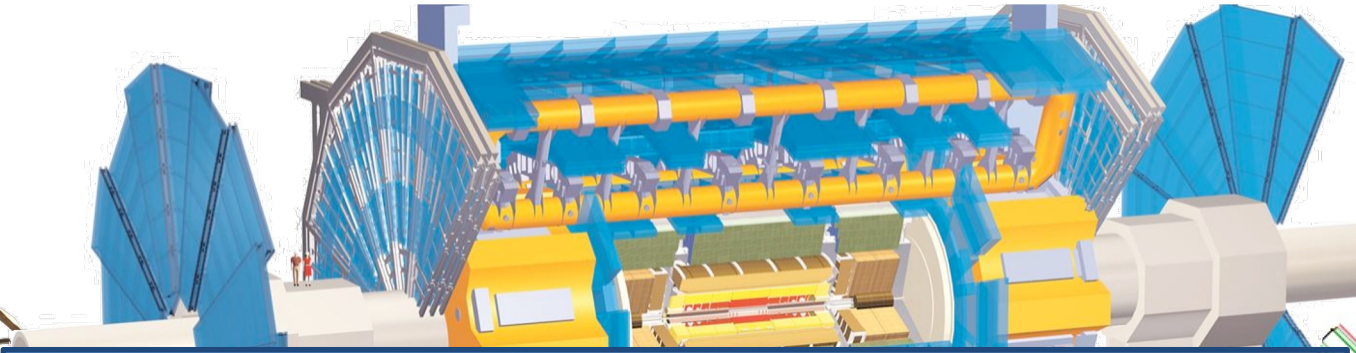
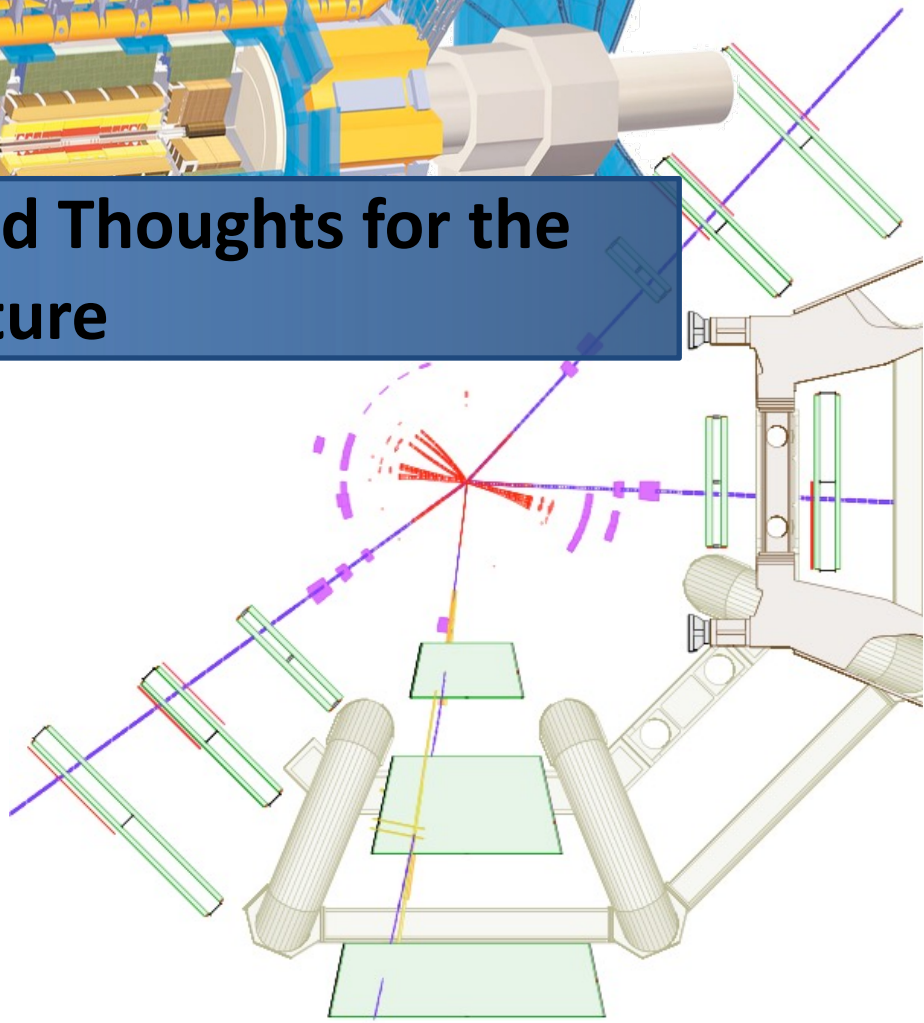


