Teaching RooFit

Jonas Rembser (CERN, EP-SFT) for the ROOT team

12 May 2022, ROOT train the trainers

Introduction

- **RooFit**: C++ library for statistical data analysis in ROOT

- It has many components itself:
  - core RooFit libraries, **RooStats**, and **HistFactory**

- Different users interact with it in different ways

- Topic of **today**: *what to consider when teaching RooFit?*
  - **Introduction** to RooFit with motivation
  - The RooFit **ecosystem**
  - **What** parts to teach?
  - Teaching how to **debug**
  - RooFit **documentation**

Why RooFit?

- ROOT function framework can handle complicated functions...
  - ...but requires writing much code
- **Normalization** of pdfs not always trivial
  - RooFit does it automatically
- In complex fit, computation **performance** is important
  - need to optimize code for acceptable performance
  - built-in optimization available in RooFit
    - evaluation only when needed
- **Simultaneous fit** to different data samples
- Provides full model description for **reusability**

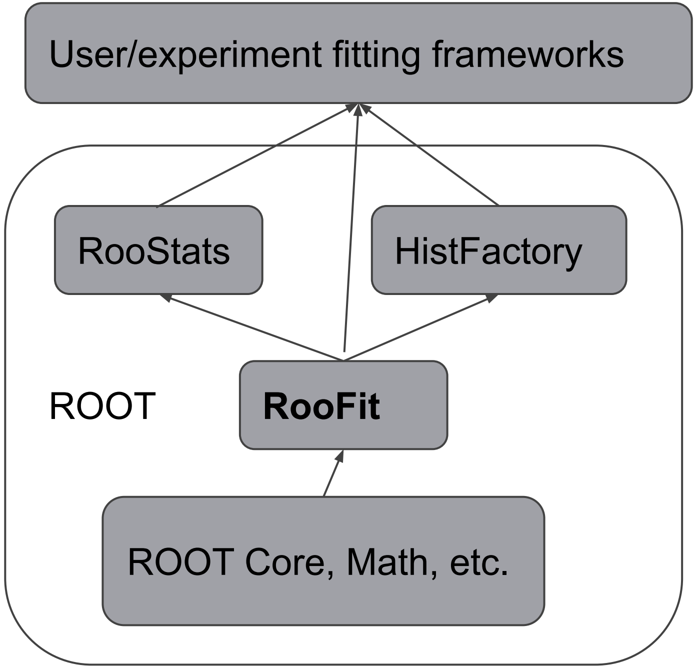
The RooFit ecosystem

It's important to show how the **RooFit ecosystem** looks like

- **RooFit**:
  - model building and fitting to data
- **RooStats**:
  - widely-used statistical procedure
- **HistFactory**:
  - specify complex binned RooFit models

Users from experiments often don't interact much with the RooFit interfaces directly:

- Many **fitting frameworks** built on RooFit



What to teach?

The functionality of the core RooFit libraries is wide:

1. **Model** building
2. **Handling of** binned and unbinned **data**
3. **Toy** dataset **generation**
4. **Test statistic** building and **minimization**
5. Data and model **visualization**
6. The `RooWorkspace` for storing data and models

- Most users use this core functionality and not RooStats/HistFactory
  - It’s better to teach basic RooFit and mention RooStats/HistFactory in passing (depending on the audience)
- Most new RooFit users use **Python** nowadays, so it’s probably better to teach in Python

