`pyhf`: pure-Python implementation of HistFactory with autograd

Matthew Feickert
matthew.feickert@cern.ch

2018 US LHC Users Association Meeting Lightning Round

October 26th, 2018

SMU

Collaborators

Lukas Heinrich

NYU

Giordon Stark

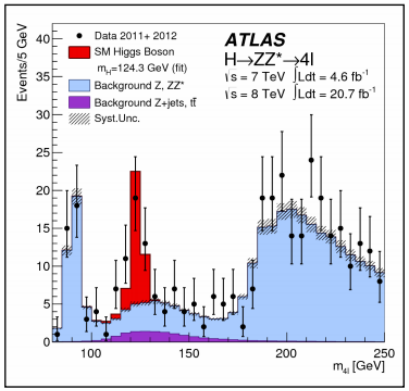
UCSC SCIPP

Kyle Cranmer

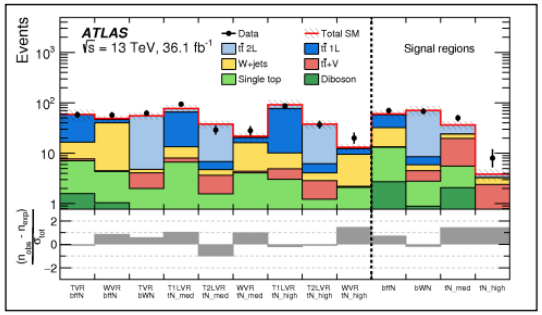
NYU

HistFactory

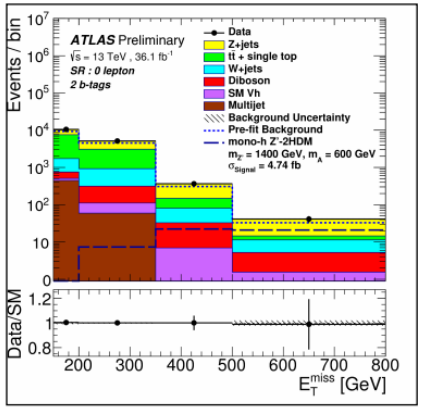
- A flexible p.d.f. template to build statistical models from binned distributions and data
- Developed by Cranmer, Lewis, Moneta, Shibata, and Verkerke [1]
- Widely used by the HEP community for standard model measurements and BSM searches



SM



SUSY



Exotics

HistFactory Template

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

This is a mathematical representation! Nowhere is any software spec defined

Until now, the only implementation of HistFactory has been in RooStats+RooFit

- To start using HistFactory p.d.f.s first have to learn ROOT, RooFit, RooStats
  - Problem for our theory colleagues (generally don't want to)
- Possible issues with scaling I/O and memory for large models
  - Not multithreaded
- Likelihood stored in the binary ROOT format
  - Challenge for long-term preservation (i.e. HEPData)
  - Why is a histogram needed for an array of numbers?

`pyhf`: HistFactory in pure Python

- First non-ROOT implementation of the HistFactory p.d.f. template
  - pure-Python library
    - `pip install pyhf[tensorflow]`
- Alternative choice to ROOT-based HistFactory to use in the analysis pipeline of HistFitter + HistFactory + RooStats
  - Not a replacement for HistFitter or RooStats
- Open source tool for all of HEP
  - Originated from a DIANA/HEP project fellowship
  - Dev team are on ATLAS and have contributions from CMS members and strong interest by theorists
  - Has already been used for reinterpretation in phenomenology paper (now on arXiv) [2]

in [109]. To compute the CL_s values, the Python-based implementation of HistFactory [110] `pyhf` was applied [111].

