Tips and tricks with RooFit

Tutorial

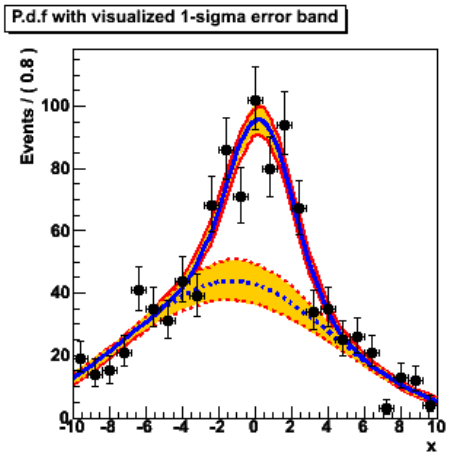
Leszek Kosarzewski, BEng, Ph.D.

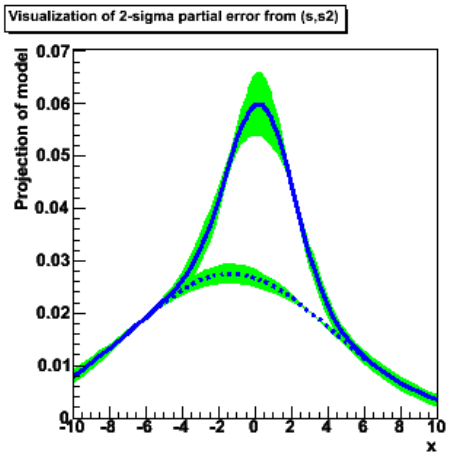
Faculty of Nuclear Sciences and Physical Engineering
Czech Technical University in Prague

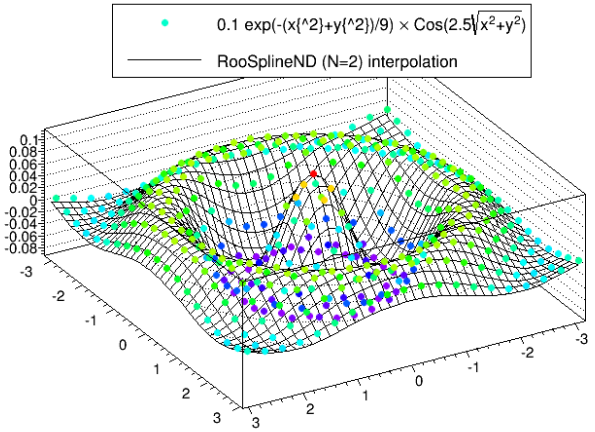
Workshop EJČF Zimni Škola, Bílý Potok 12-18.1.2020

