Day 4
RooFit

成曈光
北京航空航天大学

Reference

- Following slides copy from
  - https://root.cern/download/roofit-strasbourg-v10.pdf
- 第一届中国 CMS 冬令营
  https://indico.ihep.ac.cn/event/15418/timetable/?view=standard#15-statistical-studies-roofit
- ROOT online resources
  - tutorial:
    https://root.cern.ch/doc/master/group__tutorial__tree.html

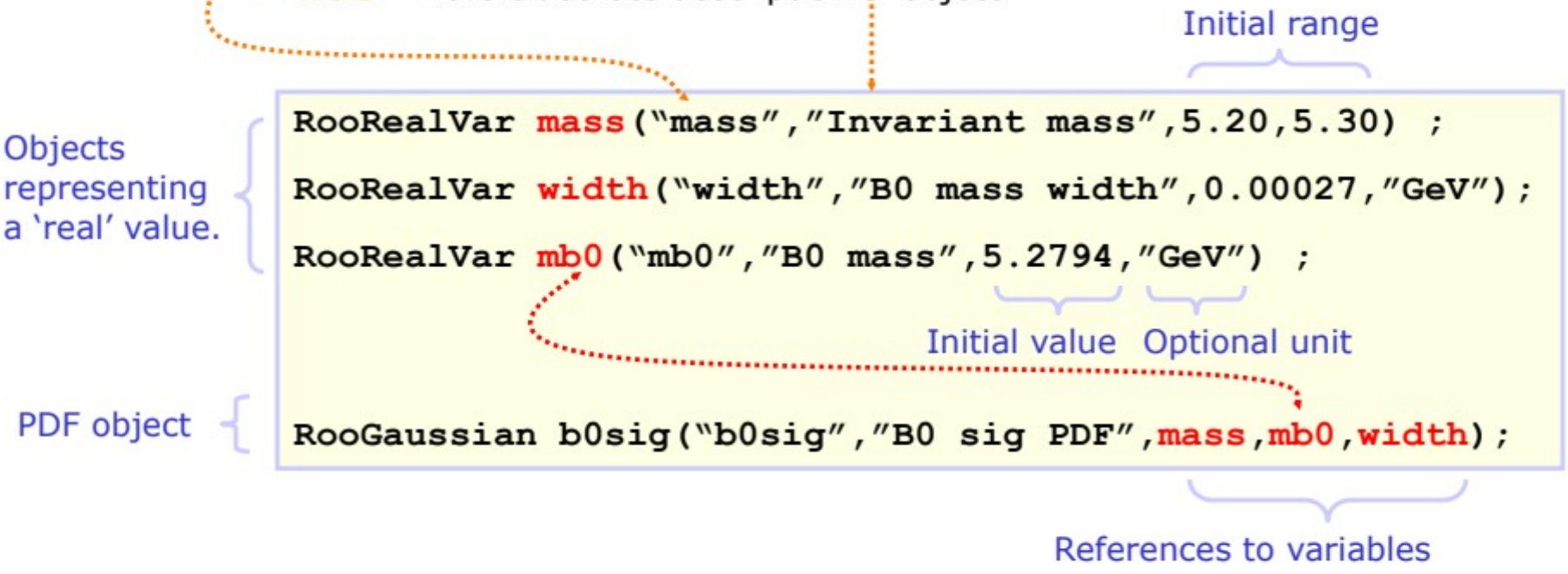
RooFit core design philosophy

- 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	$\int_{x_{\min}}^{x_{\max}} f(x)dx$	`RooRealIntegral`
list of space points		`RooAbsData`

Object-oriented data modeling

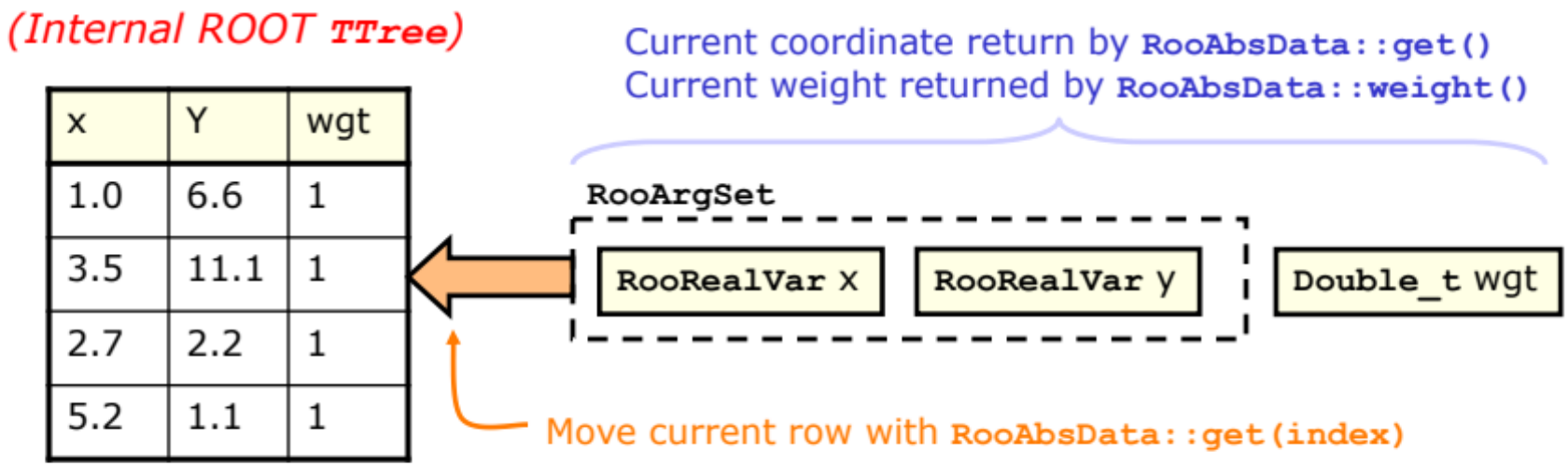
- In RooFit every variable, data point, function, PDF represented in a C++ object
  - Objects classified by data/function type they represent, not by their role in a particular setup
  - All objects are **self documenting**
    - **Name** - Unique identifier of object
    - **Title** – More elaborate description of object