HistFactory in ROOT Environment

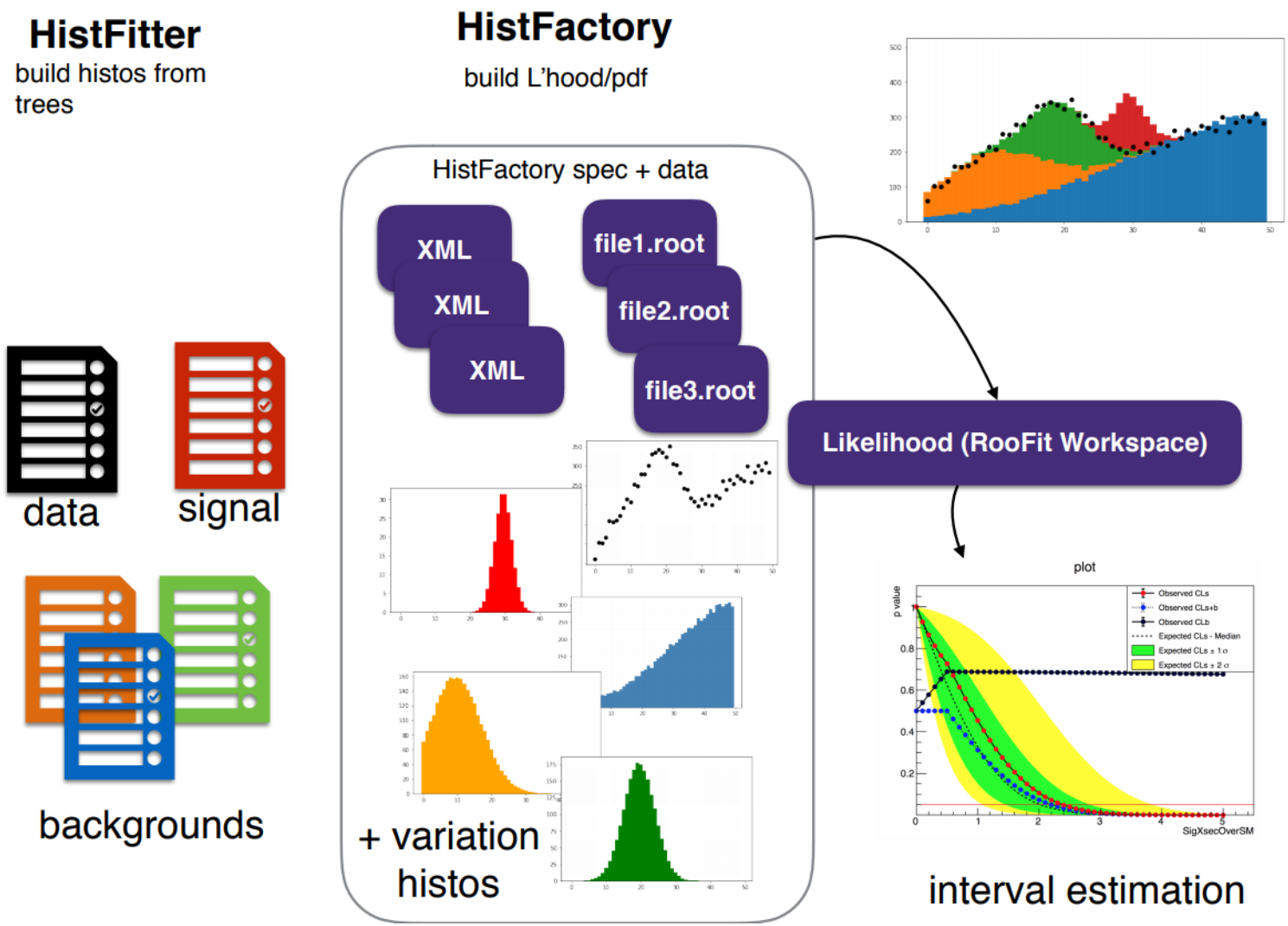


Image credit: Lukas Heinrich

HistFactory with `pyhf`

