Making data on HepData re-use friendly

CEDAR Session
MCnet Workshop April 10 2018

NEW YORK UNIVERSITY

HepData is the established data interface between hep-ph and hep-th

The re-launch of HepData (https://hepdata.net/) allows us to re-assess what experiments store on HD and how to best store it there.

Historically, HepData started with digitized versions of tables in papers, but gradually moves to more complete information (beyond the publication)

Coincides with a move from storing mainly SM data to more searches: HD as the place to obtain structured data for a given published expt result.

We should think of HepData records not only as a destination / archive of data, but make records useful for non-collaboration physicists — reinterpretation is a perfect use-case.

- move beyond table data structures
- store (simplified) likelihoods
- cross-link third-party resources (Rivet, CERN Analysis Preservation)

For many analyses, most of the information that is ultimately put on HepData is encoded in HistFactory configurations

- observed data distributions for signal and control regions
- signal and background distributions and their variations under nuisance parameters
- pre- and post-fit distributions are accessible.

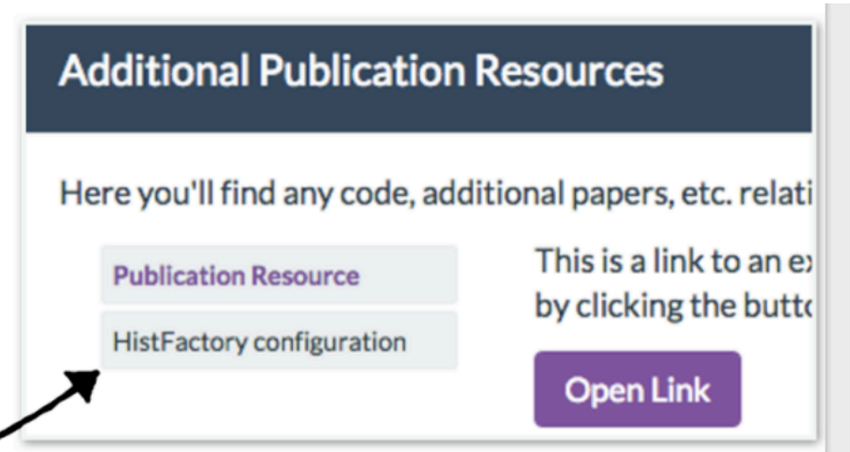
HistFactory is very structured. We can use it to systematically prepare HepData content. Two approaches:

1. produce HepData tables directly from HistFactory workspaces
2. store complete (perhaps simplified) HistFactory configurations on HepData as an additional resource to allow third parties to build new models

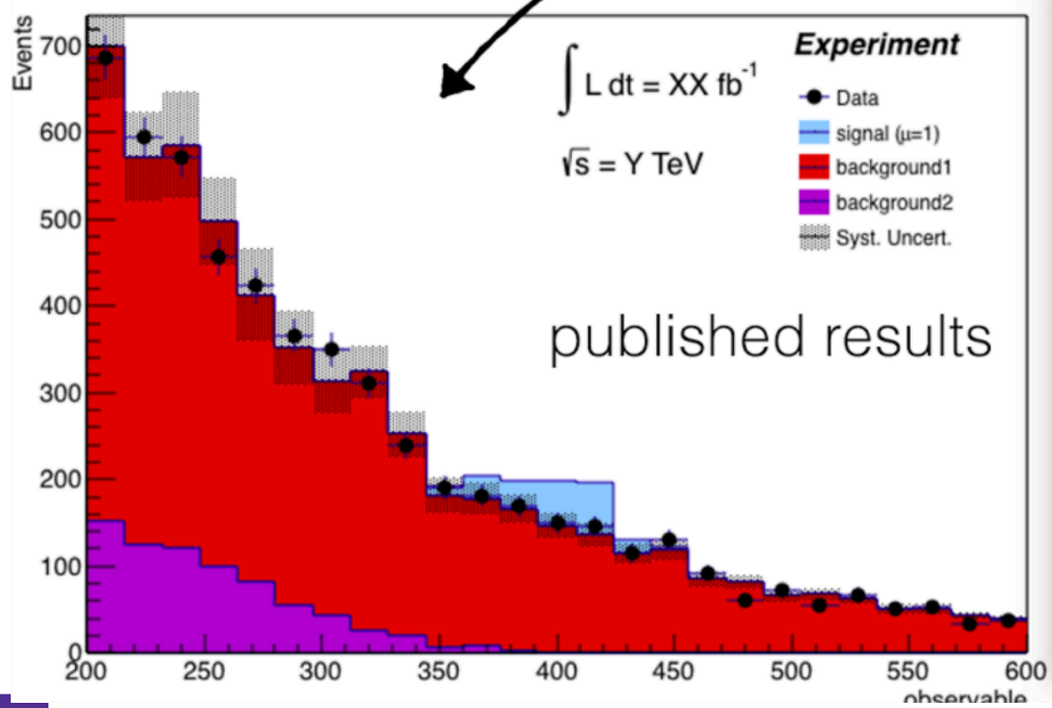
1. produce HepData tables directly from HistFactory

