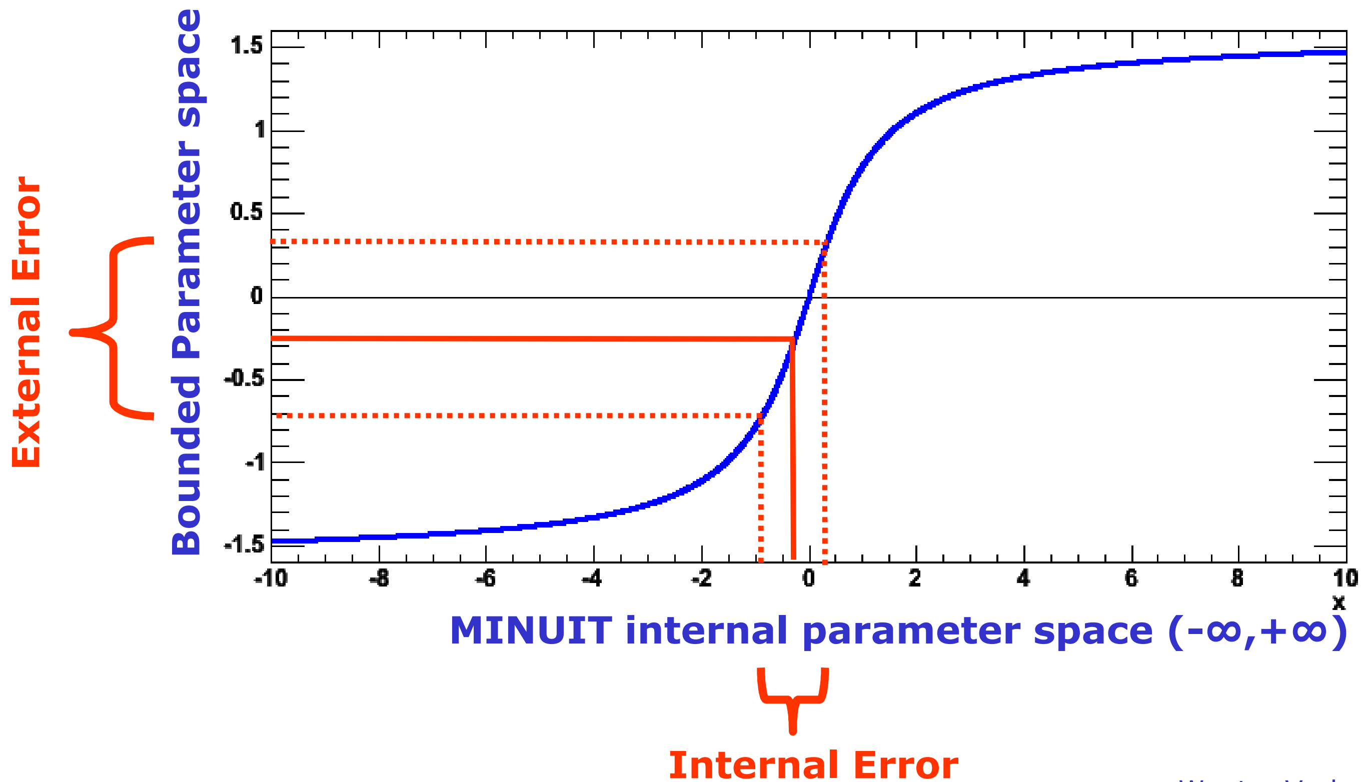
- Example of 1,2 sigma contour of two variables with problematic correlation
  - Pdf = f·G1(x,0,3)+(1-f)·G2(x,0,s) with s=4 in data





Practical estimation – Bounding fit parameters

- Sometimes is it desirable to bound the allowed range of parameters in a fit
  - Example: a fraction parameter is only defined in the range [0,1]
  - MINUIT option 'B' maps finite range parameter to an internal infinite range using an arcsin(x) transformation:

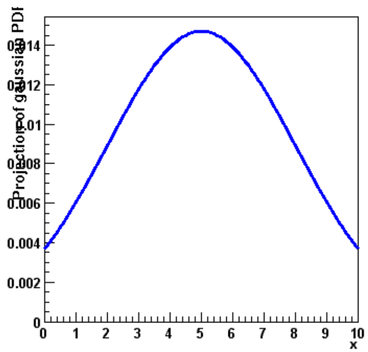


5 Multidimensional models

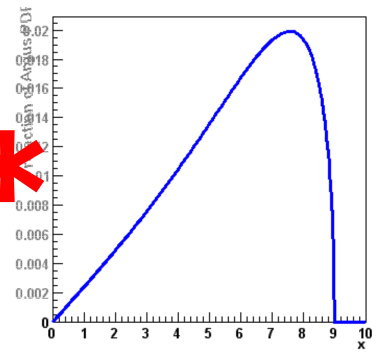
- *Uncorrelated products of p.d.f.s*
- *Using composition to p.d.f.s with correlation*
- *Products of conditional and plain p.d.f.s*

Building realistic models

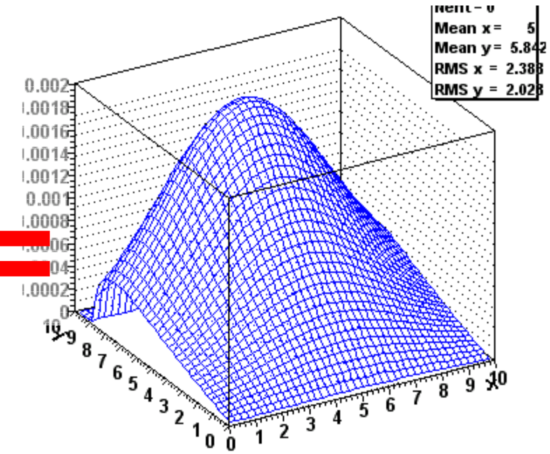
– Multiplication



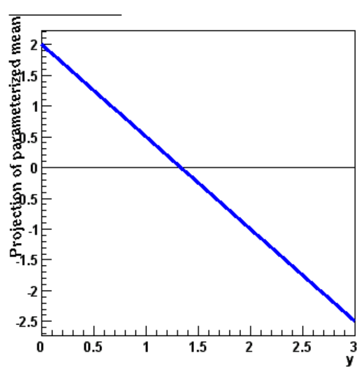
*

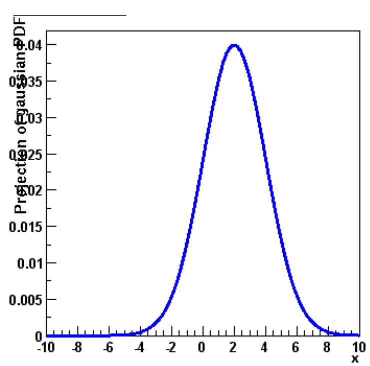


=

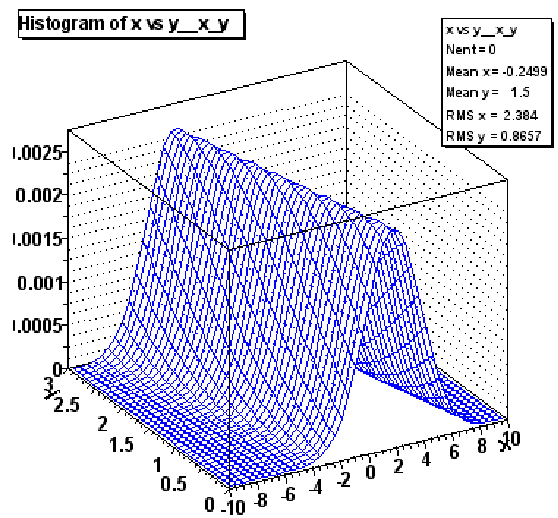


– Composition





=



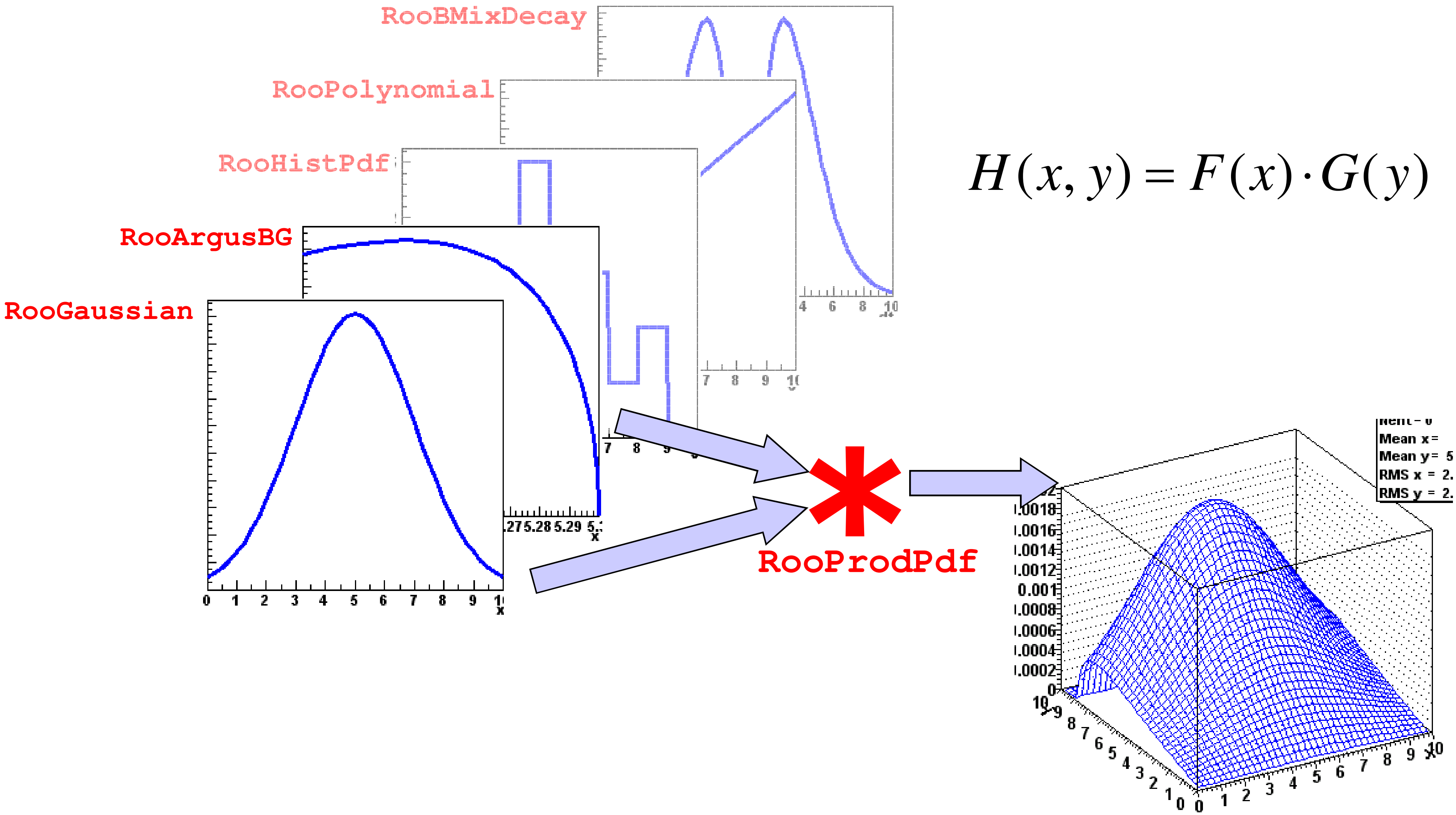
$m(y;a_0,a_1)$ $g(x;m,s)$

$g(x,y;a0,a1,s)$

Possible in *any* PDF
No explicit support in PDF code needed

Model building – Products of uncorrelated p.d.f.s

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



Uncorrelated products – Mathematics and constructors

- Mathematical construction of products of uncorrelated p.d.f.s is straightforward