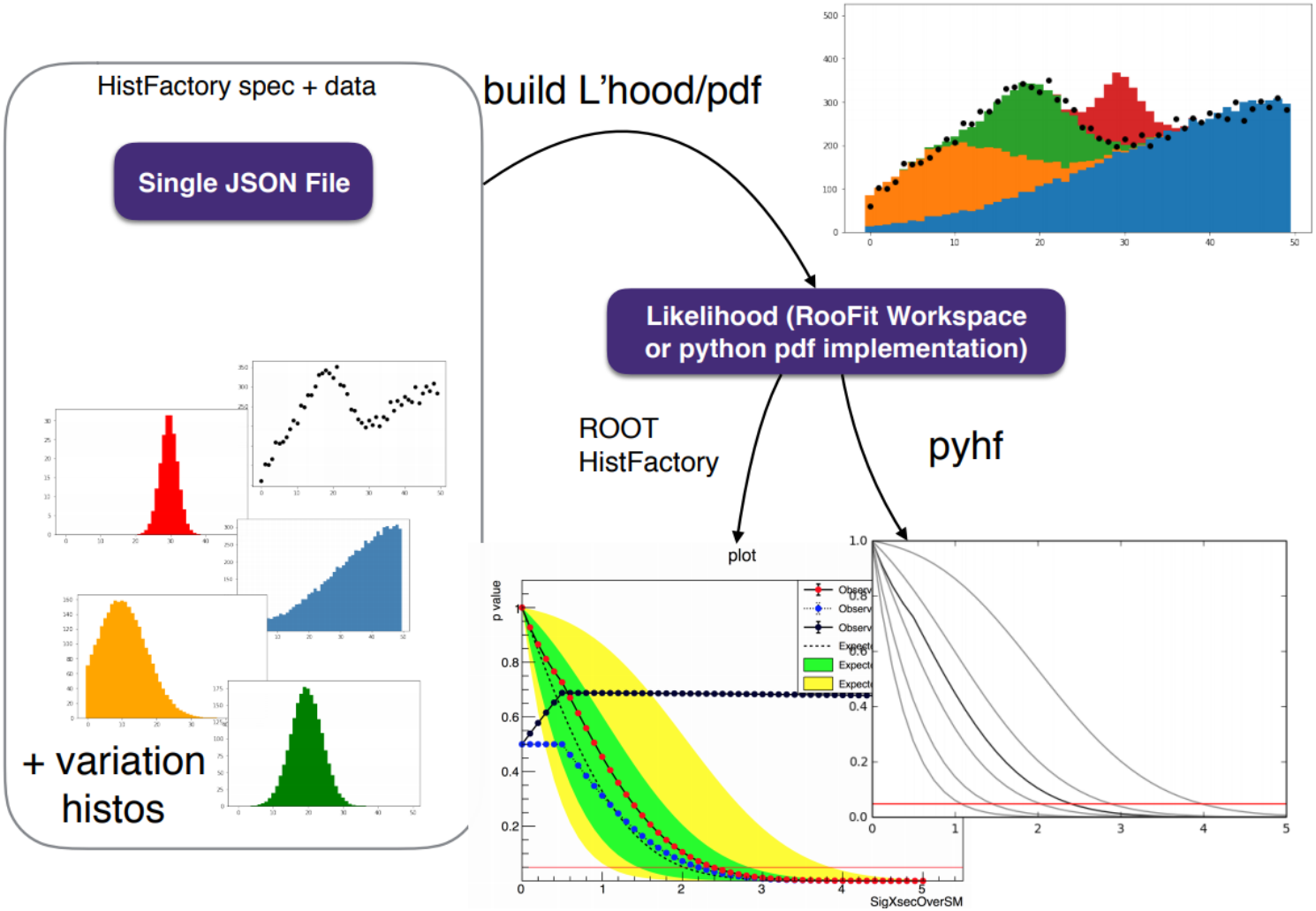


Image credit: Lukas Heinrich

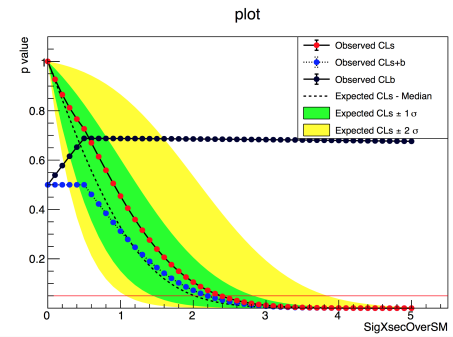
What does `pyhf` provide?