Outline

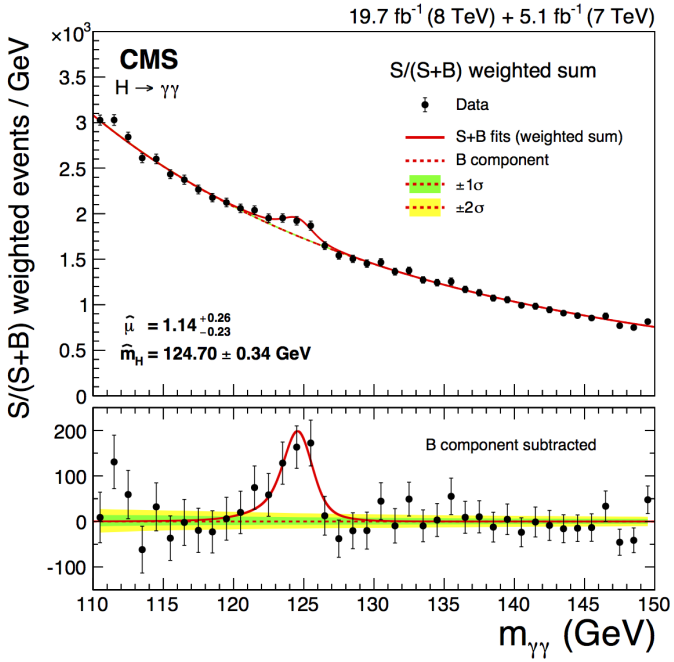
RooFit - Introduction





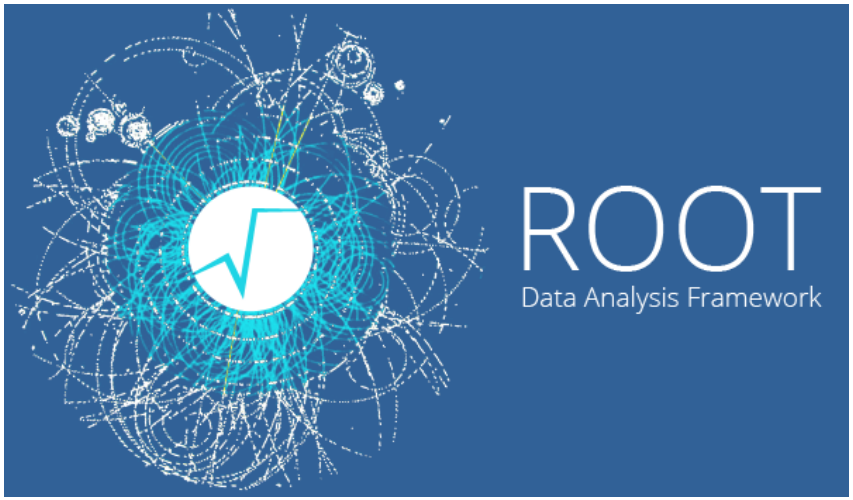


- Fitting is often performed as part of physics analysis to extract signal yields etc.
- RooFit is an easy to use package useful for:
  - Fitting complicated functions 1D and multidimensional:
    - χ^2 fits
    - Binned likelihood fits
    - Unbinned likelihood fits
  - MC data generation
  - Testing and validation of fits
  - Efficiency calculation



[CMS CR-2015/193]

- RooFit is part of ROOT
- Get it here: `https://root.cern.ch/`
- Build ROOT with roofit option ON



Manuals and links

- RooFit Users Manual:
  `http://root.cern.ch/download/doc/RooFit_Users_Manual_2.91-33.pdf`
- RooFit Quick Start Guide:
  `https://root.cern.ch/download/doc/roofit_quickstart_3.00.pdf`
- RooFit in 20 minutes: `https://root.cern.ch/roofit-20-minutes`
- Tutorials:
  - `https://root.cern.ch/doc/master/group__tutorial__roofit.html`
  - $ROOTSYS/tutorials/roofit
- Core documentation:
  `https://root.cern.ch/doc/master/group__Roofitcore.html`

RooFit vs. Math

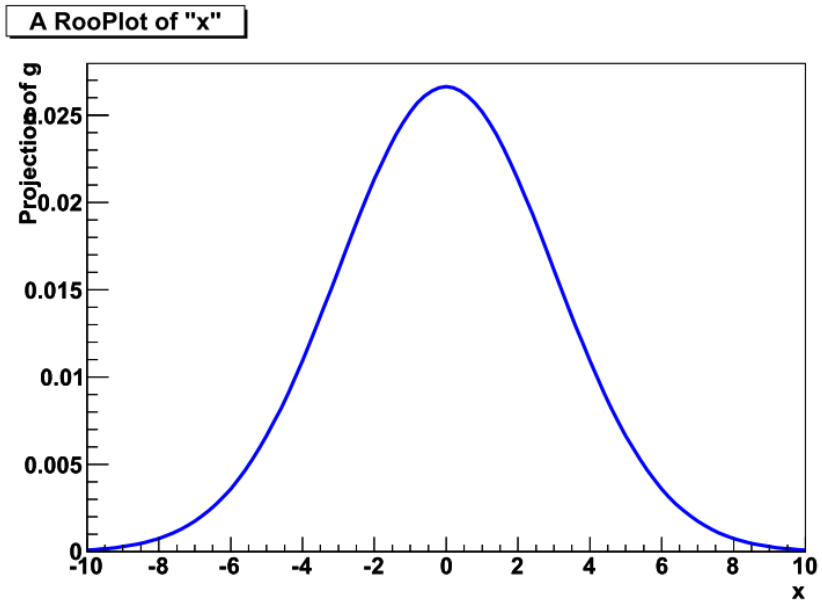
Concept	Math Symbol	`RooFit` class name
Variable	x, p	`RooRealVar`
Function	$f(\vec{x})$	`RooAbsReal`
PDF	$F(\vec{x}; \vec{p}, \vec{q})$	`RooAbsPdf`
Space point	$\vec{x}$	`RooArgSet`
Integral	$\int_{\vec{x}_{min}}^{\vec{x}_{max}} f(\vec{x}) d\vec{x}$	`RooRealIntegal`
Derivative	dF/dx	`RooDerivative`
-log(Likelihood)	$-\sum_{data} \log(F(x_i, \vec{p})$	`RooNLLVar`
List of space points	$\vec{x}_k$	`RooAbsData`

