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

Matthew Feickert
matthew.feickert@cern.ch

2019 US LHC Users Association Meeting Lightning Round

October 17th, 2019

ILLINOIS

pyhf team

Lukas Heinrich

CERN

Matthew Feickert

Illinois

Giordon Stark

UCSC SCIPP

Kyle Cranmer

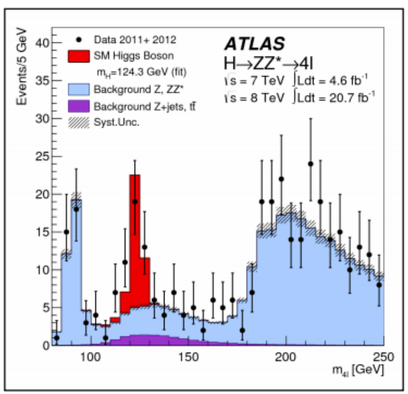
NYU

Core Developers

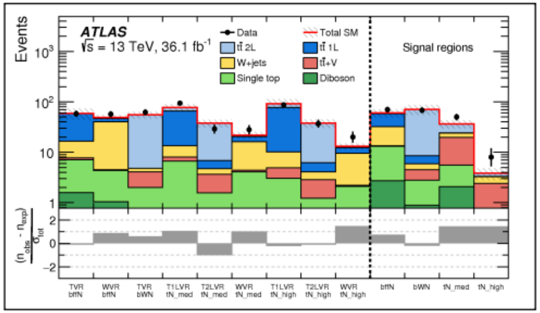
Advising

HistFactory

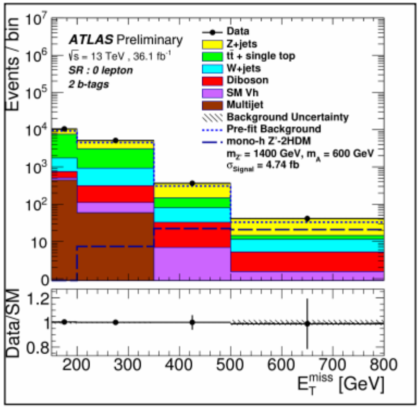
- A flexible p.d.f. template to build statistical models from binned distributions and data
- Developed by Cranmer, Lewis, Moneta, Shibata, and Verkerke [1]
- Widely used by the HEP community for standard model measurements and BSM searches



SM



SUSY



Exotics

HistFactory Template

$$f\left(\vec{n},\vec{a}|\vec{\eta},\vec{\chi}\right) = \prod_{c\,\in\,\text{channels}}\ \prod_{b\,\in\,\text{bins}_c} \text{Pois}\left(n_{cb}|\nu_{cb}\left(\vec{\eta},\vec{\chi}\right)\right) \prod_{\chi\,\in\,\vec{\chi}} c_\chi\left(a_\chi|\chi\right)$$

$$\nu_{cb}(\vec{\eta},\vec{\chi}) = \sum_{s\,\in\,\text{samples}} \underbrace{\left(\sum_{\kappa\,\in\,\vec{\kappa}} \kappa_{scb}(\vec{\eta},\vec{\chi})\right)}_{\text{multiplicative}} \left(\nu^0_{scb}(\vec{\eta},\vec{\chi}) + \underbrace{\sum_{\Delta\,\in\,\vec{\Delta}} \Delta_{scb}(\vec{\eta},\vec{\chi})}_{\text{additive}}\right)$$

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:

- Main Poisson p.d.f. for simultaneous measurement of multiple channels
- $\text{Event rates } \nu_{cb}$ from nominal rate ν^0_{scb} and rate modifiers κ and Δ
- Constraint p.d.f. (+ data) for "auxiliary measurements"
  - encoding systematic uncertainties (normalization, shape, etc)

HistFactory Template

$$f(\vec{n}, \vec{a} | \vec{\eta}, \vec{\chi}) = \prod_{c \in \text{channels}} \prod_{b \in \text{bins}_c} \text{Pois}\left(n_{cb} | \nu_{cb}(\vec{\eta}, \vec{\chi})\right) \prod_{\chi \in \vec{\chi}} c_{\chi}(a_{\chi} | \chi)$$

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

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

- Preservation: 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?
- 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)
- Difficult to use for reinterpretation

`pyhf`: HistFactory in pure Python

- First non-ROOT implementation of the HistFactory p.d.f. template
- pure-Python library as second implementation of HistFactory
  - `pip install pyhf`
  - No dependence on ROOT!