1. **Standalone pure-Python HistFactory implementation** including hooks into modern deep-learning, autodifferentiable tensor libraries

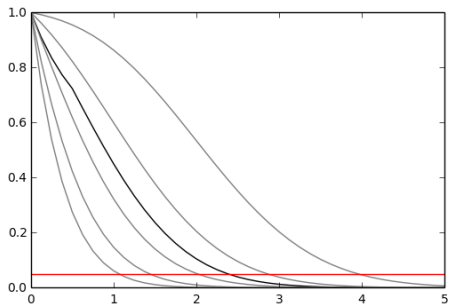
- No ROOT dependency required at all!

2. **Pure JSON schema** to distribute and archive HistFactory models ingredients without any reliance on binary formats

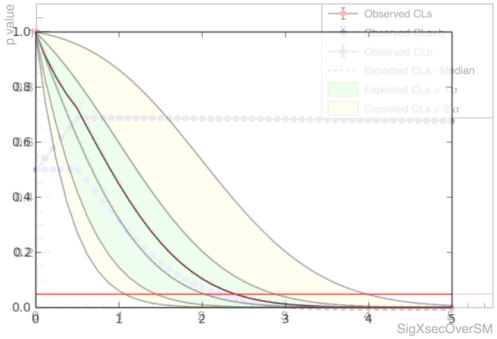
3. **Consistent results** with ROOT



ROOT



pyhf



Same Result

Easy to use Pythonic API

- Easy to use alternative implementation for non-ROOT software stack brings the useful HistFactory template to all of HEP

Hello World, `pyhf` style:

- Two bin counting experiment with a background uncertainty

```
>>> import pyhf
>>> import pyhf.simplemodels
>>> import pyhf.utils
>>>
>>> pdf = pyhf.simplemodels.hepdata_like(
...     signal_data=[12.0, 11.0], bkg_data=[50.0, 52.0], bkg_uncerts=[3.0, 7.0])
>>> CLs_obs, CLs_exp = pyhf.utils.hypotest(
...     1.0, [51, 48] + pdf.config.auxdata, pdf, return_expected=True)
>>> print('Observed: {}, Expected: {}'.format(CLs_obs, CLs_exp))
Observed: [0.05290116], Expected: [0.06445521]
```

Open industry standard file formats

JSON defining a single channel, two bin counting experiment with systematics

```
{
    "channels": [{
        "name": "singlechannel",
        "samples": [{
                "name": "sig",
                "data": [12.0, 11.0],
                "modifiers": [{ "name": "mu", "data": null, "type": "normfactor" }]
            },
            {
                "name": "bkg",
                "data": [50.0, 52.0],
                "modifiers": [{ "name": "uncorr_bkguncrt", "data": [3.0, 7.0], "type": "shapesys" }]
            }
        ]
    }],
    "data": {
        "singlechannel": [51.0, 48.0]
    },
    "toplvl": {
        "measurements": [{
            "config": { "poi": "mu" },
            "name": "singlechannel"
        }]
    }
}
```

