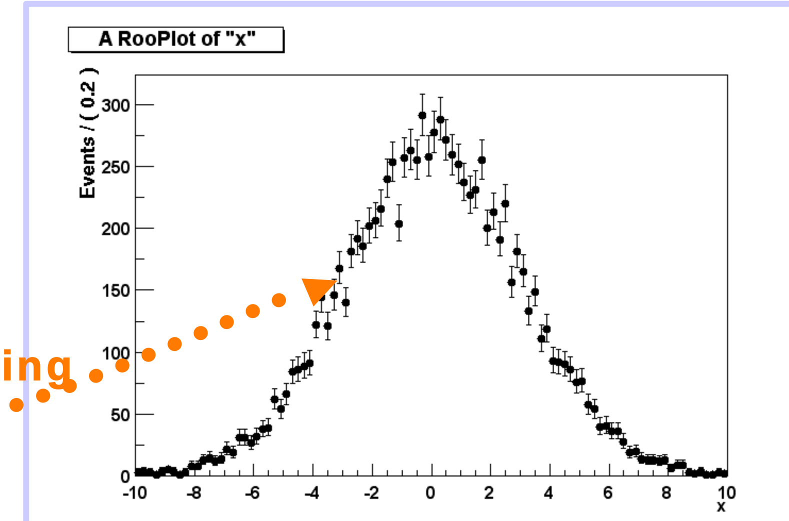
# RooFit Data Visualization Tutorial

Wouter Verkerke (UC Santa Barbara)
David Kirkby (UC Irvine)

# Data Visualization in RooFit - Overview

`RooDataSet(x,y,z)`

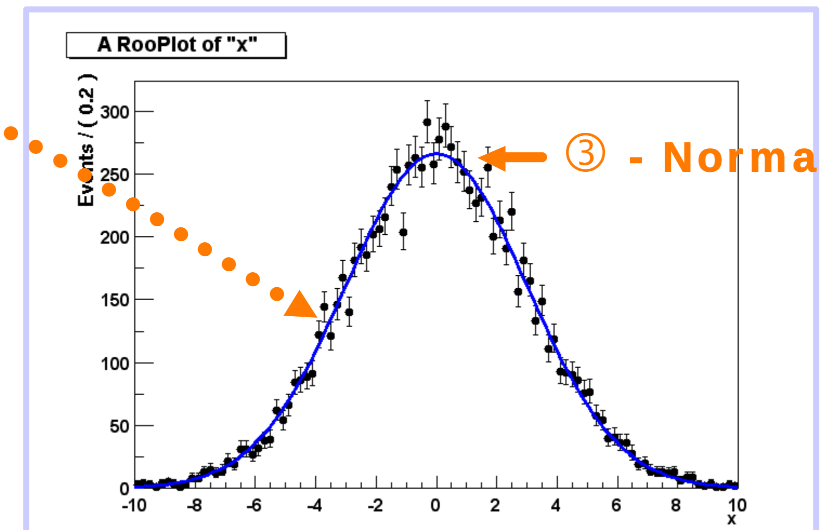
① - Binning





② - Projection (x,y,z) ® (x)

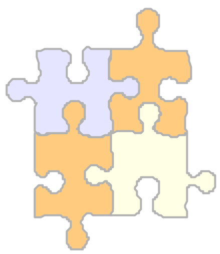
`RooAbsPdf(x,y,z)`





③ - Normalization

# 1-Dimensional plots



The basics

# 1-Dimensional plots – class **`RooPlot`**

*1-Dimensional plots are most frequently used*
*and have special support in RooFit via the RooPlot class:*

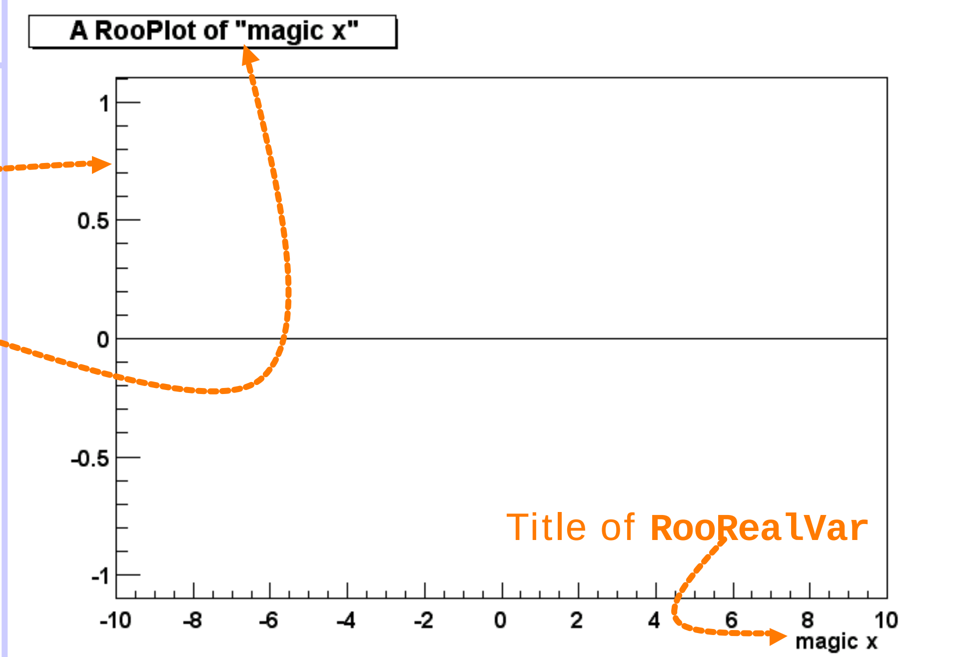
- Derives from TH1 for implementation of graphics, axes etc…
  - Container class for plotable objects: doesn't contain any data itself, TH1 member functions operating on data are non-functional
  - Persistable with ROOT I/O (including contents)
- Hold a list of objects to be plotted
  - Datasets (represented as histograms)
  - PDF projections (represented as curves)
  - Any other **`TObject`** that can be drawn (e.g. **`TArrow`**, **`TPaveText`**)
- Takes care of normalization PDF projection curves
  - Unit-normalized curve is automatically multiplied by number of events of last plotted dataset
- Facilitates automatic projection of PDFs onto plotted observable
  - RooPlot knows plotted observable and all observables of last plotted dataset.
  - PDF are automatically
    - Normalized over all known observables
    - Projected over all known observables except the plotted observables

# Using **RooPlot** – the basics

- A **RooPlot** class is easiest created from a **RooRealVar**

```
RooRealVar x("x","magic x",-10,10);
RooPlot* xframe = x.frame() ;
xframe->Draw() ;
```

```
// To change title
xframe->SetName("blah");
```



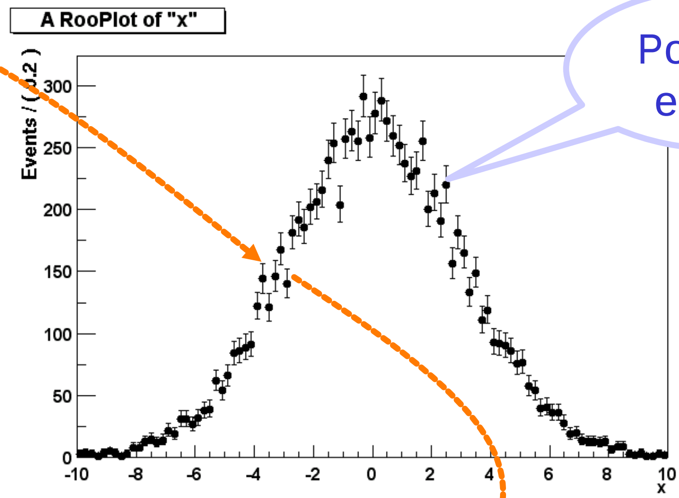

Title of **RooRealVar**

```
// Alternate frame() methods
// change default range, binning
RooPlot* xframe = x.frame(-5,5) ;
RooPlot* xframe = x.frame(40) ;
```

Default plot range = limits of **x**

# Using **RooPlot** – Adding datasets

```
// d contains x,y,z
RooDataSet *d ;
d->plotOn(frame) ;
frame->Draw() ;
```




Poisson errors

```
// list frame contents
frame->Print("v") ;
RooPlot::frame(088aa410): "A RooPlot of "magic x""
  Plotting RooRealVar::x: "magic x"
  Plot contains 1 object(s)
    (Options="P") RooHist::gData_plot__x: "Histogram of gData_plot__x"
```

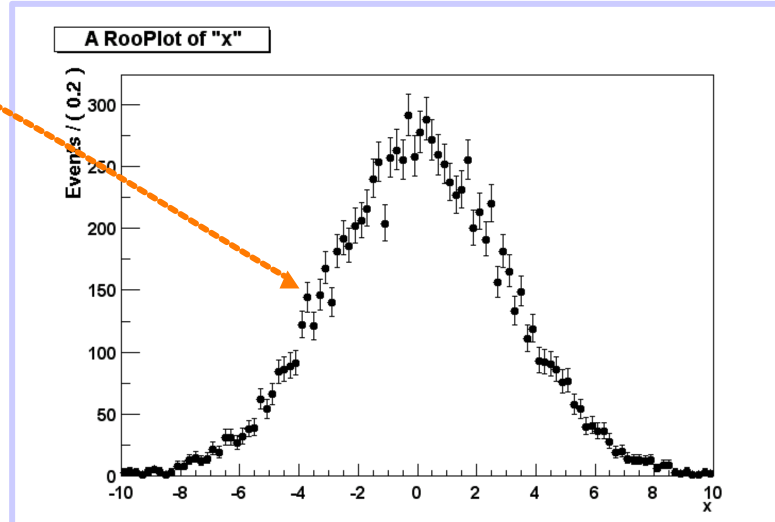
```
// Adding a dataset also updates
// the set of normalization observables
frame->getNormVars()->Print("1") ;
(x,y,z)
```

PDFs added after this dataset that depend on y,z will be normalized & projected over y,z

# Using **RooPlot** – Datasets and binning

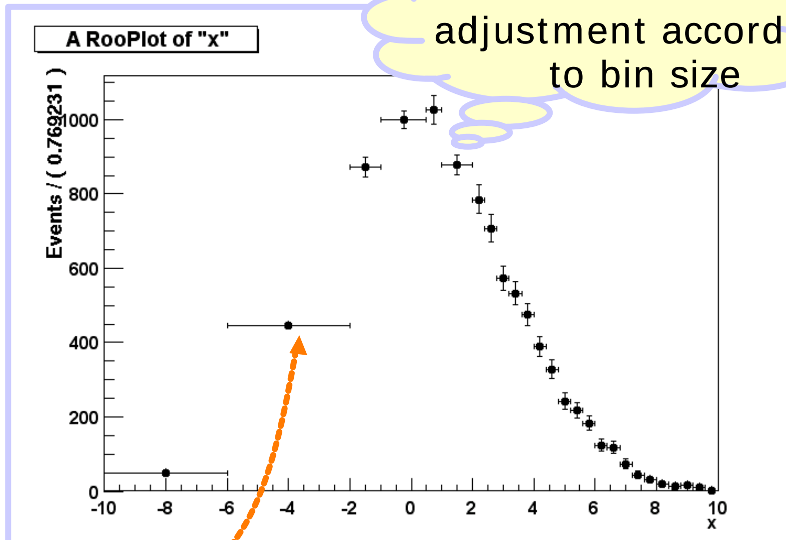
Default binning

```
RooDataSet *d ;
d->plotOn(frame) ;
frame->Draw() ;
```




Custom (non-uniform) binning

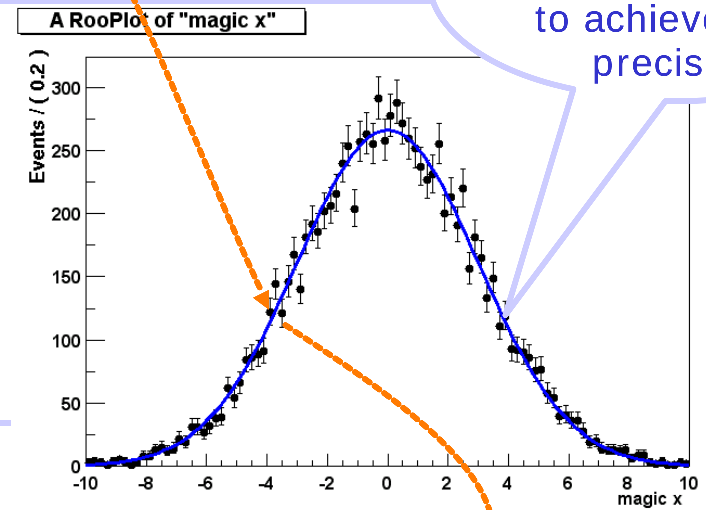
```
// Create binning object
RooBinning b(-10,10) ;

// Add single boundary
b.addBoundary(0.5) ;

// Add (x,-x) pairs of boundaries
b.addBoundaryPair(1) ;
b.addBoundaryPair(2) ;

// Add uniform patterns
b.addUniform(2,-10,-2) ;
b.addUniform(20,2,10) ;

RooDataSet *d ;
d->plotOn(frame),Binning(b)) ;
frame->Draw() ;
```