- Has a JSON spec that fully describes the HistFactory model
  - JSON: Industry standard, parsable by every language, human & machine readable, versionable and easily preserved (HEPData is JSON)
- Open source tool for all of HEP
  - Originated from a DIANA/HEP project fellowship and now an IRIS-HEP supported project
  - Used for reinterpretation in phenomenology paper [2]
  - Used internally in ATLAS for pMSSM SUSY large scale reinterpretation

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

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

NumPy

TensorFlow

PyTorch

Open industry standard file formats

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

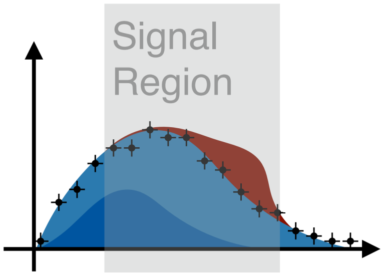
```
{
    "channels": [ # List of regions
        { "name": "singlechannel",
          "samples": [ # List of samples in region
            { "name": "signal",
              "data": [5.0, 10.0],
              # List of rate factors and/or systematic uncertainties
              "modifiers": [ { "name": "mu", "type": "normfactor", "data": null} ]
            },
            { "name": "background",
              "data": [50.0, 60.0],
              "modifiers": [ {"name": "uncorr_bkguncrt", "type": "shapesys", "data": [5.0, 12.0]} ]
            }
          ]
        }
    ],
    "observations": [ # Observed data
        { "name": "singlechannel", "data": [50.0, 60.0] }
    ],
    "measurements": [ # Parameter of interest
        { "name": "Measurement", "config": {"poi": "mu", "parameters": []} }
    ],
    "version": "1.0.0" # Version of spec standard
}
```

CL_s Example using `pyhf` CLI

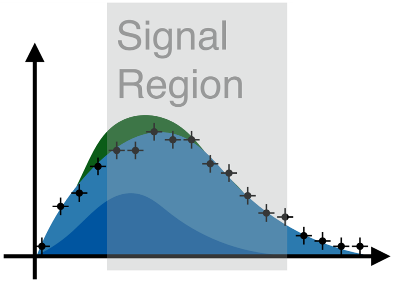
```
$ pyhf cls example.json
{
    "CLs_exp": [
        0.07807427911686152,
        0.17472571775474582,
        0.35998495263681274,
        0.6343568235898907,
        0.8809947004472013
    ],
    "CLs_obs": 0.3599845631401913
}
```

JSON Patch for new signal models

```
$ pyhf cls example.json | jq .CLs_obs
0.3599845631401913

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

$ pyhf cls example.json --patch new_signal.json | jq .CLs_obs
0.4764263982925686
```