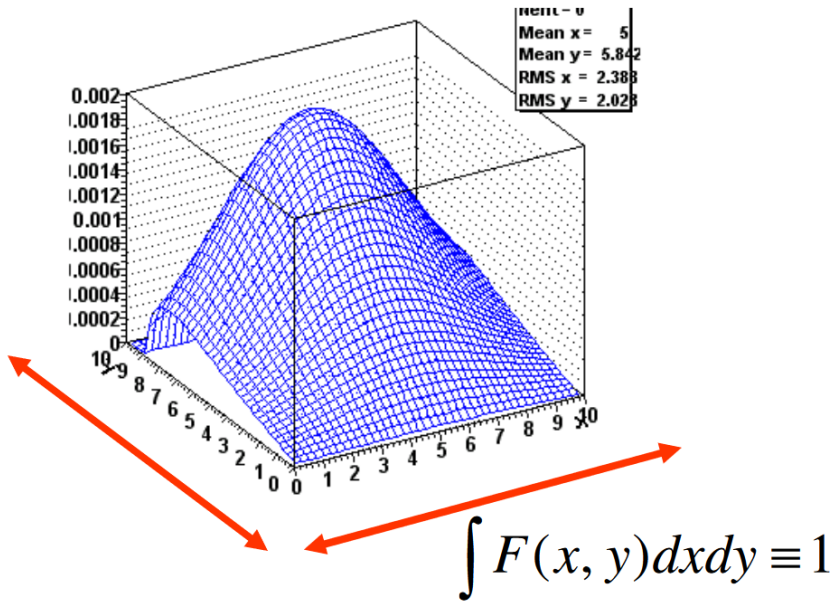
TAB. 1.1 – Correspondence between mathematical concepts and `RooFit` classes.

- RooFit objects correspond to mathematical concepts
- Implemented as C++ classes in an elegant way
- Brings in advantages of object oriented programming to fitting problems
- They can be referenced and chained together, which allows:
  - Creation of complex functions, PDFs, etc.
  - Uncertainty propagation including correlations
  - Calculation of values at the end, not needed at every step

Probability density functions: PDFs

Normalized $\int_{\vec{x}_{min}}^{\vec{x}_{max}} F(\vec{x}, \vec{p}) d\vec{x} \equiv 1$ and positive $f(\vec{x}, \vec{p}) \geq 0$