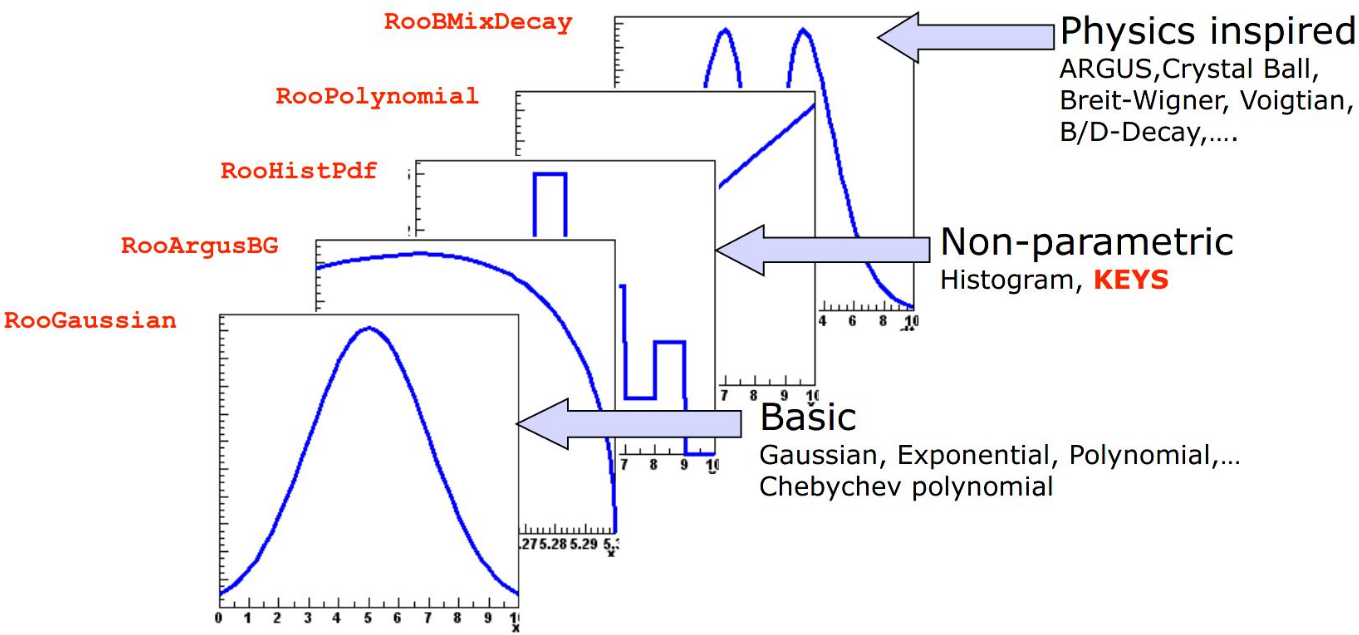
RooFit programming model

Mathematical concepts are represented by C++ objects

Mathematical concept	RooFit class
Variable x	`RooRealVar`
Function $f(x)$	`RooAbsReal`
Pdf $p(x)$	`RooAbsPdf`
Space point $\vec{x}$	`RooArgSet`
Integral $\int_{x_{min}}^{x_{max}} f(x)dx$	`RooRealIntegral`
List of space points	`RooAbsData`

More RooFit building blocks: PDFs

- RooF



Physics inspired
ARGUS,Crystal Ball,
Breit-Wigner, Voigtian,
B/D-Decay,….

Non-parametric
Histogram, **KEYS**

Basic
Gaussian, Exponential, Polynomial,…
Chebychev polynomial

More RooFit building blocks

Besides PDFs, RooFit implements many useful operations for model building:

Operation	**RooFit class**
Addition	`RooAddPdf / RooAddition for functions`
Product	`RooProdPdf / RooProduct` for functions
Convolution	`RooFFTConvPdf`
PDF or function from histogram	`RooHistPdf / RooHistFunc`
Kernel estimation	`RooKeysPdf`
Morphing PDFs for sys. variations	`RooMomentMorph / RooMomentMorphFunc`

RooFit data handling

- Unbinned data can also be imported from **ROOT TTrees**

```
data = ROOT.RooDataSet(“data”, ”data”, x, Import=myTree)
```

  - Imports TTree branch named “x”, all data is converted to double internally
  - Specify a RooArgSet to import multiple observables

- Import from a **text file** of variables (separated by white spaces)

```
data = ROOT.RooDataSet.read(“data.txt”, [x,y])
```

