PYTHON x HS^3

INFERENCE SUITE

Lorenz Gärtner, Thomas Kuhr, Giordon Stark

PUBLISHING STATISTICAL MODELS

WORKSHOP ON CONFIDENCE LIMITS

CERN, Geneva, Switzerland
17–18 January 2000

CERN 2000–005

Massimo Corradi

Does everybody agree on this statement, to publish likelihoods?

Louis Lyons

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

...[Fred James wants to be able to calculate coverage, Don Groom wants to able to calculate goodness of fit]...

Cousins

I thought the point of unanimity was that publishing the likelihood function was a *necessary* condition, not a sufficient condition.

But a practical problem remained: *How* to communicate multi-D likelihood?

HEP Statistics Serialization Standard - HS^3

HS^3 provides a **standardized format** for statistical models:

- ❏ Human-readable
- ❏ Machine-readable

} JSON(-like) format

- ❏ Software-independent
- ❏ Generic, mathematical definitions
- ❏ Only from specifications: build your own implementation in language of your choice

HS^3 - Overview of supported types and components

Carsten Burgard, Tomas Dado, Jonas Eschle, Matthew Feickert, Cornelius Grunwald, Alexander Held, Jerry Ling, Robin Pelkner, Jonas Rembser, Oliver Schulz

November 6, 2023

About this document

1 Introduction

This section is non-normative.

With the introduction of `pyhf` [2], a `JSON` format for likelihood serialization has been put forward. However, an interoperable format that encompasses likelihoods with a scope beyond stacks of binned histograms was sorely lacking. With the release of `ROOT` 6.26/00 [3] and the experimental `RooJSONFactoryWSTool` therein, this gap has now been filled.

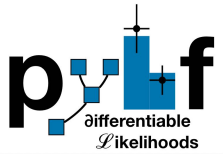
This document sets out to document the syntax and features of the HEP Statistics Serialization Standard (HS^3) for likelihoods and statistical models in general, as to be adopted by any HS^3-compatible statistics framework. The examples in this document employ the `JSON` notation, but are intended to encompass also representations as `YAML` or `TOML`.

Please note that this document as well as the HS^3 standard are still in development and can still undergo minor and major changes in the future. This document describes the syntax of version 0.2 of the HS^3 standard.

https://tinyurl.com/hs3-overleaf

https://tinyurl.com/hs3-github

CURRENT IMPLEMENTATIONS



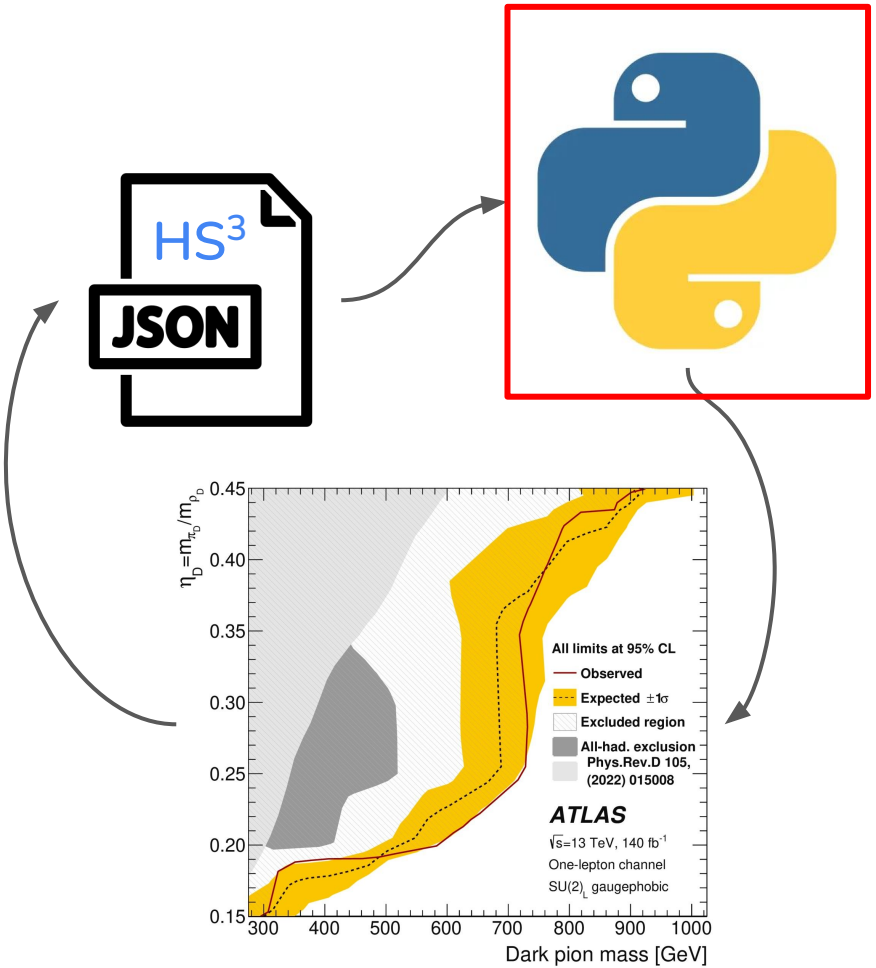
https://indico.cern.ch/event/1294577/contributions/5693326/

THE MISSING LINK

A pythonic well-maintained, user-friendly and efficient implementation of HS^3.