- PDFs can be 1D or multidimensional

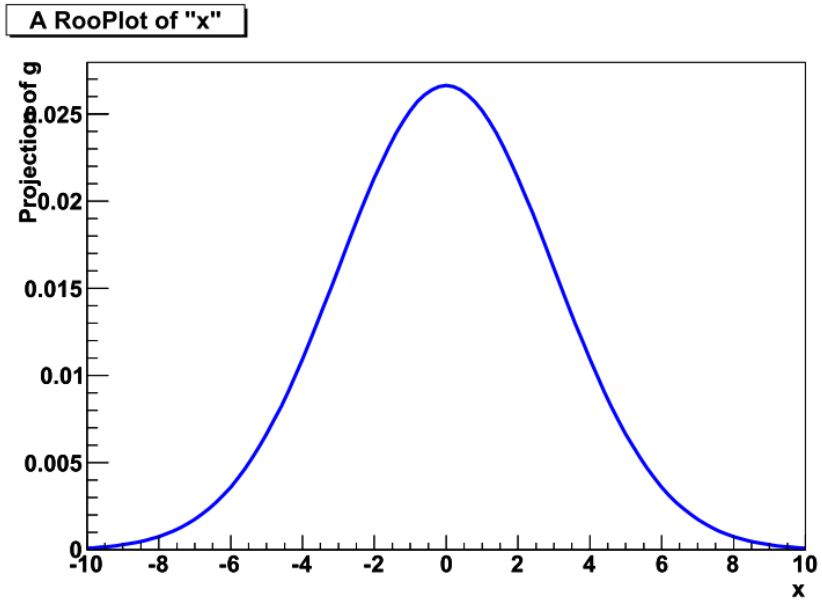
Listing 1: Gaussian

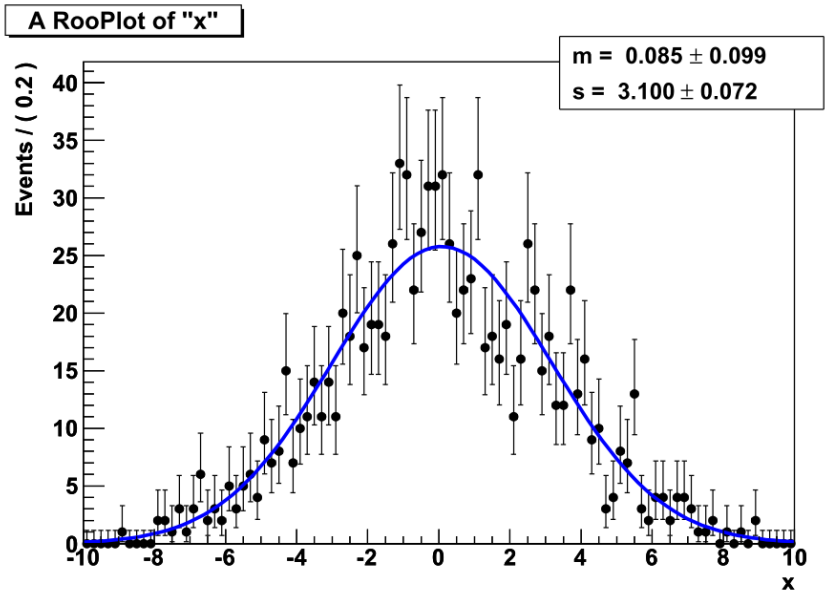
```
RooRealVar x("x", "mass", -10.0, 10.0, "GeV/c^{2}"); // variable (name,
      description, min, max, unit)
RooRealVar mean("#mu", "mean", 0.0, -10.0, 10.0); // parameter (name, description,
      value, min, max)
RooRealVar sigma("#sigma", "width", 3.0, -10.0, 10.0);
RooGaussian *gauss = new RooGaussian("gauss", "Gaussian", x, mean, sigma);
```

