Statistics for HEP
Hands-on Tutorial #1

Nicolas Berger (LAPP)

Basics

- **Goal**: implement some practical applications of the methods shown in the lectures.
- **Framework**: RooFit, a package shipped with ROOT.
  - If all is correctly installed, you should be able to
    - Open a terminal
    - Type **`root`** at the command prompt
    - Start entering root commands.
  - In case of problems, please make sure you have
    - a recent version of ROOT (5.3x or later)
    - RooFit included in your ROOT distribution: type "**`RooRealVar v`**" at the ROOT prompt, check for errors
- **Use macros**: *.C files that look like
  - Can be run as **`root mymacro.C`**

```
{
command1;
command2;
...
}
```

Defining Variables and PDFs

- **Variables**
  - `RooRealVar v("v", "", 3);`
    - v=3, fixed
  - `RooRealVar m("m", "", 3, 0, 10);`
    - m=3, can vary in (0,10)

- **PDFs**
  - `RooGaussian g("g", "", x, x0, sigma);`
    - Defines $G(x; x_0, \sigma)$
    - `x, x0, sigma` are RooRealVar's which must have been defined before
  - `RooPoisson p("p", "", n, lambda);`
    - Defines $P(n; \lambda)$

Plots

- **Making a PDF plot:**
  - `p = x.frame();`
    - Defines an empty plot for variable `x`
  - `pdf.plotOn(p);`
    - Plot pdf onto p
  - `p.Draw();`
    - Display p

- **Exercise 1:**
  - Define a variable x with range (-10,10)
  - Define a Gaussian PDF for x with x0=1, sigma=2
  - Make a plot of the PDF
  - Check the mean and RMS using the `pdf.mean(x)->getVal()` and `pdf.sigma(x)->getVal()` functions.

Data

- Generating from a PDF
  - `RooDataSet* d = pdf->generate(x, 1000);`
    - Generate 1000 events of the variable **x**, following the distribution of **pdf**.
    - **x** and **pdf** must have been defined previously
- Creating from scratch
  - `RooDataSet* d = new RooDataSet("d", "", x);`
    - Create the dataset
  - `x.setVal(3); d->add(x);`
    - add the value "3" to the dataset. Repeat as needed
- Plotting data : same as for PDFs

```
p = x.frame();
d->plotOn(p);
p.Draw();
```

Data

- **Exercise 2**
  - Start with the Gaussian PDF created in Exercise 1
  - Generate 10 events in this PDF
  - Plot the data
- **Exercise 3**
  - Same, but generate 1000 events
  - Plot the data and the PDF together:

```
p = x.frame();
pdf.plotOn(p);
data->plotOn(p);
p.Draw();
```

  - Repeat with a Poisson distribution with $\lambda=3$

Likelihood

- Compute a Likelihood
  - `RooNLLVar nll("nll", "", g, *d);`
    - This defines the -log L for the PDF g, applied to dataset d.
    - To compute L, use `exp(-nll.getVal())`
- **Exercise 4**
  - Start again from the PDF from Exercise 1
  - Create a dataset with 1 event at x=1
  - Plot the data and the PDF. Plot the PDF using

    `g.plotOn(p, RooFit::Normalization(100))`

    (with a scale factor of 100), so that it is actually visible.
  - Compute the likelihood
  - Repeat with an event at x=-1, and other values; check if the results work out as expected

Graphs

- Graphs in ROOT can be created as follows:
  - **`TGraph graph(10);`**
    - Define a graph with 10 points (index 0..9)
  - **`graph.SetPoint(0, 5, 8.2).`**
    - Set point 0 to be x=5, y=8.2
    - repeat for the over points
  - **`graph.Draw("AC");`**
    - Draw the graph (Axes and a Curve though the points)

Likelihood Scan

- **Exercise 5**
  - Create a TGraph with 11 points
  - Repeat the setup of Exercise 4:
    - A Gaussian PDF with mean **x0**
      - Make sure x0 can vary between -5 and 5 :

```
RooRealVar x0("x0", "", 1,-5,5);
```

    - A dataset with a single point at x = -1
  - Scan x0 over all integers from -5 to 5 (use a for-loop!)
    - `for (int i=0; i<11;i++) { x0.setVal(i-5); ...`
  - For each point, store the value of λ=-2logL in the graph using `graph.SetPoint(i, x0.getVal(), 2*nll.getVal());`
  - Draw the graph
  - Estimate the MLE for x0, and its 68% CL interval.
    - You can use e.g. `graph.Eval(5.2)` to get the interpolate value of the graph at x0=5.2.