A bit more detail on RooFit datasets

- A dataset is a N-dimensional collection of points
  - With optional weights
  - No limit on number of dimensions
  - Observables continuous (`RooRealVar`) or discrete (`RooCategory`)

- Interface of each dataset is 'current' row
  - Set of RooFit value objects that represent coordinate of current event



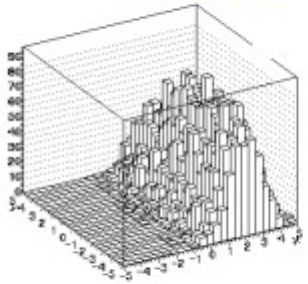
Binned data, or unbinned data (with optional weights)

- Binned or unbinned ML fit?
  - In most RooFit applications it doesn't matter

Unbinned

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

Binned



Internally binned data is represented the same way as unbinned data,
A ROOT TTree with the bin coordinates

RooDataSet **RooDataHist**

RooAbsData

- For example ML fitting interface takes abstract RooAbsData object
  - Binned data → Binned likelihood
  - Unbinned data → Unbinned likelihood
- Weights are supported in unbinned datasets
  - But use with care. Error analysis in ML fits to weighted unbinned data can be complicated!

Importing unbinned data

- From ROOT trees
  - `RooRealVar` variables are imported from /D /F /I tree branches
  - `RooCategory` variables are imported from /I /b tree branches
  - Mapping between `TTree` branches and dataset variables by name: e.g. `RooRealVar x("x","x",-10,10)` imports `TTree` branch "x"

```
RooRealVar x("x","x",-10,10) ;
RooRealVar c("c","c",0,30) ;
RooDataSet data("data","data",inputTree,RooArgSet(x,c));
```

  - Only events with 'valid' entries are imported. In above example any events with |x|>10 or c<0 or c>30 are *not* imported

- From ASCII files
  - One line per event, order of variables as given in RooArgList

```
RooDataSet* data =
   RooDataSet::read("ascii.file",RooArgList(x,c)) ;
```

Importing binned data

- From ROOT **`THx`** histogram objects

```
RooDataHist bdata1("bdata","bdata",RooArgList(x),histo1d);
RooDataHist bdata2("bdata","bdata",RooArgList(x,y),histo2d);
RooDataHist bdata3("bdata","bdata",RooArgList(x,y,z),histo3d);
```

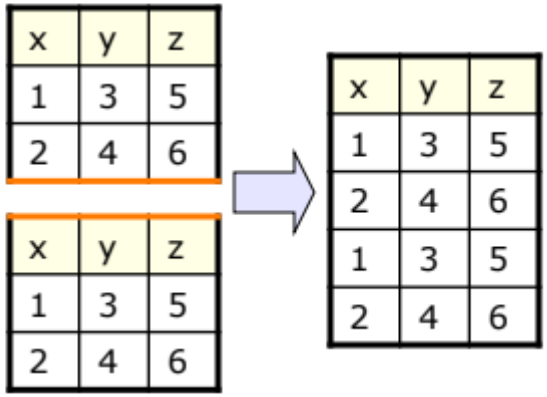
- From a **`RooDataSet`**

```
RooDataHist* binnedData = data->binnedClone() ;
```

Extending and reducing **unbinned** datasets

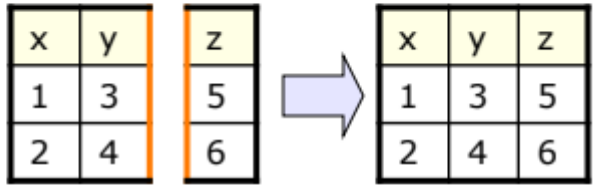
- Appending

```
RooDataSet d1("d1","d1",RooArgSet(x,y,z));
RooDataSet d2("d2","d2",RooArgSet(x,y,z));

d1.append(d2) ;
```