More PDFs

https://root.cern.ch/doc/master/group__Roofit.html

Data generation and plotting



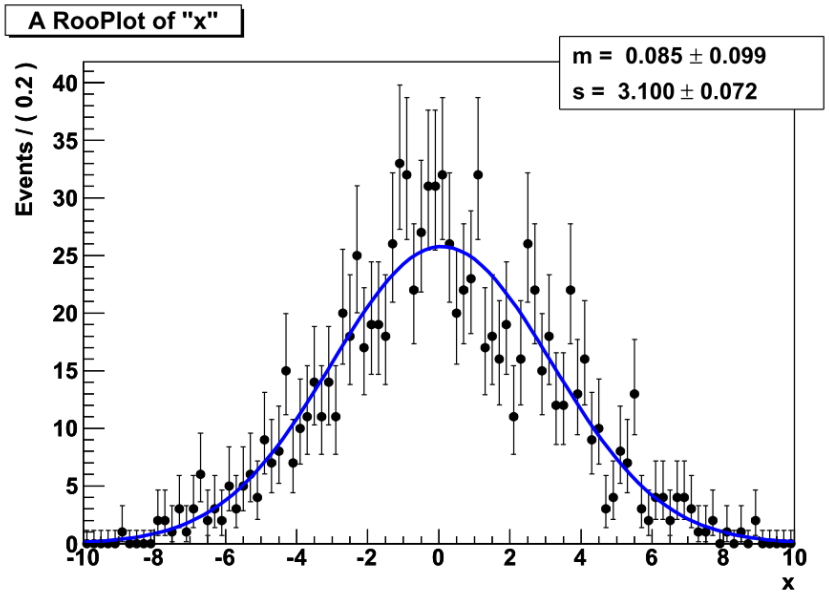


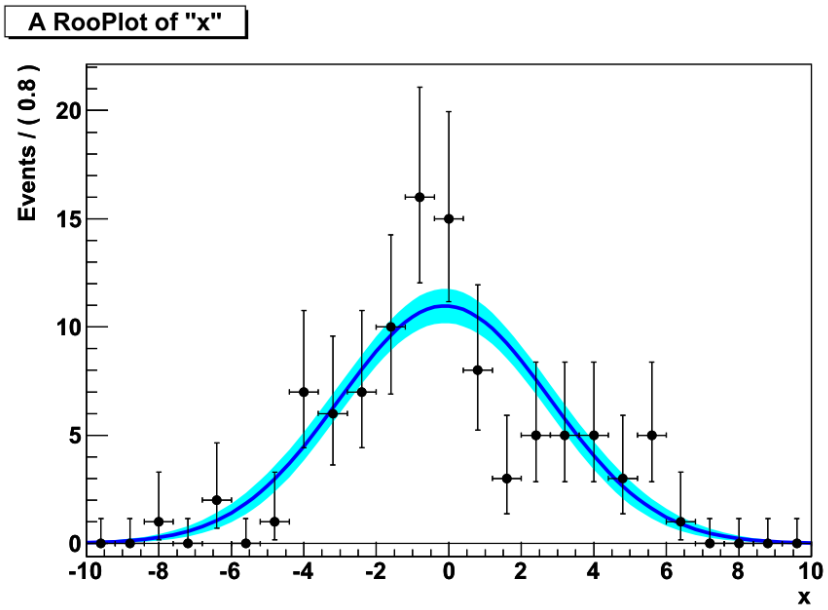
- Very easy to generate MC data and plot
- PDFs are plotted normalized to the data entries in a specified range

Listing 2: Generating data from a Gaussian

```
x.setBins(100); // set number of bins for histogram
RooDataHist *hist = gauss->generateBinned(x, 100000); // generate 100K events

RooPlot* frame = x.frame(); // create frame for plotting
frame->SetTitle("Data");
hist->plotOn(frame);
gauss->plotOn(frame); // normalized to 100K events
frame->Draw(); // automatically creates TCanvas and draws frame contents
```

Fitting





- When fitting, it is important to check the uncertainties including correlations
- Again, very easy to print or visualize - 1 line of code!

Listing 3: Fitting a Gaussian and printing results

```
x.setRange("narrow", -5.0, 5.0);
x.setRange("full", -10.0, 10.0);
RooFitResult *result = gauss->fitTo(*hist, Save()); // Likelihood fit and save the
      result
RooFitResult *result2 = gauss2->chi2FitTo(*hist, Range("narrow"), Save()); // Chi
     ^2 in a declared "narrow" range
// Print results
result->Print();
result->correlationMatrix().Print(); // check correlations!
gauss->plotOn(frame, NormRange("narrow"), Range("full"), VisualizeError(*result,
     1)); // draw 1 sigma uncertainty in the "full" range, but normalized in the "
     narrow", includes correlations!
```

- χ^2/N_{DOF} is a measure of deviation of a model from data: "'goodness"' of fit