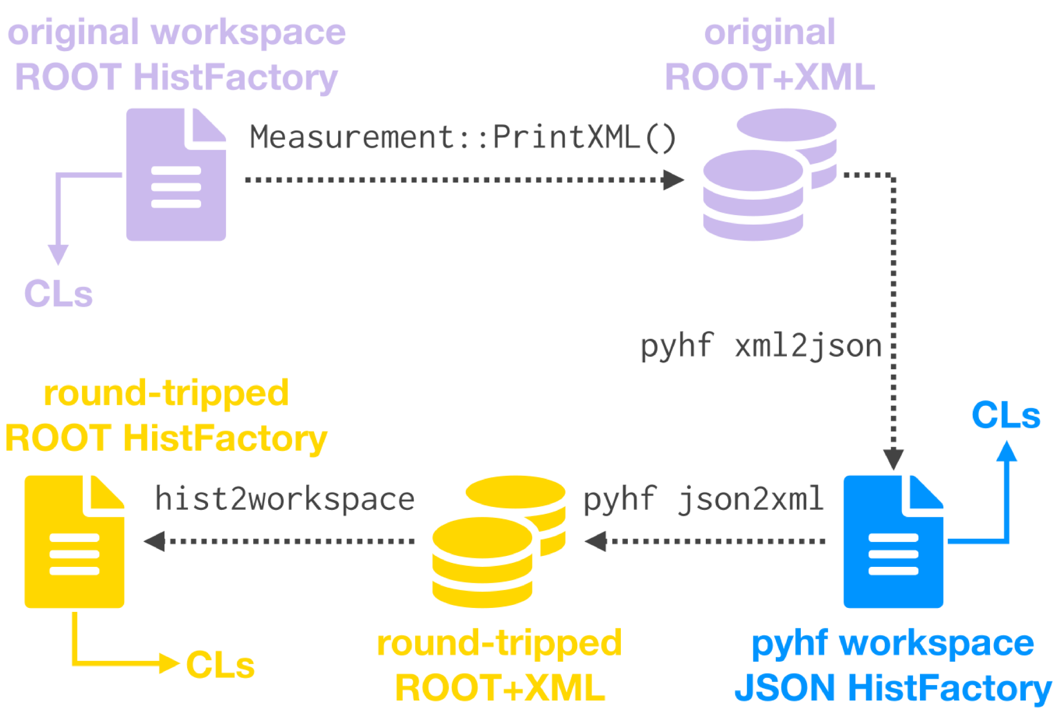
Original analysis (model A)



Recast analysis (model B)

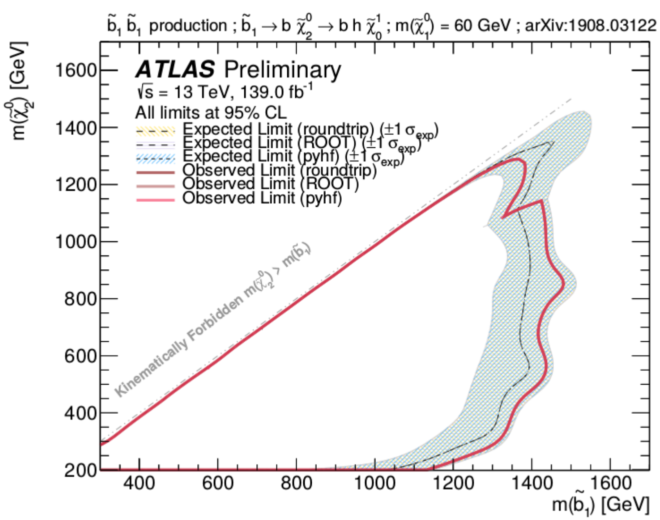
Likelihood serialization and reproduction

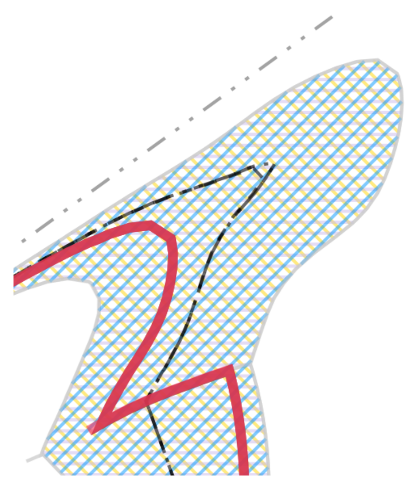
- ATLAS PUB note on the JSON schema for serialization and reproduction of results (ATL-PHYS-PUB-2019-029)
  - Contours: original ROOT+XML, pyhf JSON, JSON converted back to ROOT+XML



Likelihood serialization and reproduction

- ATLAS PUB note on the JSON schema for serialization and reproduction of results (ATL-PHYS-PUB-2019-029)
    - Contours: original ROOT+XML, pyhf JSON, JSON converted back to ROOT+XML
        - Overlay of contours nice visualization of near perfect agreement
    - Serialized likelihood and reproduced results of ATLAS Run-2 search for sbottom quarks (CERN-EP-2019-142) and published to HEPData
    - Shown to reproduce results but faster! **ROOT:** 10+ hours **pyhf:** < 30 minutes