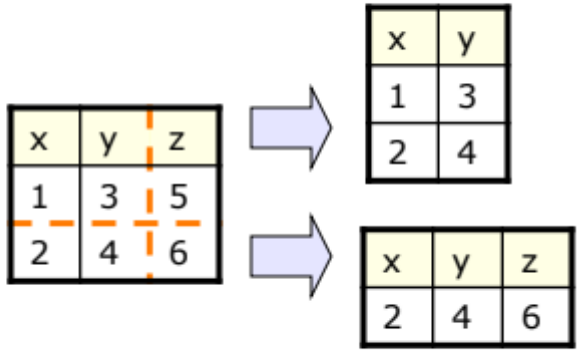
- Merging

```
RooDataSet d1("d1","d1",RooArgSet(x,y) ;
RooDataSet d2("d2","d2",RooArgSet(z)) ;

d1.merge(d2) ;
```



- Reducing

```
RooDataSet d1("d1","d1",RooArgSet(x,y,z) ;

RooDataSet* d2 = d1.reduce(RooArgSet(x,y));

RooDataSet* d3 = d1.reduce("x>1");
```



Adding and reducing **binned** datasets

- Adding

```
RooDataHist d1("d1","d1",
                RooArgSet(x,y));
RooDataHist d2("d2","d2",
                RooArgSet(x,y));

d1.add(d2) ;
```

w	y1	y2
x1	0	1
x2	1	0

w	y1	y2
x1	0	1
x2	1	0

w	y1	y2
x1	1	1
x2	1	1

- Reducing

```
RooDataHist d1("d1","d1",
                RooArgSet(x,y) ;

RooDataHist* d2 =
     d1.reduce(x);

RooDataHist* d3 =
     d1.reduce("x>1");
```

w	y1	y2
x1	0	1
x2	1	0

-	w
x1	1
x2	1

w	y1	y2
x1	0	1
x2	1	0

w	y1	y2
x1	0	0
x2	1	0

HANDS-ON

rf102_dataimport

rf403_weightedevts

Math – Extended Maximum Likelihood

- Maximum likelihood information only parameterizes *shape* of distribution
  - I.e. one can determine *fraction* of signal events from ML fit, but not *number* of signal events

$$L(\vec{p}) = \prod_i F(\vec{x}_i; \vec{p}), \quad \text{i.e.} \quad L(\vec{p}) = F(x_0; \vec{p}) \cdot F(x_1; \vec{p}) \cdot F(x_2; \vec{p}) \ldots$$

- Extended Maximum likelihood add extra term

$$-\log\left(L(\vec{p})\right) = -\sum_D \log(g(\vec{x}_i, \vec{p})) + N_{\text{exp}} - N_{obs} \log(N_{\text{exp}})$$

Log of Poisson(Nexp,Nobs) (modulo a constant)

  - Clever choice of parameters will allows us to extract N_{sig} and N_{bkg} in one pass ($N_{exp}=N_{sig}+N_{bkg}$, $f_{sig}=N_{sig}/(N_{sig}+N_{bkg})$)

Extended p.d.f form of RooAddPdf

- If extended ML term is introduced, we can fit expected number of events (N_{exp}) in addition to shape parameters
- In case of sum of p.d.f.s it is convenient to *re-parameterize* sum of p.d.f.s.

$$\begin{pmatrix} f_{sig} \\ N_{\exp} \end{pmatrix} \Rightarrow \begin{pmatrix} N_{sig} \equiv f_{sig} N_{\exp} \\ N_{bkg} \equiv (1 - f_{sig}) N_{\exp} \end{pmatrix}$$

- This transformation is applied automatically in **RooAddPdf** if equal number of p.d.f.s and coefs are given

```
RooRealVar nsig("nsig","number of signal events",100,0,10000) ;
RooRealVar nbkg("nbkg","number of backgnd events",100,0,10000) ;
RooAddPdf sume("sume","extended sum pdf",RooArgList(gauss,argus),
                                         RooArgList(nsig,nbkg)) ;
```

General features of extended p.d.f.s

- Extended term –log(Poisson(N_{obs},N_{exp})) is not added by default to likelihood
  - Use the `Extended()` argument to fit to have it added

```
// Regular maximum likelihood fit
pdf.fitTo(*data) ;

// Extended maximum likelihood fit
pdf.fitTo(*data,Extended(kTRUE)) ;
```

- If p.d.f. is extended, N_{exp} is default number of events to generate

```
// Generate pdf.expectedEvents() events
RooDataSet* data = pdf.generate(x) ;

// Generate 1000 events
RooDataSet* data = pdf.generate(x,1000) ;
```