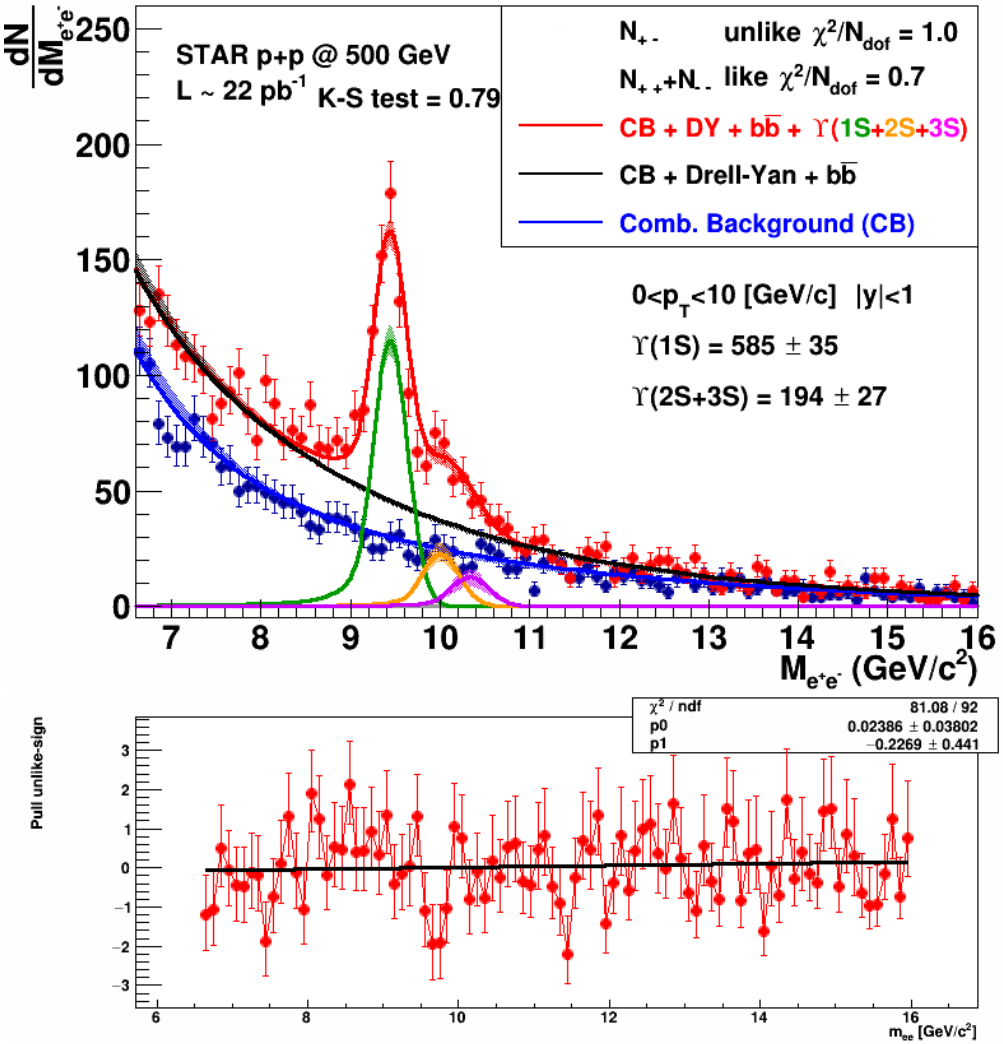
Listing 4: Calculating χ^2

```
total->plotOn(frame, LineColor(kRed), NormRange("full"), Range("chi2")); // Plot
     it last for chi2/ndf calculation in the specified "chi2" range

// Calculate chi2 after extracting the number of parameters - used to calculate
     Ndof
Int_t npar = gauss->getParameters(*hist)->selectByAttrib("Constant",kFALSE)->
     getSize(); // select floating parameters and count their number
Double_t chi2ndf = frame->chiSquare(npar); // calculate chi2/ndf (model and data
     selected from last plotted, but can be specified)

TLatex *latex = new TLatex(); // prepare text in LaTeX format
latex->SetTextSize(0.035);
latex->SetNDC();
latex->DrawLatex(0.25, 0.82, Form("#frac{#chi^{2}}{N_{dof}} = %.2f", chi2ndf)); //
      draw text
```

- Pulls g_i provide information about how well the data points x_i are described by the fit f_i scaled by the uncertainty $\sigma(x)_i$ on the data

$$g_i = \frac{x_i - f_i}{\sigma(x)_i}$$

- Useful for looking for systematic shifts in the fits - indication of problem with model, parameters etc.

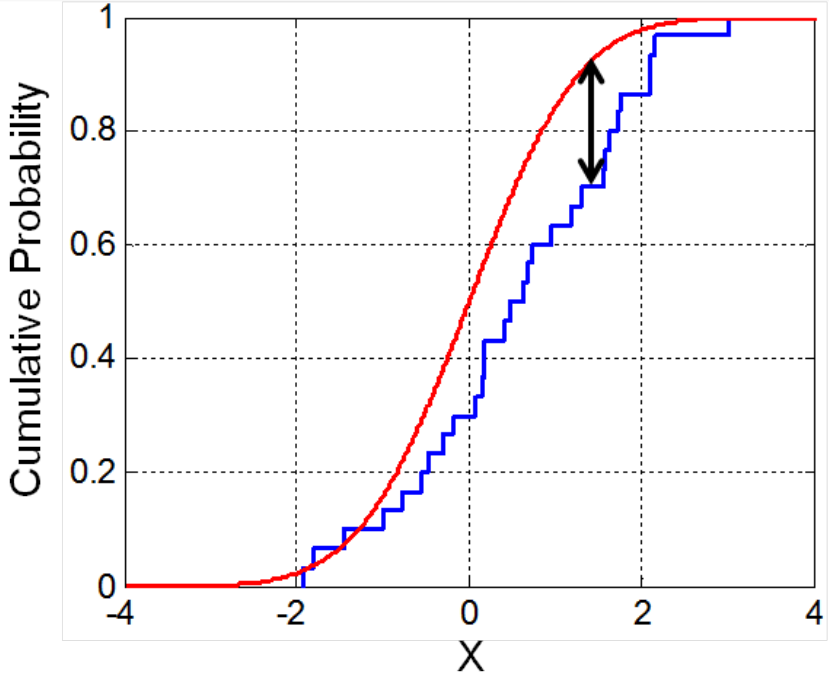


Listing 5: Calculating pulls

```
// specify histogram and curve by their names (look them up in drawn frame)
RooHist* hpull = frame->pullHist("h_hist", "gauss_Norm[m_{ee}]_Range[full]
     _NormRange[full]");
```