2D

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

nD

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

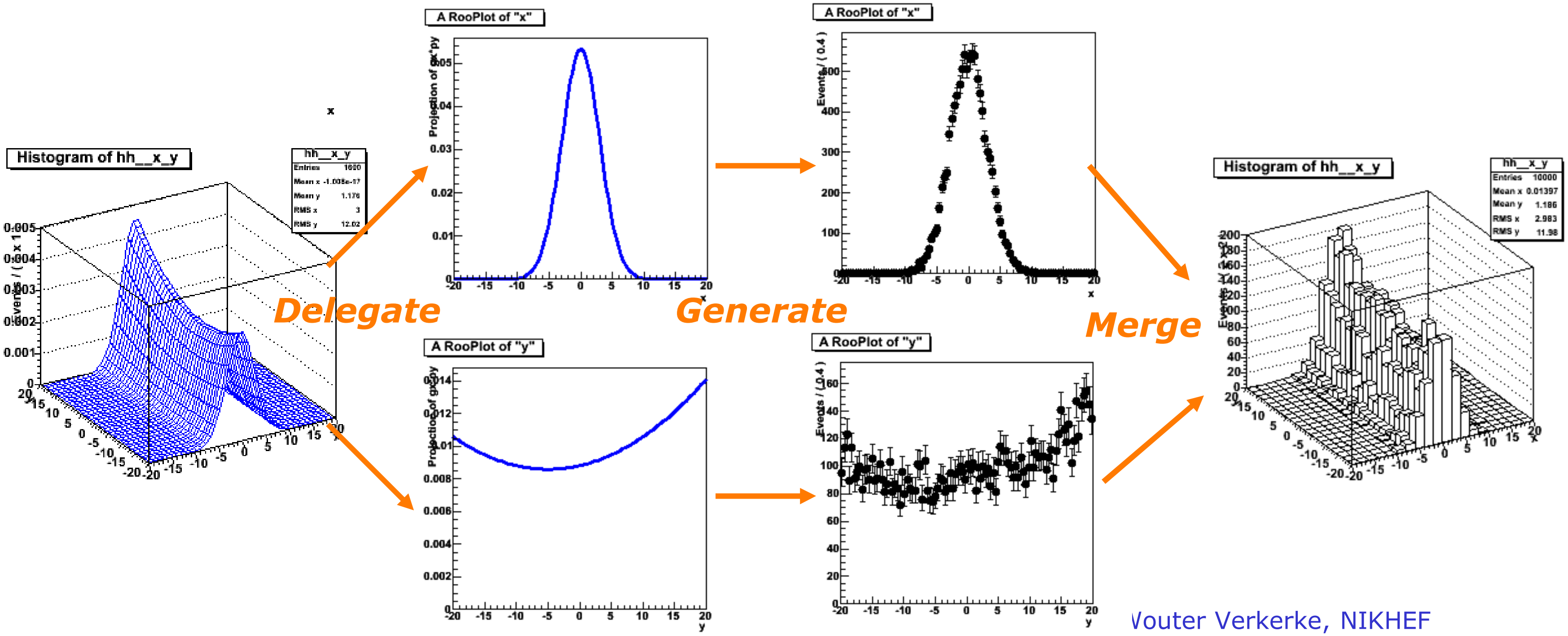
  - No explicit normalization required → If input p.d.f.s are unit normalized, product is also unit normalized
    (this is true *only* because of the absence of correlations)

- Corresponding factory operator is 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)") ;
```

How it work – event generation on uncorrelated products

- If p.d.f.s are uncorrelated, each observable can be generated separately
  - Reduced dimensionality of problem (important for e.g. accept/reject sampling)
  - Actual event generation delegated to component p.d.f (can e.g. use internal generator if available)
  - **`RooProdPdf`** just aggregates output in single dataset



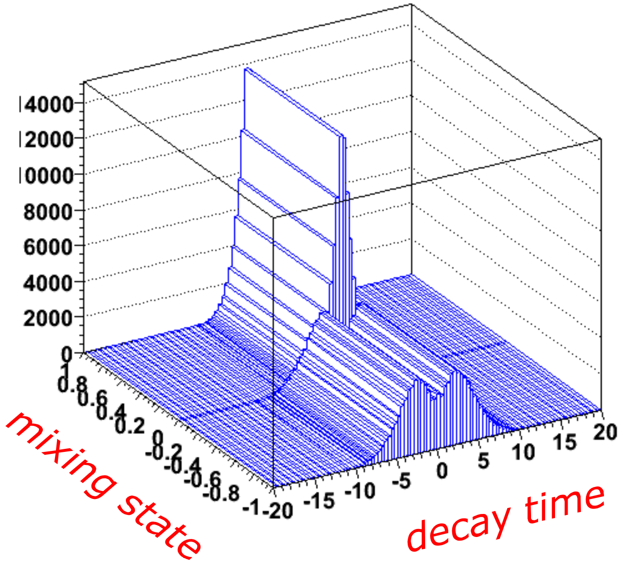
Fundamental multi-dimensional p.d.fs

- It also possible define multi-dimensional p.d.f.s that do not arise through a product construction
  - For example

```
EXPR::mypdf('sqrt(x+y)*sqrt(x-y)',x,y) ;
```

  - But *usually n-dim p.d.f.s are constructed more intuitively* through product constructs. Also correlations can be introduced efficiently (more on that in a moment)

- Example of fundamental 2-D B-physics p.d.f. **RooBMixDecay**
  - Two observables:
    decay time (t, continuous)
    mixingState (m, discrete [-1,+1])

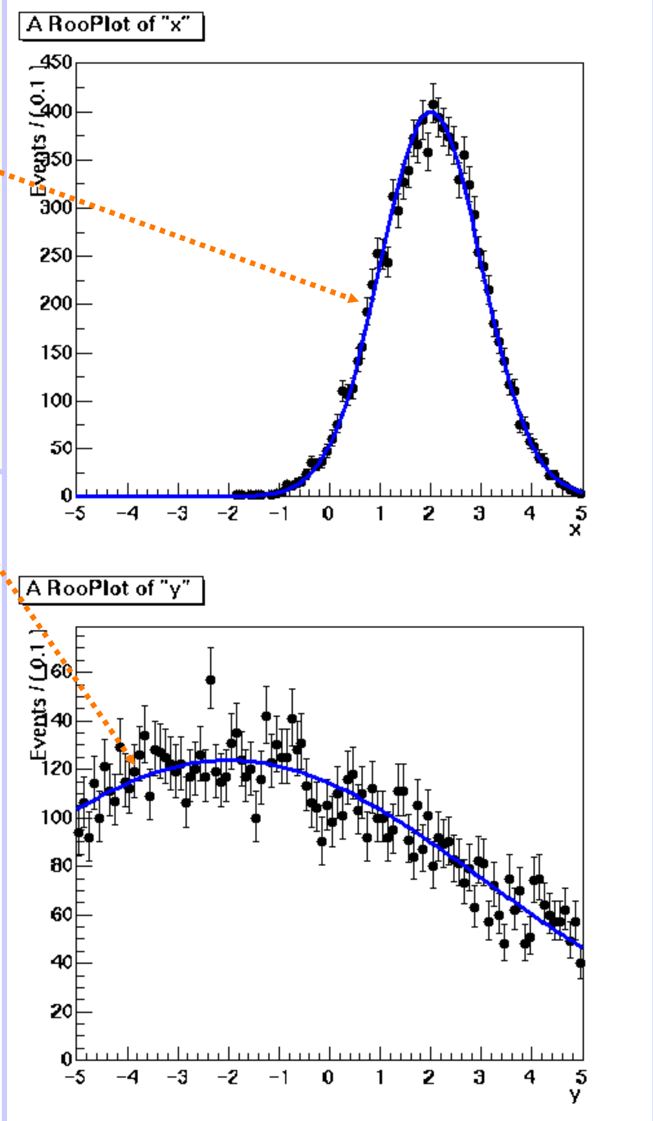


Plotting multi-dimensional PDFs

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

c->cd(2) ;
RooPlot* yframe = y.frame() ;
data->plotOn(yframe) ;
prod->plotOn(yframe) ;
yframe->Draw() ;
```

$$f(x) = \int pdf(x,y)dy$$

$$f(y) = \int pdf(x,y)dx$$



-Plotting a dataset D(x,y) versus x represents a *projection over y*

-To overlay PDF(x,y), you must plot *Int(dy)PDF(x,y)*

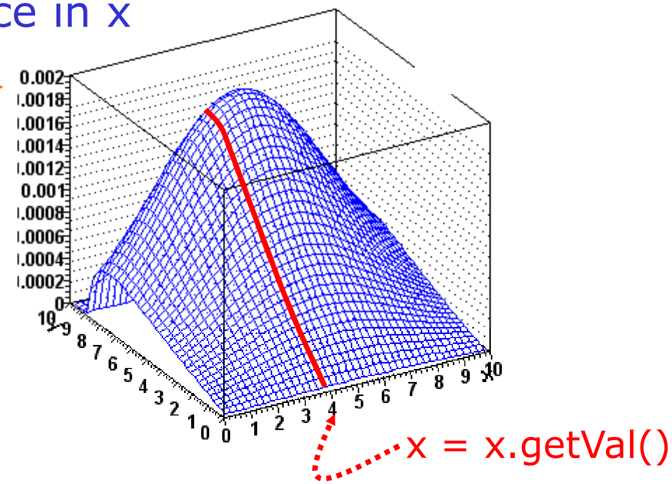
-RooFit automatically takes care of this!

- RooPlot remembers dimensions of plotted datasets

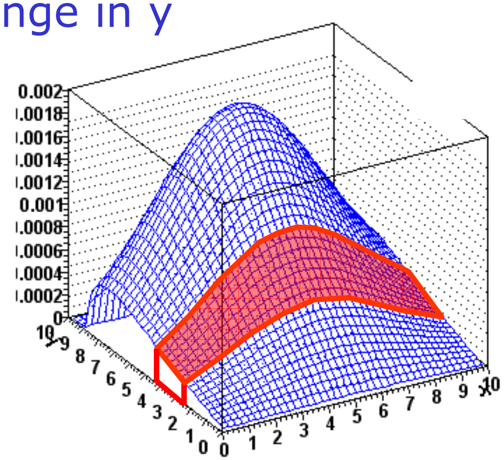
Introduction to slicing

- With multidimensional p.d.f.s it is also often useful to be able to plot a slice of a p.d.f
- In RooFit
  - A *slice* is thin
  - A *range* is thick
- Slices mostly useful in discrete observables
  - A slice in a continuous observable has no width and usually no data with the corresponding cut (e.g. "x=5.234")
- Ranges work for both continuous and discrete observables
  - Range of discrete observable can be list of >=1 state

Slice in x



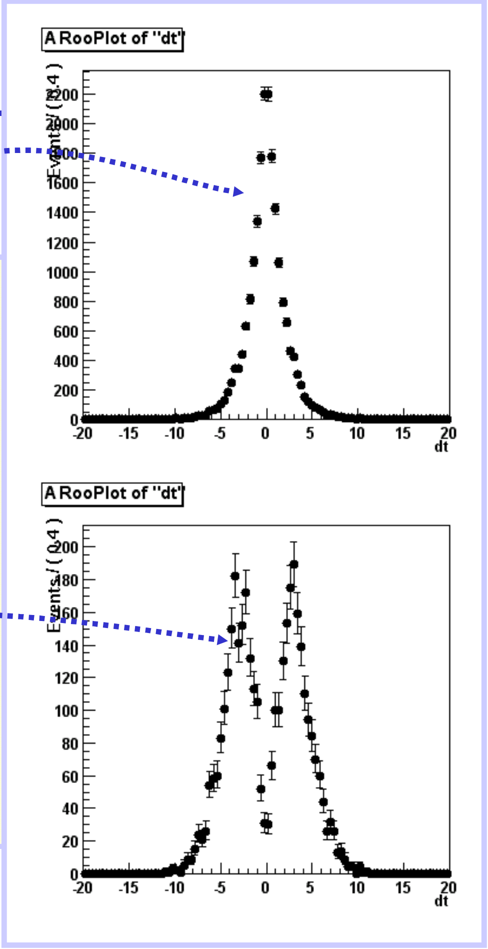
Range in y



Plotting a *slice* of a dataset

- Use the optional cut string expression

```
// Mixing dataset defines dt,mixState
RooDataSet* data ;

// Plot the entire dataset
RooPlot* frame = dt.frame() ;
data->plotOn(frame) ;

// Plot the mixed part of the data
RooPlot* frame_mix = dt.frame() ;
data->plotOn(frame,
              Cut("mixState==mixState::mixed"))
```