HANDS-ON

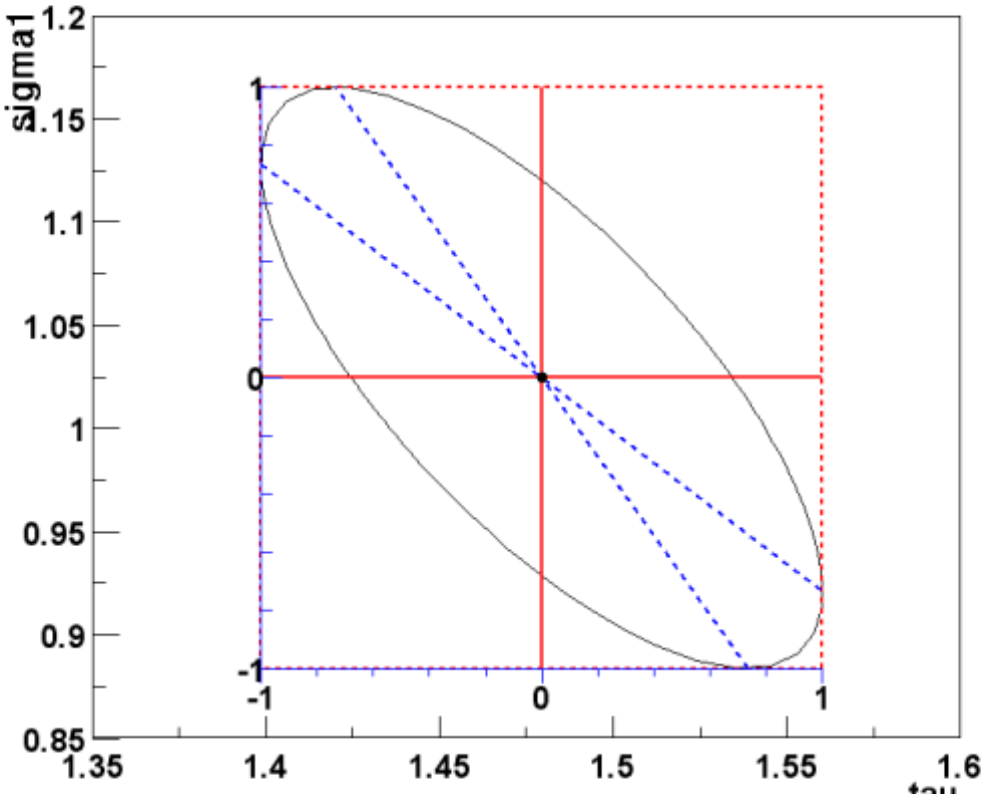
rf202_extendedmlfit.py

HiggsCombineRoofitTutorial

Visualize errors and correlation matrix elements

```
RooFitResult* r = pdf->fitTo(data,"mhvr") ;
RooPlot* f = new RooPlot(tau,sigma1,1.35,1.6,0.85,1.20) ;
r->plotOn(f,tau,sigma1,"ME12VHB") ;
f->Draw() ;
```

Works on any `RooFitResult`, Also after persistence



MINUIT contour scan is also possible with a separate interface

Browsing fit results with `RooFitResult`

- As fits grow in complexity (e.g. 45 floating parameters), number of output variables increases
  - Need better way to navigate output that MINUIT screen dump
- **`RooFitResult`** holds complete snapshot of fit results
  - Constant parameters
  - Initial and final values of floating parameters
  - Global correlations & full correlation matrix
  - Returned from **`RooAbsPdf::fitTo()`** when "**`r`**" option is supplied
- Compact & verbose printing mode

Compact Mode

Constant parameters omitted in compact mode

Alphabetical parameter listing

```
fitres->Print() ;

  RooFitResult: min. NLL value: 1.6e+04, est. distance to min: 1.2e-05

    Floating Parameter    FinalValue +/-  Error
  --------------------  --------------------------
                argpar   -4.6855e-01 +/-  7.11e-02
                g2frac    3.0652e-01 +/-  5.10e-03
                 mean1    7.0022e+00 +/-  7.11e-03
                 mean2    1.9971e+00 +/-  6.27e-03
                 sigma    2.9803e-01 +/-  4.00e-03
```

Browsing fit results with `RooFitResult`

Verbose printing mode

```
fitres->Print("v") ;

  RooFitResult: min. NLL value: 1.6e+04, est. distance to min: 1.2e-05

  Constant Parameter    Value
  --------------------  ------------
                cutoff    9.0000e+00
                g1frac    3.0000e-01

    Floating Parameter  InitialValue    FinalValue +/-  Error      GblCorr.
  --------------------  ------------  --------------------------  --------
                argpar   -5.0000e-01   -4.6855e-01 +/-  7.11e-02  0.191895
                g2frac    3.0000e-01    3.0652e-01 +/-  5.10e-03  0.293455
                 mean1    7.0000e+00    7.0022e+00 +/-  7.11e-03  0.113253
                 mean2    2.0000e+00    1.9971e+00 +/-  6.27e-03  0.100026
                 sigma    3.0000e-01    2.9803e-01 +/-  4.00e-03  0.276640
```

Constant parameters listed separately

Initial,final value and global corr. listed side-by-side

Correlation matrix accessed separately

Browsing fit results with `RooFitResult`

- Easy navigation of correlation matrix
  - Select single element or complete row by parameter name

```
r->correlation("argpar","sigma")
(const Double_t)(-9.25606412005910845e-02)

r->correlation("mean1")->Print("v")
RooArgList::C[mean1,*]: (Owning contents)
  1) RooRealVar::C[mean1,argpar] :  0.11064 C
  2) RooRealVar::C[mean1,g2frac] : -0.0262487 C
  3) RooRealVar::C[mean1,mean1]  :  1.0000 C
  4) RooRealVar::C[mean1,mean2]  : -0.00632847 C
  5) RooRealVar::C[mean1,sigma]  : -0.0339814 C
```