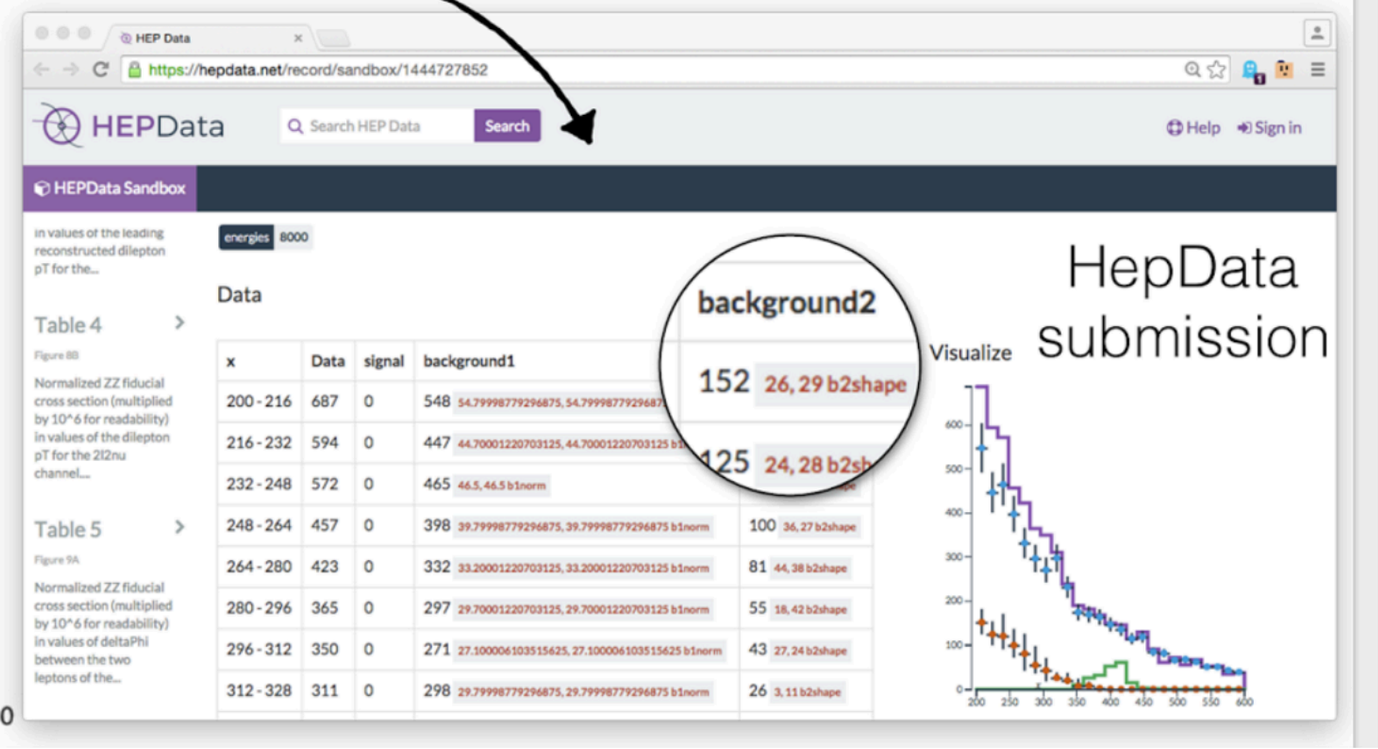
python tool (hftools) in development to prepare HD tables straight from HistFactory workspaces:

- outputs new YML format
- configurable w.r.t. which variations to include
- variations taken as differences between parameter value sets (“snapshots”)