xRooFit: Status and Thoughts for the Future

Will Buttinger



Introduction

What is xRooFit?

- A high-level API for RooFit, to assist with creation, inspection, modification, and analysis of workspaces
  - xRooFit is to RooFit as Keras is to Tensorflow
  - Included in ROOT 6.30 under ROOT::Experimental::XRooFit namespace
- Toolkit-agnostic: should work with all workspaces, regardless of how they were made
- Rule: xRooFit does not have any custom classes that are persistified
  - Everything has to be done through ROOT/RooFit/RooStats classes
  - xRooFit classes 'wrap' over these classes to provide the additional functionality
  - Result is xRooFit can be used at any stage of a workflow without forcing dependency either upstream or downstream
- Bonus: Provides a GUI (the xRooBrowser) to much of the functionality

Purpose of this presentation

- Show you some of the things you can do with xRooFit
- Share with you some of my thoughts about future developments
- **Hear your thoughts about how xRooFit could help you and your users**
  - **In particular: keen to try opening your workspaces and see what happens**

Newest features

```
from ROOT.Experimental import XRooFit as XRF
w = XRF.xRooNode("/path/to/workspace.root")
```

- Channel yield plots

```
w["simPdf"].Draw("x=channelCat")
```

- Hybrid dataset generation (asimov in some channels, obsData in others)

```
ds = w["simPdf"].reduced("*050*").generate(fr="",expected=True) # asimov dataset of current state of *050* channels
ds.Add(w["simPdf"].reduced("*050*",invert=True).datasets()["obsData"])
```

- Improved progress tracking during lengthy fits

```
fr = w["simPdf"].nll("obsData").minimize()
```

- Overlaying alternative model states

```
w["simPdf"].Draw("eauxRatio")
w.SetFitResult("") # restores prefit state
w["simPdf"].Draw("overlayModel Prefit")
```

- Impact plots (using hessian approximation) for fit results

```
fr.Draw("impact:parName") # draw impact for parameter "parName"
```

- PLR scans and automated CLs limit scanning

```
hs = w["simPdf"].nll("obsData").hypoSpace("poiName")
hs.scan("cls")
print(hs.limits())
```

Future development thoughts

- Would like to improve management of NLL Options and Fit Options
  - NLL Options: control aspects of exactly what objective function is created
    - Offsetting, Const-optimization, binnedLikelihood mode, range, …
    - Currently held in a RooLinkedList
  - Fit Options: control aspects of the minimization
    - Strategy sequences, Hesse and/or Minos activation, print level, max iterations, …
    - Extra options that are xRooFit-specific, such as how frequently to report fit progress, or size of the logging buffer when saving logs to fit result
    - Currently using a ROOT::Fit::FitConfig for this, which contains a generic MinimizerOptions() for adding arbitrary settings
    - For a number of reasons, this class has never felt quite right for this use case
  - Vague idea: ModelConfig (currently almost entirely ignored by xRooFit) could become the object that is used to fully define the objective function, and we think of a suitable object for holding the Fit options
- FitResults should be able to hold metadata
  - Currently xRooFit abuses the constPars list for this (adds extra things)
  - Could then store the ModelConfig and FitOptions names (or even pointers?), along with e.g. logging output