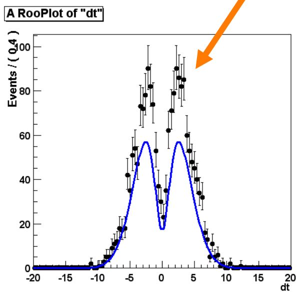


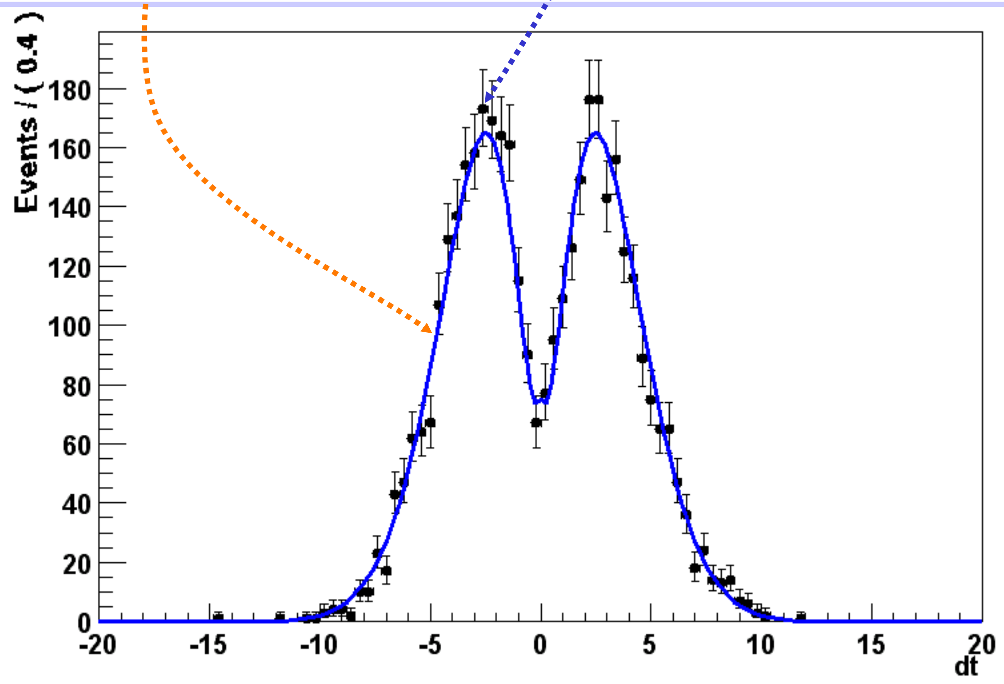
  – Works the *same* for *binned data* sets

Plotting a *slice* of a p.d.f

```
RooPlot* dtframe = dt.frame() ;
data->plotOn(dtframe,Cut("mixState==mixState::mixed")) ;

bmix.plotOn(dtframe,Slice(mixState,"mixed")) ;
dtframe->Draw() ;
```

For slices both data and p.d.f normalize with respect to full dataset. If fraction 'mixed' in above example disagrees between data and p.d.f prediction, this discrepancy will show in plot



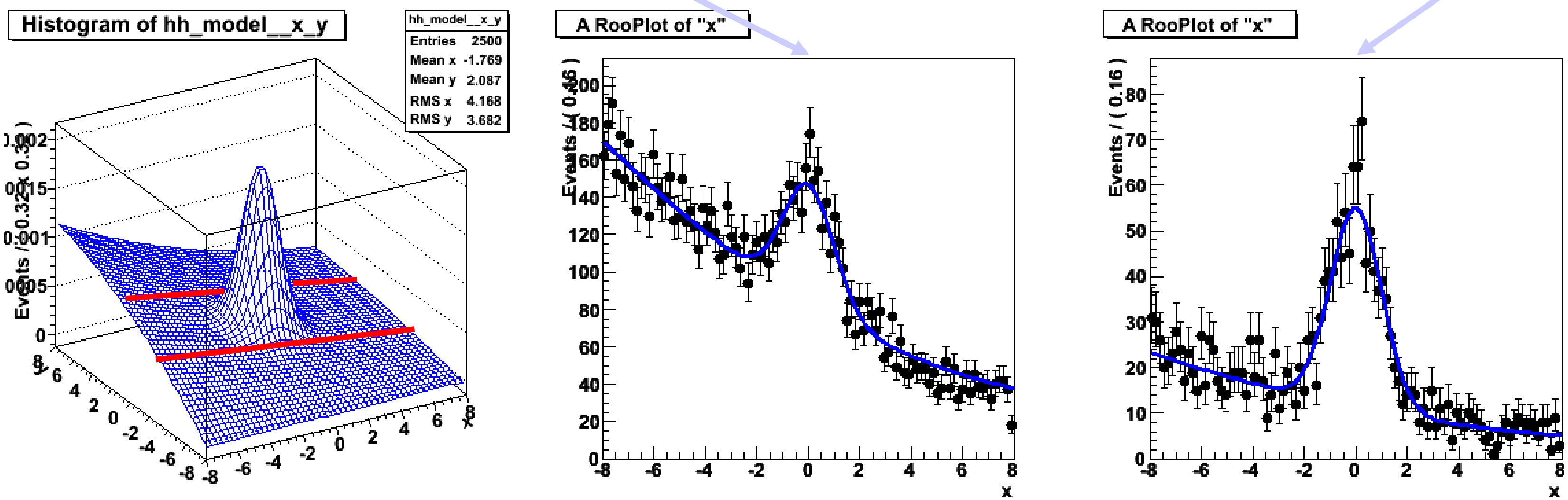


Plotting a *range* of a p.d.f and a dataset

model(x,y) = gauss(x)*gauss(y) + poly(x)*poly(y)

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

```
y.setRange("sig",-1,1) ;
RooPlot* xframe2 = x.frame() ;
data->plotOn(xframe2,CutRange("sig")) ;
model.plotOn(xframe2,ProjectionRange("sig")) ;
```



→ *Works also with >2D projections (just specify projection range on all projected observables)*

→ *Works also with multidimensional p.d.fs that have correlations*

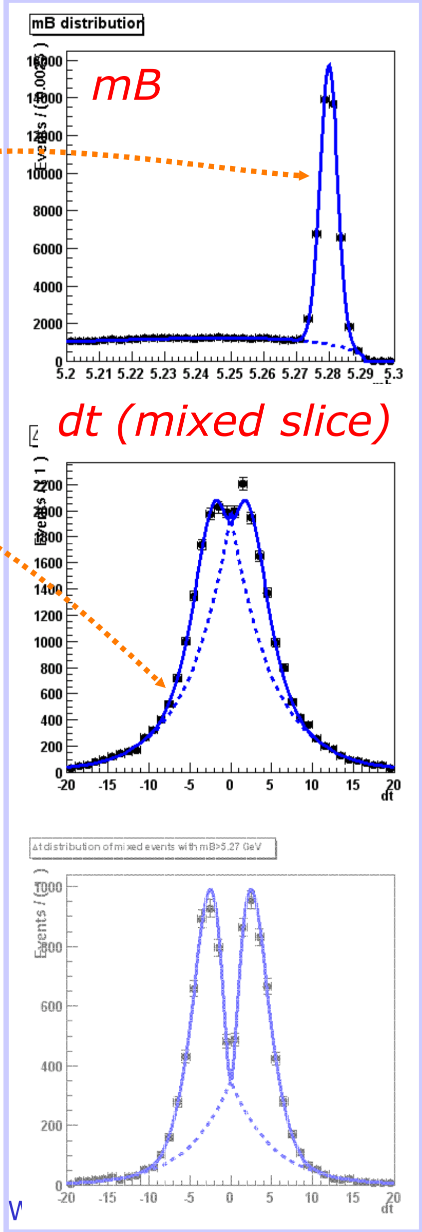
Physics example of combined range and slice plotting

Example setup:

Argus(mB)*Decay(dt) + (background)

Gauss(mB)*BMixDecay(dt) (signal)

```
// Plot projection on mB
RooPlot* mbframe = mb.frame(40) ;
data->plotOn(mbframe) ;
model.plotOn(mbframe) ;

// Plot mixed slice projection on deltat
RooPlot* dtframe = dt.frame(40) ;
data>plotOn(dtframe,
            Cut(”mixState==mixState::mixed”)) ;
model.plotOn(dtframe,Slice(mixState,”mixed”)) ;
```



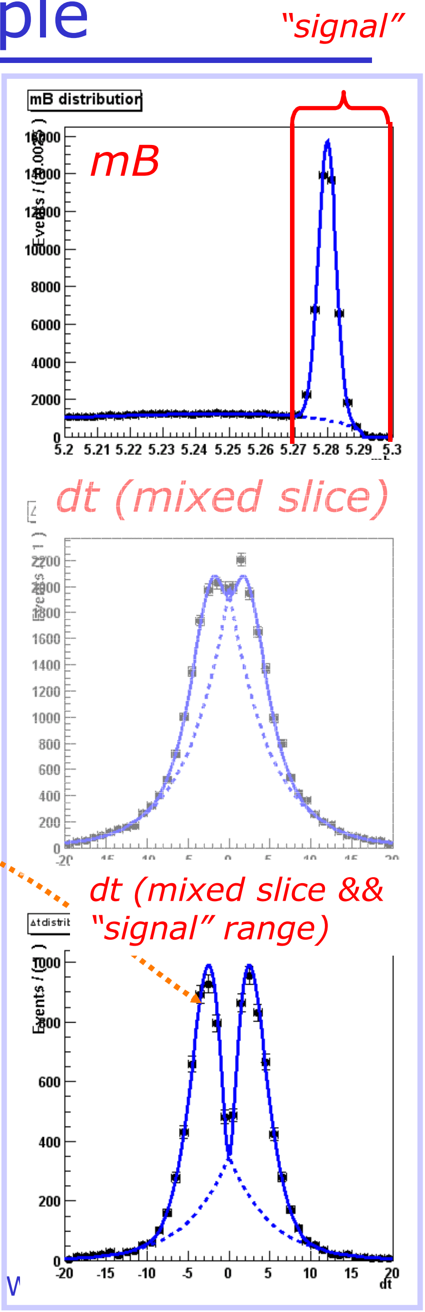
Plotting slices with finite width - Example

Example setup:

Argus(mB)*Decay(dt) + (background)

Gauss(mB)*BMixDecay(dt) (signal)

```
mb.setRange("signal",5.27,5.30) ;

mbSliceData->plotOn(dtframe2,
   Cut("mixState==mixState::mixed"),
   CutRange("signal"))

model.plotOn(dtframe2,Slice(mixState,"mixed"),
             ProjectionRange("signal"))
```

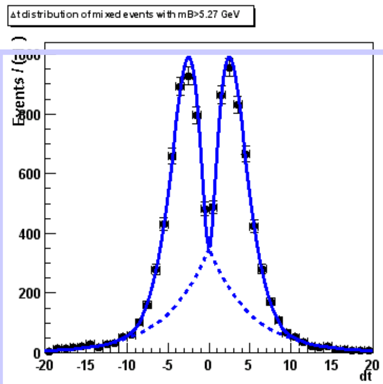


Plotting slices with finite width - Example

- We can also plot the finite width slice with a different technique → toy MC integration

```
// Generate 80K toy MC events from p.d.f to be projected
RooDataSet *toyMC =
       model.generate(RooArgSet(dt,mixState,tagFlav,mB),80000);

// Apply desired cut on toy MC data
RooDataSet* mbSliceToyMC = toyMC->reduce("mb>5.27");

// Plot data requesting data averaging over selected toy MC data
model.plotOn(dtframe2,Slice(mixState),ProjWData(mb,mbSliceToyMC))
```

$$\int M(x,y,z)\,dy\,dz \approx \frac{1}{N}\sum_{D(y,z)} M(x,y_i,z_i)$$



Plotting non-rectangular PDF regions

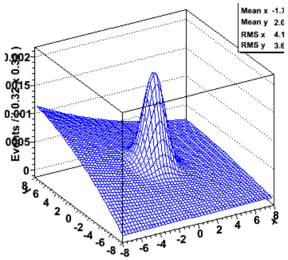
- Why is this interesting? Because with this technique we can trivially implement projection over arbitrarily shaped regions.
  - Any cut prescription that you can think of to apply to data works

$$(x-5)^2+(y-3)^2<4 \qquad \text{'donut'}$$

- Example: Likelihood ratio projection plot
  - Common technique in rare decay analyses
  - PDF typically consist of N-dimensional event selection PDF, where N is large (e.g. 6.)
  - Projection of data & PDF in any of the N dimensions doesn't show a significant excess of signal events
  - To demonstrate purity of selected signal, plot data distribution (with overlaid PDF) in one dimension, while selecting events with a cut on the likelihood ratio of signal and background in the remaining N-1 dimensions

Likelihood ratio plots

- Idea: use information on S/(S+B) ratio in projected observables to define a cut
- Example: generalize previous toy model to 3 dimensions