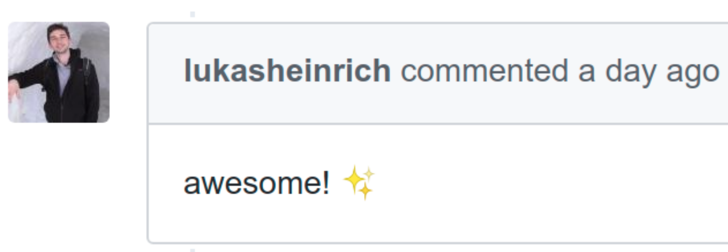




Summary

Through pyhf are able to provide:

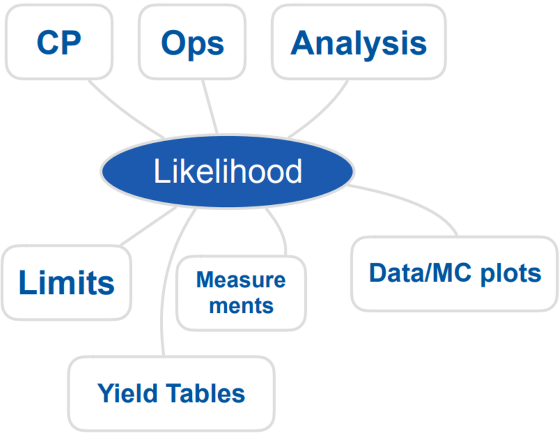
- First non-ROOT implementation of the HistFactory p.d.f. template in **pure Python**
- **JSON specification** of likelihoods that fully describes the model
  - human/machine readable, versionable, HEPData friendly, orders of magnitude smaller
  - long long term support
- **Fast and flexible** analysis
  - Analysis groups in ATLAS looking to use now
  - Part of IRIS-HEP Analysis Systems pipeline — "reducing time to insight"
- Publication for the first time of the **full likelihood** of a search for new physics
- Transparent **open development** on GitHub



Backup

Why is the likelihood important?

- High information-density summary of analysis
- Almost everything we do in the analysis ultimately affects the likelihood and is encapsulated in it
  - Trigger
  - Detector
  - Systematic Uncertainties
  - Event Selection
- Unique representation of the analysis to preserve



Likelihood serialization...

...making good on 19 year old agreement to publish likelihoods

Massimo Corradi

It seems to me that there is a general consensus that what is really meaningful for an experiment is *likelihood*, and almost everybody would agree on the prescription that experiments should give their likelihood function for these kinds of results. Does everybody agree on this statement, to publish likelihoods?

Louis Lyons

Any disagreement ? Carried unanimously. That's actually quite an achievement for this Workshop.

(1st Workshop on Confidence Limits, CERN, 2000)

This hadn't been done in HEP until now

- In an "open world" of statistics this is a difficult problem to solve
- What to preserve and how? All of ROOT?
- Idea: Focus on a single more tractable binned model first

JSON Patch for new signal models

```
{
    "channels": [
        { "name": "singlechannel",
          "samples": [
            { "name": "signal",
              "data": [5.0, 10.0],
              "modifiers": [ { "name": "mu", "type": "normfactor", "data": null} ]
            },
# Rest of the model...
}
```

Original model

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

New Signal (JSON Patch file)

```
{
    "channels": [
        { "name": "singlechannel",
          "samples": [
            { "name": "signal",
              "data": [5.0, 6.0],
              "modifiers": [ { "name": "mu", "type": "normfactor", "data": null} ]
            },
# Rest of the model...
}
```