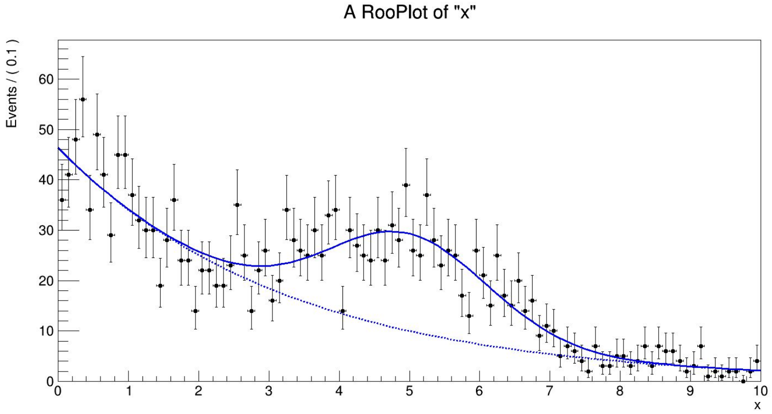
- Binned data can be imported from **ROOT histograms**

```
data =  ROOT.RooDataHist(“data”, ”data”, x, Import=myTH1)
```

  - Imports values, binning definition and bin errors (if defined)
  - Specify a RooArgList of observables when importing a TH2/3.
- Data can be imported/exported from/to **NumPy** and **Pandas** (see this tutorial)
- Data can be imported from **RDataFrame** (see this tutorial)

Toy generation, fitting and visualization

- Typical workflow
  - a pdf with a signal and a background component
  - expected number of events considered in the likelihood (**extended fit**)
  - model building, toy generation, fitting, and plotting
  - even though dataset is binned for plot, the fit is **unbinned**



```
# Observable and parameters
x = ROOT.RooRealVar("x","x", 0.0, 0.0, 10.0)
sigmean = ROOT.RooRealVar("sigmean", "sigmean", 5.0, 0.0, 10.0)
sigwidth = ROOT.RooRealVar("sigwidth", "sigwidth", 1.0, 0.01, 10.)
bkgc = ROOT.RooRealVar("bkgc","bkgc", -0.3, -10.0, 0.1)

# Build a Gaussian pdf and exponential background pdf:
signal = ROOT.RooGaussian("signal","signal",x,sigmean,sigwidth)
background = ROOT.RooExponential("background","background", x, bkgc)

# Construct the added pdf with expected nr. of events for extended fit:
nsig = ROOT.RooRealVar("nsig", "nsig", 200, 0., 10000)
nbkg = ROOT.RooRealVar("nbkg", "nbkg", 600, 0., 10000)
model = ROOT.RooAddPdf("model","model", [signal, background],
                                        [nsig, nbkg])

# Generate a toy MC sample from composite PDF:
data = model.generate(x, 2000)

# Perform extended ML fit of composite PDF to toy data:
model.fitTo(data)

# Plot toy data and composite PDF overlaid:
xframe = x.frame()
data.plotOn(xframe)
model.plotOn(xframe)
model.plotOn(xframe, Components=background, LineStyle=”--”)
xframe.Draw()
```

The RooWorkspace

- RooWorkspace: container of all RooFit objects
  - **full model** with pdfs, functions and variables
  - (multiple) data sets
- possible to save entire model in a ROOT file
- all information is available for further analysis
- possible to join workspaces for **combined fits**
  - common format for sharing physics results
- The RooWorkspace also enables the **factory syntax** the build models, for example for Gaussian pdf:

```
ws.factory("Gaussian::gauss(x[0.,0.,10.],mean[5.,0.,10.],width[1.,0.01.,10.])");
```

  - More details on the factory syntax can be found for example in this presentation or in the RooFit tutorials

Importing and **saving** the model and data from the previous example:

```
RooWorkspace ws("ws", "ws");
ws.import(model);
ws.import(*data);
ws.writeToFile("myWorkspace.root");
```

Tree **printing** mode (`ws.Print("t")`) of **workspace** reveals model structure:

```
variables
---------
(bkgc,nbkg,nsig,sigmean,sigwidth,x)

p.d.f.s
-------
RooAddPdf::model[ nsig * signal + nbkg * background ] = 0.750001
  RooGaussian::signal[ x=x mean=sigmean sigma=sigwidth ] =
3.72665e-06
  RooExponential::background[ x=x c=bkgc ] = 1

datasets
--------
RooDataSet::modelData(x)
```

C++/Python or factory language?