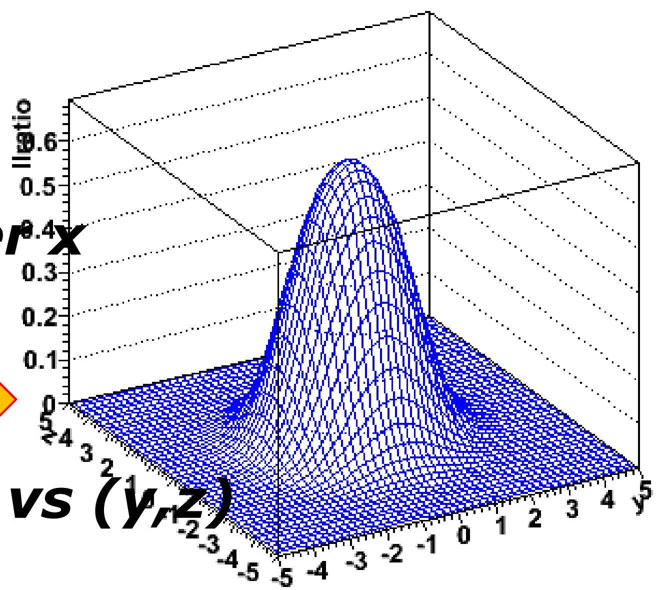
- Express information on S/(S+B) ratio of model in terms of integrals over model components

$$LR(x,y,z) = \frac{f \cdot S(x,y,z)}{\left[f \cdot S(x,y,z) + (1-f)B(x,y,z)\right]}$$

- *Integrate over x*

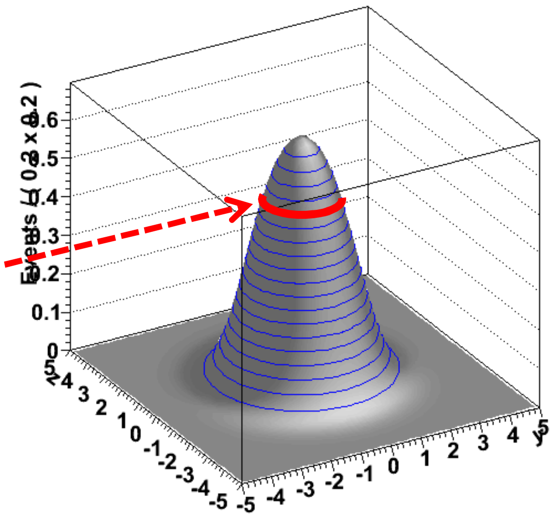
$$LR(y,z) = \frac{\int f \cdot S(x,y,z)dx}{\int \left[f \cdot S(x,y,z) + (1-f)B(x,y,z)\right]dx}$$

- *Plot LR vs (y,z)*



Likelihood ratio plots

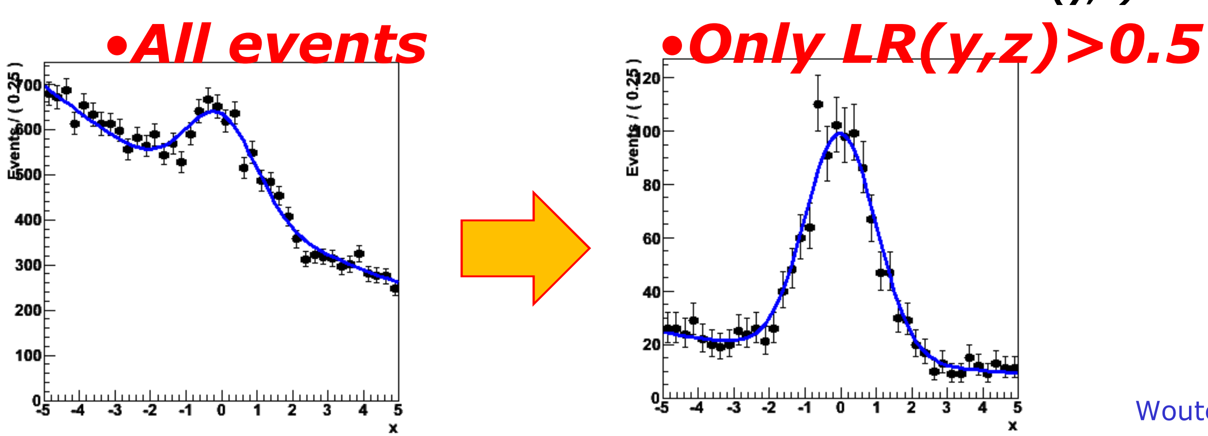
- Decide on s/(s+b) purity contour of LR(y,z)
  - Example s/(s+b) > 50%



- Plot both data and model with corresponding cut.
  - For data: calculate LR(y,z) for each event, plot only event with LR>0.5
  - For model: using Monte Carlo integration technique:

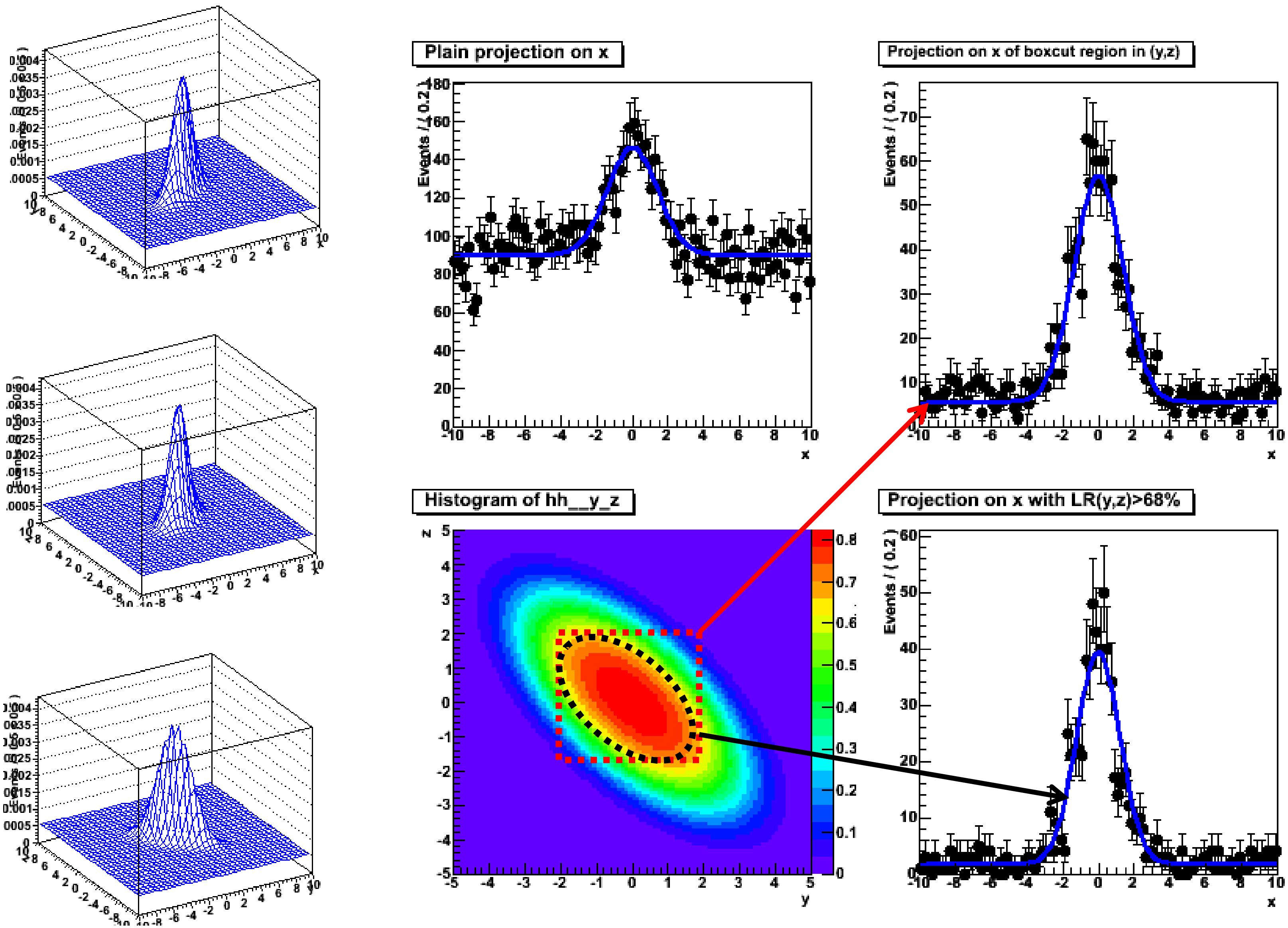
$$\int_{LR(y,z)>0.5} M(x,y,z)\,dy\,dz \approx \frac{1}{N}\sum_{D(y,z)} M(x,y_i,z_i)$$

•Dataset with values of (y,z) sampled from p.d.f and filtered for events that meet LR(y,z)>0.5



Wouter Verkerke, NIKHEF

Likelihood ratio plot on model with correlations



Likelihood ratio plots – Coded example

```
// Construct likelihood ratio in projection on (y,z)
w.factory("expr::LR('fsig*psig/ptot',fsig,
                    PROJ::psig(sig,x),PROJ::ptot(model,x))") ;
```

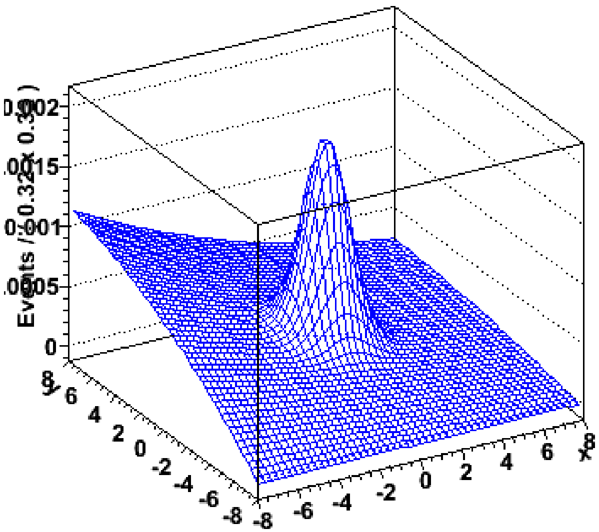
$$LR(y,z) = \frac{\int f \cdot S(x,y,z)dx}{\int \left[f \cdot S(x,y,z) + (1-f)B(x,y,z) \right] dx}$$

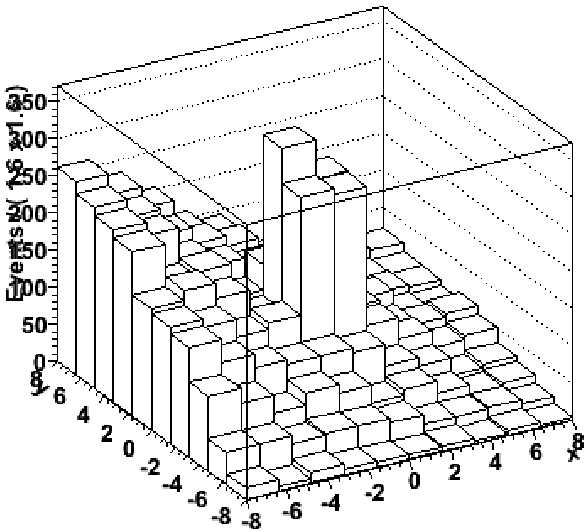
```
// Generate toy dataset for MC integration over region with LR>68%
RooDataSet* tmpdata = model.generate(RooArgSet(x,y,z),10000) ;
tmpdata->addColumn(*w.function("LR")) ;
RooDataSet* projdata = (RooDataSet*) tmpdata->reduce(Cut("LR>0.68")) ;

// Add LR to observed data so we can cut on it
data->addColumn(*w.function("LR")) ;
RooDataSet* seldata = (RooDataSet*) data->reduce(Cut("LR>0.68")) ;

// Make plot for data and pdf
RooPlot* frame3 = x.frame(Title("Projection with LR(y,z)>68%")) ;
seldata->plotOn(frame3) ;
model.plotOn(frame3,ProjWData(*projdata)) ;
```

Plotting in more than 2,3 dimensions

- No equivalent of RooPlot for >1 dimensions
  - Usually >1D plots are not overlaid anyway
- Easy to use **createHistogram()** methods provided in both **RooAbsData** and **RooAbsPdf** to fill ROOT 2D,3D histograms

```
TH2D* ph2 = pdf.createHistogram(“ph2”,x,YVar(y)) ;

TH2* dh2 = data.createHistogram(“dg2",x,Binning(10),
                                YVar(y,Binning(10)));
ph2->Draw("SURF") ;
dh2->Draw("LEGO") ;
```





rkerke, NIKHEF

Building models – Introducing correlations

- Easiest way to do this is
  - start with 1-dim p.d.f. and change on of its parameters into a function that depends on another observable