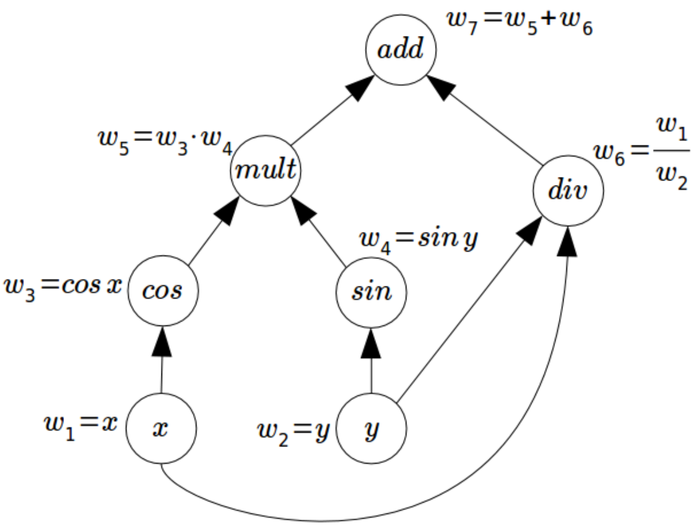
Reinterpretation

Automatic differentiation

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

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

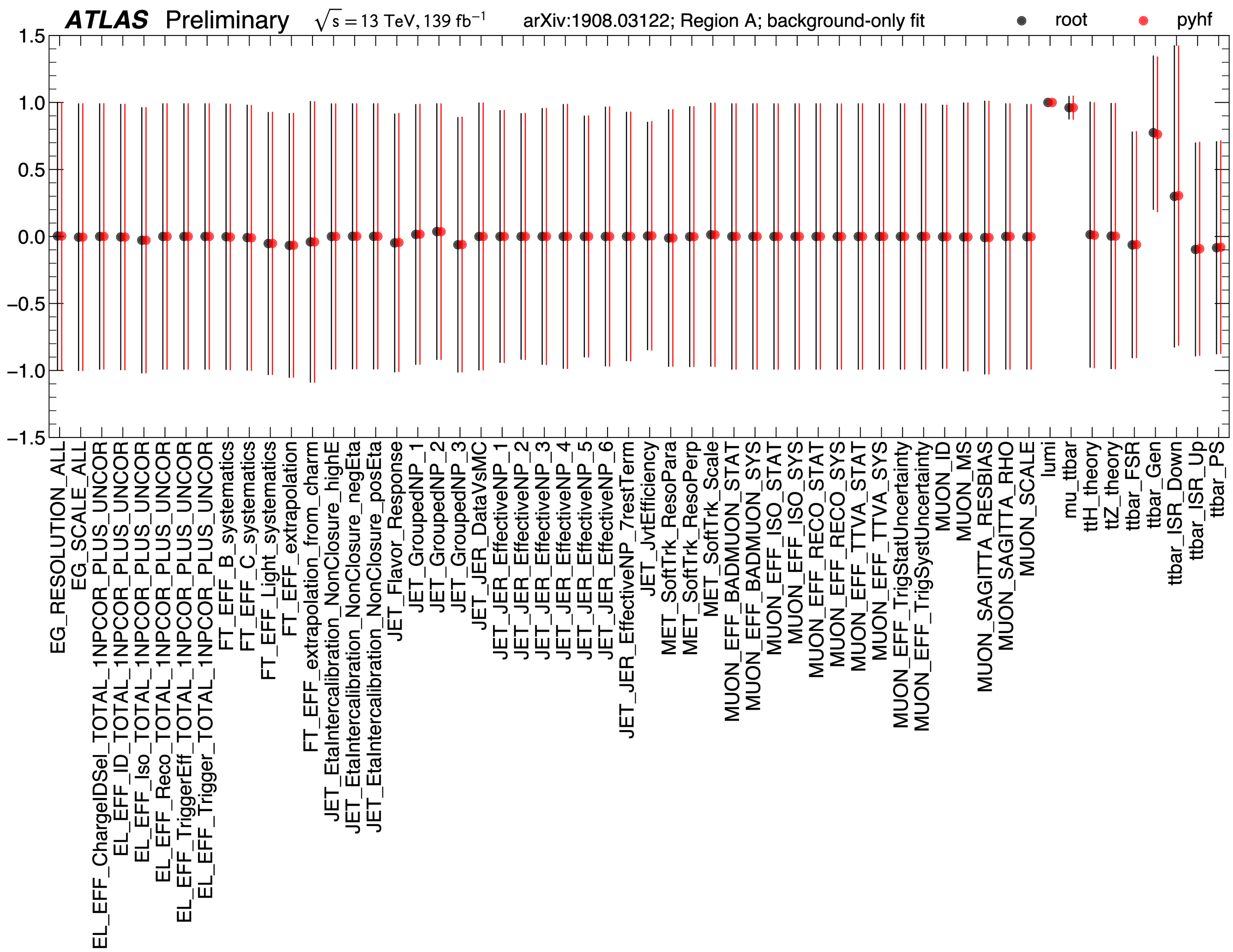
Gain this through the frameworks creating computational directed acyclic graphs and then applying the chain rule (to the operations)