Future development thoughts

- Storage of scan results should also be improved
  - Currently using a RooStats::HypoTestInverterResult, which can hold collections of HypoTestResults
  - But HypoTestResult's don't have dedicated capability to store RooFitResults
    - Again, xRooFit does some abusive things to store partial or full fit results through them

- Add support for Chi2 objective function (currently only NLL function supported)
  - Could have a generic method for creating any type of function

- Would love to make a proper dedicated GUI rather than the current approach of figuring out how to hijack/utilise existing TBrowser GUI
  - Just not enough capacity to work on this
  - And the GUI is ultimately not the primary interface to xRooFit, despite it being very useful to quickly get familiar with someone's workspace (as well as showcasing xRooFit functionality)

- If nothing else, I will work on adding documentation to the ROOT webpages, as well as the dedicated readthedocs I have been working on

Backup

What is xRooFit

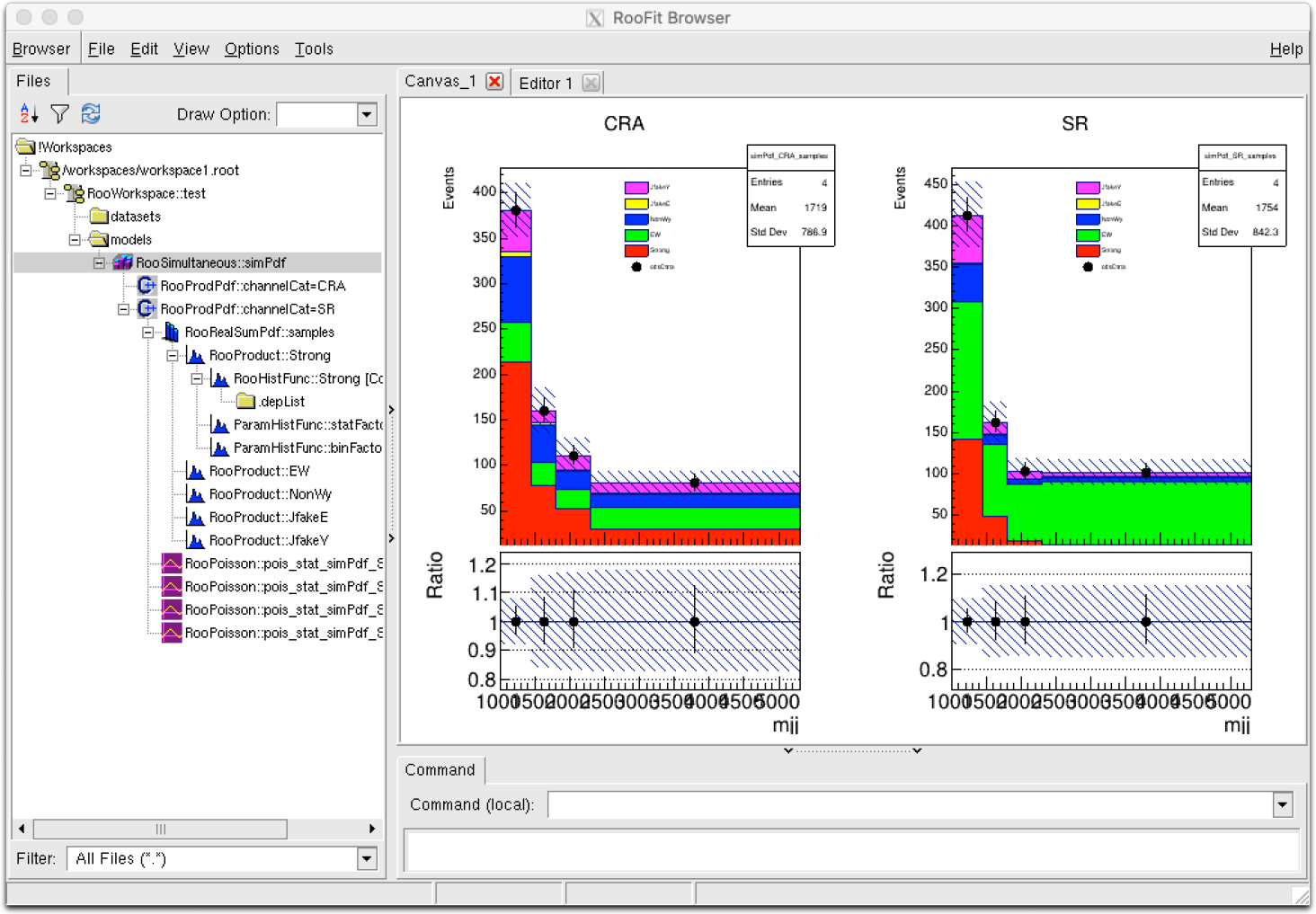
- Attempt at a user interface for RooFit that I could use to add functionality I wanted without creating custom classes that had to be persistified.
  - My previous creation, TRooFit, involved defining a bunch of RooFit classes (RooAbsPdf) that had ROOT-like interface (SetBinContent, Draw, etc).
  - For xRooFit I created a “wrapper” class, xRooNode, that has methods that behave according to the object they are wrapping.