Testing - Kolmogorov-Smirnov test

- Provides a probability that the data come from a distribution described by the model
- If probability is:
  - ≈ 1 the distributions are compatible
  - ≈ 0 the distributions are incompatible
- Warning! ROOT implements a 2-histogram comparison. More details:
  - `https://root.cern.ch/doc/master/classTH1.html#aeadcf087afe6ba203bcde124cfabbee4`
  - `https://www.itl.nist.gov/div898/handbook/eda/section3/eda35g.htm`

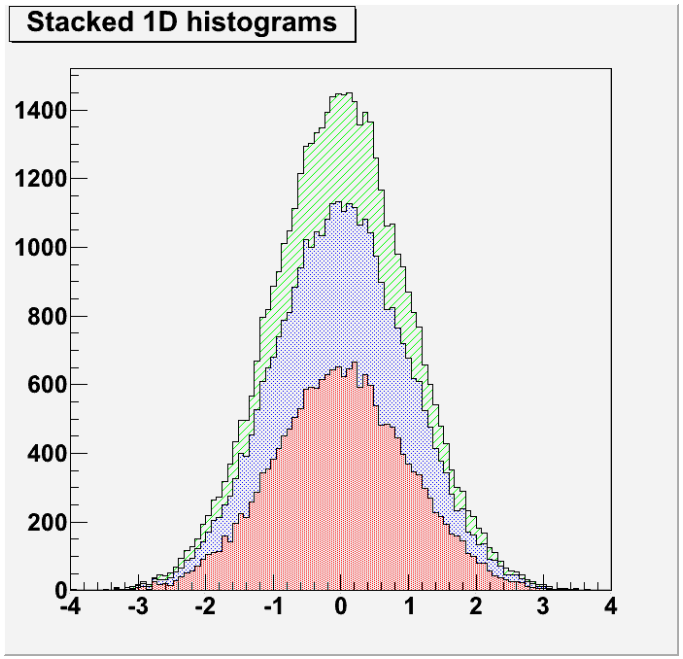


Listing 6: Performing Kolmogorov-Smirnov test

```
// Get ROOT histograms
TH1* h_func = gauss->createHistogram("h_func", x);
TH1* h_data = hist->createHistogram("h_data", x);

double KStest = h_data->KolmogorovTest(h_func);
latex->DrawLatex(0.25, 0.75, Form("K-S test = %.2f", KStest)); // K-S test = 1
     means very high probability of data coming from the distribution described by
     the model
h_data->SetMarkerColor(kRed);

TCanvas *cnv_KS = new TCanvas(); cnv_KS->cd();
h_data->Draw();
h_func->Draw("same");
```

Importing data

- Binned data can be imported from ROOT histograms
- Unbinned data can be imported from ROOT trees