- RooFit models can be built either:
  - directly from RooAbsArg objects in **C++/Python**:
    - more concise and benefits from type system of programming language
  - inside a RooWorkspace with the **RooFit factory language**
    - more expressive, but everything happens inside strings
- When teaching RooFit, try to not mix both ways too much

```
# building Gaussian PDF from objects:

x = ROOT.RooRealVar("x", "x", 5.20, 5.30)
mean = ROOT.RooRealVar("mean", "mean", 5.28, 5.20, 5.30
sigma = ROOT.RooRealVar("sigma", "sigma", 0.0027, 0.001, 1.)
gauss = ROOT.RooGaussian("gauss", "gauss", x, mean, sigma)

# building Gaussian PDF with factory language:

ws = ROOT.RooWorkspace("ws", true)
ws.factory("Gaussian::gauss(x[5.20,5.30], mean[5.28,5.2,5.3], sigma[0.0027,0.001,1])")
```

RooFit pythonizations

- PyROOT bindings **more pythonic** in *6.26*
- Now you can for example:
  - use **Python keyword arguments** instead of RooFit command arguments
  - pass around **Python sets or lists** instead of `RooArgSet` or `RooArgList`
  - pass **Python dictionaries** to functions that take **`std::map<>`**
  - implicitly convert floats to `RooConstVar` in `RooArgList/Set` constructors
- All pythonizations are [documented]()
- Some Pythonizations to help with C++/Python lifetime issue
  - Still there are memory leaks when returning owning pointers
- *See also this [ROOT meeting presentation]()*

Example code from the [rf316_llratioplot.py]() tutorial showcasing the pythonizations:

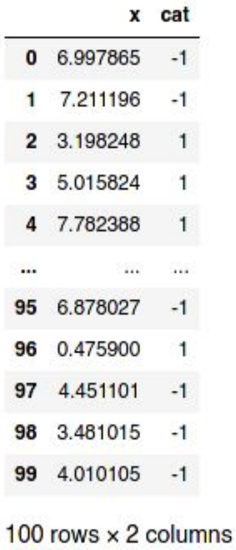
```python
# Create background pdf poly(x)*poly(y)*poly(z)
px = ROOT.RooPolynomial("px", "px", x, [-0.1, 0.004])
py = ROOT.RooPolynomial("py", "py", y, [0.1, -0.004])
pz = ROOT.RooPolynomial("pz", "pz", z)
bkg = ROOT.RooProdPdf("bkg", "bkg", [px, py, pz])

# Create composite pdf sig+bkg
fsig = ROOT.RooRealVar("fsig", "signal fraction",
                       0.1, 0., 1.)
model = ROOT.RooAddPdf("model", "model",
                       [sig, bkg], [fsig])

data = model.generate((x, y, z), 20000)

# Make plain projection of data and pdf on x observable
frame = x.frame(Title="Projection on X", Bins=40)
data.plotOn(frame)
```

RooFit with NumPy, Pandas, and RDF

- ROOT *v6.26* **new converters** between NumPy arrays/Pandas dataframes and **RooDataSet/RooDataHist**
    - No translation from RooDataHist to dataframe because histograms are in general multi-dimensional
    - Tutorial in Python
- New `RooRealVar.bins()` function to get RooFit **bin boundaries** as NumPy array
- Creating **RooFit datasets** from **RDataFrame**
    - Works for both `RooDataSet` and `RooDataHist`
    - Weighted filling still needs to be implemented
    - Tutorial in C++ and Python

Example of exporting RooDataSet to Pandas:

```
from ROOT import RooRealVar, RooCategory, RooGaussian

x = RooRealVar("x", "x", 0, 10)
cat = RooCategory("cat", "cat",
                  {"minus": -1, "plus": +1})

mean = RooRealVar("mean", "mean",
                  5, 0, 10)
sigma = RooRealVar("sigma", "sigma",
                   2, 0.1, 10)

gauss = RooGaussian("gauss", "gauss",
                    x, mean, sigma)


data = gauss.generate((x, cat), 100)

df = data.to_pandas()
```