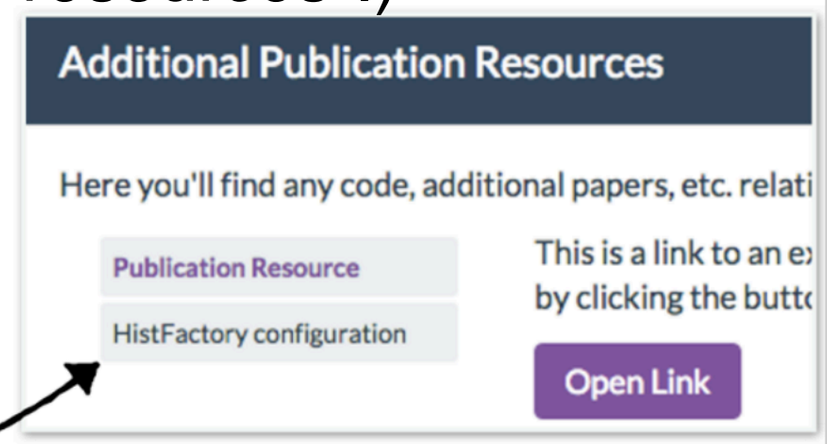
HistFactory



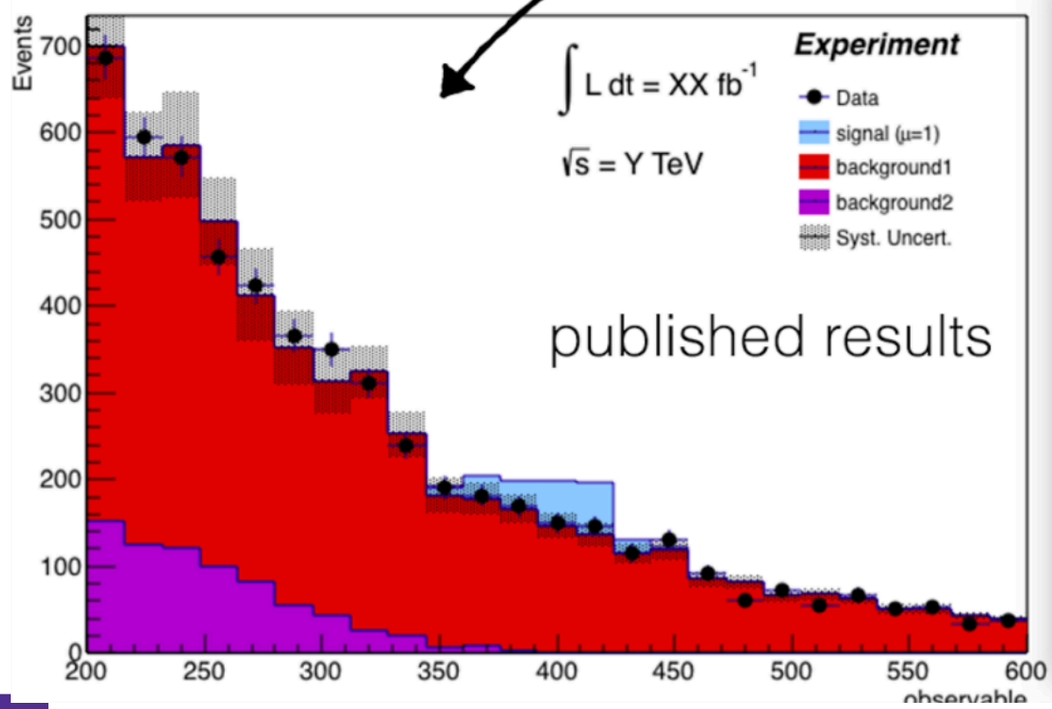


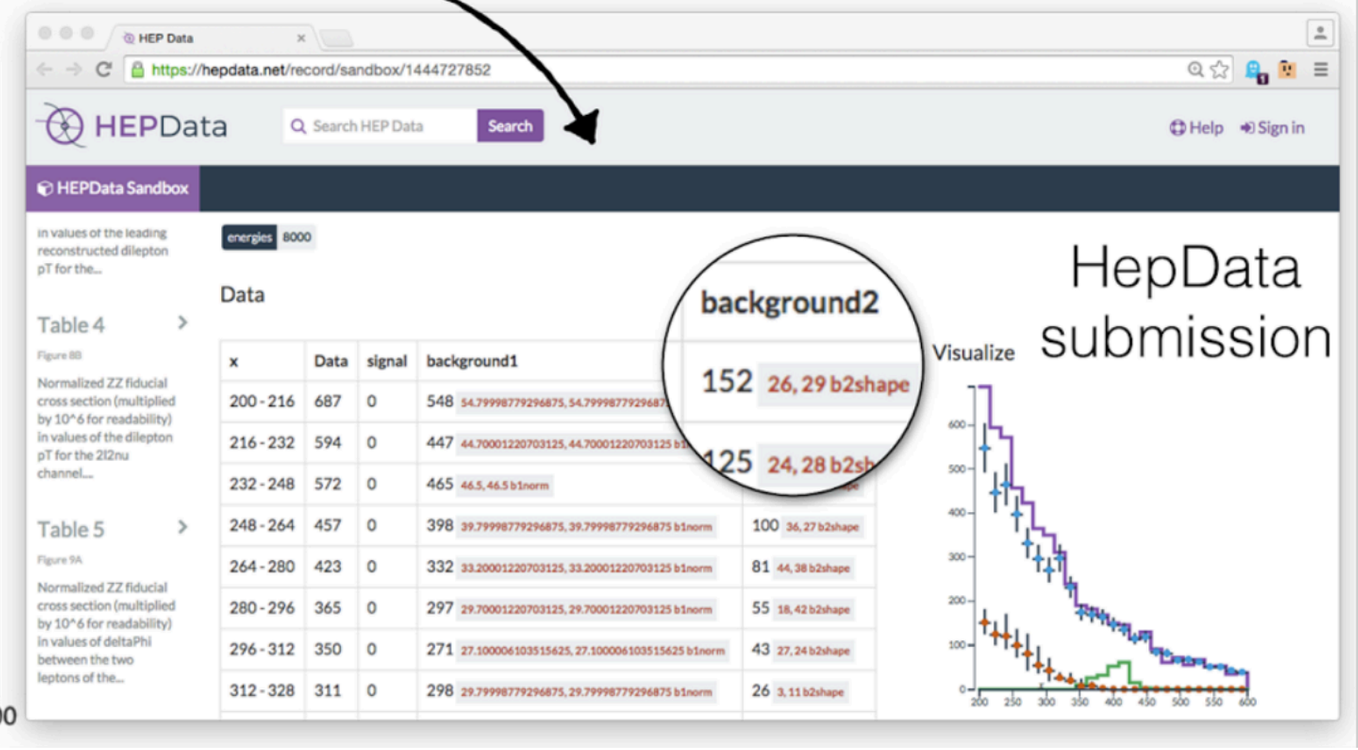
Storing HistFactory configurations on HepData

- can already be done by just storing zip file of base directory (that includes xml and input root files as “additional resources”.)
- working with HepData team to make records that have such richer resources more discoverable (search filters, badge similar to Rivet)



HistFactory





This appraoch is what's used most often today. But these are "lossy" projections of the likelihood p(x | θ)

- Analyzers pick a set of parameter settings, e.g.
  - the nominal parameter set θ_0 : nominal distributions for signal and standard model backgrounds
  - the best-fit point θ*: i.e. the post-fit distribution / fitted background and signal
- Errors according to some convention
  - either pre-fit uncertainties (e.g. what was the uncertainty on the background going into the fit — corresponds to constraint term in the likelihood). But information lost on type of constraint (correlated across all bins?, shared constraind across multiple distributions?)
  - uncertainty on the fitted value (e.g. $\theta^{*}{}_{\theta_{down}}^{\theta_{up}}$),

Useful archival information — but third-party tools like Rivet + Contur / CheckMate, MadAnalysis are often actually interested in reconstructing a realistic likelihood — not possible to reconstruct it from this lossy data.