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

  - Natural way to think about it

- Example problem
  - Observable is reconstructed mass M of some object.
  - Fitting Gaussian g(M,mean,sigma) some background to dataset D(M)
  - But reconstructed mass has *bias* depending on some other observable X
  - Rewrite fit functions as g(M,meanCorr(mtrue,X,alpha),sigma) where meanCorr is an (emperical) function that corrects for the bias depending on X

Introducing correlations through composition

- 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

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

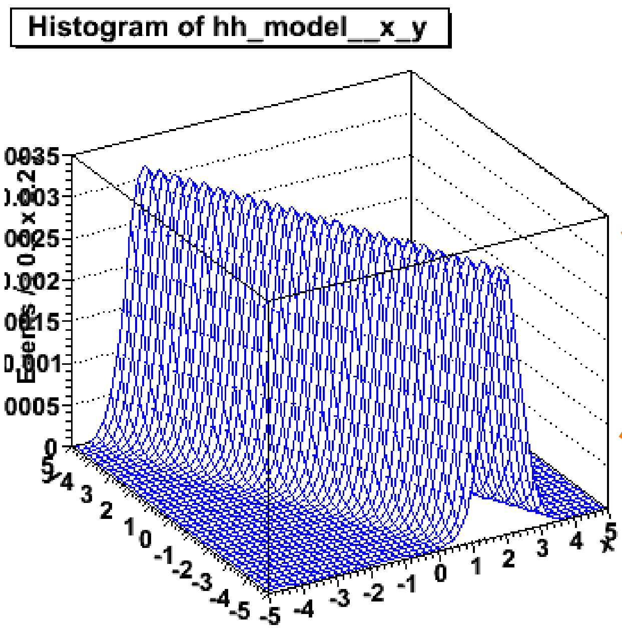
  - Example: Gaussian with shifting mean

```
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])") ;
```

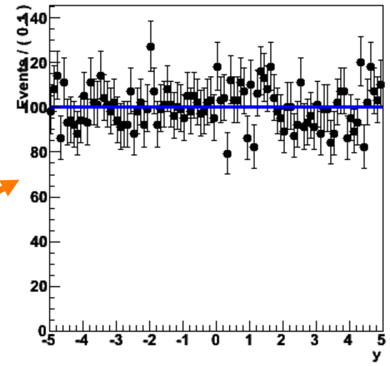
  - No assumption made in function on a,b,x,y being observables or parameters, any combination will work

What does the example p.d.f look like?

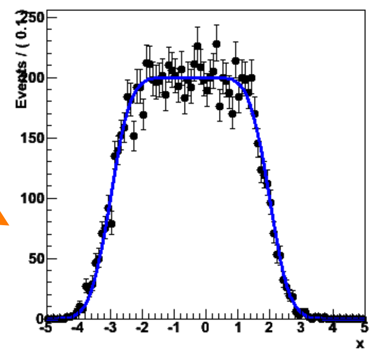
- Use example model with x,y as observables



Projection on Y



Projection on X



- Note flat distribution in y. Unlikely to describe data, solutions:
  1. Use as conditional p.d.f g(x|y,a,b)
  2. Use in conditional form multiplied by another pdf in y: g(x|y)*h(y)

Conditional p.d.f.s – Formulation and construction

- Mathematical formulation of a conditional p.d.f
  - A conditional p.d.f is not normalized w.r.t its conditional observables

$$F(\vec{x} \mid \vec{y}; \vec{p}) = \frac{f(\vec{x}, \vec{y}, \vec{p})}{\int f(\vec{x}, \vec{y}, \vec{p}) d\vec{x}}$$

  - Note that denominator in above expression *depends on y* and is thus in general different for each event

- Constructing a conditional p.d.f in RooFit
  - Any RooFit p.d.f can be used as a conditional p.d.f as objects have no internal notion of distinction between parameters, observables and conditional observables
  - Observables that should be used as conditional observables have to be specified in use context (generation, plotting, fitting etc…)

Method 1 – Using a conditional p.d.f – fitting and plotting

- For fitting, indicate in fitTo() call what the conditional observables are

```
pdf.fitTo(data,ConditionalObservables(y))
```

$$F(x\,|\,y) = \frac{f(x,y)}{\int f(x,y)d\vec{x}}$$

  - You may notice a performance penalty if the normalization integral of the p.d.f needs to be calculated numerically. For a conditional p.d.f it must evaluated again for each event

- Plotting: You cannot project a conditional *F*(*x*|*y*) on *x* without external information on the distribution of *y*
  - Substitute integration with averaging over y values in data

Integrate over *y*

$$P_p(x) = \frac{\int p(x,y)dy}{\int p(x,y)dxdy}$$

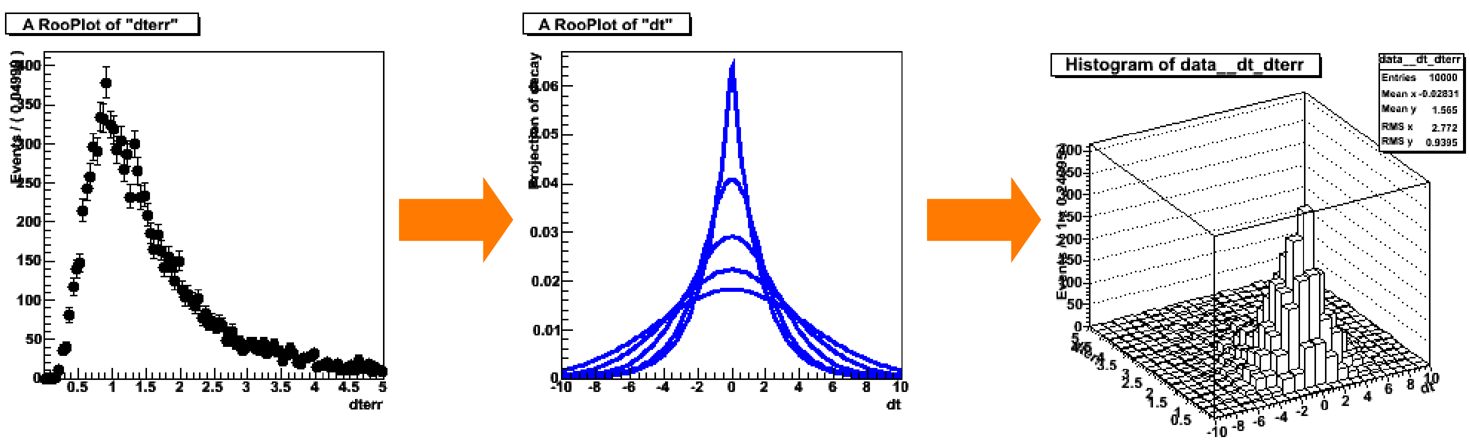
⇒

Sum over all y_i in dataset *D*

$$P_p(x) = \frac{1}{N}\sum_{D}^{i=1,N} \frac{p(x,y_i)}{\int p(x,y_i)dx}$$

How it works – event generation with conditional p.d.f.s

- Just like plotting, event generation of conditional p.d.f.s requires external input on the conditional observables
  - Given an external input dataset ***P***(*dt*)
  - For each event in ***P***,
    set the value of *dt* in $F(d|dt)$ to dt_i
    generate one event for observable *t* from $F(t|dt_i)$
  - Store both t_i and dt_i in the output dataset



Physics example with conditional p.d.f.s

- Want to fit decay time distribution of B0 mesons (exponential) convoluted with Gaussian resolution

$$F(t) = D(t;\tau) \otimes R(t, m, \sigma)$$

- However, resolution on decay time varies from event by event (e.g. more or less tracks available).
  - We have in the data an error estimate dt for each measurement from the decay vertex fitter ("*per-event error*")
  - Incorporate this information into this physics model

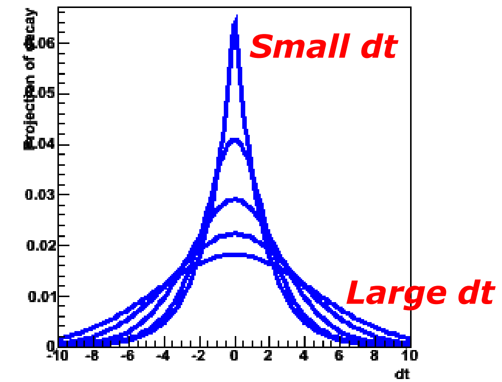
$$F(t \mid \delta t) = D(t;\tau) \otimes R(t, m, \sigma \cdot \delta t)$$

  - Resolution in physics model is adjusted for each event to expected error.
  - Overall scale factor σ can account for incorrect vertex error estimates (i.e. if fitted $\sigma > 1$ then dt was underestimate of true error)
  - Physics p.d.f must used conditional conditional p.d.f because it give no sensible prediction on the distribution of the per-event errors

Physics example with conditional p.d.f.s

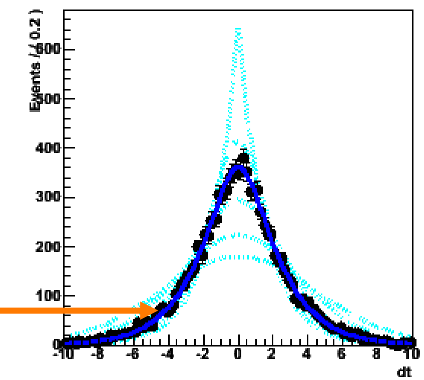
- Some illustrations of decay model with per-event errors
  - Shape of $F(t|\delta t)$ for several values of δt

$$F(t\,|\,\delta t) = D(t;\tau) \otimes R(t, m, \sigma \cdot \delta t)$$



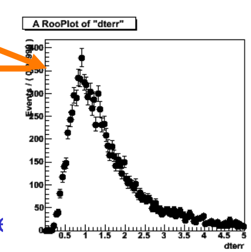
- Plot of *D(t)* and *F(t|dt)* projected over *dt*

```
// Plotting of decay(t|dterr)
 RooPlot* frame = dt.frame() ;
 data->plotOn(frame2) ;
 decay_gm1.plotOn(frame2,ProjWData(*data)) ;
```



Note that projecting over large datasets can be slow. You can speed this up by projecting with a binned copy of the projection data

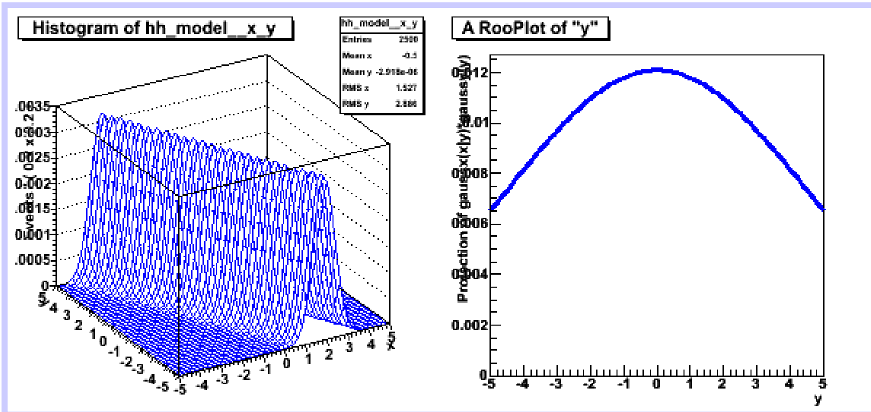
$$P_p(x) = \frac{1}{N} \sum_{D}^{i=1,N} \frac{p(x, y_i)}{\int p(x, y_i)\,dx}$$



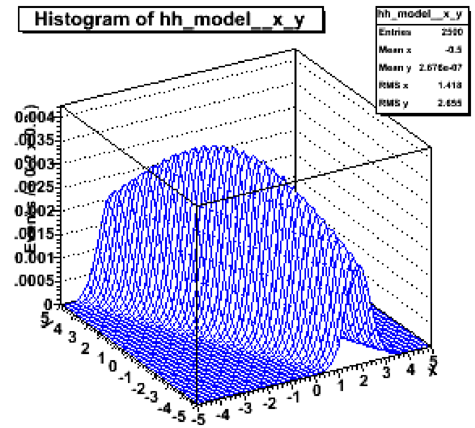
Method 2 – Building products with conditional pdfs

- Use of conditional pdf in fitting, plotting, event generation has some *practical* drawbacks
  - Need external dataset with distribution in conditional observable in all operations
- But there is also a *fundamental* issue
  - If your model has both a signal and a background component, the model assumes that the distribution of the conditional observable (e.g. the per-event error) is the same for signal and background
  - This may not be a valid assumption (‘Punzi effect’)
  - Way out: Construct a product F(x|y)*G(y) separately for signal and background

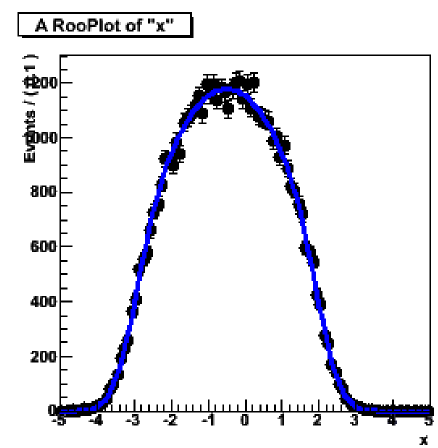
Example with product of conditional and plain p.d.f.



gx(x|y) * gy(y) = model(x,y)




```
// I - Use g as conditional pdf g(x|y)
w::g.fitTo(data,ConditionalObservables(w::y)) ;

// II - Construct product with another pdf in y
w.factory("Gaussian::h(y,0,2)") ;
w.factory("PROD::gxy(g|y,h)") ;
```



$$\int gx(x|y)g(y)dy$$

Example with product of conditional and plain p.d.f.