# Using **RooPlot** – Adding PDF projections

```
// pdf depends on x,y,z
RooAbsPdf* p
p->plotOn(frame) ;
RooAbsReal::plotOn(f) plot on x integrates over variables (y,z)
frame->Draw() ;
```

Automatic because RooPlot remembers dimensions of last plotted dataset (x,y,z)




Adaptive spacing of curve points to achieve 1 ‰ precision

```
// list frame contents
frame->Print("v") ;
RooPlot::frame(088aa410): "A RooPlot of "magic x""
  Plotting RooRealVar::x: "magic x"
  Plot contains 2 object(s)
    (Options="P") RooHist::gData_plot__x: "Histogram of gData_plot__x"
    (Options="L") RooCurve::curve_gProjected: "Projection of g"
```

# Using **RooPlot** – Adding PDF projections

Change draw option (e.g. 'Fill')

Modify the default normalization in various ways

(Re)define manually which of the PDF variables are observables

```
p->plotOn(xframe,DrawOption("F"))

// Correction w.r.t default normalization
p->plotOn(xframe,Norm(0.7)) ;

// Override number of events for PDF normalization
p->plotOn(xframe,Norm(RooAbsReal::NumEvent,10000)) ;

// Use expected number of events of extended PDF
p->plotOn(xframe,Norm(RooAbsReal::RelativeExpected,1.0))

// Raw scale factor (no bin width correction is applied)
p->plotOn(xframe,Norm(RooAbsReal::Raw,5.27));

// No variables are projected by default when PDF
// is plotted on an empty frame

// Enter custom definition of observables
xframe->updateNormVars(RooArgSet(x,y,z)) ;
p->plotOn(xframe) ;
```

# Other **RooPlot** features

- Change attributes of last added plot elements

```
frame->getAttLine()->setLineColor(kRed)
frame->getAttMarker()->setMarkerType(22)
```

- Change the plotting order of contained objects

```
frame->drawAfter(“objectName1”,”objectName2”) ;
```

Use
**frame->Print(“v”)**
to see list of object names

- Add non-RooFit objects

```
TArrow *a = new TArrow(0,0,5,7) ;
frame->addObject(a) ;
```

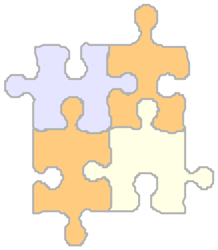
- Merge contents from another RooPlot

```
frame->merge(frame2) ;
```

- Curve/histogram $\chi^2$ calculation

```
frame->chiSquare() ;
frame->chiSquare(“curveName”,”histName”) ;
```

# Projecting out dimensions



Projection via Integration

Projecting discrete vs real observables

Projection via data averaging

Mixing projection methods

# Projecting out hidden dimensions - Integration

- PDF is always **normalized** over **all** observables
  - Normalization set ***n*** = variables PDF and dataset have in common
- PDF is **projected** over all **unplotted** observables
  - The plot variable set ***x*** = the plotted dimensions of the PDF (for a 1-D RooPlot this is always 1 variable)
  - The projection set ***p*** is ***n – x***
  - The projected PDF function is

$$P_f(\vec{x}) = \frac{\int f(\vec{x},\vec{p})\,d\vec{p}}{\int f(\vec{x},\vec{p})\,d\vec{x}\,d\vec{p}}$$

*Projected observables*

*Plotted observables*

# Projecting out hidden dimensions

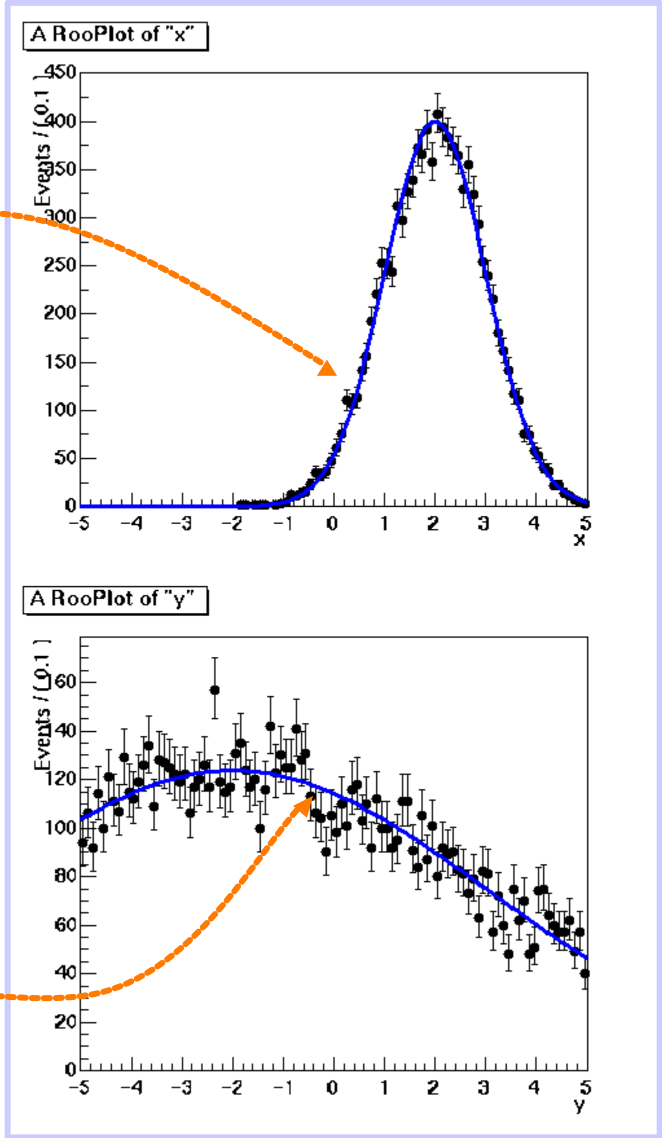
- Example in 2 dimensions
  - 2-dim dataset D(x,y)
  - 2-dim PDF P(x,y)=gauss(x)*gauss(y)

- 1-dim plot versus x

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

- 1-dim plot versus y

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



# RooProdPdf automatic optimization

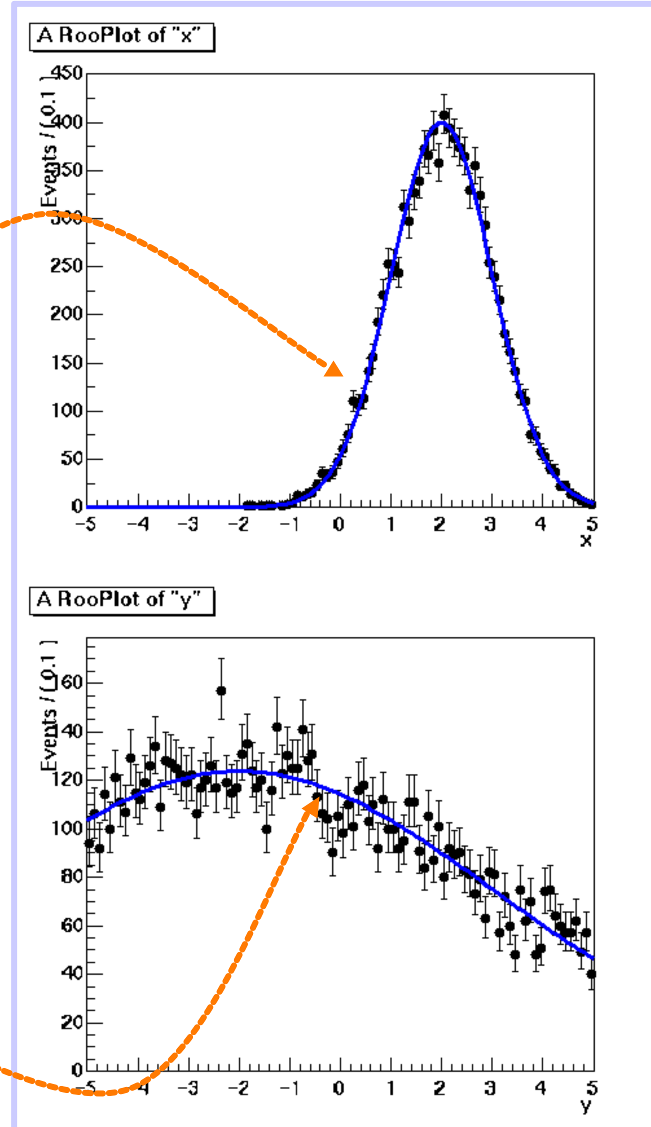
- Example in 2 dimensions
  - 2-dim dataset D(x,y)
  - 2-dim PDF P(x,y)=gaus(x)*gauss(y)

- 1-dim plot versus x

$$P_p(x) = \frac{\int g(x)g(y)dy}{\int g(x)g(y)dxdy} = \frac{g(x)\cancel{\int g(y)dy}}{\int g(x)dx\cancel{\int g(y)dy}} = \frac{g(x)}{\int g(x)dx}$$

- 1-dim plot versus y

$$P_p(y) = \frac{\int g(x)g(y)dx}{\int g(x)g(y)dxdy} = \frac{\cancel{\int g(x)dx}\cdot g(y)}{\cancel{\int g(x)dx}\int g(y)dy} = \frac{g(y)}{\int g(y)dy}$$



# Projecting out discrete observables

- Works the same way as for real observables
  - Projected discrete dimension is summed over all its states
- Example: B-Decay with mixing
  - dataset(dt,mixState) & PDF(dt,mixState)
  - 1-dim plot versus dt:

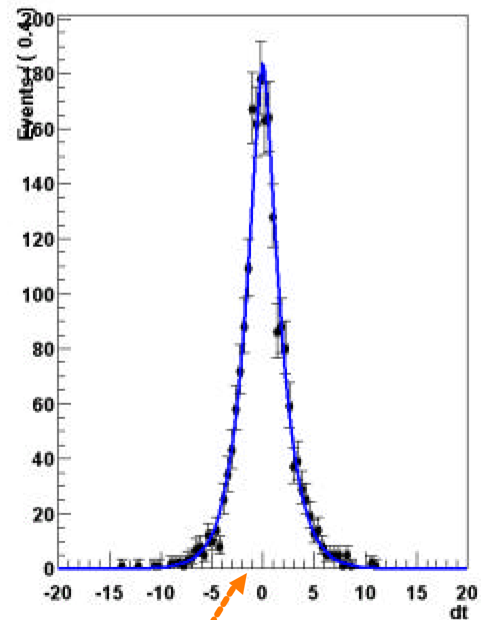
$$P_p(t) = \frac{\int p(t,M)dM}{\int p(t,M)dtdM}$$

*Use summation instead of integration for discrete states*

$$= \frac{\sum_{mS} p(t,M)}{\sum_{mS}\int p(t,M)dt}$$

*Expand summation*

$$= \frac{p_{mixed}(t) + p_{unmixed}(t)}{\int p_{mixed}(t)dt + \int p_{unmixed}(t)dt}$$




**Projection works universally for real and discrete observables**

# Projecting out observables – Data averaging

- An alternative method to project out observables is to construct a data weighted average function:

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*

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

- The summed variable (y) is treated as a parameter
  - PDF is *not* normalized over y in above example
- Can be used to cancel the effect of a disagreement between data and PDF in a projected observable
  - Example: per-event errors:
    PDF is usually flat in dtErr, distribution in data is usually peaked.