2. Store Likelihoods in HepData

Instead of taking a likelihood (which we do have when producing a result) and tasking students on generating "views"/"projections" of it in the form of HepData tables, we could rather store the likelihood directly.

What's in a likelihood:

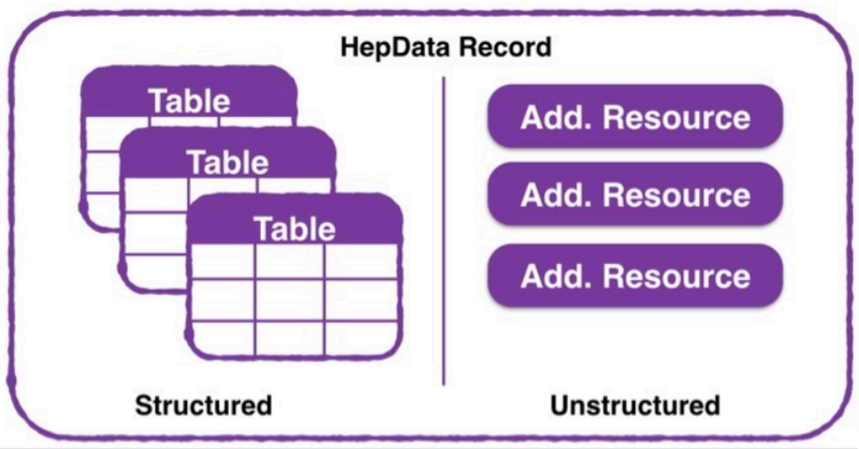
- defines structure of the likelihood (what samples are present, what signal regions / control regions are present
- nominal component distribution (for individual signal and background contributions, for all regions)
- uncertainty data (includes type of uncertaintly, e.g. correlation information, how are nuisance parameters shared across samples, etc.)

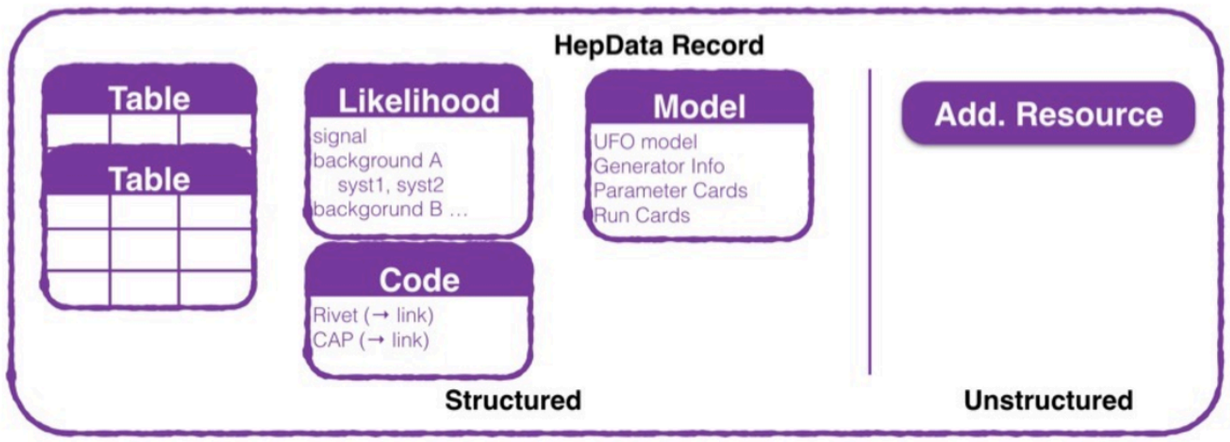
What do we gain

- Likelihoods add important semantic information on distributions in HD.
- If we want to re-use e.g. for a reinterpretation, we want to be able to e.g. reuse most of the likelihood, but switch out the signal distribution. Need to know, which histogram is the "signal" histogram etc.

How do we get this inso HepData? Could just add is as a “auxiliary resource” (short-term, but unstructured, hard to query, etc). Long term it would be good to extend HD schemas for new data structures