- Following the ‘conditional product’ formalism you can now choose different distributions for the conditional observable for signal and background e.g.

$$F(t,dt) = S(t\,|\,dt)s(dt) + B(t\,|\,dt)b(dt)$$

- At this point F(t,dt) is a plain pdf: fitting plotting and event generation works ‘as usual’ without external input
- You may want to use an empirical pdf for s(dt) or b(dt) if these distributions are difficult to model
  - Histogram based pdf (RooHistPdf)
  - Kernel estimatin pdf (RooKeysPdf) → Set next slide

Special pdfs – Kernel estimation model

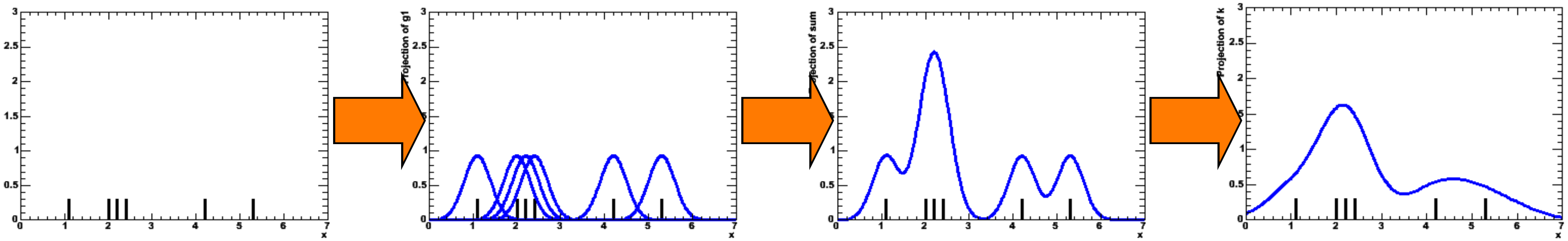
- Kernel estimation model
  - Construct smooth pdf from unbinned data, using kernel estimation technique

Sample of events

Gaussian pdf for each event

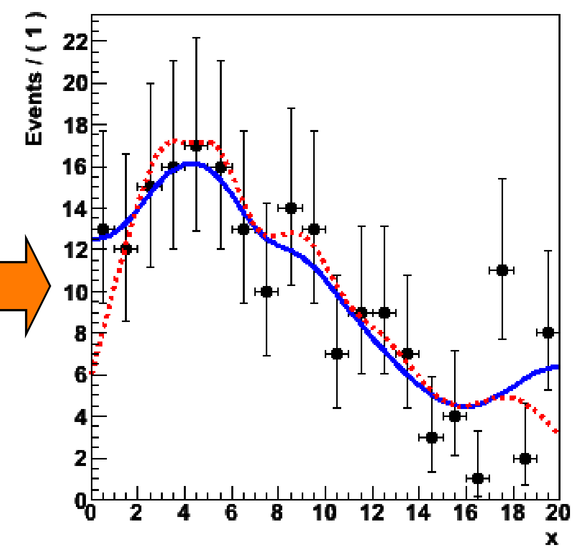
Summed pdf for all events

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

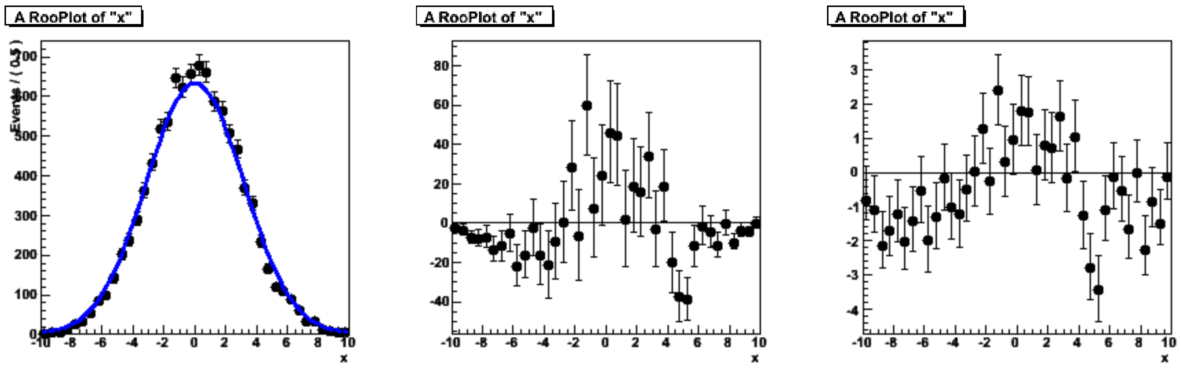
6

Fit validation, Toy MC studies

- *Goodness-of-fit, c2*
- *Toy Monte Carlo studies for fit validation*

How do you know if your fit was 'good'

- Goodness-of-fit broad issue in statistics in general, will just focus on a few specific tools implemented in RooFit here
- For one-dimensional fits, a χ^2 is usually the right thing to do
  - Some tools implemented in RooPlot to be able to calculate χ^2/ndf of curve w.r.t data



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

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

GOF in >1D, other aspects of fit validity

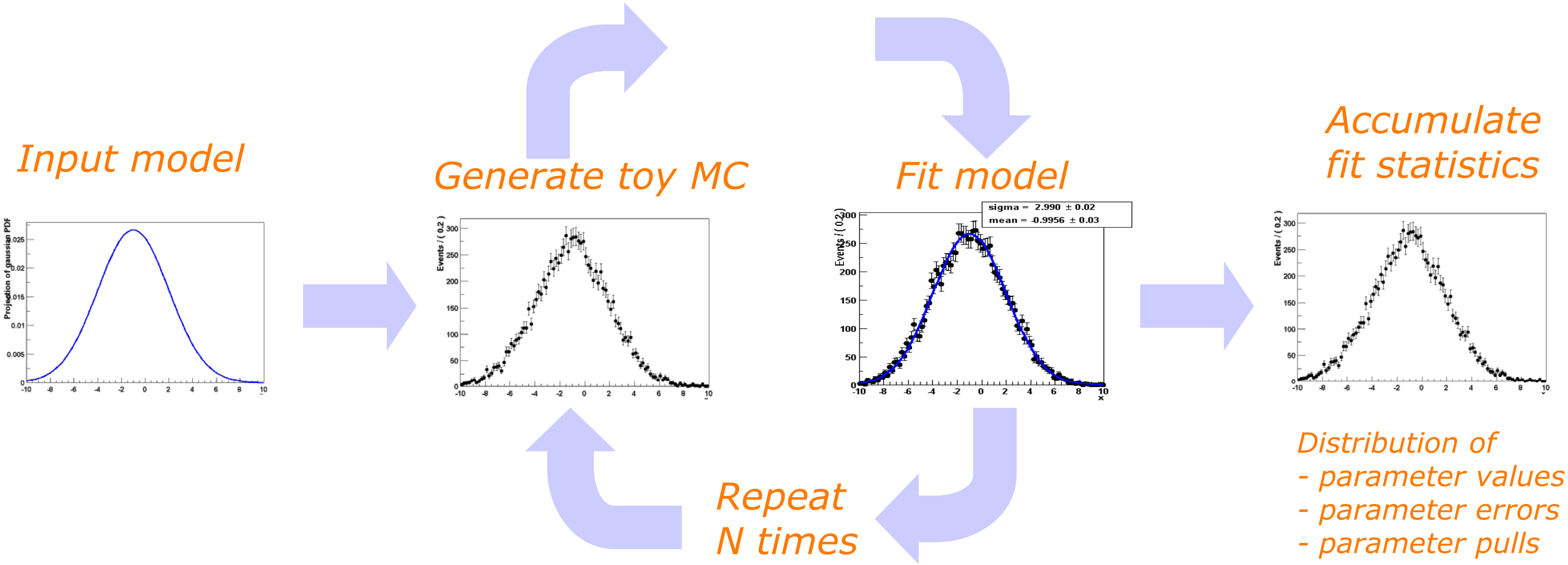
- No special tools for >1 dimensional goodness-of-fit
  - A χ^2 usually doesn't work because empty bins proliferate with dimensions
  - But if you have ideas you'd like to try, there exists generic base classes for implementation that provide the same level of computational optimization and parallelization as is done for likelihoods (**`RooAbsOptTestStatistic`**)

- But you can study many other aspect of your fit validity
  - Is your fit unbiased?
  - Does it (often) have convergence problems?

- You can answer these with a toy Monte Carlo study
  - I.e. generate 10000 samples from your p.d.f., fit them all and collect and analyze the statistics of these 10000 fits.
  - The **`RooMCStudy`** class helps out with the logistics

Advanced features – Task automation

- Support for routine task automation, e.g. goodness-of-fit study




```
// Instantiate MC study manager
RooMCStudy mgr(inputModel) ;

// Generate and fit 100 samples of 1000 events
mgr.generateAndFit(100,1000) ;

// Plot distribution of sigma parameter
mgr.plotParam(sigma)->Draw()
```

How to *efficiently* generate multiple sets of ToyMC?

- Use **`RooMCStudy`** class to manage generation and fitting
- Generating features
  - *Generator overhead only incurred once*
    → Efficient for large number of small samples
  - Optional Poisson distribution for #events of generated experiments
  - Optional automatic creation of ASCII data files
- Fitting
  - Fit with generator PDF or different PDF
  - Fit *results* (floating parameters & NLL) automatically *collected in summary dataset*
- Plotting
  - Automated plotting for distribution of parameters, parameter errors, pulls and NLL
- Add-in modules for optional modifications of procedure
  - Concrete tools for variation of generation parameters, calculation of likelihood ratios for each experiment
  - Easy to write your own. You can intervene at any stage and offer proprietary data to be aggregated with fit results

A `RooMCStudy` example

- Generating and fitting a simple PDF

```
// Setup PDF
RooRealVar x("x","x",-5,15) ;
RooRealVar mean("mean","mean of gaussian",-1) ;
RooRealVar sigma("sigma","width of gaussian",4) ;
RooGaussian gauss("gauss","gaussian PDF",x,mean,sigma) ;


// Create manager
RooMCStudy mgr(gauss,gauss,x,"","mhv") ;


// Generate and fit 1000 experiments of 100 events each
mgr.generateAndFit(1000,100) ;
RooMCStudy::run: Generating and fitting sample 999
RooMCStudy::run: Generating and fitting sample 998
RooMCStudy::run: Generating and fitting sample 997
…
```