# Selecting data averaging as the projection method

```
// PDF and data defined elsewhere,
// observables:dt,dtErr,mixState
RooAbsData* data ;
RooAbsPdf* bmixPdf ;

// Create frame and plot data as usual
RooPlot* dtframe = dt.frame() ;
data->plotOn(dtframe) ;

// Plot bmixPdf, projecting dterr  with data
bmixPdf->plotOn(dtframe,ProjWData(dterr,projData)) ;
RooAbsReal::plotOn(bmixPdf) plot on dt integrates
over variables (mixState)
RooAbsReal::plotOn(bmixPdf) plot on dt averages
using data variables (dtErr)
```

The **ProjWData()** modifier overrides the projection method of selected variables:

Observable **dterr** will be averaged over the values in dataset **projData**

**ProjWData** only controls *how* observables are projected. It does *not* override *which* observables are projected

# Example: integration vs. data averaging on per-event errors

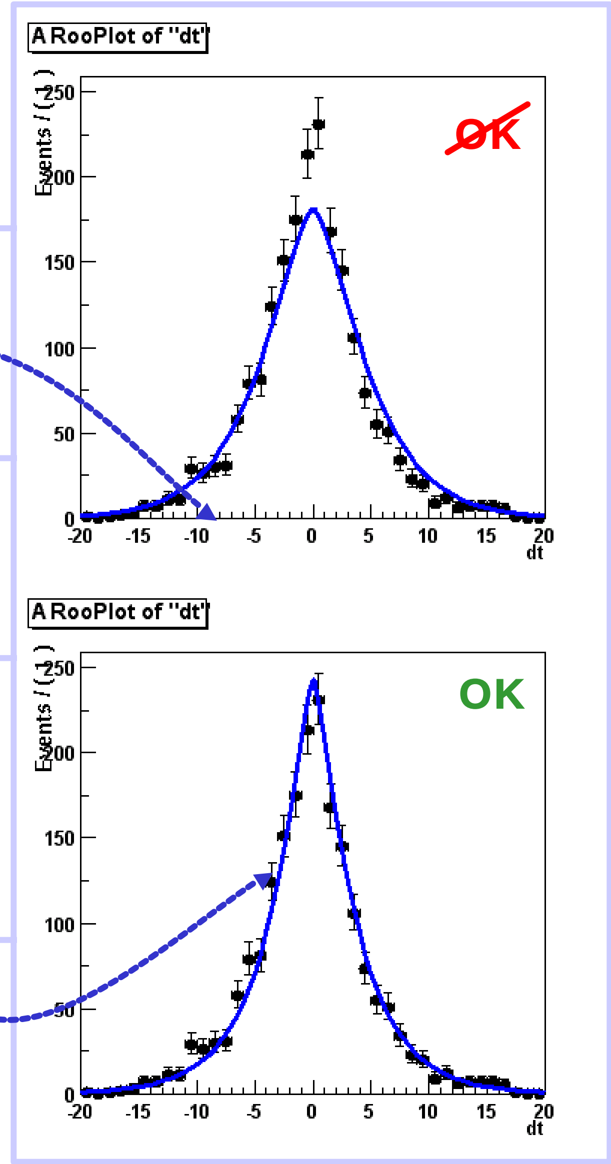
*Special property of per-event errors:*
*Distribution in data and PDF do not agree*

Integrating out per-event errors

```
RooPlot* dtFrame = dt.frame() ;
data->plotOn(dtFrame) ;
bmixPdf.plotOn(dtFrame) ;
dtFrame->Draw() ;
```

Projecting per-event errors with data

```
RooPlot* dtFrame = dt.frame() ;
data->plotOn(dtFrame) ;
bmixPdf.plotOn(dtFrame,
               ProjWData(dterr,projData));
dtFrame->Draw() ;
```



# Selecting data averaging as the projection method

- Projection via data averaging may be applied to *any* observable
  - Also discrete valued observables
- Choosing data averaging instead of integration changes the meaning of the projected function
  - The theoretical model / experimental data distinction is blurred: the plotted curve takes part of its behavior from the dataset
  - Often applied to non-physics observables (e.g. per-event errors)
    - Shape of per-event error distribution irrelevant to physics and may be hard to model correctly in a PDF
  - Can also to applied to well-modeled physics observable:

    Example: plot $\delta$t distribution of B-mixing PDF while

    - projecting the mix state via integration –
      True model/experimental data comparison
    - projecting the mix state with data averaging -
      Compare only dt shape aspect of model with data

    Any effects that purely arise from PDF/data discrepancy in $B^0/\overline{B^0}$ counter are taken out

# Data average projection - Performance

- Data-averaged projections can be computationally expensive
  - Effectively the sum of N curves is plotted (N = #evts in projection dataset)
- Projections with large datasets can be accelerated enormously by using binned projection data sets
  - Works the same way, just provide a binned dataset

```
RooPlot* dtFrame = dt.frame() ;
data->plotOn(dtFrame) ;
dterr.setFitBins(50) ;
RooDataHist projData("projData","projData",dtErr,data) ;
bmixPdf.plotOn(dtFrame,ProjWData(projData));
```

  - Minor loss of precision may occur, but with sufficient data and a prudent binning net loss may be less than plotting precision
  - Example: unbinned projection with 20K events: **51.2 sec**
    binned projection with 100 bins: **0.2 sec**
- Also possible when projecting >1 dimensions, and/or discrete dimensions
  - Simply create a multi-dimensional binned dataset

# Integration vs. data averaging - Summary

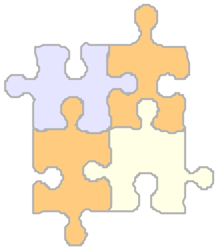
- Default projection method for all observables is integration

- To override integration method with data averaging method, provide a projection dataset with observables to be averaged
  - Projection dataset only controls method of projection, not which variables are projection
  - Projection dataset may contain both *discrete and real observables*
  - *Projection dataset may be binned (speed vs accuracy tradeoff)*

- *Any* projected PDF observable *may be averaged with data* instead of integrated

- Final projection may be *combination* of data-averaging & integration

# Slicing & Cutting



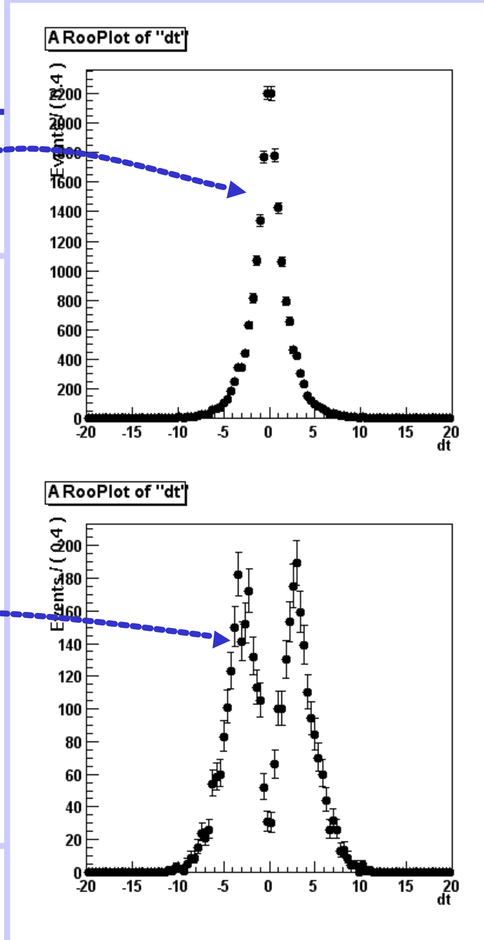
Plotting a slice in real & discrete dimensions