Simple example graph (full likelihood too big to show here)

Image credit: Alan Du

Best-fit parameter values

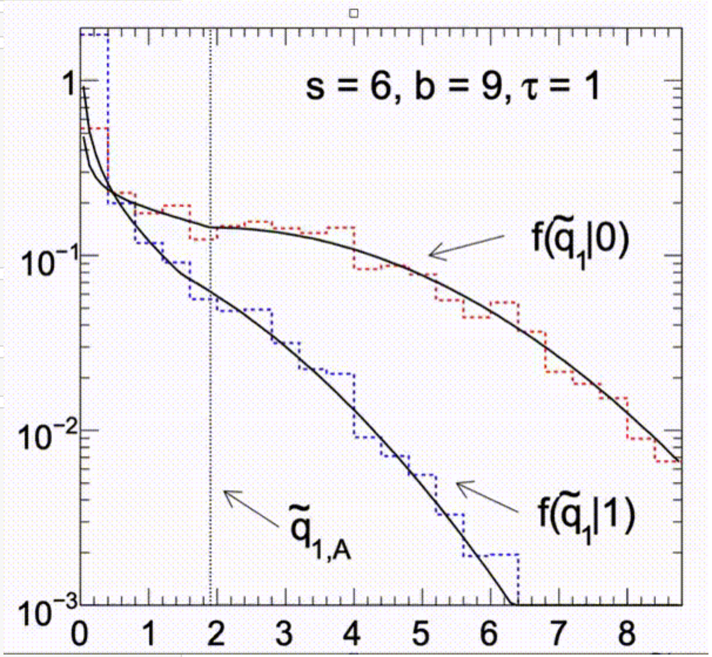


Upcoming Release v0.1.3

Pseudoexperiment generation (toys!)

In just a few lines of code are able to reproduce Figure 5b of arXiv:1007.1727!

```
In [1]: import pyhf
        import numpy as np
        import matplotlib.pyplot as plt

In [2]: np.random.seed(0)

In [3]: model = pyhf.simplemodels.hepdata_like([6], [9], [np.sqrt(9)])

        signal_pars = model.config.suggested_init()
        signal_pars[model.config.poi_index] = 1.0

        bkg_pars = model.config.suggested_init()
        bkg_pars[model.config.poi_index] = 0.0

        signal_pdf = model.make_pdf(signal_pars)
        bkg_pdf = model.make_pdf(bkg_pars)

        sample_shape = (10000,)

        signal_sample = signal_pdf.sample(sample_shape)
        bkg_sample = bkg_pdf.sample(sample_shape)

In [4]: def q_mu_tilde(poi_test, data, pdf):
            return pyhf.utils.hypotest(
                poi_test, data, pdf, qtilde=True, return_test_statistics=True
            )[1][0]

In [5]: signal_qtilde = np.asarray([q_mu_tilde(1.0, sample, model) for sample in signal_sample])
        bkg_qtilde = np.asarray([q_mu_tilde(1.0, sample, model) for sample in bkg_sample])

In [6]: fig, ax = plt.subplots(figsize=(5, 5))

        ax.hist(
            signal_qtilde,
            bins=np.linspace(0, 10, 26),
            histtype="step",
            density=True,
            linestyle="dashed",
            label=r"$f\,(\tilde{q}_1|1)$",
        )
        ax.hist(
            bkg_qtilde,
            bins=np.linspace(0, 10, 26),
            histtype="step",
            density=True,
            label=r"$f\,(\tilde{q}_1|0)$",
        )

        ax.set_xlim(0, 9)
        ax.set_ylim(1e-3, 2)
        ax.semilogy()
        ax.set_xlabel(r"$\tilde{q}_1$", fontsize=20)
        ax.set_ylabel(r"$f\,(\tilde{q}_1|\theta)$", fontsize=20)
        ax.legend(loc="best", fontsize=14);
```



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.