- **`RooFitResult`** persistable with ROOT I/O
  - Save your batch fit results in a ROOT file and navigate your results just as easy afterwards

HANDS-ON

rf607_fitresult
rf610_visualerror

Model building – Generic expression-based PDFs

- If your favorite PDF isn't there
  and you don't want to code a PDF class right away
  → use **RooGenericPdf**
- Just write down the PDFs expression as a C++ formula

```
// PDF variables
RooRealVar x("x","x",-10,10) ;
RooRealVar y("y","y",0,5) ;
RooRealVar a("a","a",3.0) ;
RooRealVar b("b","b",-2.0) ;

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

- Numeric normalization automatically provided

Model Building – Writing your own class

- Factory class exists (`RooClassFactory`) that can write, compile, link C++ code for RooFit p.d.f. and function classes
- **Example 1**:
  - Write class `MyPdf` with variable x,y,a,b in files `MyPdf.h, MyPdf.cxx`

```
RooClassFactory::makePdf("MyPdf","x,y,a,b");
```

  - Only need to fill `evaluate()` method in `MyPdf.cxx` in terms of a,b,x
  - Can add optional code to support for analytical integration, internal event generation

Model Building – Writing your own class

- **Example 2**:
  - Functional equivalent to `RooGenericPdf`: Write class `MyPdf` with prefilled one-line function expression, compile and link p.d.f, create and return instance of class

Compiled code

```
RooAbsPdf* gp = RooClassFactory::makePdfInstance("gp",
                          "exp(x*y+a)-b*x",RooArgSet(x,y,a,b));
```

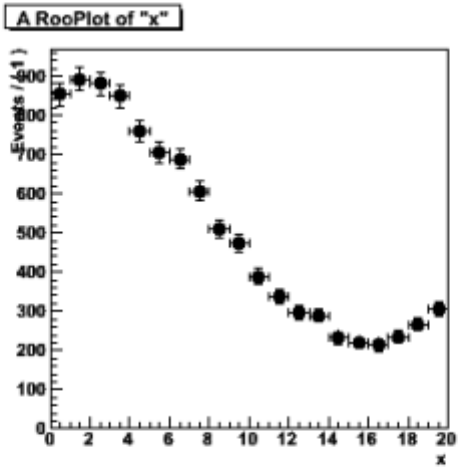
Interpreted code

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

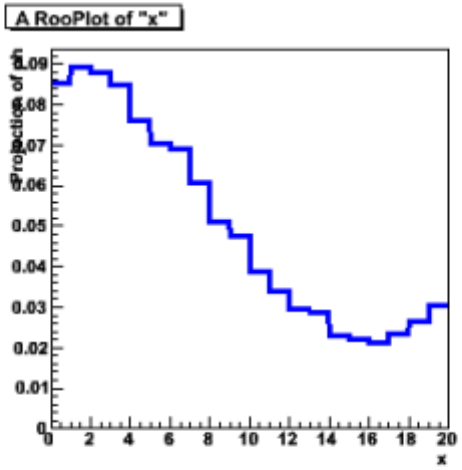
Highlight of non-parametric shapes - histograms

- Will highlight two types of non-parametric p.d.f.s
- Class **`RooHistPdf`** – a p.d.f. described by a histogram

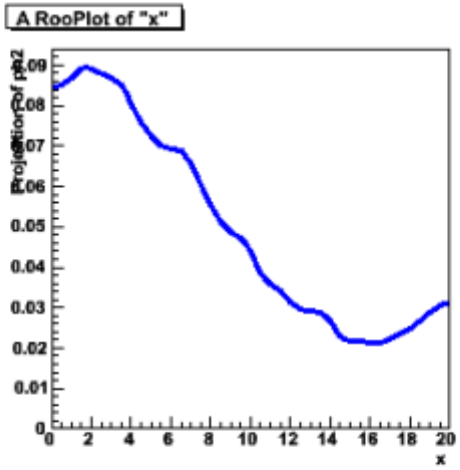
`dataHist`



`RooHistPdf(N=0)`



`RooHistPdf(N=4)`