Understanding normalization in slicing

# Plotting a slice of a dataset

- Use the optional cut string expression

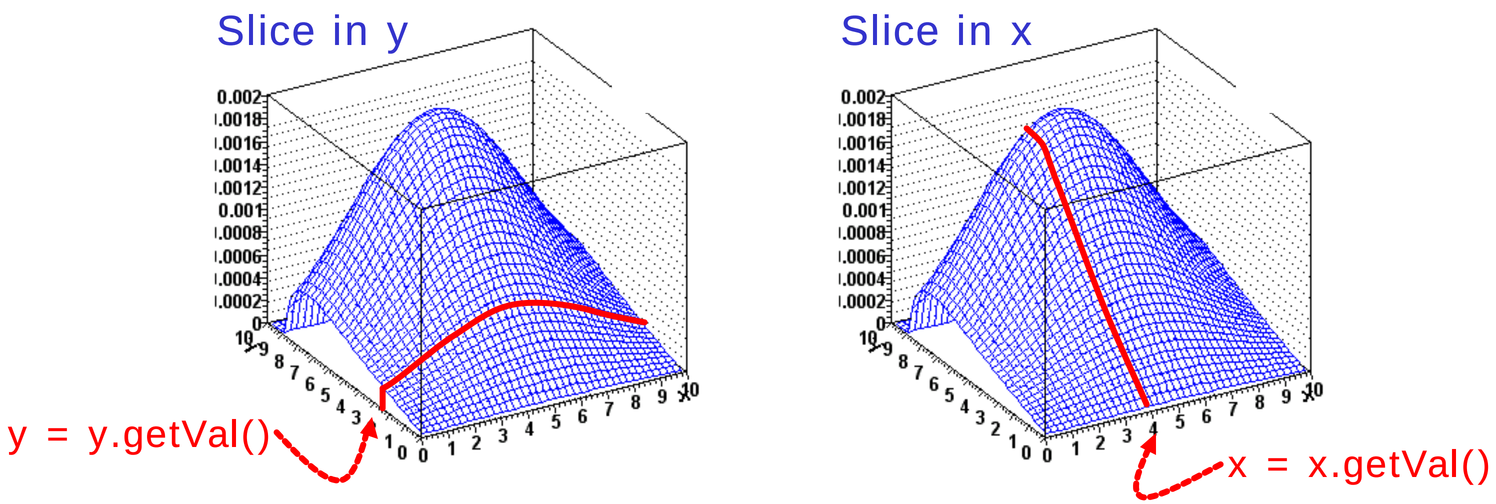
```
// Mixing dataset defines dt,mixState
RooDataSet* data ;

// 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
  – The target **RooPlot** will retain the *total number of events* for future *PDF normalizations* (*not* the number of events in the slice)
    - More about this later

# Plotting a slice of a PDF – plotSliceOn()

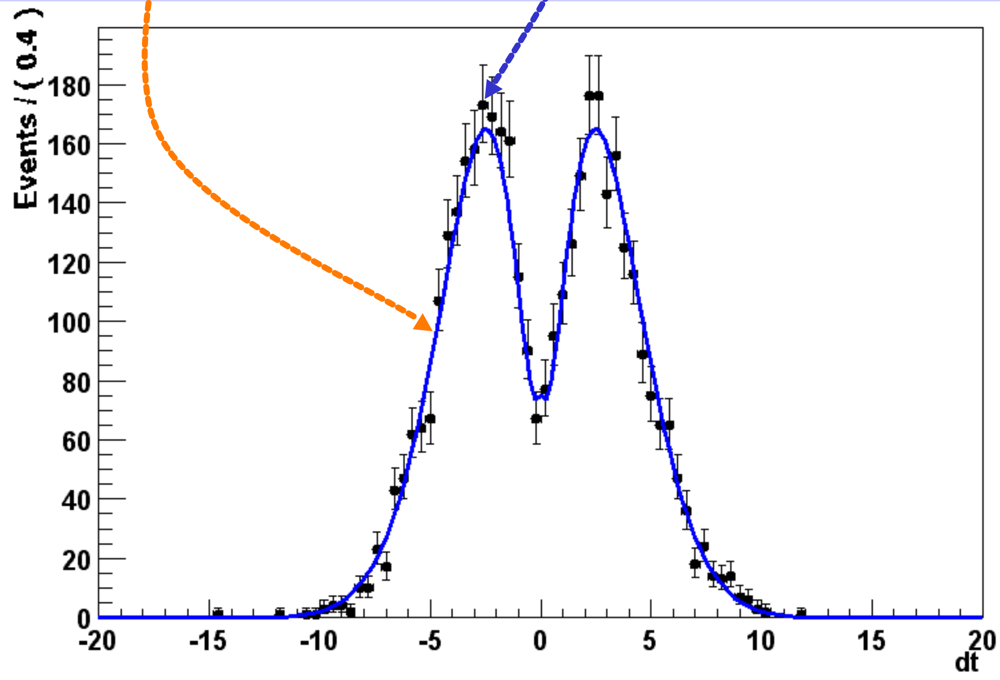
- To plot a (projection of a) slice of a PDF use **`Slice()`**
  - **`RooAbsReal::plotOn(frame,Slice(sliceSet),…)`** overrides default set of observables to project out
  - Argument **`sliceSet`** specifies the set of observables that should *not* be *projected out*
  - Position of slice is determined by the current value of slice observable




  - Slicing can be done in real and discrete dimensions
  - Slice set can have an any number of dimensions

# Example: plotting mixed-only slice of data and PDF

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

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



## Understanding the normalization for PDF/data slices

**$f_{mixed}$**

A PDF plotted with plotSliceOn() is normalized to *all* observables, *including the sliced observables*, therefore

$$\int P_f(t,M)dt \neq 1 = \int dt \left( \frac{p(t,M)}{\sum_M \int p(t,M)dt} \right)$$

$$= \frac{\int p(t,M)dt}{\sum_M \int p(t,M)dt} = f_M$$

**The integral of a PDF slice projection is not 1!**

*Integral of PDF projection = Fraction of mixed events predicted by PDF*

**$N_{total}$**

The **`RooAbsData::plotOn()`** function with cut gives the *full (uncut) number of events* to the **`RooPlot`** so that the final normalization comes out as

$$C_f(\vec{x}) = P_f(\vec{x}) \cdot N^{\mathbf{total}}_{\mathbf{data}} \cdot f^{PDF}_{mixed} \cdot V_{bin}$$

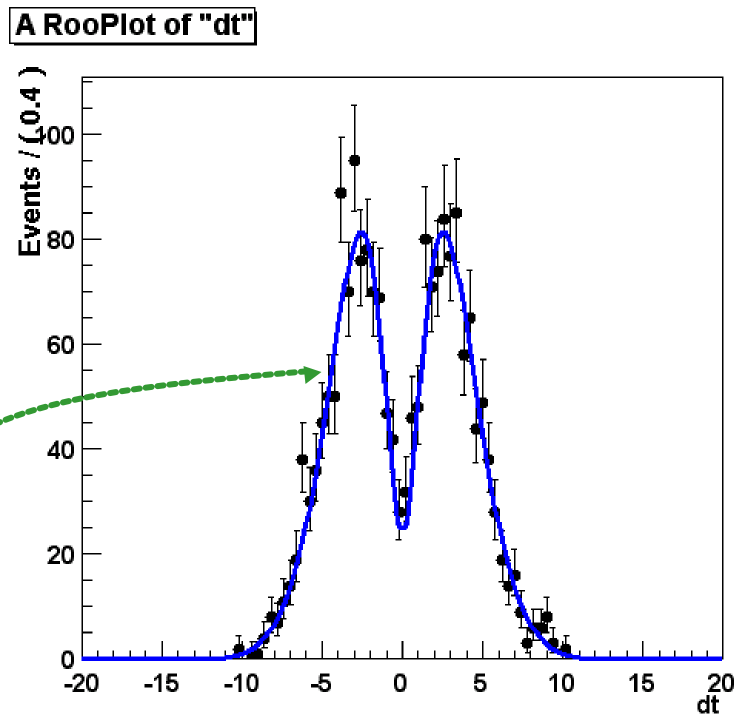
$$= P_f(\vec{x}) \cdot N^{\mathbf{slice}}_{\mathbf{PDF}} \cdot V_{bin}$$

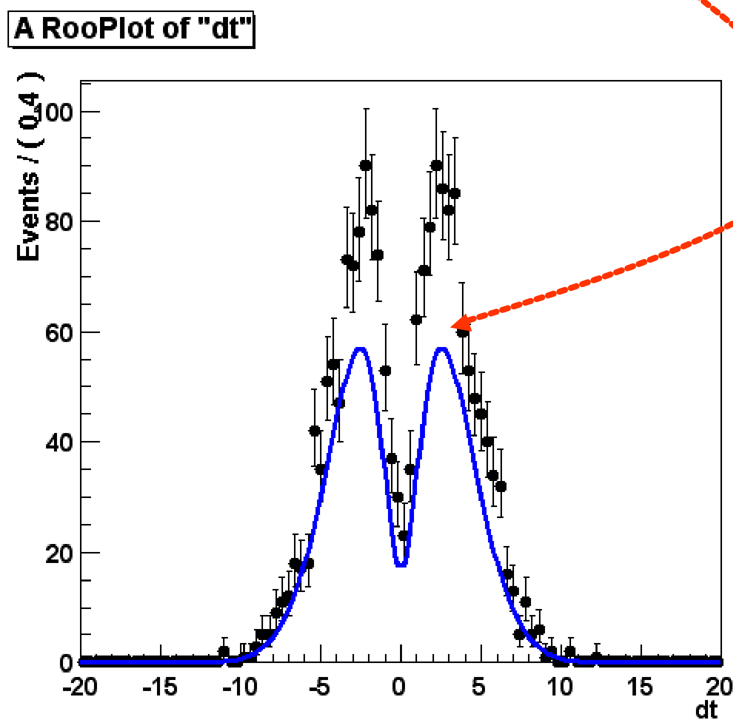
$$\neq N^{\mathbf{slice}}_{\mathbf{data}}$$

**The normalization of the PDF slice curve *reflects* *the PDFs prediction of the slice fraction***

# Understanding the normalization for PDF/data slices

Data has large fraction of mixed events than PDF predicts



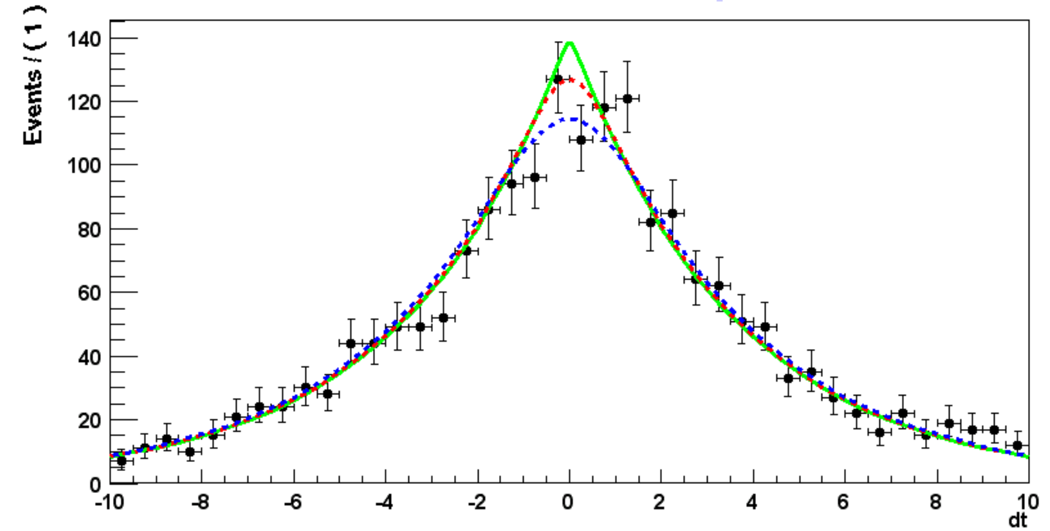

PDF and data agree on fraction of mixed events

# Slices in a real-valued observable

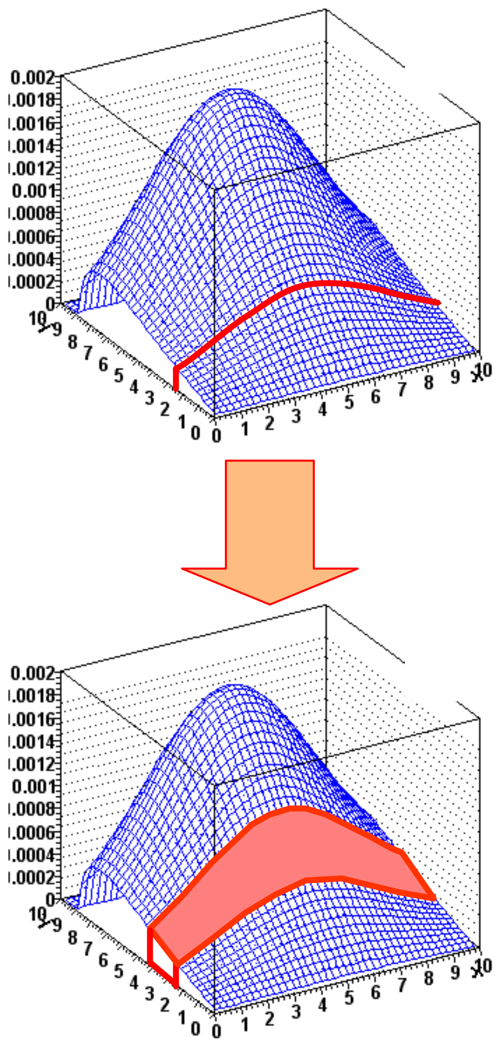
- Real-valued slices have *no width*
  - Usually not that useful (equivalent slices in data are usually empty)
  - *Finite width slices* can be made with *different technique* (see later)

- Example plot: effect of per-event error

```
RooPlot* dtframe = dt.frame() ;
data->plotOn(dtframe) ; // not a slice
dtErr=0.1 ; bmix.plotOn(dtframe,Slice(dtErr)) ;
dtErr=0.5 ; bmix.plotOn(dtframe,Slice(dtErr)) ;
dtErr=1.0 ; bmix.plotOn(dtframe,Slice(dtErr)) ;

dtframe->Draw() ;
```



# Plotting slices with finite width - Introduction



- Problem: analytic calculation of the projection of a ‘band’ of a PDF often very hard or impossible
- Solution: Numeric solution via ToyMC approach
  - Construct finite width slice as weighted average of no-width slices:

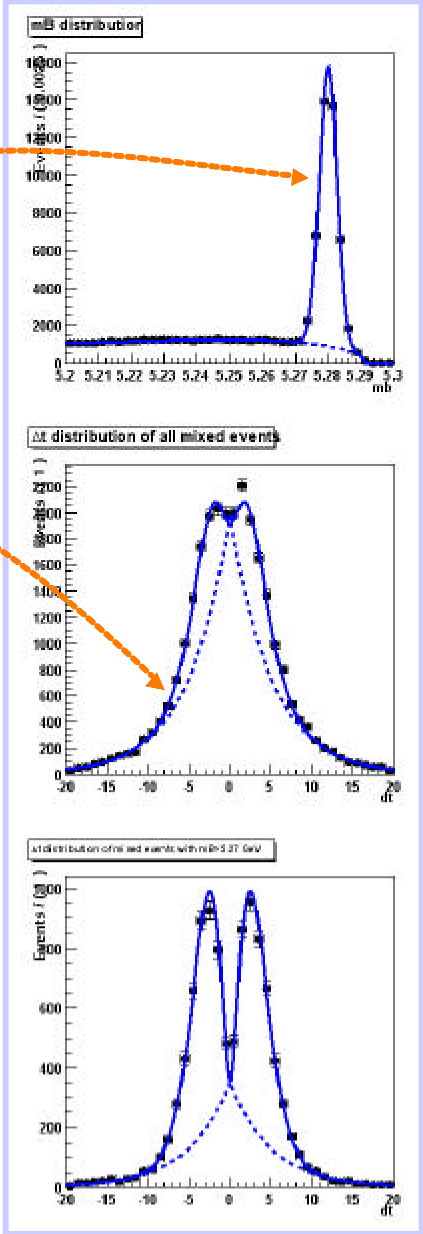
1) Generate a sufficiently large ToyMC sample to be plotted
2) Reduce the ToyMC data to the band to be plotted
3) Plot the PDF the usual way, projecting out *all* unplotted observables via data averaging. Use the reduce ToyMC set as weighting dataset

# Plotting slices with finite width - Example

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”)) ;
mixState=“mixed” ;
model.plotOn(dtframe,Slice(mixState)) ;
```



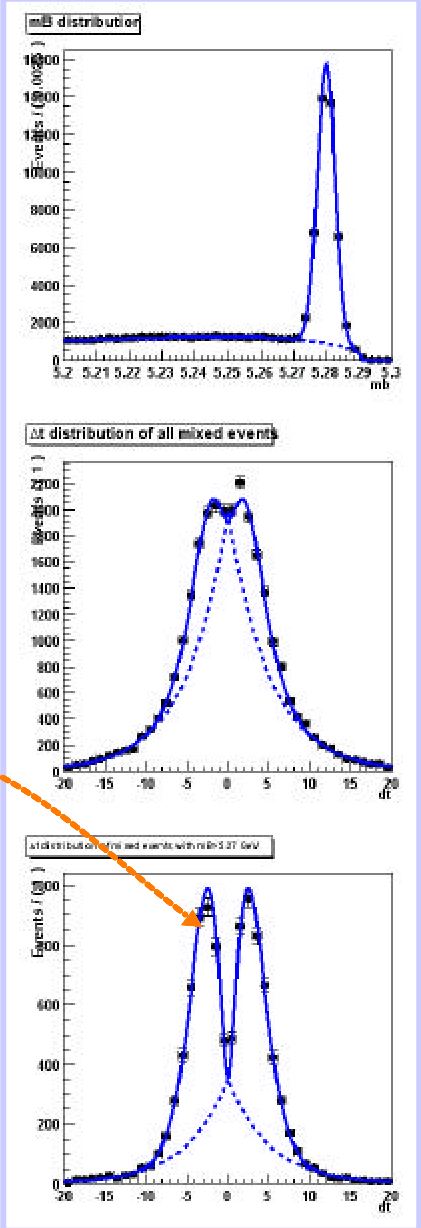
# Plotting slices with finite width - Example

Example setup:

**`Argus(mB)*Decay(dt) +`** (background)
**`Gauss(mB)*BMixDecay(dt)`** (signal)

0. Reduce dataset before plotting
1. Generate a sufficiently large ToyMC sample to be plotted
2. Reduce the toyMC data to the band to be plotted
3. Plot the PDF the usual way, projecting out all unplotted observables via data averaging.

```
RooDataSet* mbSliceData =
   data->reduce("mb>5.27") ;

mbSliceData->plotOn(dtframe2,
   "mixState==mixState::mixed“)

RooDataSet *toyMC = model.generate(
   RooArgSet(dt,mixState,tagFlav,mB),
   80000);

RooDataSet* mbSliceToyMC =
   toyMC->reduce(“mb>5.27”);

model.plotOn(dtframe2,Slice(mixState),
                      ProjWData(mb,mbSliceToyMC))
```