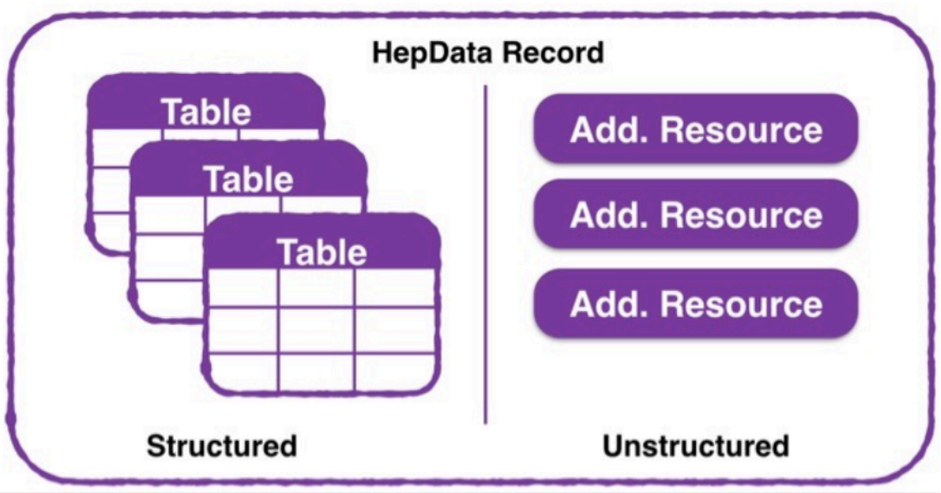
- During HepData re-write, switch from home-made ASCII format to industry standard JSON/YAML
- As a consequence, HepData is now easily extensible to add new types of data records beyond the table by just adding new JSON schemas.
- Not everything is best represented by tables (e.g. likelihoods, model information, mass spectra)

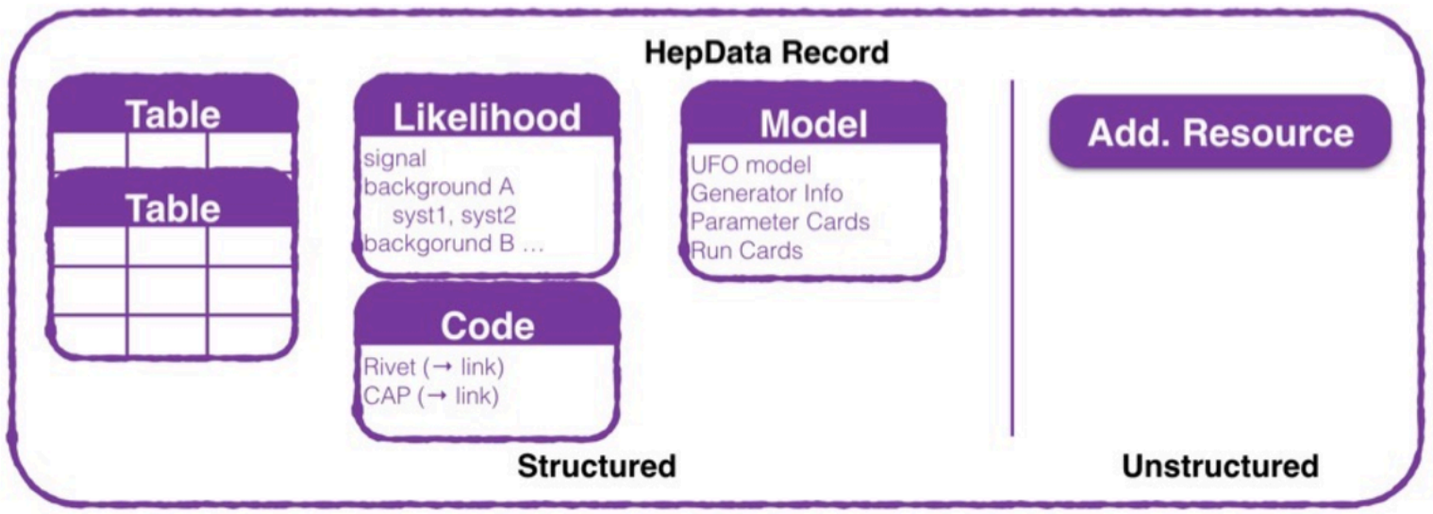




Extending the HepData schema — Example HistFactory

- HistFactory already comes with a schema (XML) that can be translated to a HepData schema
  - XML → YML
  - Histograms in ROOT files → Histograms in HepData table format
- Prototype XML→YML conversion exists but would need testing