Generator PDF

Generator Options

Fitting PDF

Observables

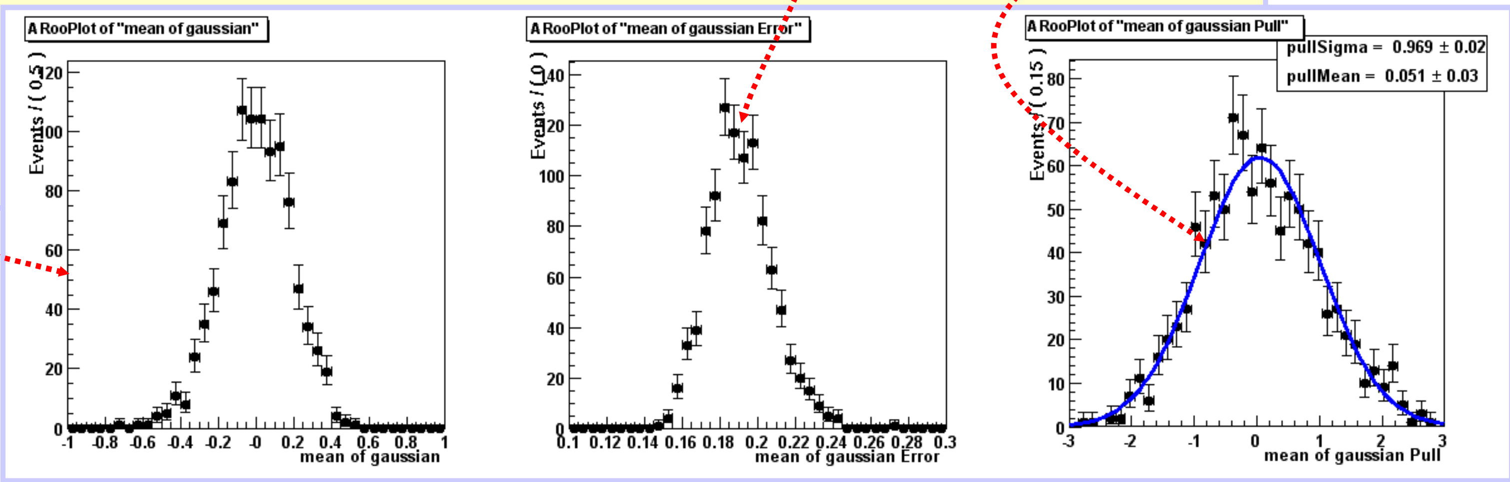
Fitting Options

A `RooMCStudy` example

- Plot the distribution of the value, error and pull of *mean*

```
// Plot the distrution of the value
RooPlot* mframe = mean.frame(-2,0) ;
mgr.plotParamOn(mframe) ;
mframe->Draw() ;

// Plot the distrution of the error
RooPlot* meframe = mgr.plotError(mean,0.,0.1) ;
meframe->Draw() ;

// Plot the distrution of the pull
RooPlot* mpframe = mgr.plotPull(mean,-3,3,40,kTRUE) ;
mpframe->Draw() ;
```

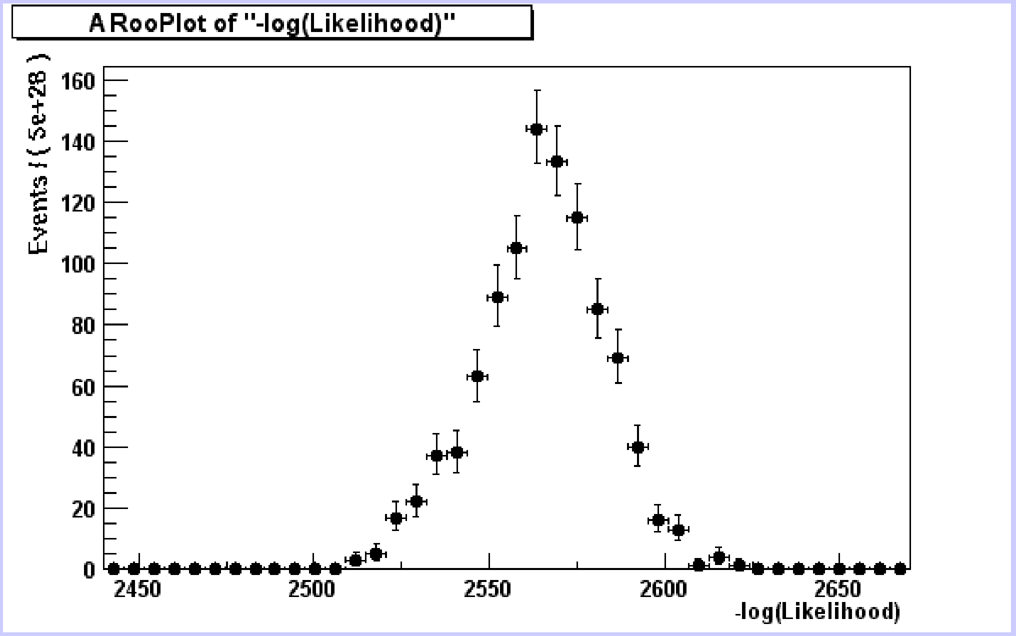
Add Gaussian fit



A **RooMCStudy** example

- Plot the distribution of –log(L)

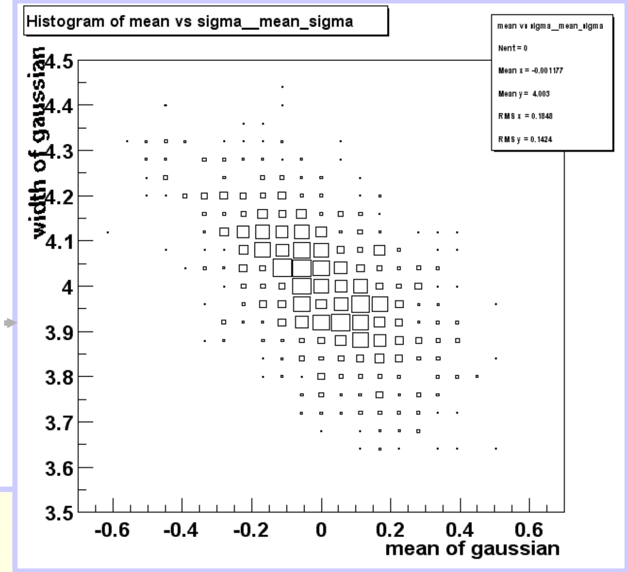
```
// Plot the distribution of the NLL
mgr.plotNLL(mframe) ;
mframe->Draw() ;
```



- NB: likelihood distributions cannot be used to deduce goodness-of-fit information!

A `RooMCStudy` example

- For other uses, use summarized fit results in **`RooDataSet`** form

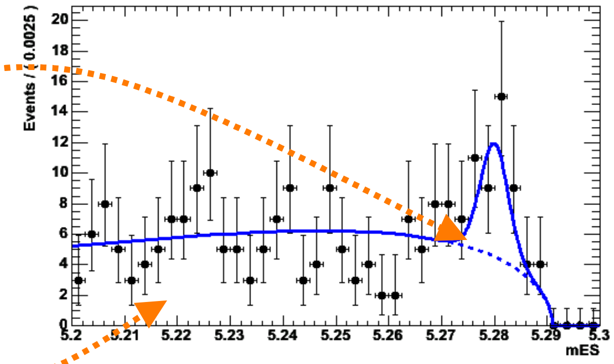



```
mgr.fitParDataSet().get(10)->Print("v") ;
RooArgSet:::
  1) RooRealVar::mean      :  0.14814 +/- 0.191 L(-10 - 10)
  2) RooRealVar::sigma     :  4.0619 +/- 0.143 L(0 - 20)
  3) RooRealVar::NLL       :  2585.1 C
  4) RooRealVar::meanerr   :  0.19064 C
  5) RooRealVar::meanpull  :  0.77704 C
  6) RooRealVar::sigmaerr  :  0.14338 C
  7) RooRealVar::sigmapull :  0.43199 C

TH2* h = mean.createHistogram("mean vs sigma",sigma) ;
mgr.fitParDataSet().fillHistogram(h,RooArgList(mean,sigma)) ;
h->Draw("BOX") ;
```

Pulls and errors have separate entries for easy access and plotting

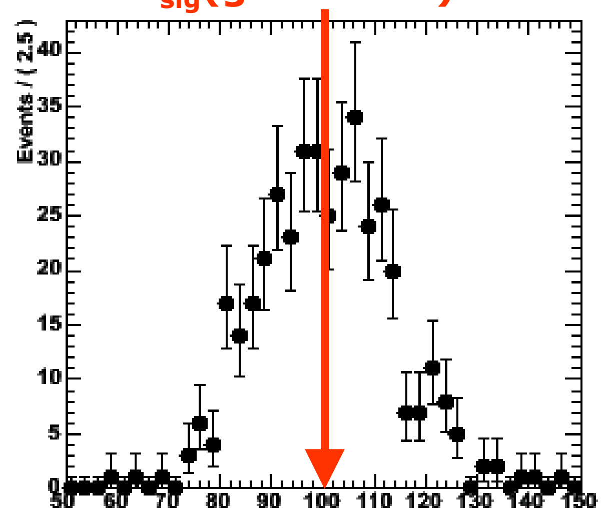
Fit Validation Study – Practical example

- Example fit model in 1-D (B mass)
  - Signal component is Gaussian centered at B mass
  - Background component is Argus function (models phase space near kinematic limit)



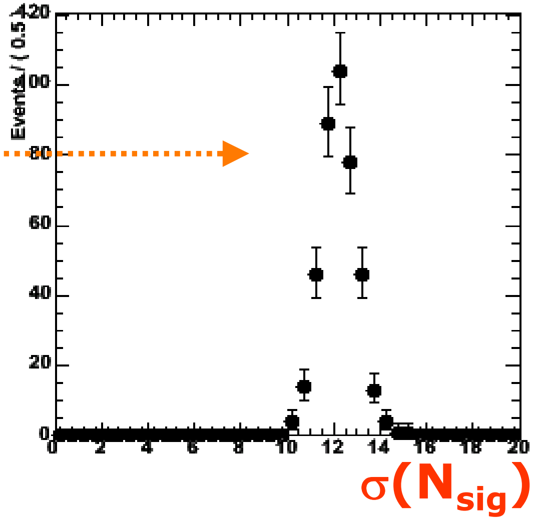
$$F(m; N_{\text{sig}}, N_{\text{bkg}}, \vec{p}_S, \vec{p}_B) = N_{\text{sig}} \cdot G(m; p_S) + N_{\text{bkg}} \cdot A(m; p_B)$$

- Fit parameter under study: $\boldsymbol{N_{sig}}$
  - Results of simulation study: 1000 experiments with **N_{SIG}(gen)=100, N_{BKG}(gen)=200**
  - Distribution of N_{sig}(fit)
  - This particular fit looks unbiased…

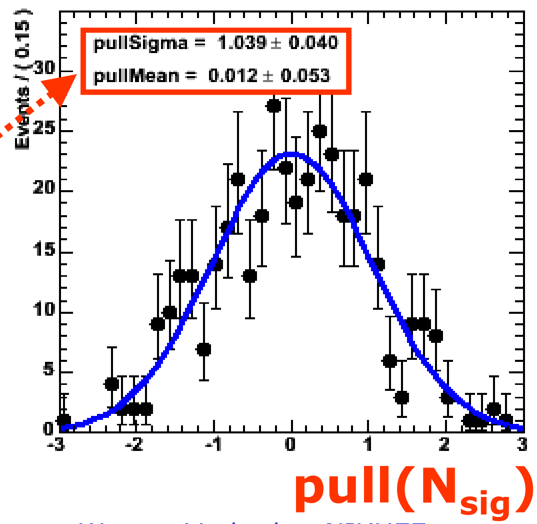


Fit Validation Study – The pull distribution

- What about the validity of the error?
  - Distribution of error from simulated experiments is difficult to interpret…
  - We don't have equivalent of N_{sig}(generated) for the error



- Solution: look at the ***pull distribution***
  - Definition: $$\text{pull}(N_{sig}) = \frac{N_{sig}^{fit} - N_{sig}^{true}}{\sigma_N^{fit}}$$
  - Properties of pull:
    - **Mean is 0 if there is no bias**
    - **Width is 1 if error is correct**
  - In this example: no bias, correct error within statistical precision of study



Wouter Verkerke, NIKHEF

Fit Validation Study – Low statistics example

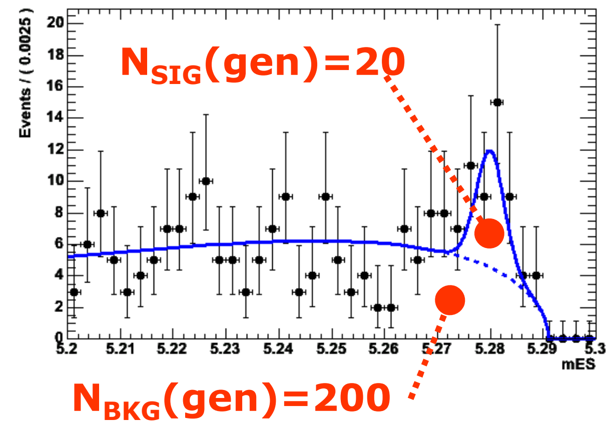
- Special care should be taken when fitting small data samples
  - Also if fitting for small signal component in large sample

- Possible causes of trouble
  - χ^2 estimators may become approximate as Gaussian approximation of Poisson statistics becomes inaccurate
  - ML estimators may no longer be efficient
    → error estimate from 2nd derivative may become inaccurate
  - Bias term proportional to 1/N of ML and χ^2 estimators may no longer be small compared to 1/sqrt(N)

- In general, absence of bias, correctness of error can not be assumed. How to proceed?
  - Use unbinned ML fits only – most robust at low statistics
  - **Explicitly verify the validity of your fit**