# Plotting non-rectangular PDF regions

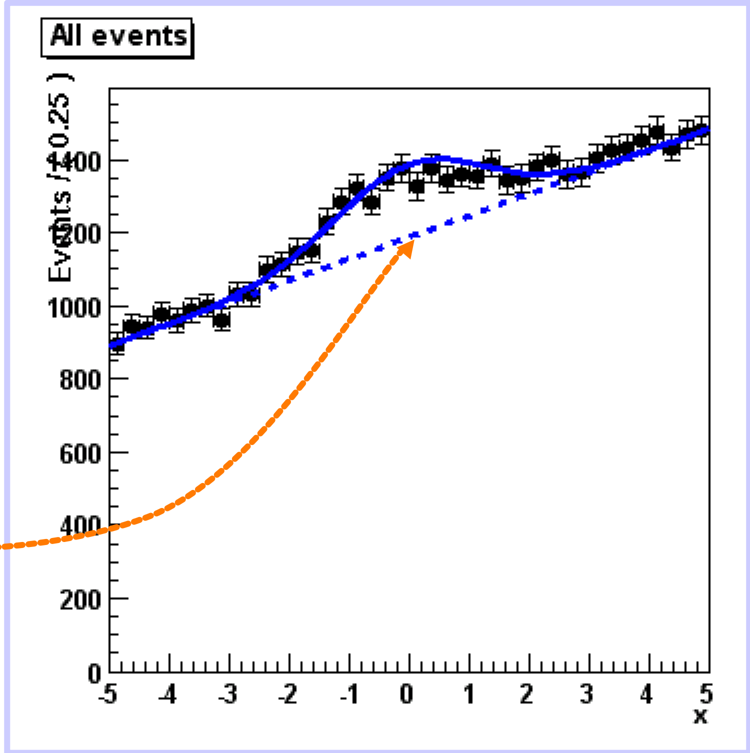
- The ToyMC projection technique makes no assumptions on the shape of the selected region
  - Regions of arbitrary size, shape and dimension can be selected

- Example: Likelihood 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 in the remaining N-1 dimensions

# Plotting data & PDF with a likelihood cut

- Simple example
  - 3 observables (x,y,z)
  - Signal shape: gauss(x)·gauss(y)·gauss(z)
  - Background shape: (1+a·x)(1+b·y)(1+c·z)
  - Plot distribution in x with cut on likelihood in (y,z)

```
// Plot x distribution of all events
RooPlot* xframe1 = x.frame(40) ;
data->plotOn(xframe1) ;
sum.plotOn(xframe1) ;
```




Integrated projection of data/PDF on X doesn't reflect signal/background discrimination power of PDF in y,z

# Plotting data & PDF with a likelihood cut

```
RooDataSet* data = sum.generate(RooArgSet(x,y,z),50000) ;

RooAbsReal* pdfProj = sum.createProjection(RooArgSet(y,z),x) ;

RooFormulaVar nllFunc("nll","-log(    lihood)","-log(@0)",*pdfProj) ;
RooRealVar* nll = data->ad        (nllFunc) ;

                                        ) ;

                                        ArgSet(x,y,z),"nll<5.2") ;

                                        at::Relative,sliceData) ;
```

The **`createProjection()`** method create a projection of sum over x, with (y,z) as observables:

$$P_f(y,z,\vec{p}) = \frac{\int f(x,y,z,\vec{p})dx}{\int f(x,y,z,\vec{p})dxdydz}$$

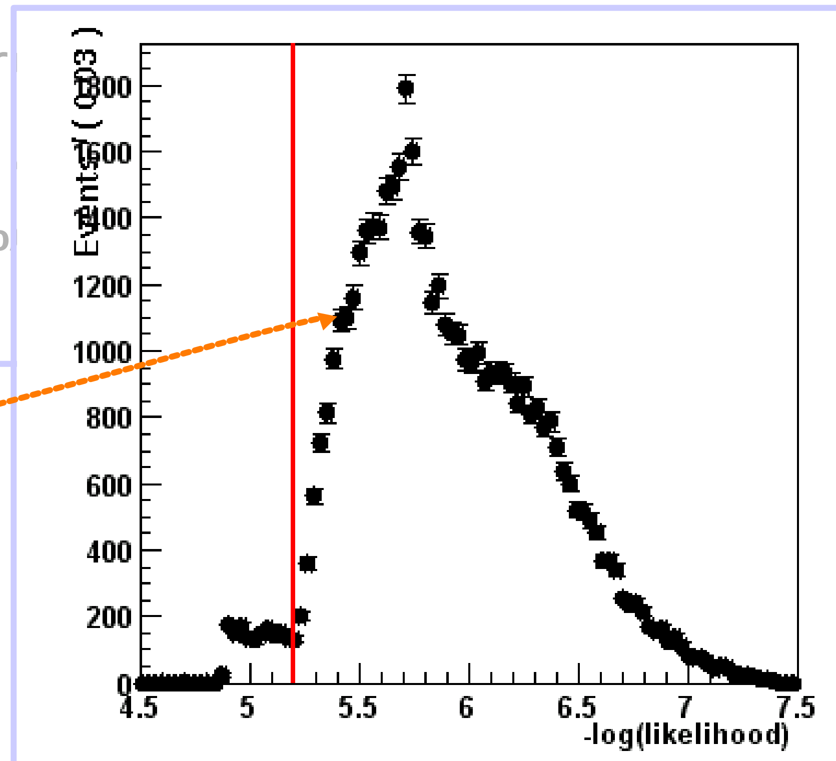
Automatic optimization:
If f factorizes as g(x)*h(y,z):

$$\begin{aligned} P_f(y,z,\vec{p}) &= \frac{\int \cancel{g(x,\vec{p}_g)} h(y,z,\vec{p}_h)dx}{\int \cancel{g(x,\vec{p}_g)} h(y,z,\vec{p}_h)dxdydz} \\ &= \frac{h(y,z,\vec{p}_h)}{\int h(y,z,\vec{p}_h)dydz} \end{aligned}$$

# Plotting data & PDF with a likelihood cut

Construct per-event likelihood and add as pre-calculated column to the dataset

```
Set(x,y,z),50000) ;

tion(RooArgSet(y,z),x) ;

RooFormulaVar nllFunc("nll","-log(likelihood)","-log(@0)",*pdfProj) ;
RooRealVar* nll = data->addColumn(nllFunc) ;

RooPlot* pframe = nll->frame(4.5,7.5,100) ;
data->plotOn(pframe) ;

RooDataSet* sliceData = data->r

RooPlot* xframe2 = x.frame(40) ;
sliceData->plotOn(xframe2) ;
sum.plotOn(xframe2,"L",1.0, Roo
```



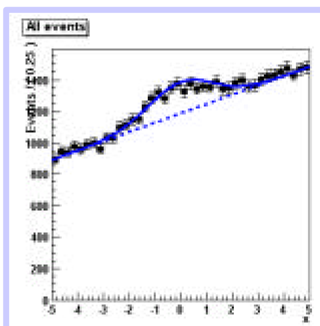

Plot per-event likelihood distribution to tune cut

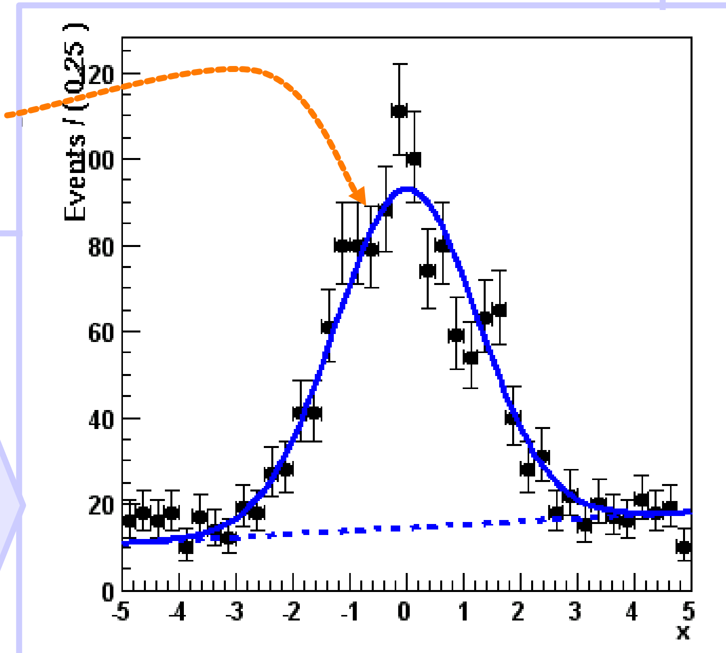
# Plotting data & PDF with a likelihood cut

```
RooDataSet* data = sum.generate(RooArgSet(x,y,z),50000) ;

RooAbsReal* pdfProj = sum.createProjection(RooArgSet(y,z),x) ;

RooFormulaVar nllFunc("nll","-log(likelihood)","-log(@0)",*pdfProj) ;
RooRealVar* nll = data->addColumn(nl

RooPlot* pframe = nll->frame(4.5,7
data->plotOn(pframe) ;

RooDataSet* sliceData = data->reduce(RooArgSet(x,y,z),"nll<5.2") ;

RooPlot* xframe2 = x.frame(40) ;
sliceData->plotOn(xframe2) ;
sum.plotOn(xframe2,ProjWData(sliceData))
```

Reduce ToyMC projection dataset with cut on per-event likelihood

Plot PDF with selected ToyMC events



# Summary of slice plotting

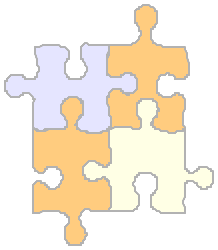
- To project category slices (or no-width real slices) use

```
RooAbsData::plotOn(frame,Cut("slice_cut_expr"),…)
RooAbsPdf::plotOn(frame,Slice(sliceSet),…) ;
```

  - Normalization of PDF slice projection will reflect the PDF information on $f_{slice}$, not the $f_{slice}$ of the data

- To plot bands, likelihood slices or arbitrarily shaped regions
  - Use ToyMC projection technique
  - If the number of projected observables is low (<=2) binning the ToyMC projection dataset can speed up the plotting process.
  - Can be used in combination with **`Slice()`** to slice in observables no participating in the region cut

# Component plotting

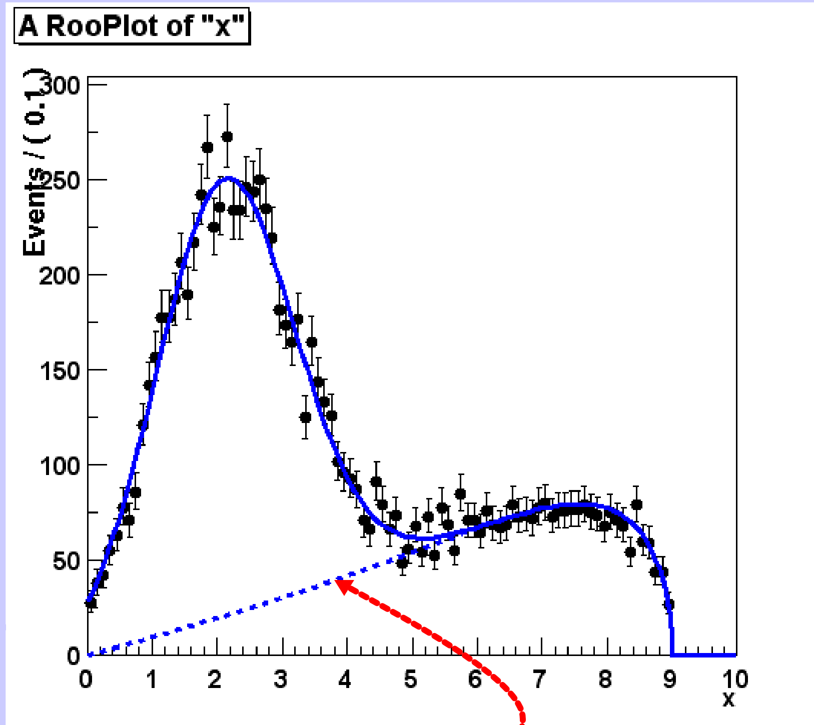


Selecting components to be plotted

Slices vs components

# Component plotting - Introduction

- A PDF that is explicitly constructed as a sum of components via **RooAddPdf** can plot its components separately
  - Use Method **Components()**




- Example:
  Argus + Gaussian PDF

```
// Plot data and full PDF first

// Now plot only argus component
sum->plotOn(xframe,
             Components(argus), LineStyle(kDashed)) ;
```

# Component plotting – Selecting components

There are various ways to select single or multiple components to plot

```
// Single component selection
pdf->plotOn(frame,Components(argus)) ;
pdf->plotOn(frame,Components("gauss")) ;

// Multiple component selection
pdf->plotOn(frame,Components(RooArgSet(pdfA,pdfB))) ;
pdf->plotOn(frame,Components("pdfA,pdfB")) ;

// Wild card expression allowed
pdf->plotOn(frame,Components("bkgA*,bkgB*")) ;
```

Wildcard option particularly useful for
simultaneous PDFs built by RooSimPdfBuilder.