```
// Histogram based p.d.f with N-th order interpolation
RooHistPdf ph("ph","ph",x,*dataHist,N) ;
```

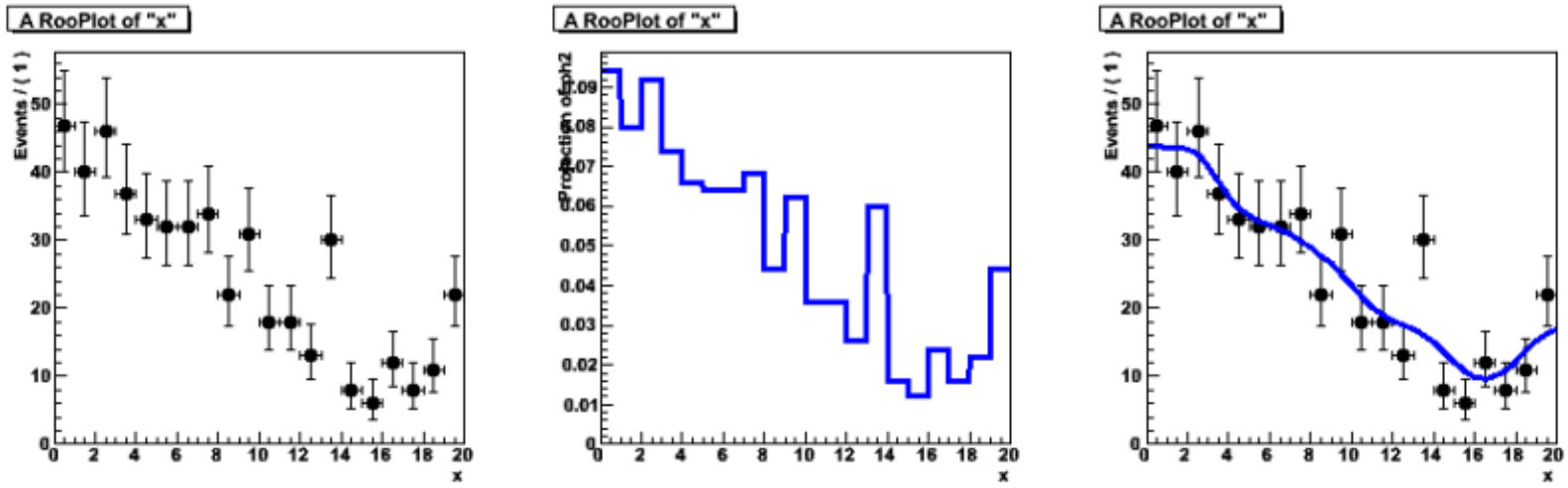
- Not so great at low statistics (especially problematic in >1 dim)

Highlight of non-parametric shapes – kernel estimation

- Example with comparison to histogram based p.d.f
  - Superior performance at low statistics
  - Can mirror input data over boundaries to reduce 'edge leakage'
  - Works also in >1 dimensions (class `RooNDKeysPdf`)

```
// Adaptive kernel estimation p.d.f
 RooKeysPdf k("k","k",x,*d,RooKeysPdf::MirrorBoth) ;
```

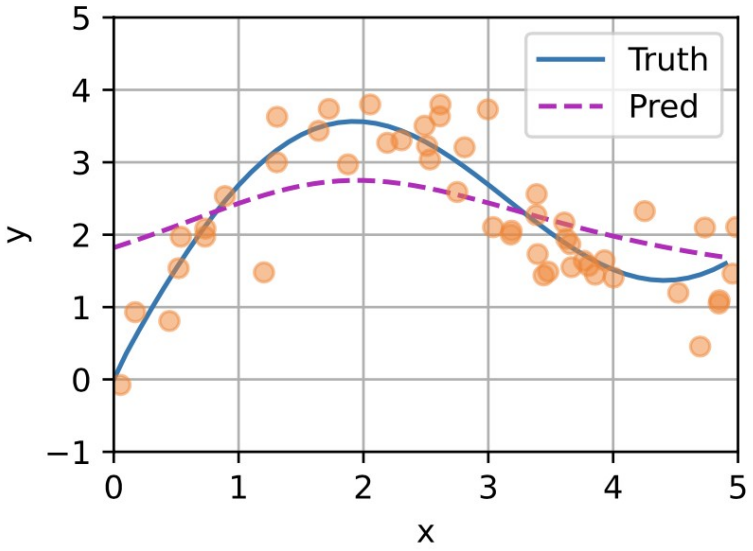
Data (N=500) **RooHistPdf(data)** **RooKeysPdf(data)**



$$f(x) = \sum_{i=1}^{n} \frac{K(x - x_i)}{\sum_{j=1}^{n} K(x - x_j)} y_i$$

$$K(u) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{u^2}{2}).$$

$$\begin{aligned} f(x) &= \sum_{i=1}^{n} \alpha(x, x_i) y_i \\ &= \sum_{i=1}^{n} \frac{\exp\left(-\frac{1}{2}(x - x_i)^2\right)}{\sum_{j=1}^{n} \exp\left(-\frac{1}{2}(x - x_j)^2\right)} y_i \\ &= \sum_{i=1}^{n} \mathrm{softmax}\left(-\frac{1}{2}(x - x_i)^2\right) y_i. \end{aligned}$$



x - query

x_i - key

y_i - value

HANDS-ON

rf706_histpdf
rf707_kernelestimation

Backup
likelihood scan, MINUIT, profiled likelihood

Fitting and likelihood minimization

- What happens when you do `pdf->fitTo(*data)`
  - 1) Construct object representing –log of (extended) likelihood
  - 2) Minimize likelihood w.r.t floating parameters using MINUIT

- Can also do these two steps explicitly by hand

```
// Construct function object representing -log(L)
RooNLLVar nll("nll","nll",pdf,data) ;