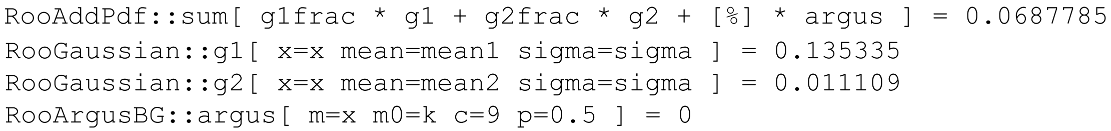
	x	cat
0	6.997865	-1
1	7.211196	-1
2	3.198248	1
3	5.015824	1
4	7.782388	1
...	...	...
95	6.878027	-1
96	0.475900	1
97	4.451101	-1
98	3.481015	-1
99	4.010105	-1

100 rows × 2 columns

Teaching how to debug

It's important to teach **how to debug** models and fits.

- Remind to always **read logs** (especially errors and warnings)
- Explain the **message logger** (like in this tutorial)
- Options to get more minimization output (`PrintLevel()` in `RooAbsPdf.fitTo()`)
  - Finding fit convergence problems is a whole lecture in itself
- Two important methods to print model structure:
  - `model.Print(“t”) # “t” for “tree”, also “v” for “verbose” is useful`
  - `workspace.Print() # to get all the content in a RooWorkspace`



```
RooAddPdf::sum[ g1frac * g1 + g2frac * g2 + [%] * argus ] = 0.0687785
RooGaussian::g1[ x=x mean=mean1 sigma=sigma ] = 0.135335
RooGaussian::g2[ x=x mean=mean2 sigma=sigma ] = 0.011109
RooArgusBG::argus[ m=x m0=k c=9 p=0.5 ] = 0
```