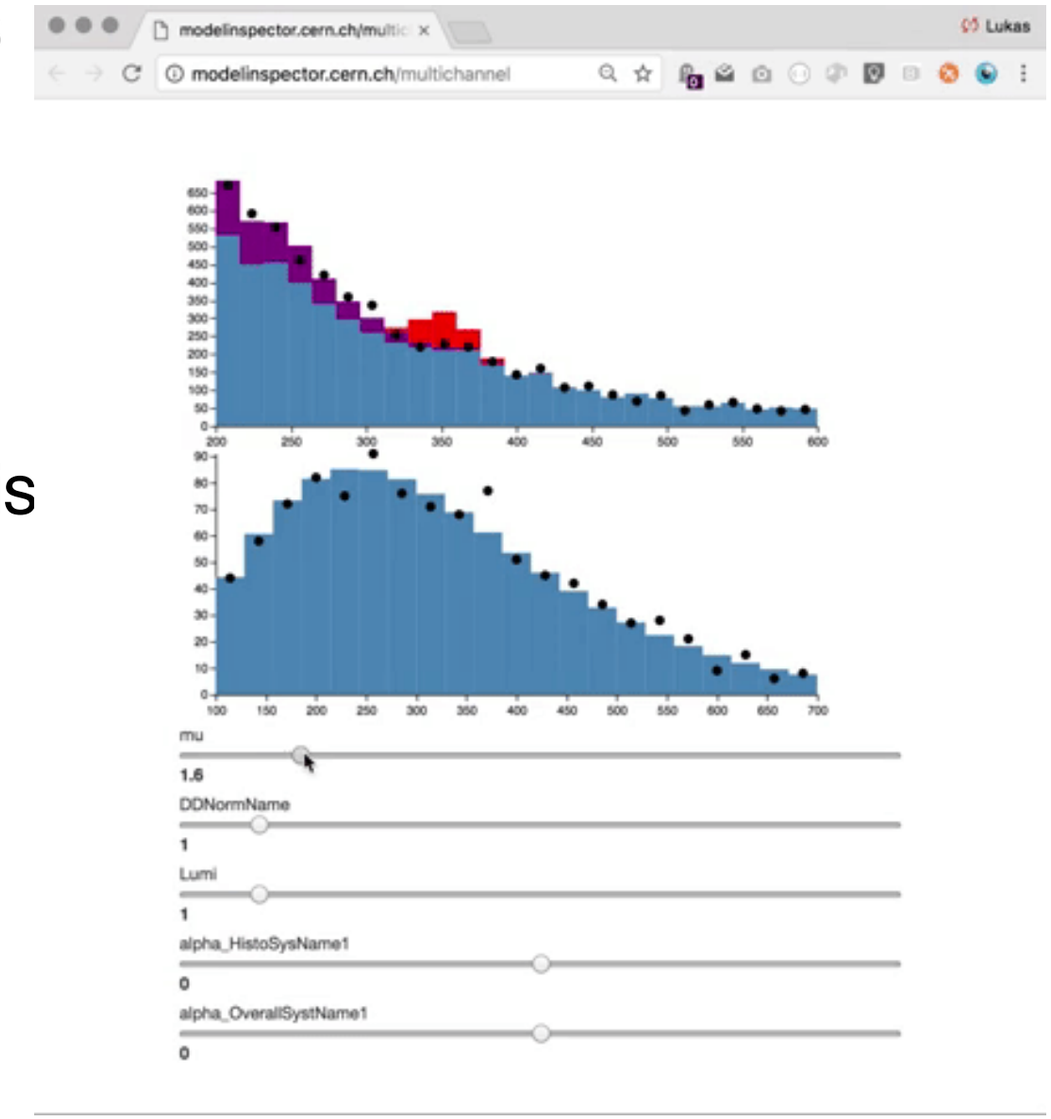




Interactive Visualization of HistFactory models

Build proto-type for visualization module for HistFactory models in HepData.

Highlights how one has full control over all nuisance parameters / gives a useful overview what nuisance parameters are present, which are most important
etc.



Very happy to work with HepData to get this supported natively / advide e.g. summer student to implement it

GIF Animation

Youtube Video

Making re-using HistFactory likelihoods easy

HistFactory historically closely connected ROOT / RooFit / RooStats. But it's a generic likelihood template:

[CERN-OPEN-2012-016]

$$\mathcal{P}(n_c, x_e, a_p \mid \phi_p, \alpha_p, \gamma_b) = \prod_{c \in \text{channels}} \left[\text{Pois}(n_c|\nu_c) \prod_{e=1}^{n_c} f_c(x_e|\boldsymbol{\alpha}) \right] \cdot G(L_0|\lambda, \Delta_L) \cdot \prod_{p \in \mathbb{S}+\boldsymbol{\Gamma}} f_p(a_p|\alpha_p) \quad (5)$$

ROOT so-far comes in as

- well-tested implementation of the template providing e.g. minimization / interval estimation etc *based on that template*
- a *data storage technology* to store histograms or a "compiled form of the likelihood" (a RooFit workspace)

Know-how how to use e.g. ROOT / RooFit / HistFactory mostly within experiments.

Making re-using HistFactory likelihoods easy

$$\mathcal{P}(n_c, x_e, a_p \mid \phi_p, \alpha_p, \gamma_b) = \prod_{c \in \text{channels}} \left[\text{Pois}(n_c|\nu_c) \prod_{e=1}^{n_c} f_c(x_e|\boldsymbol{\alpha}) \right] \cdot G(L_0|\lambda, \Delta_L) \cdot \prod_{p \in \mathbb{S}+\boldsymbol{\Gamma}} f_p(a_p|\alpha_p) \quad (5)$$

We're now working on a stand-alone pure-python implementation of HistFactory

- data storage: ROOT histograms → JSON / numpy arrays (no other part of histogram structure except for bin contents is used, arrays are fine)
- ROOT/RooFit pdf implementation → standalone scipy+numpy implementation