Fits

- Maximum-likelihood fits of a PDF to data
  - **`g.fitTo(*d)`**
    - Adjust the parameters of g to their Maximum-likelihood value in d
    - The parameters must be free to float: make sure they are defined as e.g. **`RooRealVar v("v", "", 3, -5, 5);`** for a variable varying in (-5,5) (with v=3 as initial value)
  - **`g.fitTo(*d, RooFit::Minos())`**
    - Same, but use a more precise estimation of the parameter uncertainties:
      - MINOS uses a likelihood scan
      - HESSE (default) uses a parabolic approximation near the minimum of λ.

Fits

- **Exercise 6**
  - Start with the same setup as Exercise 5
  - Instead of scanning by hand, fit the Gaussian to the data

```
g.fitTo(*d, RooFit::Minos());
```

  - Check the best-fit value and uncertainties on x0:

```
cout << x0.getVal() << endl;
cout <<x0.getError() << endl; // Parabolic error
cout << x0.getErrorLo() << endl;
cout << x0.getErrorHi() << endl;
```

  - Verify that these are the expected results
  - You can also check directly that the 68% CL interval agrees with the results of Exercise 4:

```
cout << x0.getVal()+x0.getErrorLo() << endl;
cout << x0.getVal()+x0.getErrorHi() << endl;
```

Shape analysis

- **More PDFs**
  - `RooExponential e("e", "", x, alpha);` $P(x) = \alpha \cdot \exp(-\alpha x)$
  - `RooAddPdf p("p", "", RooArgList(pS,pB), RooArgList(nS, nB) );`
    - Defines the PDF sum $P(x) = N_S\, P_S(x) + N_B\, P_B(x)$

- **Setup for the rest of the tutorial:**
  - a variable m with range (100, 160)
  - Signal PDF : G(m; mH=125, sigma=1); mH varies in (110, 150)
  - Background PDF: exponential with alpha=-0.02
  - Yields: NS=200 (varies in (0,500)) , NB=10,000 (varies in (0, 50000))

- Implement the setup on your own, or use the prepared version here: **http://nberger.web.cern.ch/nberger/IDPASC/Exercises/shape_setup.C**

Shape Analysis Exercises

- **Exercise 7**
  - Setup the shape analysis, generate 10000 events
  - Plot the data and the PDF, compute the log-likelihood using

    ```
    RooNLLVar nll("nll", "", pT, *d, RooFit::Extended());
    ```
  - Set mH to 110 (`mH.setVal(110)`), redo the plot and likelihood computations, check that the result makes sense.
  - Fit the PDF to the data
  - Print the best-fit mass (`mH.getVal()`) and its error (`mH.getError()`)

Shape Analysis Exercises

- **Exercise 8**
  - Setup the shape analysis, generate 10,000 events as above
  - Scan over mH: first `mH.setConstant();` For each point
    - `mH.setVal(...);`
    - `g.fitTo(*d);` // profile over NS and NB
    - `graph.SetPoint(i, mH.getVal(), 2*nll.getVal());`
  - Estimate the best-fit mH and its error, check with the fit above

    One way to do this is to fit the graph using a quadratic function:

```
f = new TF1("f", "(x - [0])^2/[1]^2 + [2]", 124, 126);
graph.Fit(f, "", "", 124, 126);
```

    and check the fitted values of parameters 0 and 1.

Solutions

- Solutions to the exercises can be found here:

http://nberger.web.cern.ch/nberger/IDPASC/Exercises/exercise1.C

http://nberger.web.cern.ch/nberger/IDPASC/Exercises/exercise2.C

http://nberger.web.cern.ch/nberger/IDPASC/Exercises/exercise3.C

http://nberger.web.cern.ch/nberger/IDPASC/Exercises/exercise4.C

http://nberger.web.cern.ch/nberger/IDPASC/Exercises/exercise5.C

http://nberger.web.cern.ch/nberger/IDPASC/Exercises/exercise6.C

http://nberger.web.cern.ch/nberger/IDPASC/Exercises/exercise7.C

http://nberger.web.cern.ch/nberger/IDPASC/Exercises/exercise8.C