technische universität
dortmund

HS^3 - Past, present and future

The Big Picture

- Especially in HEP, ROOT has been the dominant modeling ecosystem for 2 decades
  - default format: RooWorkspace
  - essentially, RooFit is a DSL for statistical modeling
  - old-school C++: object oriented, inheritance-based (RooAbsPdf, RooAbsReal, …)
  - big benefit: pretty feature-complete, highly efficient, serializable
  - big drawbacks: not purely descriptive, serialization to exclusive binary format
- Recently, some new “contenders” have emerged
  - pyhf: stacks of histograms in python, serializable to JSON
  - zfit & juliahep: heavy on analytical modeling, very elegant, but no serialization

Within ATLAS: Endavour to perform apples-to-apples-comparison pyhf vs. RooFit

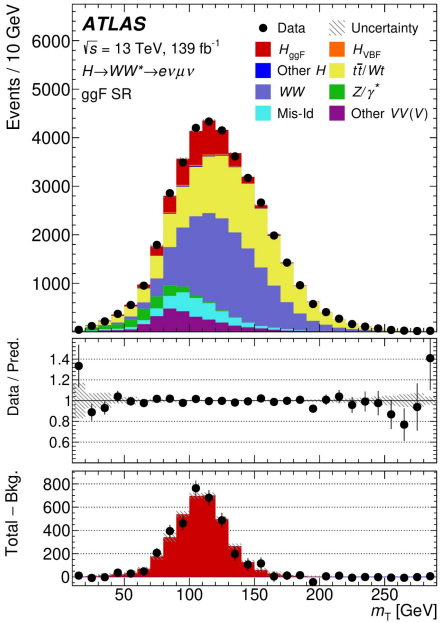
- do the efficiency claims hold up?
- can we use it practically also for larger models?

Where we came from

- RooFit philosophy: Likelihood is a computational graph
  - variables & parameters are leaves
  - operations are branching points
  - the likelihood is the trunk
- RooFit philosophy: Everything has a name
  - variables (parameters) are correlated by name
  - variables (observables) can be matched to dataset by name
  - merging (combination) of models easy
  - drawback: no inherent support for vector-valued parameters (1 name = 1 number)
- pyhf paradigm: The model is static
  - purely descriptive JSON syntax
  - high emphasis on schema validation
- pyhf paradigm: keep it simple
  - one likelihood and one dataset per file
  - limited scope: histfactory

What is HistFactory?

- A Likelihood is a simultaneous function within several orthogonal binned channels
- Each channel is a stack of histograms, mapping to one data histogram, all sharing the same binning
- The Pdf in each bin is a poisson estimate
- Each histogram in the stack can be subject to a set of modifiers
  - correlated shape changes [$\times(1+\varepsilon_i\theta)$]
  - uncorrelated shape changes [$\times\theta_i$]
  - normalization factors [$\times\theta$]
  - additive normalization changes [$\times(1+\varepsilon\theta)$]



Requirements for HS^3

- Purely descriptive language
  - no for-loops, no order-sensitive statements
- Feature complete with respect to ROOT
  - full support of all binned and unbinned types of functions, pdfs and datasets
  - support for multiple pdfs & multiple datasets per file
  - round-trip-capable with ROOT Workspace (retain names, default values, …)
  - corollary: no hard distinction between “parameters” (floating variables) and “observables” (variables contained in the dataset)
- Close to pyhf JSON
  - should be trivially constructible from pyhf JSON

Design decisions

- Embrace “computation graph” logic
  - distributions can be composed of distributions or functions
  - functions can be composed of functions
  - distributions and functions can depend on variables
- Embrace “reference by name”
  - all objects need a unique name (function name, distribution name, variable name, …)
  - allow easy transformation from and to ROOT
  - for now, no vector-valued parameters (complicated to handle, but not impossible)
- Allow “function macros” (“high-level” distribution and function types)
  - do not restrict to basic operators
  - allow frameworks to implement relatively complicated distributions / functions
  - possibility to allow for optimization
  - allow more compact representation

Shortcomings of the current draft

- no vector-valued parameters
- no stringent handling of “auxiliary measurements” (constraint terms)
- no explicit dependency tracking (leads to implicit order requirement)
- no real governance: currently, just me doing my best to be nice
  - do we need a committee? a project lead?
  - RFCs? release schedule?
- no full implementation outside of ROOT
  - julia draft exists, but not super actively developed
  - pyhf draft promised, but nothing produced so far