- xRooNodes can have child xRooNodes (often are servers in the case of RooAbsArg)

- Soon realized it was possible to add an xRooNode to ROOT’s TBrowser and have it be browsable …

Demo….

- You should be able to run yourself with the StatAnalysis releases, available e.g. on docker: `docker run -it -e DISPLAY=host.docker.internal:0  gitlab-registry.cern.ch/atlas/statanalysis:xroofit`



Functionality in xRooFit

- Workspace building
  - Can create/modify models and datasets

- Graph Modifiers: Methods that alter the 'graph' representing the likelihood function
  - `Add(...)`
  - `Multiply(...)`
  - `Vary(...)`
  - `Constrain(...)`
  - `Remove(...)`
  - `Combine(...)`

- Object Modifiers: Modify the object that the node wraps (or potentially one of the objects of the child nodes)
  - `SetBinContent(bin, value [,parName, parVal] )`
  - `SetBinError(bin, value)`
  - `SetBinData(bin, value [,dsName])`
  - `SetXaxis(name,title,nBins,low,high)` : fixed bin widths
  - `SetXaxis(name,title,nBins,bins)` : variable bin widths

- Model navigation
  - Finding out the nodes related to another node

- Related nodes: these methods return the collection of nodes related to this node in some way:
  - `components()` : the nodes that "add" together to make this node
  - `factors()` : the nodes that "multiply" together to make this node
  - `variations()` : the nodes that are "varied" (interpolated) between to make this node
  - `constraints()` : the nodes that "constrain" this node (relevant for parameter nodes)
  - `datasets()` : the nodes that represent data corresponding to this node (relevant for pdf nodes)

  - `deps()` : the fundmanental (leaf) nodes that this node depends on (=obs+pars) [note: will replace this with `vars()` in future]
  - `obs()` : the leaf nodes that are observables (robs+globs)
  - `globs()` : the leaf nodes that are global observables (subset of observables)
  - `robs()` : the leaf nodes that are regular observables
  - `pars()` : the leaf nodes that are parameters (i.e. not observables)
  - `floats()` : the parameters that are not constant and so would float in a fit
  - `args()` : the parameters that are currently constant [note: may replace this with `consts()` in future]

  - `coefs()` : Return the coefficients (if any) that multiply this node given its inclusion in its parent (these are distinct from factors because coefs are not children of this node - they are a bit like a context-dependent factor).
  - `coords()` : Return the observables with their values that this node corresponds to (e.g. if the node is a channel, the coords() will be the channelCat with its value set to this channel)