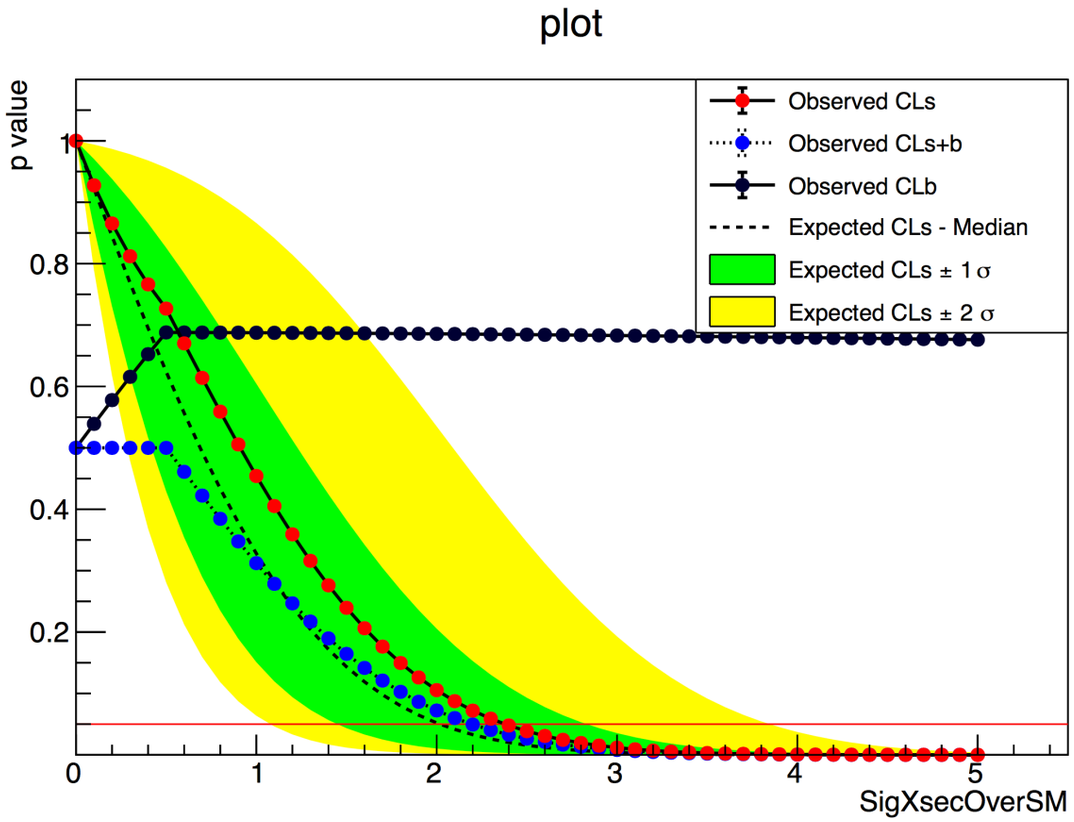
github.com/diana-hep/pyhf

try it yourself in the cloud!

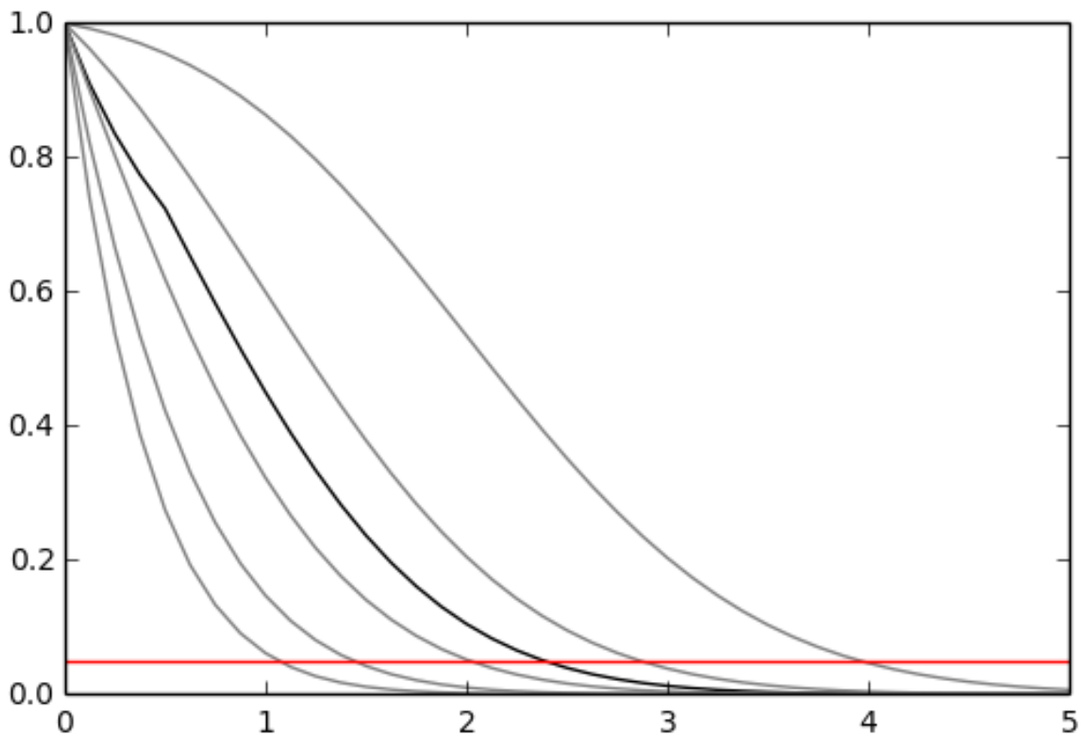
launch binder

Making re-using HistFactory likelihoods easy

- most primitives of HF implemented
- work in progress, interfaces still fluid, happy to incorporate input. feel free to join.