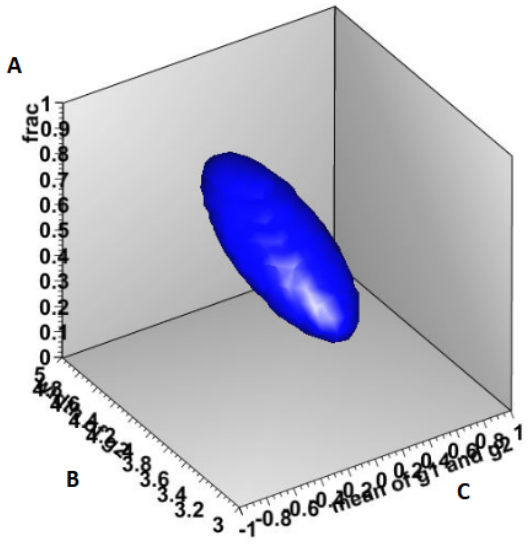


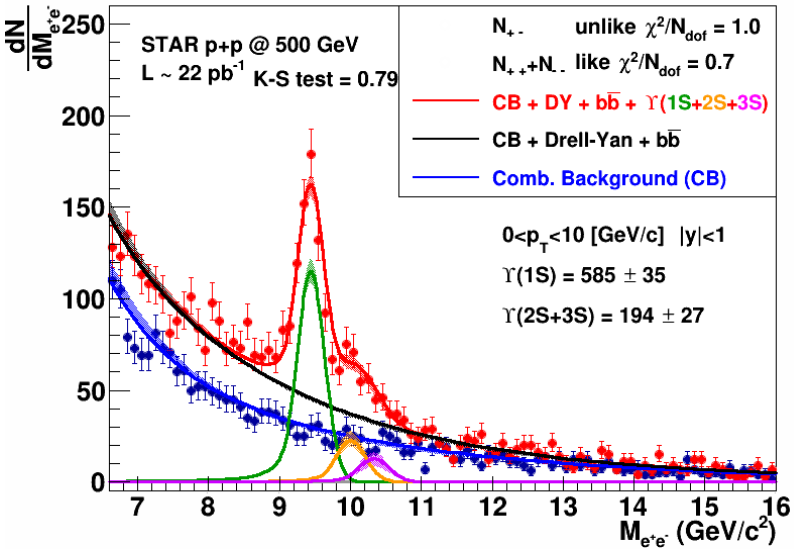
Listing 7: Importing data from ROOT

```
RooDataHist data_hist = new RooDataHist("data_hist", "binned data", x, hist);
RooDataSet data_set = new RooDataSet("data_set", "Unbinned data", Import(*tree));
```

Constrained fits

- Fits can be constrained using PDF of a parameter variable as an external constraint
- Multidimensional PDFs are also possible as constraints
- Such PDF (HessePDF) can be obtained from an initial fit to part of the data (external constraint)
  - For $\geq 2D$ includes correlations between parameters





Listing 8: Using Hesse PDF as an external constraint

```
RooFitResult *result0 = gauss->fitTo(*data, Range("full"), Save()); // Do initial
    fit
RooAbsPdf *paramPDF = result0->createHessePdf(RooArgSet(mean)); // get PDF for
    parameter mean
// Apply external constraints from paramPDF
RooFitResult *result = gauss->fitTo(*data, Range("full"), ExternalConstraints(*
    paramPDF), Save());
```

Composite PDFs

- Simple PDFs can be added
- Allows more complex composite models
- Easy to handle
- Best performance when using recursive fractions f_i
- Recursive fractions have a benefit of being defined relative to each other
- Naturally limited $0 \leq f_i \leq 1$
- For 3 $F_i(x)$ PDF components a sum $S_3(x)$ is:

$$S_3(x) = f_1 F_1(x) + (1 - f_1)(f_2 F_2(x) + (1 - f_2)F_3(x))$$

Listing 9: Composite PDFs

```
RooRealVar sig_frac("sig_frac","sig_frac", 0.4, 0.0, 1.0); // f1
RooRealVar bkg_frac("bkg_frac","bkg_frac", 0.3, 0.0, 1.0); // f2
// Add PDFs using recursive fractions
RooAddPdf *total = new RooAddPdf("total", "signal+background", RooArgList(*gauss,
     *exp, *exp2), RooArgList(sig_frac, bkg_frac), kTRUE);
```

- Fits are normalized to the number of entries in fitting/specified range
- Functions of parameters can be defined an easily calculated
- Uncertainty can be propagated using covariance matrix

Listing 10: Yields and uncertainty propagation

```
Double_t yield_hist = hist->sum(kFALSE); // Get number of entries in the fitting
     range
RooGenericPdf yieldbkg2("yieldbkg2", "background 2 yield from fit", "(1.0-sig_frac
     )*bkg_frac",RooArgSet(sig_frac, bkg_frac));
// Get background 2 yield and uncertainty
Double_t yield_bkg2 = yield_hist*yieldbkg2.getVal();
Double_t yield_bkg2_err = yield_hist*yieldbkg2.getPropagatedError(*result);
```

Exercise

1. Download and run the MC generation macro: generateToyMC.C
2. Copy it and modify in order to perform fitting
3. Make it open the .root output file produced by generateToyMC.C
4. Implement the features discussed in this presentation

Listing 11: Compiling and running macro with ROOT

```
root -l generateToyMC.C+
```

Summary

- Presented features of RooFit
- RooFit provides many useful tools for fitting and testing fits
- An illustrative exercise has been prepared

Thank you for your attention!

BACKUP

Backup 1