- Workspace inspection
  - Printing and visualizing contents, extracting yields etc

- Inspection methods: tell you about the node and move to related nodes
  - `Print([option])`: lists the child nodes (components/factors/variations) of a node. Use "depth=X" where X is a number as the option to control depth
  - `Draw([option])`: Visualize the node. Option can control what is visualized depending on the type of node. Some examples:
    - E : adds error bars (based on the currently loaded fit result)
    - RATIO : adds a ratio pad
    - SIGNIFICANCE : adds a significance pad
    - PULL : adds an interactive pull plot (to investigate parameter dependencies)
  - `Browse()`: open the node in an Browser window for interactive exploration.

  - `find("name")` or `operator[]("name")`: return child with given name. Name can be in the form of a path to navigate quickly e.g. "modelName/channelName/sampleName".
  - `reduced("list,of,regex")`: for certain nodes this can return a subset shallow-copy of the node e.g. a node with some of the samples of a channel.

  - `GetBinContent(bin)`: return the bin value of this node
  - `GetBinData(bin[,dsName])`: return bin value of dataset of this node (equivalent to `datasets()[dsName].GetBinContent(bin)`)
  - `GetBinError(bin[,fitResult])`: get the error in given bin, using the covariances in the optionally provided fit result (returns uncorrelated error calculation otherwise using the currently loaded parameter errors, unless `SetFitResult` has been called).
  - `IntegralAndError([fitResult])`: return the integral and error (as a pair) of the current node, calculating the error with the given fitResult covariance matrix if provided (returns uncorrelated error calculation otherwise, unless `SetFitResult` has been called).
  - `GetXaxis()`: returns a `TAxis` for the x-axis observable of this node, if relevant.

- Model Fitting
  - NLL construction, minimization, dataset generation

    - Fitting:
      - `nll("dataset" [, {options}])`: create NLL using the given dataset. This `xRooNLLVar` object has several special methods:
        - `minimize()`: returns a FitResult (results can be cached to a TFile).
        - `generate([expected])`: generate a toy or asimov dataset.
      - `SetFitResult(fitResult)`: load a fit result into a model: all parameter values are set to final values and covariance matrix will be used for calculating errors.

Lessons Learned

- The primary types of object in a statistical analysis are:
  - Models (RooAbsPdf): functions of variables (obs and pars) representing a PDF
  - Datasets (RooDataSet): observed or generated, toy or Asimov in the latter case
  - FitResults (RooFitResult): hold a “model state” (set of parameter values along with covariances between the parameters)
    - Doing various hacky things to cram more info into FitResults at the moment
  - NLLOptions (RooLinkedList): specialized options used in NLL function construct
  - FitConfig (ROOT::Fit::FitConfig): minimization hyperparameters
- Terminology:
  - vars: obs (globs and robs), pars (floats and consts)
- Would be nice if HistFactory copied over more metadata:
  - Things like using histogram axis titles to set obs titles, a way to give channels a title, etc. – all the RooFit objects can already carry this sort of thing.
  - Consider using var attributes to flag the obs and pars rather than the ModelConfig (how likely is it that a HistFactory model will be used with its obs switched to pars or vice versa?)

Future plans for xRooFit

- Would very much support adding functionality from xRooFit into ROOT
  - But deciding which bits could be contentious, factorizing things may be possible, not sure….
  - Also how to add it – part of me thinks the functionality should go into the classes xRooNode has been wrapping, but that might be changing RooFit too much, so perhaps keeping it all in a wrapper class is most straightforward
- But conscious that xRooFit's development has been somewhat organic and in some areas its still evolving/changing
  - Certainly things I would consider doing differently, would want to talk it through with others
- Would like to have some more people using it, as that helps figure out what functionality is good and what needs more work.
  - StatAnalysis releases are there to help distribute this functionality while its in this state of development.