// Minimize nll w.r.t its parameters
RooMinuit m(nll) ;
m.migrad() ;
m.hesse() ;
```

Constructing the likelihood function

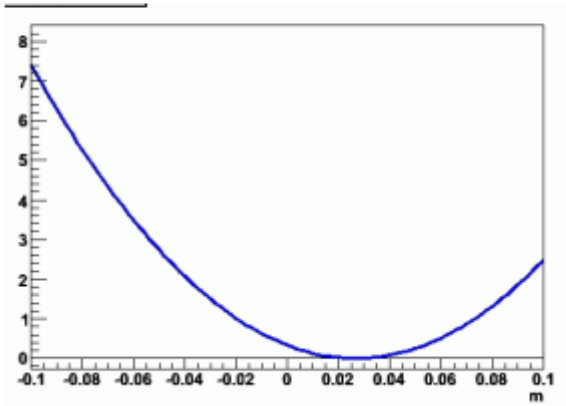
- Class **RooNLLVar** works universally for all p.d.f.s and all types of data
  - Binned data → Binned likelihood
  - Unbinned data → Unbinned likelihood

- Can add named arguments to constructor to control details of likelihood definition and mode of calculation

```
RooNLLVar nll("nll","nll",pdf,data,Extended()) ;
```

  - Works like a regular RooFit function object, i.e. can retrieve value and make plots as usual

```
Double_t val = nll.getVal() ;
RooArgSet* vars = nll.getVariables()

RooPlot* frame = p.frame() ;
nll.plotOn(frame) ;
```



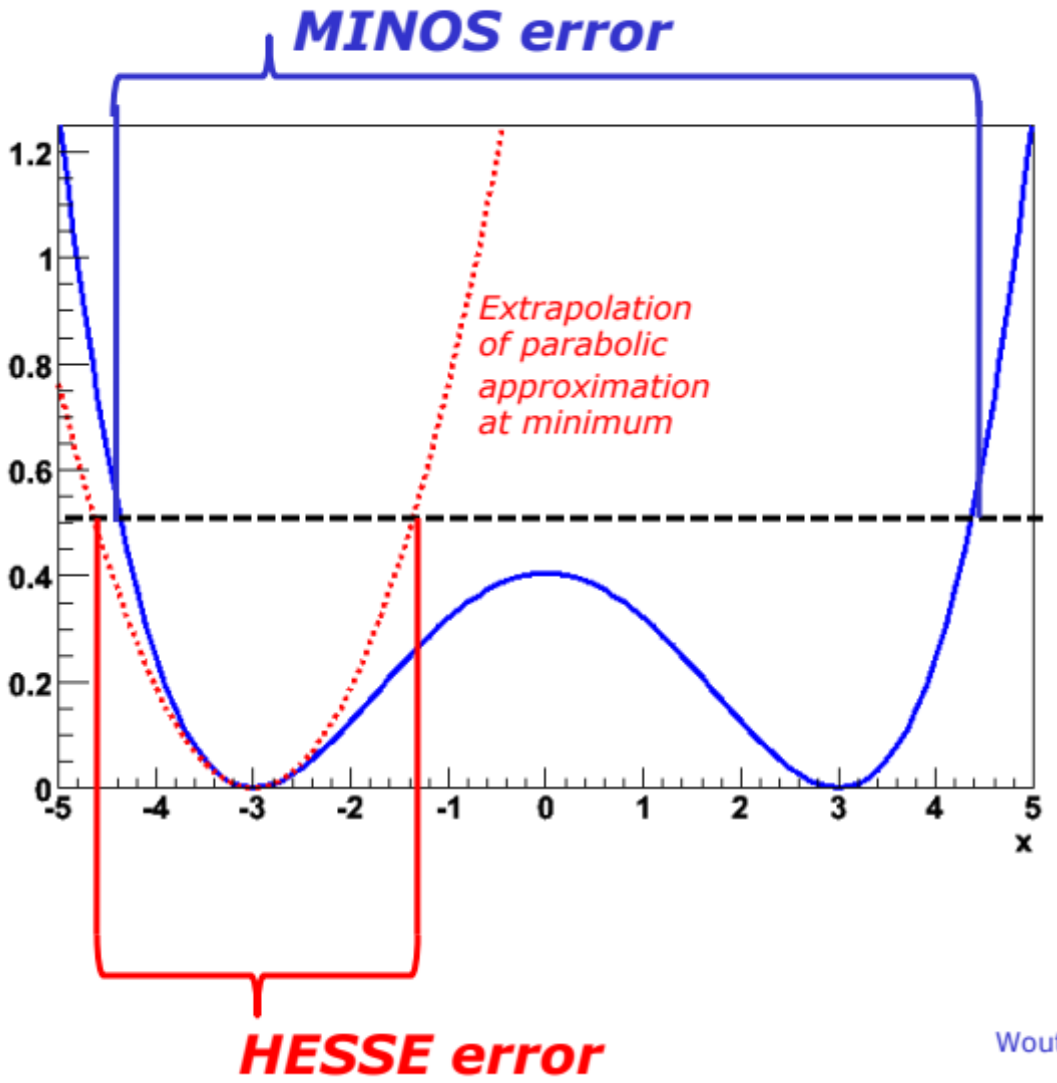
A brief description of MINUIT functionality

- 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

Illustration of difference between HESSE and MINOS errors

- 'Pathological' example likelihood with multiple minima and non-parabolic behavior



Demonstration of RooMinuit use

```
// Start Minuit session on above nll
RooMinuit m(nll) ;

// MIGRAD likelihood minimization
m.migrad() ;