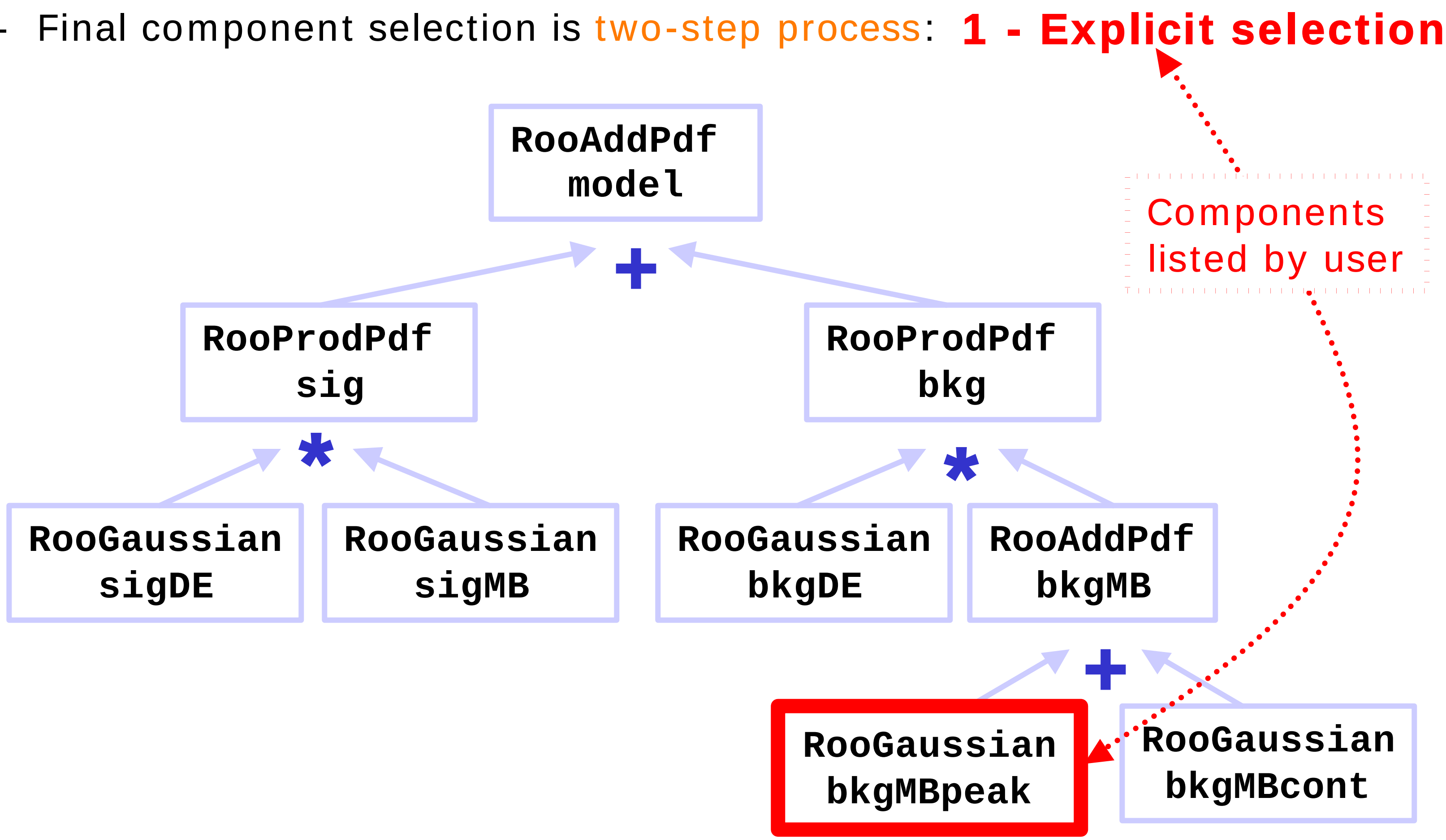
Example: simultaneous Gauss+Argus fit over 4 tagging categories

```
// plot data and full PDF
data->plotOn(frame) ;
pdf->plotOn(frame) ;
pdf->plotOn(frame,Components("Argus_*")) ;
```

Plots sum of all background PDFs
Syntax independent of number and
names of index category states

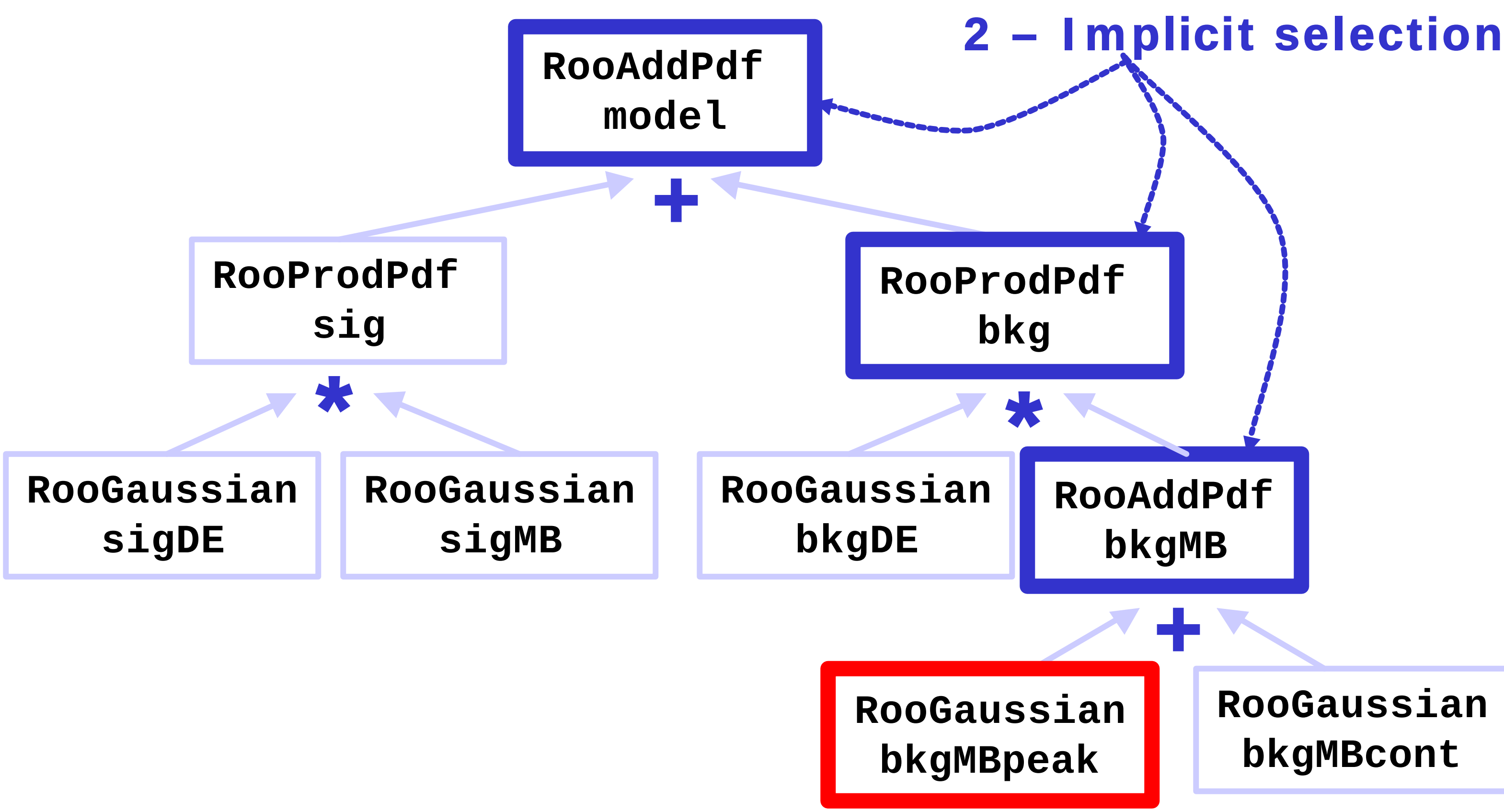
# Component plotting – Multi layer selection

- Method **plotCompOn()** can be called on any PDF, and also works for nested **RooAddPdf** structures
  - Selection mechanism works recursively
  - Final component selection is two-step process:



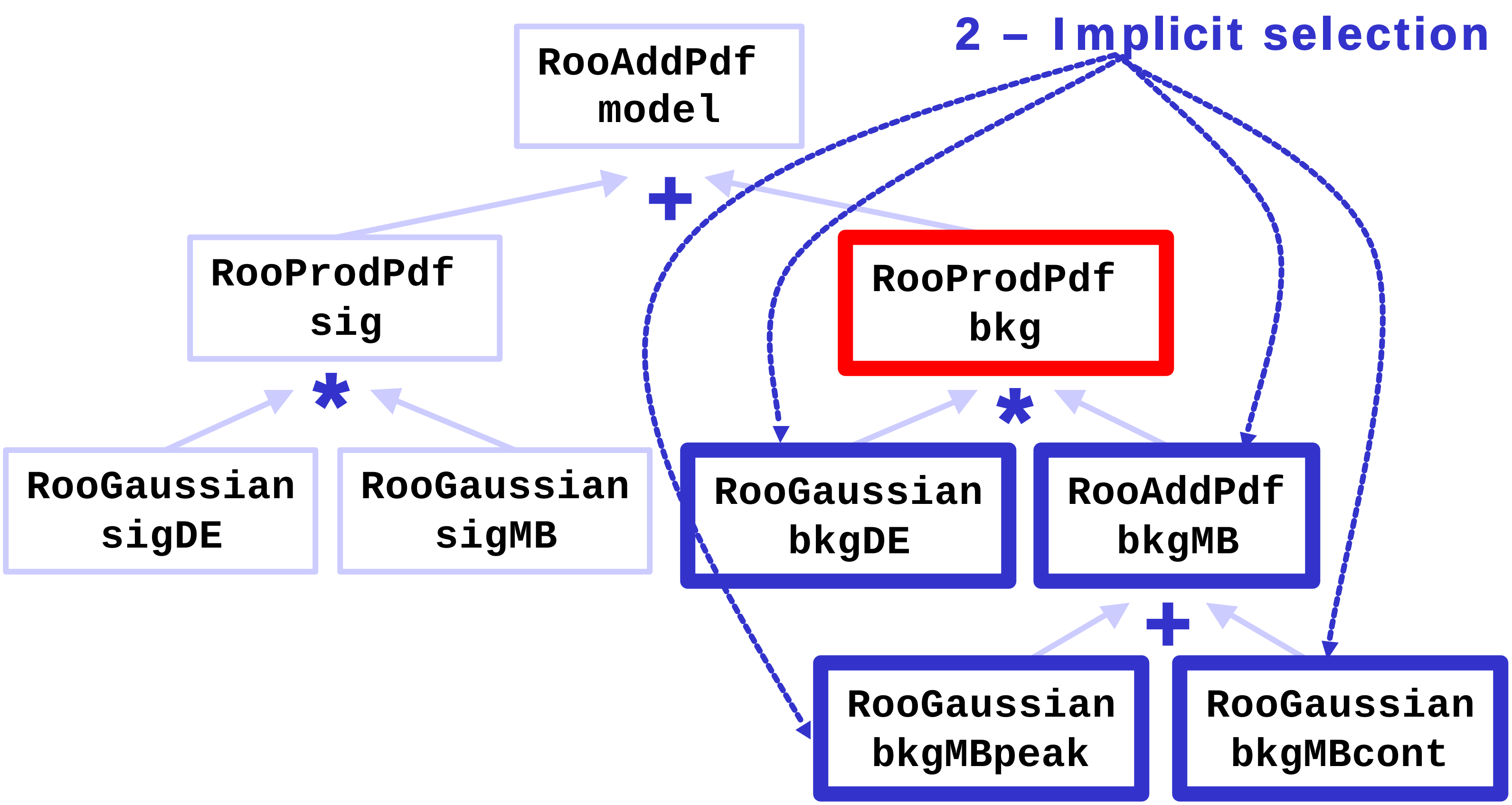
# Component plotting – Implicit selection

- All nodes in the path between each selected node and the top-level node is implicitly selected