ROOT HistFactory



pyhf

Making re-using HistFactory likelihoods easy

minimal example: one-bin hypothesis test experiment with background uncertainty at signal strength μ=1

```
p = pyhf.simplemodels.hepdata_like(
    signal_data = [10.], bkg_data = [50.], bkg_uncerts = [7.]
)
CLs_data = pyhf.runOnePoint(mu = 1.0, data = [50.0])
```

i.e. can provide pre-made HF specs for common use-cases (such as “hepdata-like” stat. models where you only have a overall background uncertainty with uncorrelated bins, similar to the Contur stat. model) — but at the same time support for e.g. full fledged ATLAS likelihoods

Vision/Goal: Load likelihood and data from HepData

```
new_signal = yoda.load('rivet.yoda')['signal']
pdf, data = pyhf.sources.hepdata('insXXXYYYZZZ')
pdf.setSample('signal', new_signal)
recast_CLs = pyhf.runOnePoint(mu = 1.0, data)
```

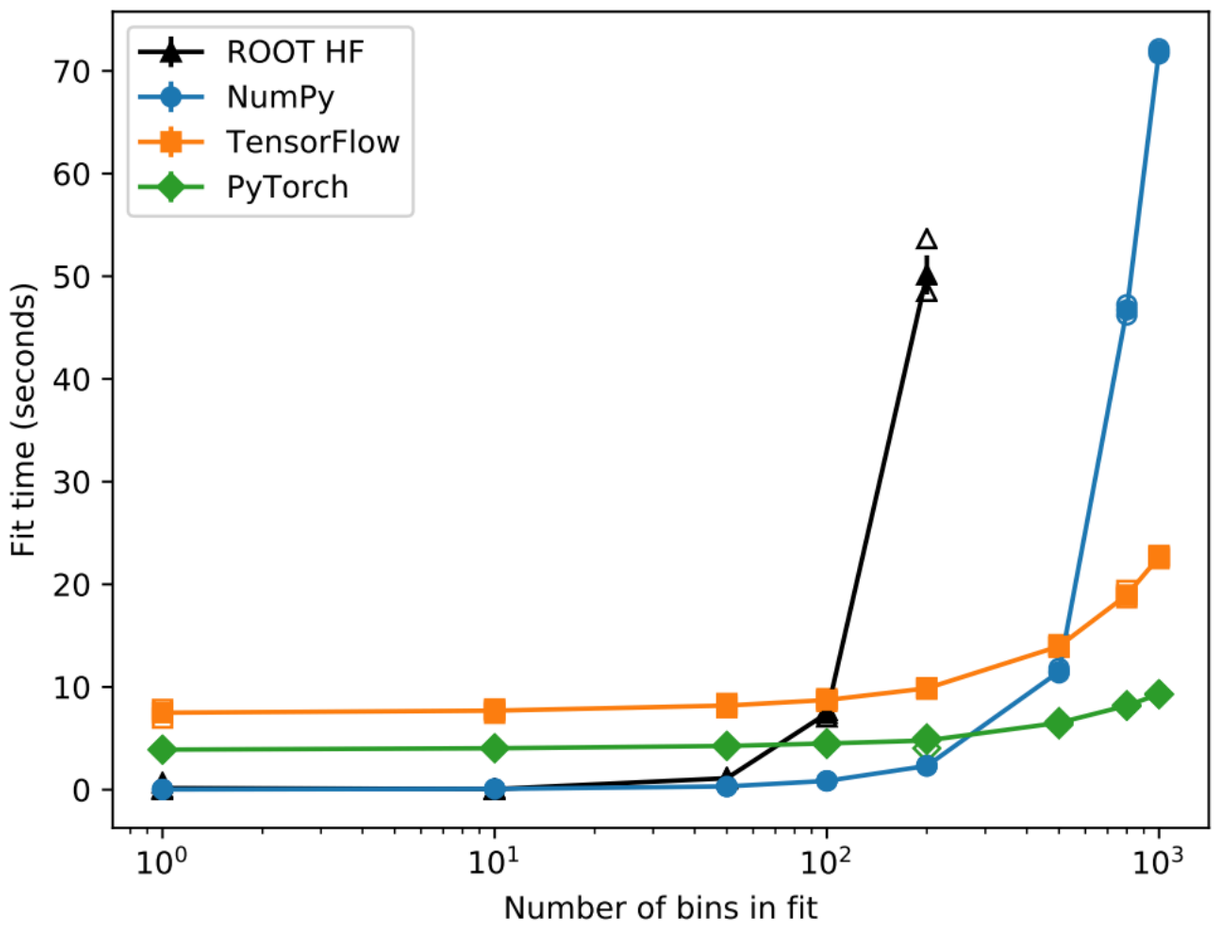
Scaling Behaviour, Running HistFactory on GPUs

added advantage of pure python implementation is that it gives us easy access to hardware acceleration, modern, auto-differentiable tensor libraries such as TensorFlow / PyTorch / MXNet that can transparently switch to GPUs

already good scaling with just CPU. Expect GPUs to add additional increase.

Often Fitting time in modern analyses not negligible (~1h) can gain a lot from acceleration.

Auto-differentiation gives us access to exact derivatives of the L'hood instead of minimizing via finite differences / MINUIT.



[Matthew Feickert, DIANA-HEP Fellow]

Conclusion

Cohesion with 3rd party tools:

If we have likelihoods, 3rd party tools like Rivet/CheckMate have a clear target of what distributions to provide as part of their analysis implementation.

Simplifying likelihoods

HistFactory is just a *likelihood format*. i.e. **not making a statement on content.** Could be reasonable for experiments to not publish the full likelihood, but some simplified version of it (but still in HF format), with only a subset of nuisance parameters / variations. Considerations

- do people want 100+ nuisance pars in their likelihood
- what are experiments comforatble with supporting / maintaining

Conclusion

Bonus:

first project excercising pheno-style recasting
custom UFO →SHERPA→RIVET → pyhf

with J. Turner, H. Schulz, Y. Zhou