// Run HESSE error analysis
m.hesse() ;

// Set sx to 3, keep fixed in fit
sx.setVal(3) ;
sx.setConstant(kTRUE) ;

// MIGRAD likelihood minimization
m.migrad() ;

// Run MINOS error analysis
m.minos()

// Draw 1,2,3 'sigma' contours in sx,sy
m.contour(sx,sy) ;
```

Working with *profile* likelihoods

- Given a likelihood, the profile likelihood is defined as
  - Where the hatted quantities represent the value of that parameter at which the –log(L) is minimal

$$PL(p) = \frac{L(p,\hat{q})}{L(\hat{p},\hat{q})}$$

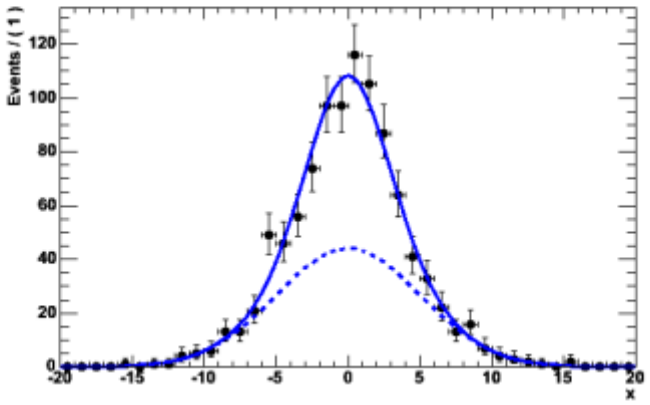
- Represented in RooFit with class **RooProfileLL**

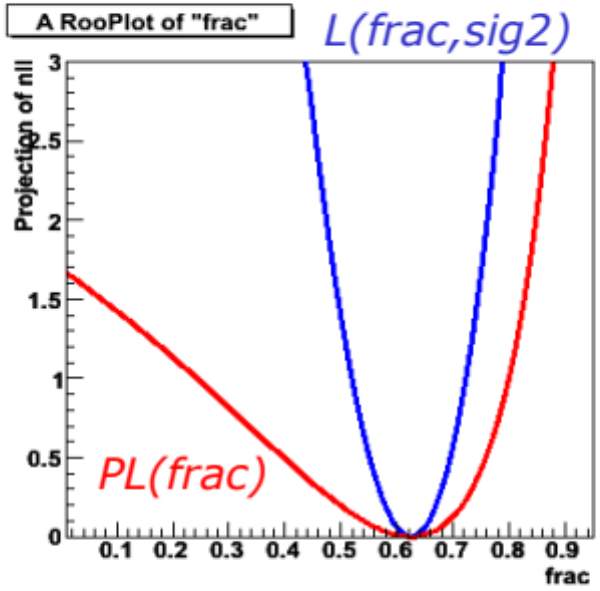
```
// First make regular likelihood object (with parameters frac and sg2)
RooNLLVar nll("nll","nll",model,*data) ;

// Now make profile likelihood in frac
RooProfileLL pll_frac("pll_frac","pll",nll,frac) ;
```

  - A profile likelihood is a regular function object in RooFit, you can plot it with plotOn() etc.
  - *However it is expensive*, as each function evaluation requires a MIGRAD minimization step!

Plotting a profile likelihood

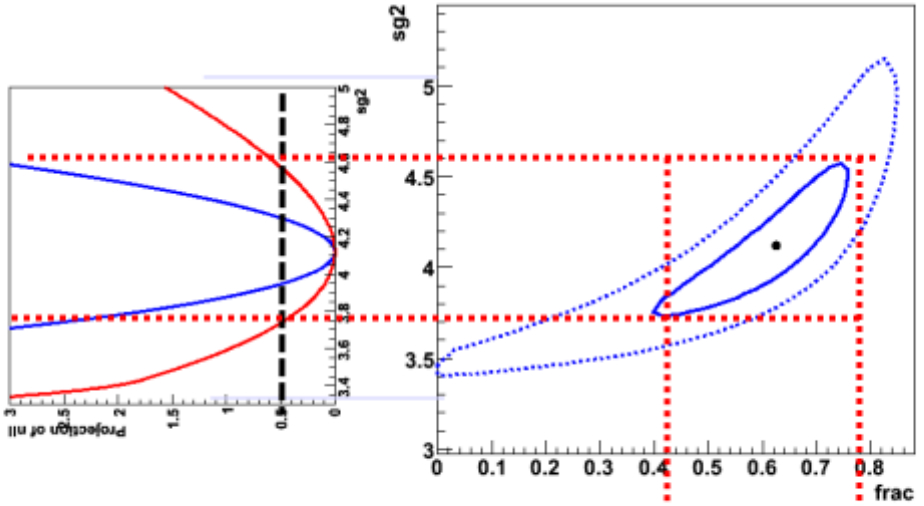
- Example with strong correlations:
  - sum of two Gaussians with similar widths and floating fraction





- Profile likelihood much broader than likelihood because changing the width the 2nd gaussian can largely compensate for off-value of fraction

Relation between MINOS errors and profile likelihood



- MINOS error box and profile likelihood give same error for multi-dimensional likelihood