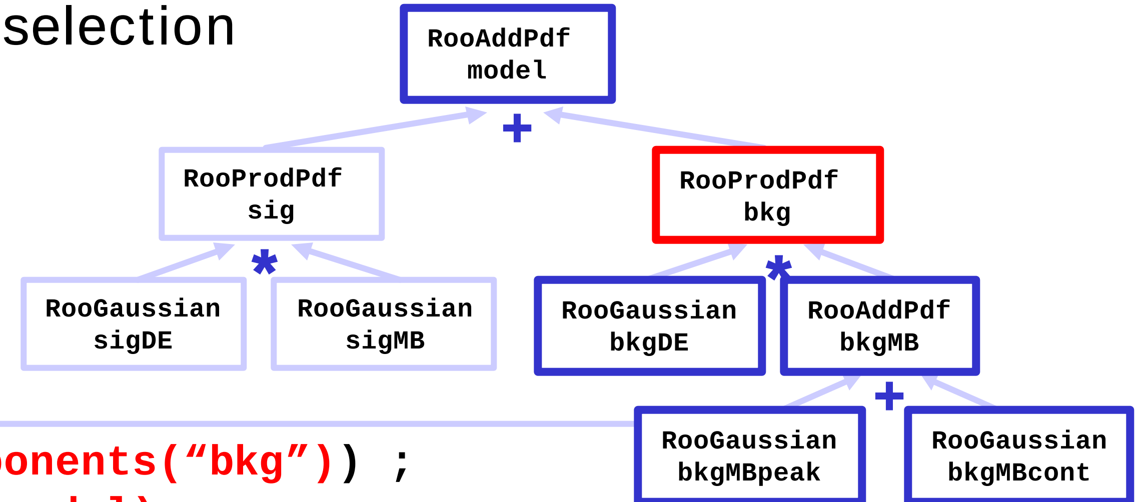
# Component plotting – Implicit selection

- All nodes below each selected node is implicitly selected



# Component plotting – Code example

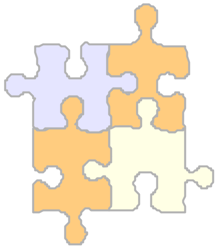
- Component selection gives feedback on explicit/implicit selection




```
pdf->plotOn(frame,Components("bkg")) ;
RooAbsPdf::plotCompOn(model)
   directly selected PDF components: (bkg)
RooAbsPdf::plotCompOn(model) indirectly selected
   PDF components: (bkgMBPeak,bkgMBCont,bkg,model)
```

- Component selection in a PDF slice projection
  - Use `plotOn(frame, Components("compList"),Slice(sliceSet),…)`
  - No special issues, just combine features of **`Slice()`** and **`Components()`**

# RooSimultaneous



Projecting and slicing RooSimultaneous PDFs

## Plotting **RooSimultaneous** PDFs

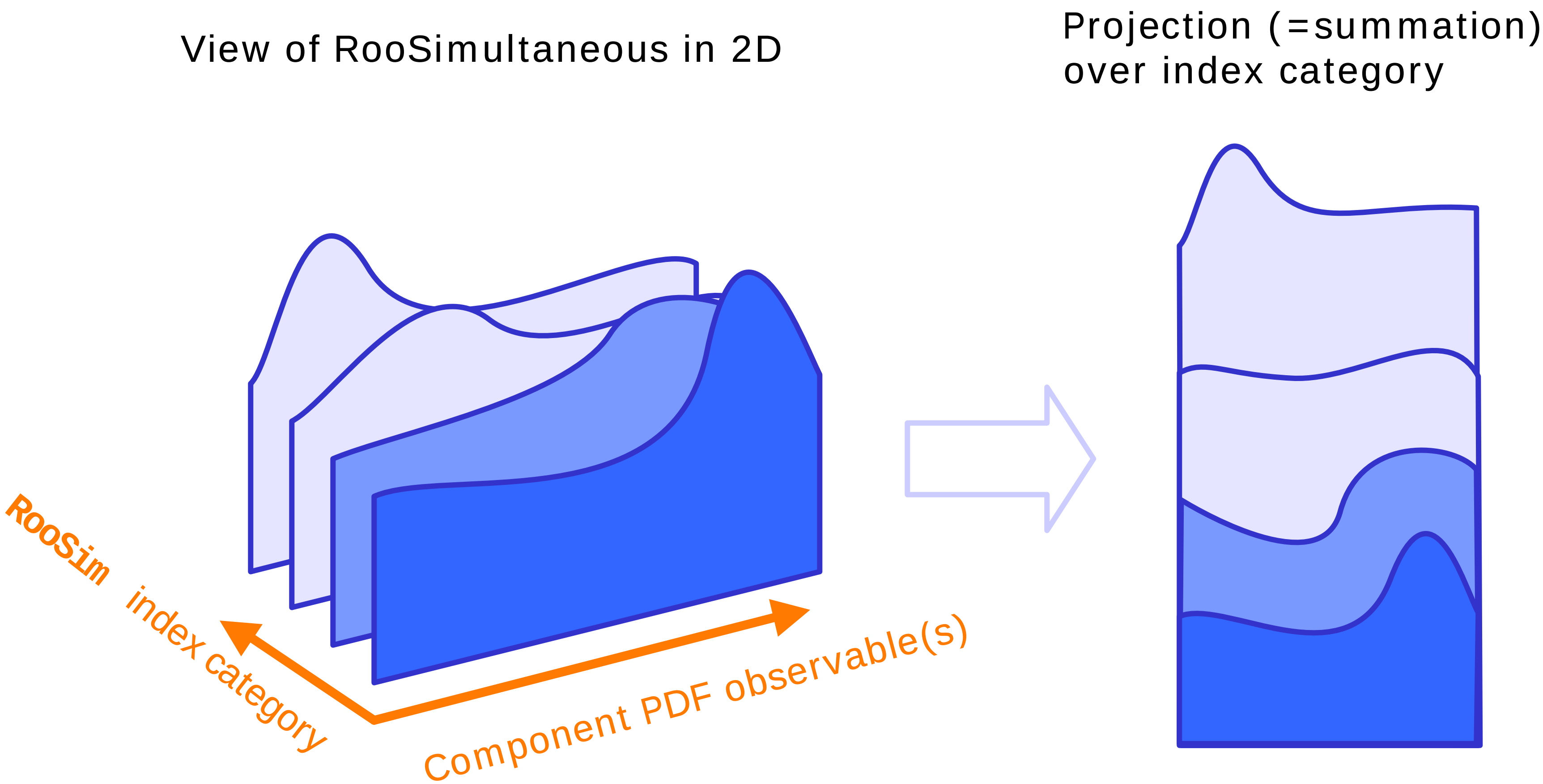
- Plotting of **RooSimultaneous** PDFs is not different from any other PDF
  - **Everything works the same as for regular PDFs**, except that the index category cannot be projected out via *integration*
  - Always provide a projection dataset for the index category (or its components if the index category is composite)
  - Otherwise, treat the RooSimultaneous index category as a regular observable

Simultaneous PDF for (A,B) – plot sum of A,B

```
RooAbsPdf *pdfA, *pdfB; // variables (x,p)
RooCategory *cat ;      // with state "A","B"
RooDataSet* data        // containing (x,cat)
RooSimultaneous sim("sim","sim",
                RooArgList(pdfA,pdfB),*cat) ;

// Plot data/PDF for A+B
RooPlot *frame = x.frame() ;
data->plotOn(frame) ;
sim->plotOn(frame,ProjWData(*cat,data)) ;
```

Needed to project out cat

# Plotting **RooSimultaneous** PDFs



# Plotting a component PDF of a RooSimultaneous

- A component PDF of a **RooSimultaneous** is a slice of the **RooSimultaneous** in the index category.
  - Use **Slice()** *not* **Components()!**

Simultaneous PDF for (A,B) – plot A only

```
// Plot data/PDF for A only
RooPlot *frame = x.frame() ;
data->plotOn(frame,Cut("cat==cat::A")) ;
cat="A" ;
sim->plotOn(frame,Slice(cat),ProjWData(cat,data)) ;
```

Needed to calculate $f_A$

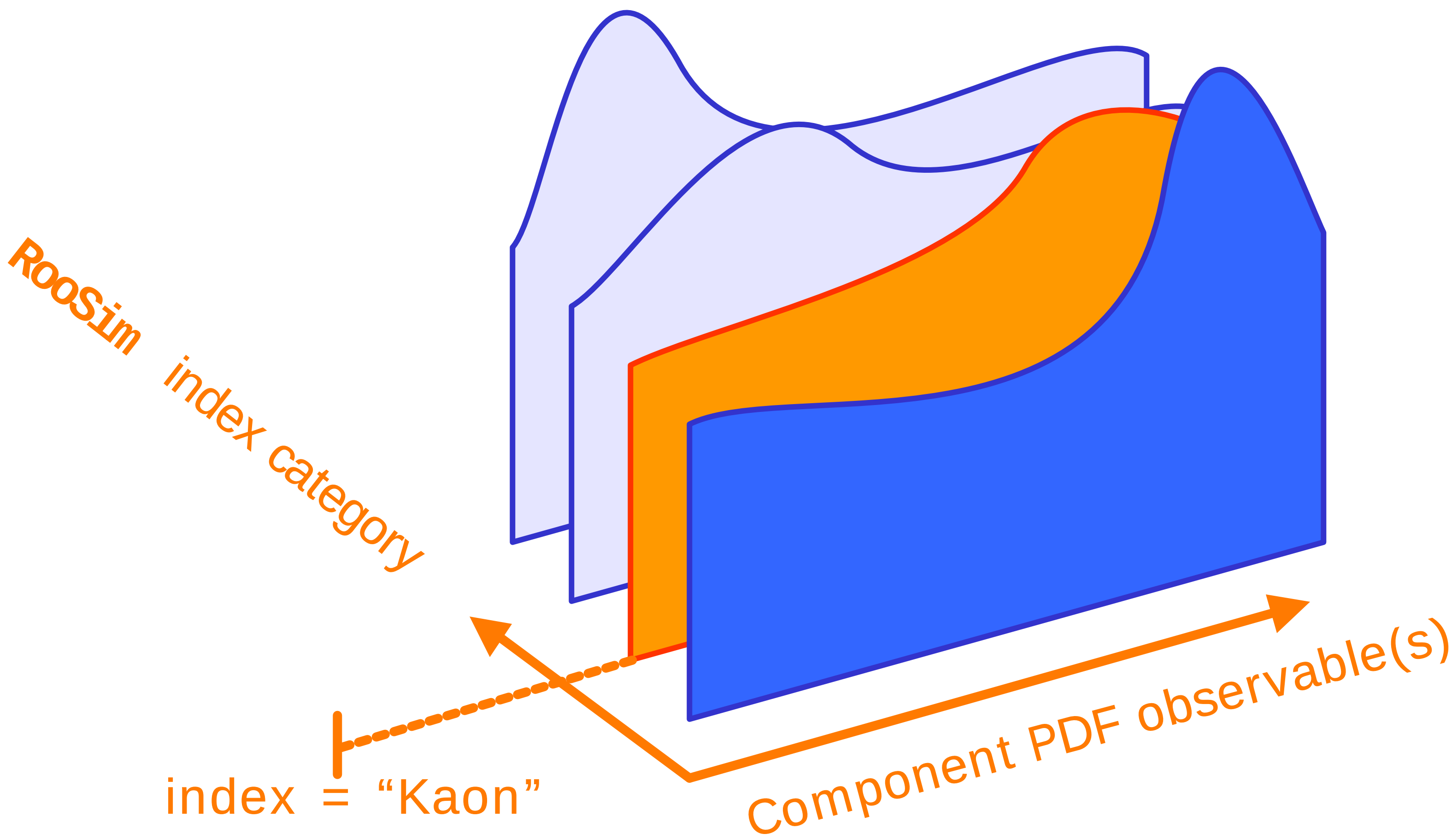
  - Why does **plotSliceOn()** need data?