Machine Learning Frameworks for Computational Backends

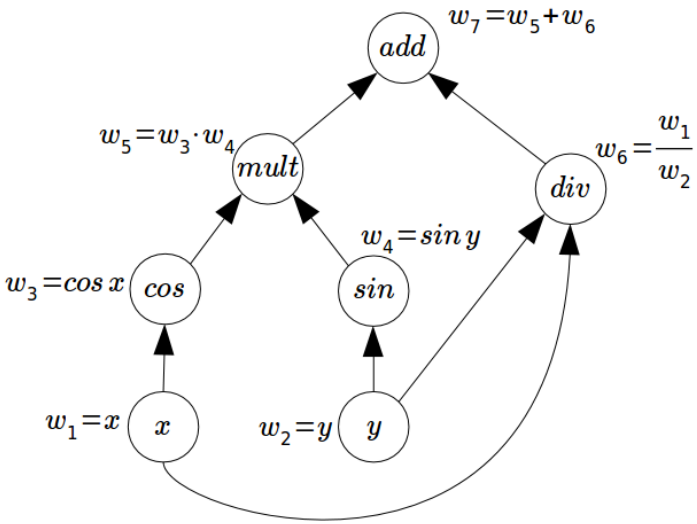
- All numerical operations implemented in **tensor backends** through an API of n-dimensional array operations
- Using deep learning frameworks as computational backends allows for **exploitation of auto differentiation (autograd) and GPU acceleration**
- As huge buy in from industry we benefit for free as these frameworks are **continually improved** by professional software engineers

Automatic differentiation

With ML backends gain access to exact (higher order) derivatives — accuracy is only limited by floating point precision

$$\frac{\partial L}{\partial \mu}, \frac{\partial L}{\partial \theta_i}$$

Gain this through the frameworks creating directed computational directed acyclic graphs and then applying the chain rule (to the operations)



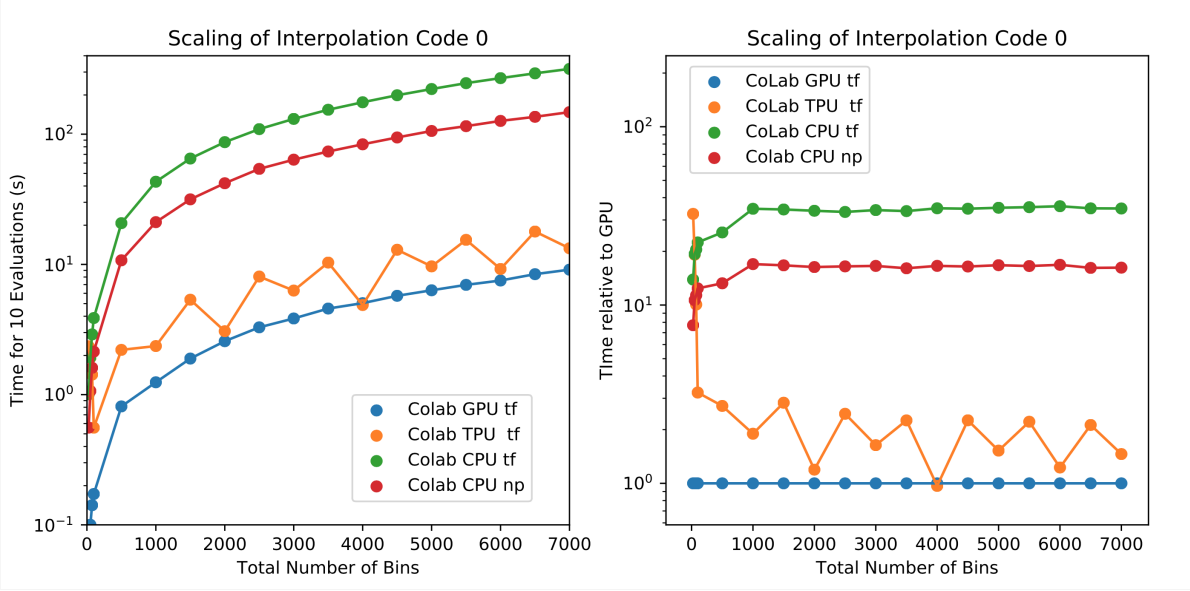
Simple example graph

Image credit: Alan Du

Preliminary Benchmarking

Fit Benchmark	`pyhf`	C++ ROOT
(Old) Single Channel	Faster	
(Old) Multichannel		Faster
(**New**) Single Channel	Faster (100X old)	
(**New**) Multichannel	Faster	

GPU Acceleration: First GPU and TPU results running on Google Colab show order of magnitude speedup for large models



Single channel fit with n bins (uncorrelated nuisance parameters).