Example of the “pdf” section in a RooWorkspace printout

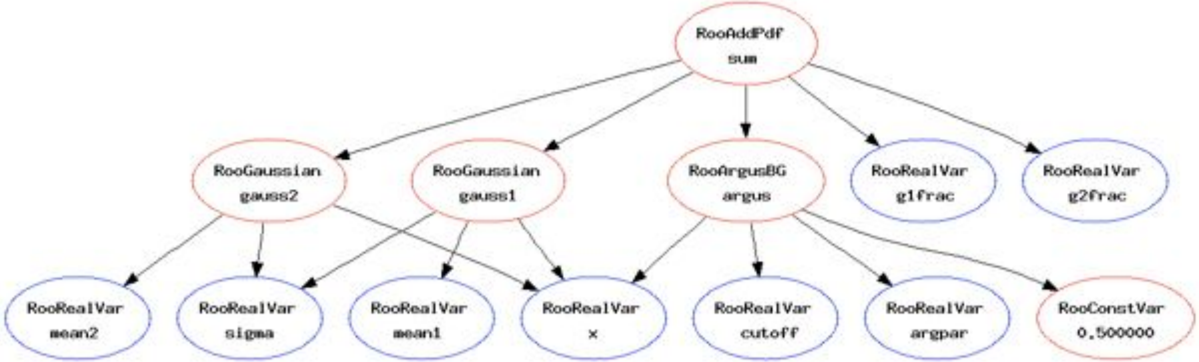
RooFit Model Visualization

- Model visualization can be also useful for debugging

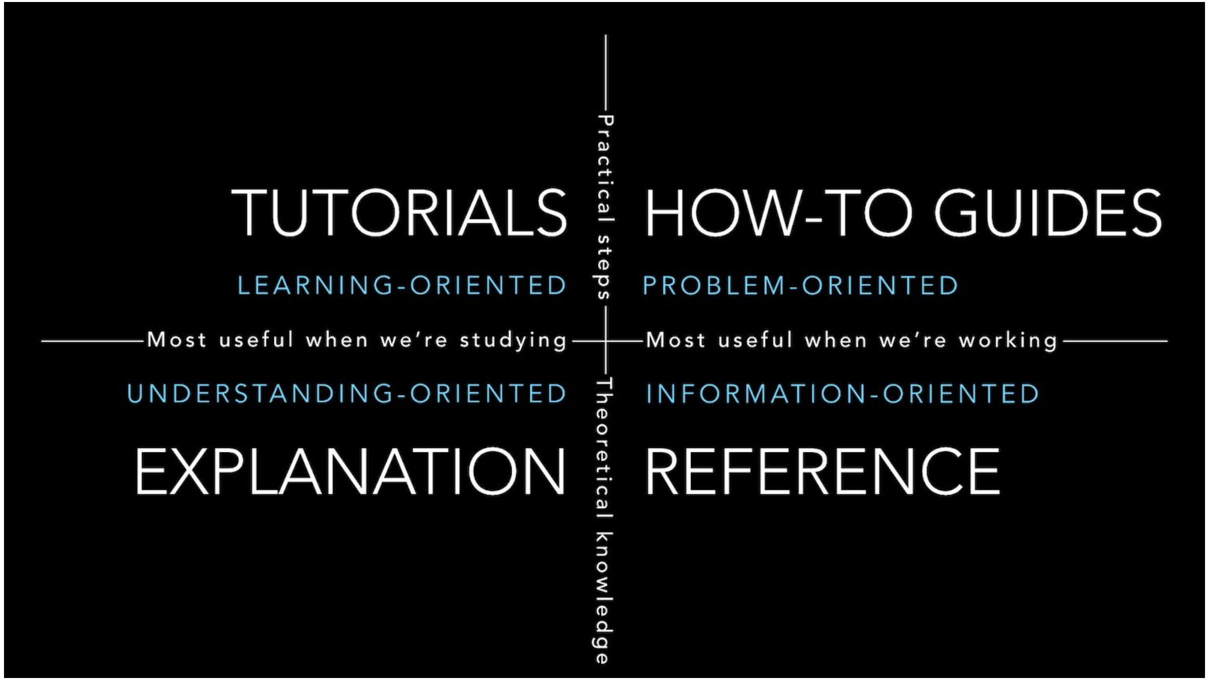
- GraphViz visualization of RooFit models:
  - `model.graphVixxTree(“model.dot”)`

- The dot file can be converted to `.png` file:

```
dot -Tgif -o model.gif model.dot     # Directed graph

fdp -Tgif -o model_fdp.gif model.dot # Spring balanced model
```



The unified theory of documentation



TUTORIALS

LEARNING-ORIENTED

HOW-TO GUIDES

PROBLEM-ORIENTED

Practical steps

Most useful when we're studying

Most useful when we're working

Theoretical knowledge

UNDERSTANDING-ORIENTED

EXPLANATION

INFORMATION-ORIENTED

REFERENCE

graphics from https://documentation.divio.com/

RooFit documentation

- **How-to guides** *(problem-oriented)*:
  - **ROOT tutorials**: **RooFit** (C++ and Python), **RooStats** (C++), **HistFactory** (C++)
- **Tutorials** *(learning-oriented)*:
  - The **RooFit manual** on the **ROOT website**
  - RooFit tutorial from the **CMS** data analysis **school**
- **Explanation** *(understanding-oriented)*:
  - The RooFit **quick start** guide
  - **RooStats** users' **guide**
- **Reference** *(information-oriented)*:
  - The RooFit sections in the **ROOT reference guide** (doxygen)
  - The **RooFit** users' **manual**
  - **HistFactory** manual
  - "*Practical statistics for the LHC*" (as reference for methods implemented in **RooStats**)

Summary

- RooFit is a component of ROOT with sub components (like **RooStats** and **HistFactory**)

- RooStats and HistFactory are less used than the core RooFit libraries
  - Training should focus on model building, data handling, fitting and visualization

- The new **pythonizations** can make RooFit more accessible in Python
  - Students can use RooFit with things they are familiar with (e.g. NumPy arrays)

- Important to teach how to **debug** your model and fits

- There is plenty of **documentation** available, especially how-to guides and references

- Don’t forget to mention the RooFit/RooStats topic on the **ROOT forum** as the best address to get help!