  Normalization works like in regular **plotSliceOn()**

    - **RooAbsData::plotOn(frame,Cut("cutExpr"))** stores *total* number of events without cut
    - **RooAbsPdf::plotOn(frame,Slice())** normalizes projection to 1 * $f_{slice}$
    - **RooSimultaneous** needs projection dataset to calculate $f_{slice}$

# Plotting **RooSimultaneous** PDFs

A slice in the RooSimultaneous index category selects a component PDF



# RooSimultaneous - Projection a slice with data averaging

- **`RooSimultaneous,Slice()`** and component data averaging
  - RooSimultaneous needs projection dataset for entire dataset
  - Component PDF needs projection dataset for events in slice only

| | |
|---|---|
| A | 0.12 |
| A | 0.23 |
| A | 0.17 |
| A | 0.43 |
| B | 0.34 |
| B | 0.07 |
| B | 0.19 |
| B | 0.13 |
| B | 0.22 |
| B | 1.05 |

- Apparent problem: need 2 projection dataset with different sizes
- Solution: RooSimultaneous::plotOn automatically trims the dataset when passing it on to the components plotOn()

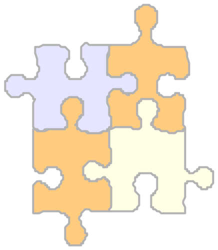
# RooSimultaneous - Projection a slice with data averaging

```
// Plot data for index A
RooPlot *frame = x.frame() ;
data->plotOn(frame,"cat==cat::A") ;

// Plot PDF slice for index A, project out per-event errors
sim->plotSliceOn(frame,ProjWData(RooArgSet(cat,dterr),data)) ;

RooSimultaneous::plotOn(sim) plot on x averages
                  with data index category (cat)
RooAbsReal::plotOn(sim) plot on dt averages
                  using data variables (dterr)
RooAbsReal::plotOn(sim) reducing given projection
                  dataset to entries with cat==A
RooAbsReal::plotOn(sim) only the following components of
                  the projection data will be used: (dterr)
```

RooSimultaneous index projection uses entire dataset

Component dterr projection uses subset of dataset with cat==A

# Miscellaneous



Asymmetry plots

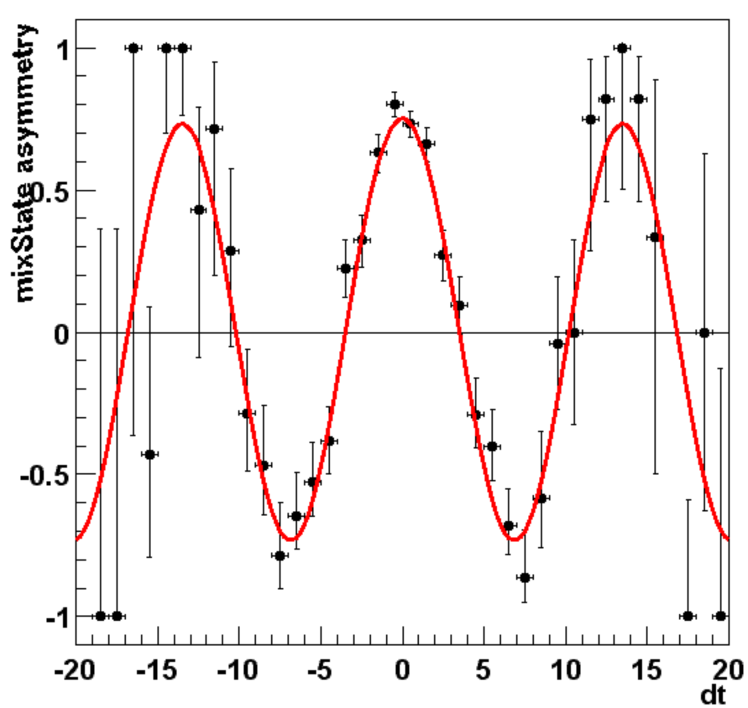
Likelihood plots

Plots in more 1 dimension

# Asymmetry plots

- RooFit supports generic asymmetry plotting
  in *any* **RooCategory** with (+1,-1) or (+1,0,-1) states
  - Example: mixState asymmetry of BMixing PDF & data

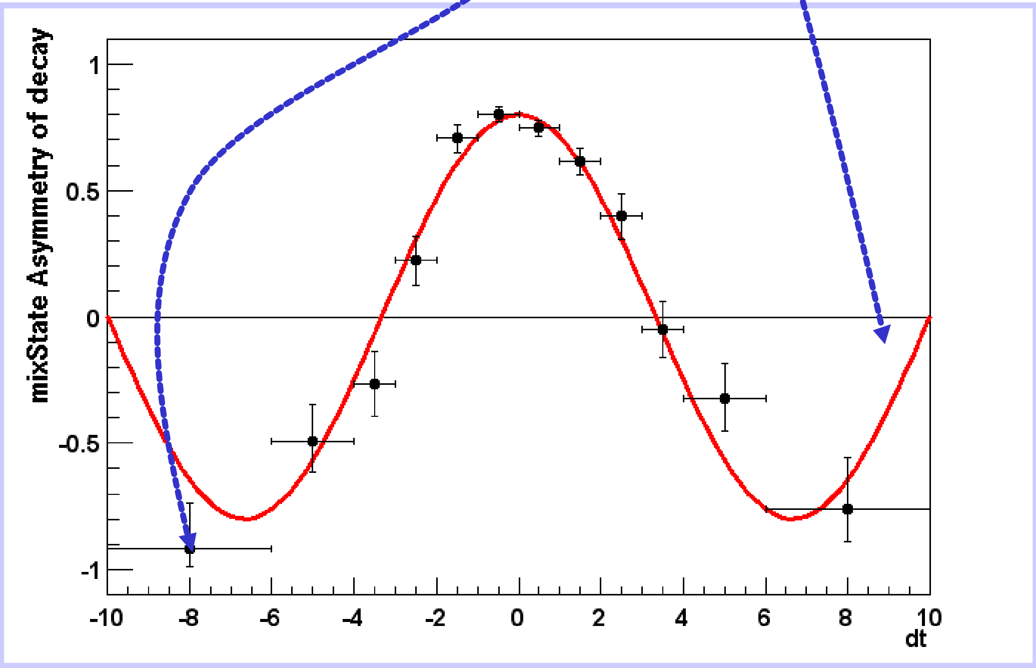
```
RooPlot* dtframe = dt.frame(40) ;
data->plotOn(dtframe,Asymmetry(mixState)) ;
bmix->plotOn(dtframe,Asymmetry(mixState),
                     ProjWData(dterr,data)) ;
```




Can be combined with other plot arguments

# Asymmetry plots - Features

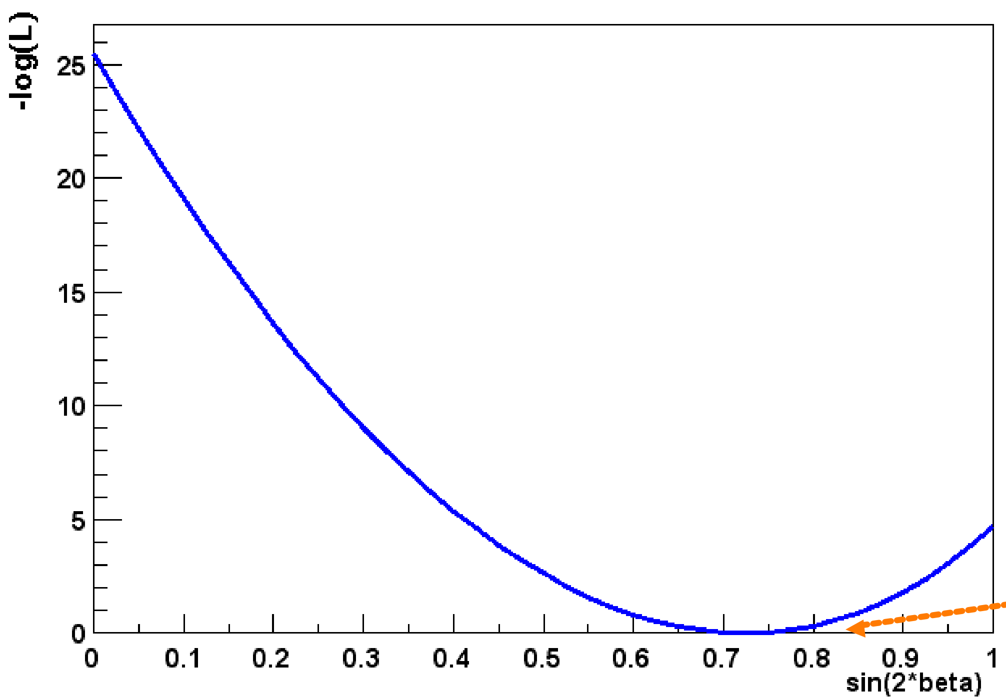
- **`RooAbsData::plotOn(Asymmetry())`**
  - Non-uniform bin sizes OK
  - Points have binomial errors instead of Poisson errors
- **`RooAbsReal::plotOn(Asymmetry())`**
  - *All* regular PDF projection techniques work:
    - Projection via integration
    - Projection with  data averaging
    - Slice plotting
    - ToyMC region plotting
    - ...



# Likelihood scans in 1 dimension

- Plot *–log(L)* for a PDF/dataset on a frame

```
// cpmixPdf and cpmixData previously defined
RooPlot* frame = sin2b.frame(0,1,20) ;
cpmixPdf->plotNLLOn(frame,cpmixData,1.0,kTRUE) ;
```



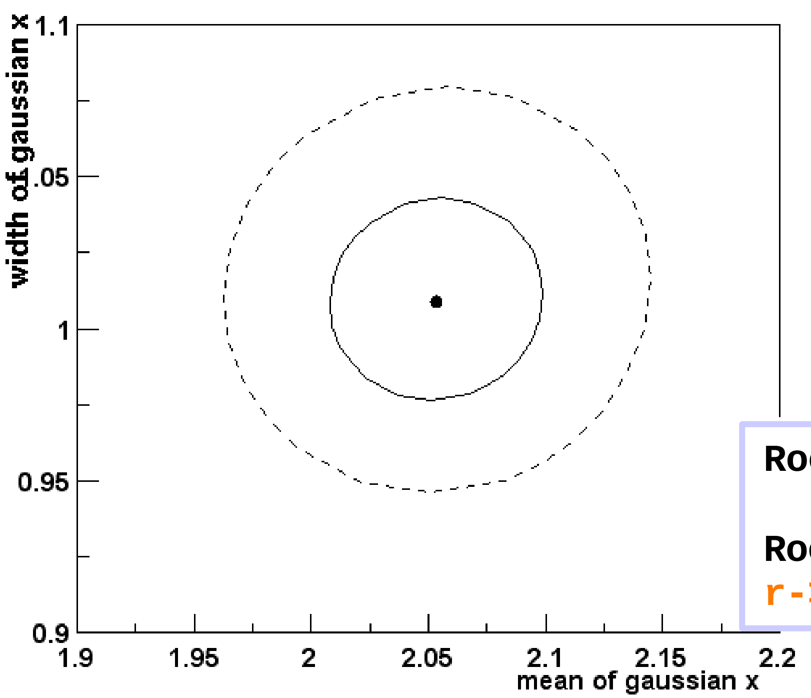

Adaptive NLL sampling used (standard for all RooPlot curves). Explicit control over resolution tunes CPU/precision tradeoff

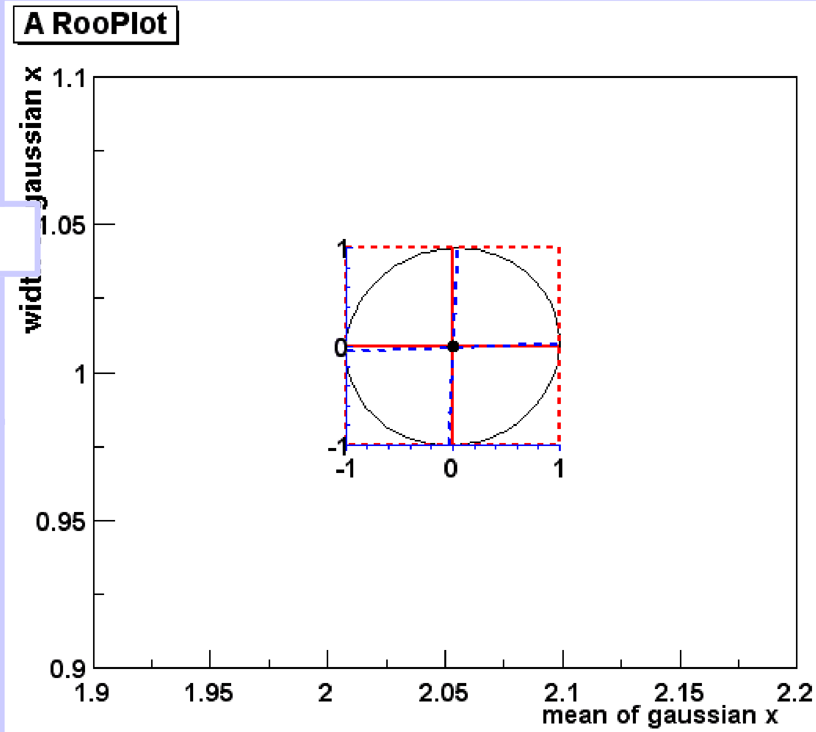
Optional automatic baseline shift to zero

# Likelihood contours in 2 dimensions

- Interface to MINUIT contour plots

```
prod.plotNLLContours(data,meanx,sigmax) ;
```




```
RooFitResult* r =
          prod.fitTo(*data,"mhvr") ;
RooPlot* frame = new RooPlot(…)
r->plotOn(frame,meanx,sigmax,"ME12VHB") ;
```

- Quick contours from corr. coefs

# Plotting in more than 2,3 dimensions

- No equivalent of RooPlot for >1 dimensions
  - Usually >1D plots are not overlaid anyway
  - Methods provided to produce 2/3D ROOT histograms from datasets and PDFs/functions

```
TH2* ph2 = x.createHistogram("x vs y pdf",y,0,0,0,bins) ;
prod.fillHistogram(ph2,RooArgList(x,y)) ;
ph2->Draw("SURF") ;

TH2* dh2 = x.createHistogram("x vs y data",y,0,0,0,bins) ;
data->fillHistogram(dh2,RooArgList(x,y)) ;
dh2->Draw("LEGO") ;
```