Open Development

diana-hep / **pyhf**

Unwatch 4 | Unstar 41 | Fork 9

Code | Issues 60 | Pull requests 6 | Projects 2 | Wiki | Insights

pure-python implementation of some (maybe someday all?) HistFactory models http://diana-hep.github.io/pyhf

328 commits | **20** branches | **15** releases | **1** environment | **5** contributors | Apache-2.0

- Openly developed on GitHub
- User base growing and looking to use in Run2 analysis papers!
- Contributions welcome!
  - Just a PR away

lukasheinrich commented a day ago

awesome! ✨

Summary

- Developed and continuing to improve the first non-ROOT implementation of the HistFactory p.d.f. template in pure Python
  - Simple but powerful Pythonic API
  - **Large speedup** over ROOT coming from vectorization and deep learning framework computational backends
- Enabling fast and flexible analysis
  - Excited users wanting to use for LHC Run2 papers
- JSON specification ideal for preservation and reinterpretation
  - Fully describe model with single simple text based (versionable) file
  - Robust **long long term support**
- Openly developed on GitHub

Backup

HistFactory Template

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

Use: Multiple disjoint channels (or regions) of binned distributions with multiple samples contributing to each with additional (possibly shared) systematics between sample estimates

Main pieces:

- Poisson p.d.f. for bins observed in all channels
- Constraint p.d.f. (+ data) for "auxiliary measurements"
  - encoding systematic uncertainties (normalization, shape, etc)