Demonstration of fit bias at low N – pull distributions

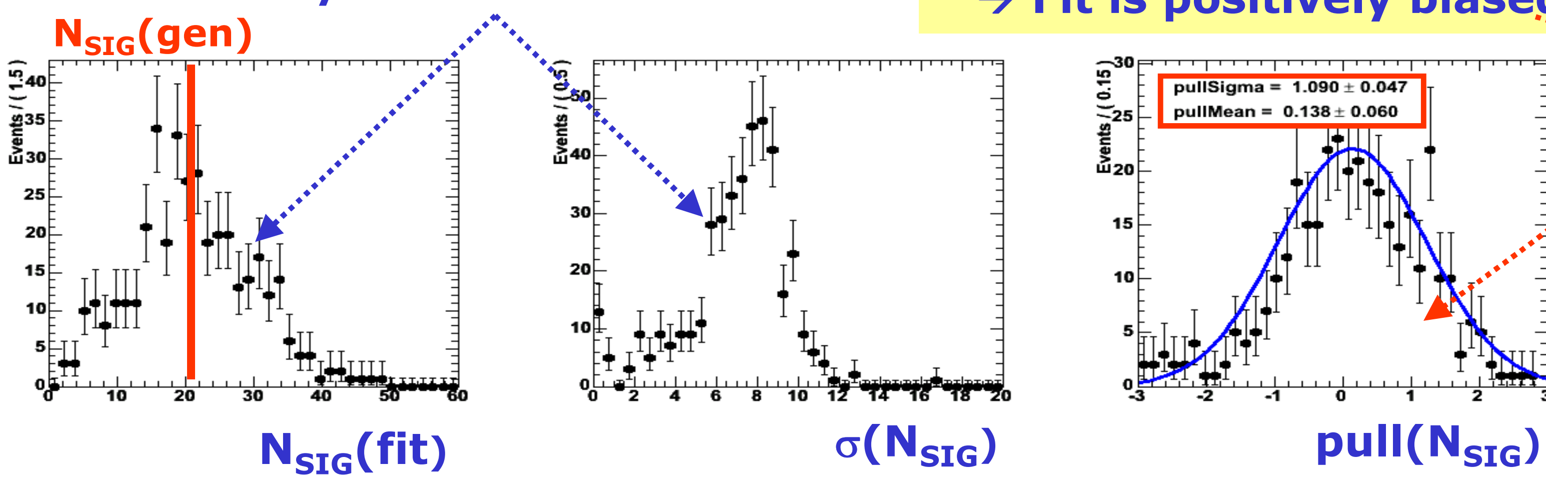
- Low statistics example:
  - Scenario as before but now with 200 bkg events and **only 20 signal events** (instead of 100)



- Results of simulation study

Distributions become asymmetric at low statistics

Pull mean ~2σ away from 0 → Fit is positively biased!



- *Absence of bias, correct error at low statistics not obvious*

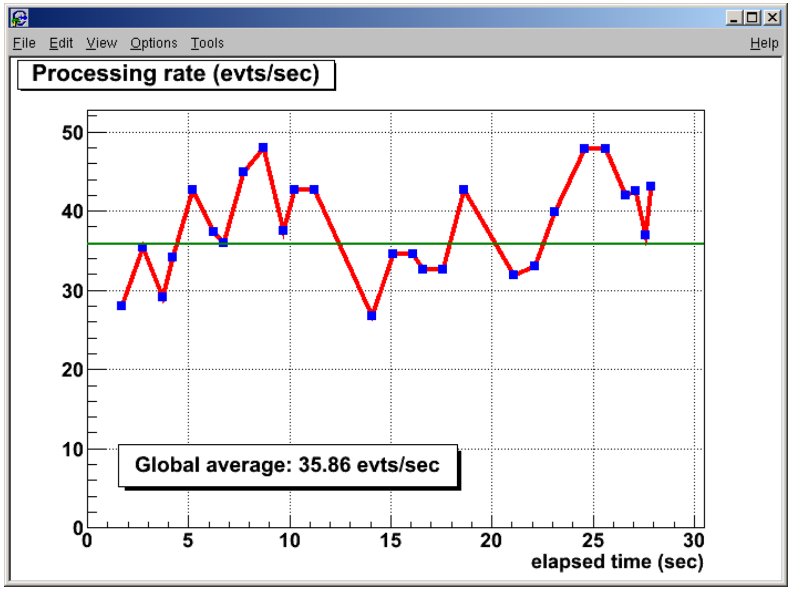
New developments for automated studies

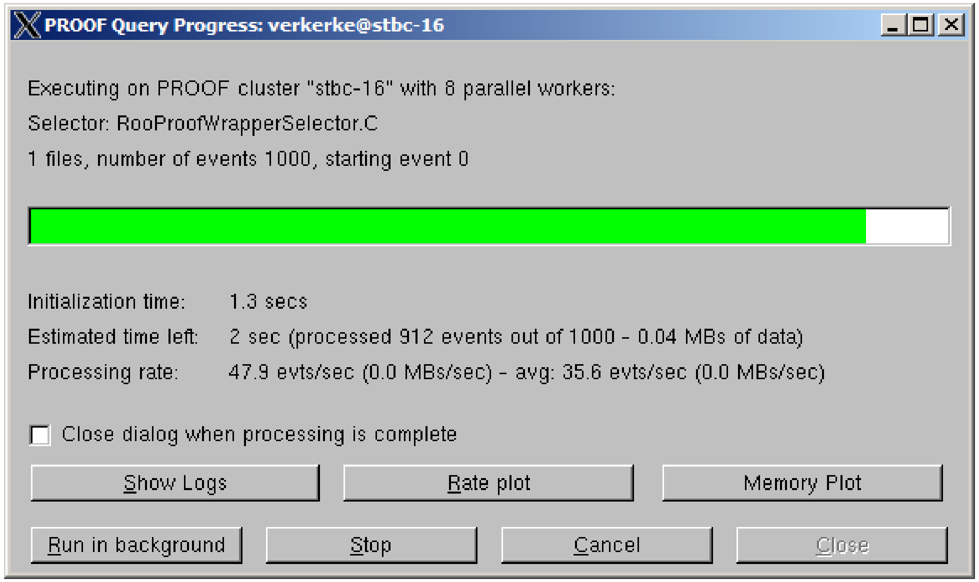
- A new alternative framework is being put in place to replace class RooMCStudy.
  - Class RooStudyManager manages logistics of repeated studies, but does not implement content of study.
  - Abstract concept of study interfaced through class RooAbsStudy
  - Class RooGenFitStudy manages implementation of 'generate-and-fit' style studies (functionality of RooMCStudy)
- Greater flexibility in choice of study (you can put in anything you want)
- Support for multiple backend implementations
  - Inline calculation (as done in RooMCStudy)
  - Parallelized execution through PROOF (lite)
  - Almost complete automation of support for batch submission
  - Just need to change one line of your macro to change back-end

Demo of parallelization with PROOF-lite

- Example – Factor 8 speed up on a dual-quad core box.
  - Works with out-of-the box ROOT distribution
  - Also: Graceful early termination when users presses 'Stop'

```
RooStudyManager mcs(*w,gfs) ;
mcs.run(1000) ; // inline running
mcs.runProof(1000,"") ; // empty string is PROOF-lite
mcs.prepareBatchInput("default",1000,kTRUE) ;
```





- Much larger gains can be made with 'real' PROOF farms

* Exercises

Exercise 3

- Take input file ex3.C, look at it and run it.
  - This file defines a signal pdf and a background pdf summed in a combined pdf.
  - The signal pdf is a B decay distribution with mixing in observable t and a Gaussian in observable m_{ES}.
  - The background pdf is a plain decay distribution in observable t and an Argus shape in observable m_{ES}
  - Both signal and background decay distributions are convoluted with a Gaussian resolution of fixed width.
  - The macro generates 1000 events, fits the model to this data and makes plots of the distributions for mES, t(mixed events) and t(unmixed events)
- Step 1 – Introduce per-event errors
  - Now we modify the pdf to included per-event errors. Class RooGaussModel has multiple constructors (look at the code in $ROOTSYS/include). We will now use the 'second' constructor which takes an extra argument in the constructor, which multiplies both the width and mean of the Gaussian. Create a new observable dt with range[0.1,5] and supply it as 4^{th} argument to the factory string that makes the resolution model

Exercise 3

– We have now modified the resolution model so that the width is scaled with the per-event error, which is different for each event. The total pdf 'model' is now ready to me used as conditional pdf F(t,mes|dt)

– To proceed generation/fitting/plotting part of the macro we also need to generate a dummy dataset with per-event errors to be used later for event generation, fitting and plotting operations. Add the following pdf to the workspace

```
Landau::sig_dt(dt,1,0.5)
```

using the factory and generate a RooDataSet named dtdata from it with 1000 events.

  - Put a 'return' statement in the macro and verify that the code runs OK up to here.

– Modify the generation call to make the 'main' dataset to take *dtdata as argument instead of 1000. This will instruct the generator to take the dt values from dtdata as input in the generation step (It is no longer necessary to specify the number of events to generate as this is implicit from the size of dtdata)

  - Put a 'return' statement in the macro and verify that the code runs OK up to here.

Exercise 3

- Modify the call to fitTo() by adding argument ConditionalObservables(*w.var("dt")), which will change the normalization of the pdf used in the fit: instread of normalizing w.r.t. (t,dt), the normalization is only performed over t, but recalculated for each value of dt.
  - Put a 'return' statement in the macro and verify that the code runs OK up to here.
- Modify the plotting code. Add to each plotOn() call for the pdf an argument 'ProjWData(*dtdata)' which will instruct the plotting operation to perform the projection over dt by averaging over the values in the provided dataset instead of integrating the pdf over dt. Verify that all plots look OK.
- In the step before the projections over dt are performed using the unbinned dataset and take relatively long. Replace each ProjWData(*dtdata) with ProjWData(*dtdata,kTRUE) to request averaging over a binned dataset in dt (default = 100 bins) which will speed up the projections by a factor 10.
  - The solution of step 1 is available in ~verkerke/solutions/ex3step1.C

- Step 2 – Add plots for the signal region
  - Define the signal region in mES as follows

    w.var("mes")->setRange("signal",5.27,5.29) ;

Exercise 3

- Replicate the code that makes plots frame1 and frame2 (dt distribution for mixed and unmixed) and modify the replica to make plots frame3 and frame4. Change the canvas layout from a (3,1) to a (3,2) layout (change the size of the canvas accordingly) and plot frame3 and frame4 on pads 5 and 6 respectively (pad 4 will remain empty)
- Now modify the code that makes plots frame3 and frame4 as follows: to the data->plotOn() calls add an argument CutRange("signal"), to the pdf->plotOn() calls add an argument ProjectionRange("signal").
  - The solution of step 2 is available in ~verkerke/solutions/ex3step2.C

- Step 3 – Add a pdf for dt to the model
  - In this step we will introduce an explicit model for the distribution of dt in the pdf so that we construct a plain pdf F(t,dt,mes) = F(t|dt)*G(dt)*H(mes) for both signal and background
  - Move the factory line that makes sig_dt above the line that constructs the signal product pdf. Modify the product construction such that it says 'PROD::sig(sig_m,sig_t|dt,sig_dt)'. Replicate the line that makes 'sig_dt' to make an identical pdf named 'bkg_dt'. Then modify the background product pdf similar to what as done for the signal pdf.

Exercise 3

  - Now the pdf has been modified to a regular pdf we can revert the code that uses the pdf to its original state: 1) In the event generation step replace *dtdata with 1000. 2) In fitTo() remove the ConditionalObservables() argument. 3) In all of the plotOn() calls remove the ProjWData() arguments. Now run again.
    - The solution of step 3 is available in ~verkerke/solutions/ex3step3.C

- Step 4 – Add observable deltaE to the model.
  - Add Gaussian signal model for observable deltaE to the workspace

    ```
    Gaussian::sig_de(de[-1,1],demean[0,-1,1],
                              dewidth[0.1,0.01,1])
    ```

  - Add a flat background model for observable deltaE to the workspace

    ```
    Polynomial::bkg_de(de)
    ```

  - Add pdfs sig_de and bkg_de to the products sig and bkg respectively
  - Add observable de to the list of observables defined by RooArgSet obs
  - Define a signal range in de (just below the def. of that range in mes)

    ```
    w.var("de")->setRange("signal",-0.2,0.2)
    ```

  - Increase the number of events generated to 10000 and run the macro again.
    - The solution of step 4 is available in ~verkerke/solutions/ex3step4.C