Additionally allows for interactive visualization

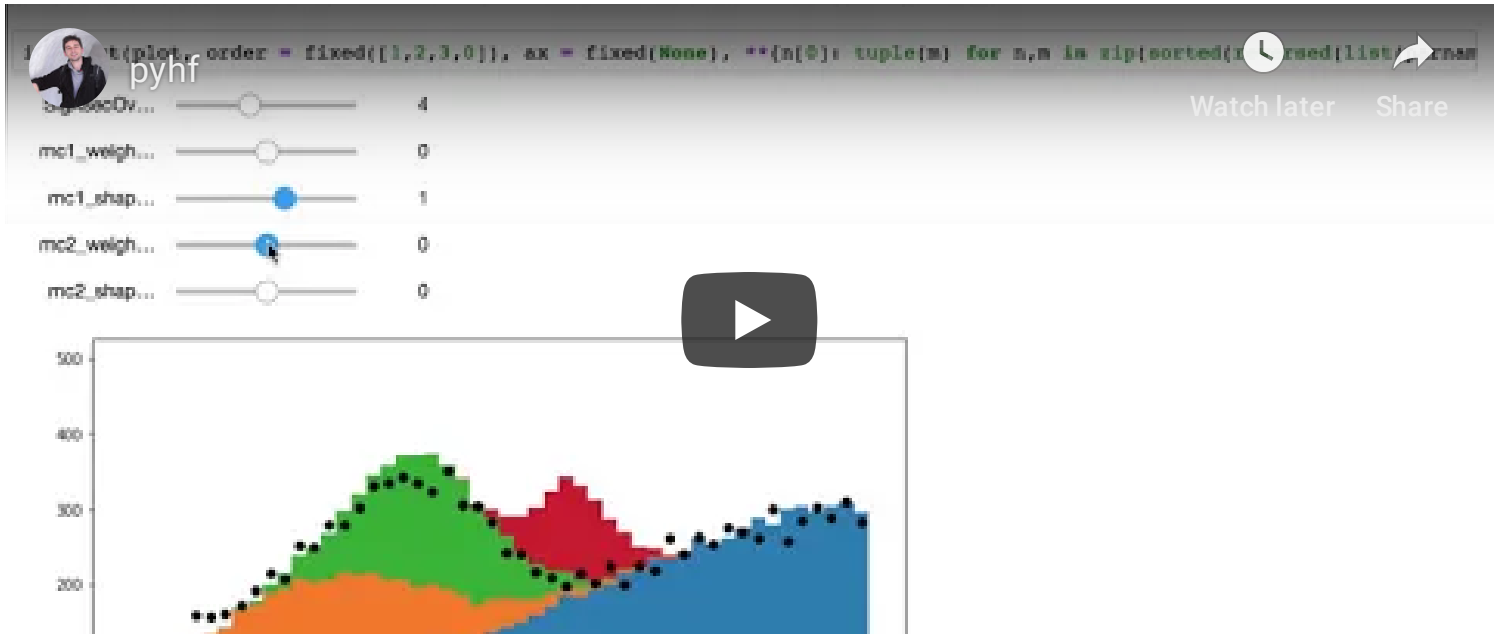


Image credit: Lukas Heinrich

Enough talk...live demo time!

Just click the button!

launch binder

CL_s Example using `pyhf` CLI

JSON defining a single channel, two bin counting experiment with systematics

```
{
    "channels": [{
        "name": "singlechannel",
        "samples": [{
                "name": "sig",
                "data": [12.0, 11.0],
                "modifiers": [{ "name": "mu", "data": null, "type": "normfactor" }]
            },
            {
                "name": "bkg",
                "data": [50.0, 52.0],
                "modifiers": [{ "name": "uncorr_bkguncrt", "data": [3.0, 7.0], "type": "shapesys" }]
            }
        ]
    }],
    "data": {
        "singlechannel": [51.0, 48.0]
    },
    "toplvl": {
        "measurements": [{
            "config": { "poi": "mu" },
            "name": "singlechannel"
        }]
    }
}
```

CL_s Example using `pyhf` CLI

```
$ pyhf cls demo.json
{
    "CLs_exp": [
        0.002606408505279359,
        0.013820656047622592,
        0.0644552079856191,
        0.23526102499555396,
        0.573041803728844
    ],
    "CLs_obs": 0.05290116065118097
}
```

CL_s with Reinterpretation

Original

```
$ pyhf cls demo.json | jq .CLs_obs
0.05290116065118097
```

Consider a new signal to test

```
# new_signal.json
[{
    "op": "replace",
    "path": "/channels/0/samples/0/data",
    "value": [5.0, 6.0]
}]
```

Apply the patch with the new signal to update the likelihood: $L \to L'$

```
$ pyhf cls demo.json --patch new_signal.json | jq .CLs_obs
0.3401578753020146
```

JSON for Statistical Models

- Long term support for non-binary format (think **preservation**)
  - HEPData now based on JSON schema
- Language agnostic
  - Reimplement likelihood in a new language more easily (c.f. `pyhf`)
  - JSON contains information to build the HistFactory likelihood
- Human and machine readable
- Offer auto-generation support from existing XML+ROOT configuration

```
$ pyhf xml2json /path/to/config.xml > hf.json
```

- Much smaller size for archiving
  - **Realistic Example:** ATLAS multi b-jet analysis workspace

`pyhf` JSON schema

```
744K mbj.json
 32K mbj.json.gz
```

ROOT

```
260K config
7.5M data
1.5M results/workspace.root
```

Realistic Example Use Case

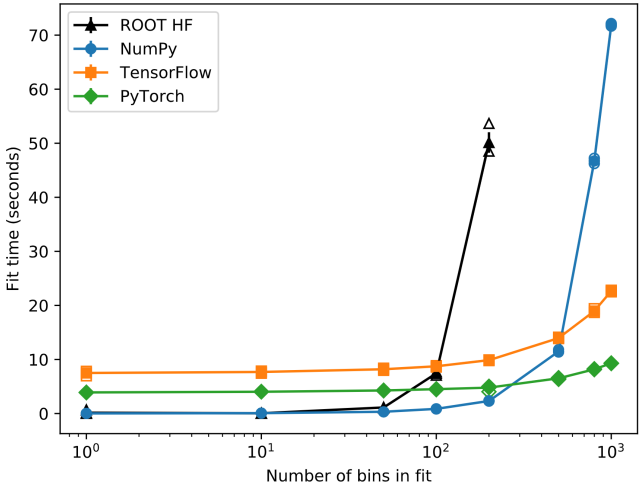
CPU NumPy backend of `pyhf` vs. ROOT HistFactory on ATLAS multi b-jet Analysis

- multi b-jet `HistFitter` configuration has 23 Channels

```
$ time pyhf cls mbj.json | jq .CLs_obs
0.2614638795780821

16.32s user 0.93s system 95% cpu 18.152 total

$ time root_cls.py atlas-conf-2018-041/workspace.root | jq .CLs_obs
0.25606989834647437

40.01s user 1.38s system 86% cpu 47.978 total
```

Preliminary Benchmarking

- Changing with many updates
- For a single channel with n nuisance parameters already seeing performance boosts
- For many channels ROOT **was** faster. With latest PRs `pyhf` now faster than ROOT in all cases.
- Needs to be revisited with recent updates that properly **implement vectorization** and graph structure
  - Seeing over a $100x$ speedup to that seen in image
- Still need to finish benchmarking on GPUs
  - Already see a $10x$ speedup



old benchmark: Single channel fit with n bins (uncorrelated nuisance parameters) with CPU backends. Lower is better.

Use in Reinterpretation

arXiv.org > hep-ph > arXiv:1810.05648

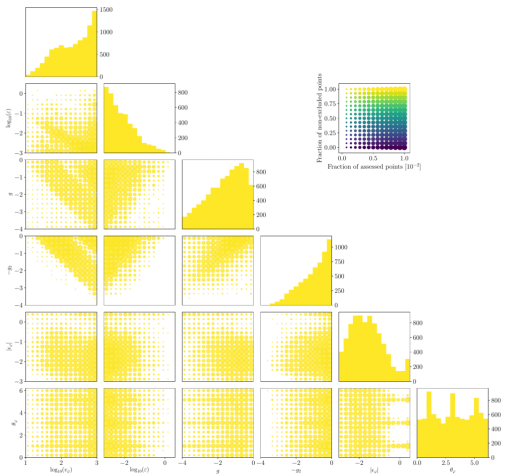
High Energy Physics - Phenomenology

Constraining A_4 Leptonic Flavour Model Parameters at Colliders and Beyond

Lukas Heinrich, Holger Schulz, Jessica Turner, Ye-Ling Zhou

(Submitted on 12 Oct 2018)

in which $p(q_\mu|\mu')$ is the distribution of the test statistic q_μ for data, which is populated according to a signal strengh μ' which we compute using the asymptotic formulae derived in [109]. To compute the CL_s values, the Python-based implementation of HistFactory [110] `pyhf` was applied [111].



Will `pyhf` extend to unbinned models?

- The project scope is to implement HistFactory as best as possible. While implementing unbinned models is not an impossibility, it is not in the project goals at the current time.
- There are already projects like GooFit, which nicely handle unbinned models.
  - DOI 10.5281/zenodo.1469544
- While contributions are welcome, it might be worth starting a different project or contributing to projects already focused on unbinned models.

The JSON has both the modifiers and the data. Is this a good idea?

- The `pyhf` dev team views this as a good feature, not a bug.
- However, the `pyhf` dev team is considering using parts of `histbook` in the future, leading to some separation.
  - DOI 10.5281/zenodo.1321926

References

1. ROOT collaboration, K. Cranmer, G. Lewis, L. Moneta, A. Shibata and W. Verkerke, *HistFactory: A tool for creating statistical models for use with RooFit and RooStats*, 2012.
2. L. Heinrich, H. Schulz, J. Turner and Y. Zhou, *Constraining A_4 Leptonic Flavour Model Parameters at Colliders and Beyond*, 2018.

The end.