Build a common core for probability model serialization and inference.

- Python-based package that enables statistical inference using HS^3-compatible inputs.
- Interoperable with the HS^3 schema for reading and writing.



LEARNING FROM PAST SUCCESS …

HISTFACTORY

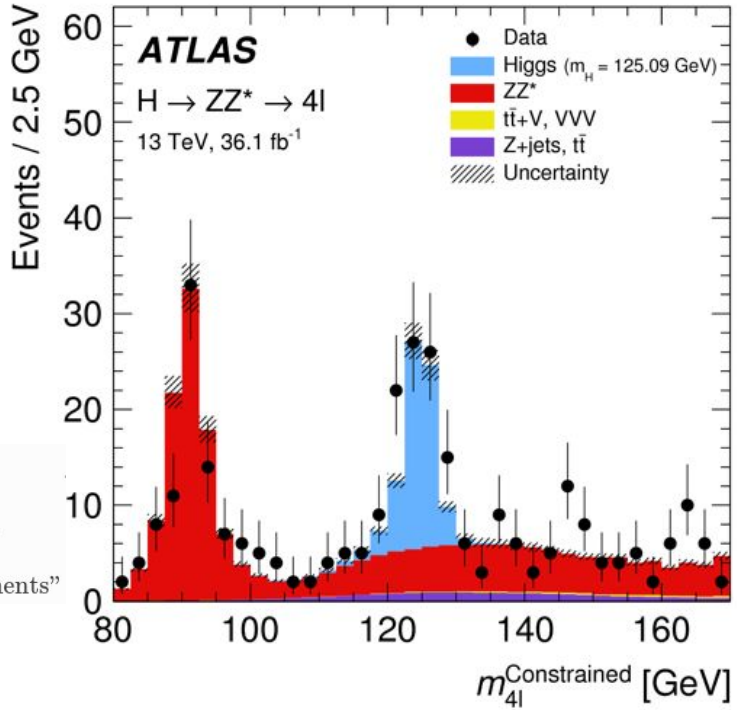
HistFactory: A tool for creating statistical models for use with RooFit and RooStats

Kyle Cranmer, George Lewis, Lorenzo Moneta, Akira Shibata, Wouter Verkerke

June 20, 2012

- A mathematical p.d.f template specification for the building of statistical models from binned distributions and data
- Widely used by the HEP community for standard model measurements and BSM searches
- Independent of its implementation in ROOT

$$f(\boldsymbol{n}, \boldsymbol{a} \mid \boldsymbol{\eta}, \boldsymbol{\chi}) = \underbrace{\prod_{c \in \text{channels}} \prod_{b \in \text{bins}_c} \text{Pois}\left(n_{cb} \mid \nu_{cb}\left(\boldsymbol{\eta}, \boldsymbol{\chi}\right)\right)}_{\text{Simultaneous measurement of multiple channels}} \underbrace{\prod_{\chi \in \boldsymbol{\chi}} c_\chi(a_\chi \mid \chi)}_{\text{constraint terms for “auxiliary measurements”}}$$



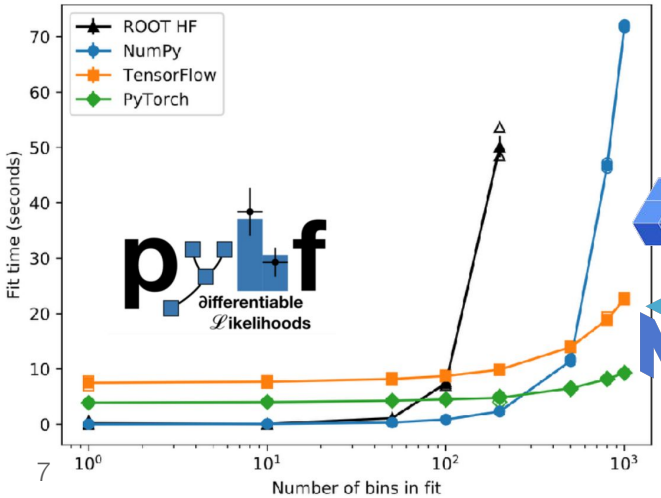
pyhf
∂ifferentiable
ℒikelihoods



Lukas Heinrich Matthew Feickert Giordon Stark

A python-only implementation of the HistFactory model + inference methods, along with a JSON model serialization.

Orders-of-magnitude faster inference than ROOT

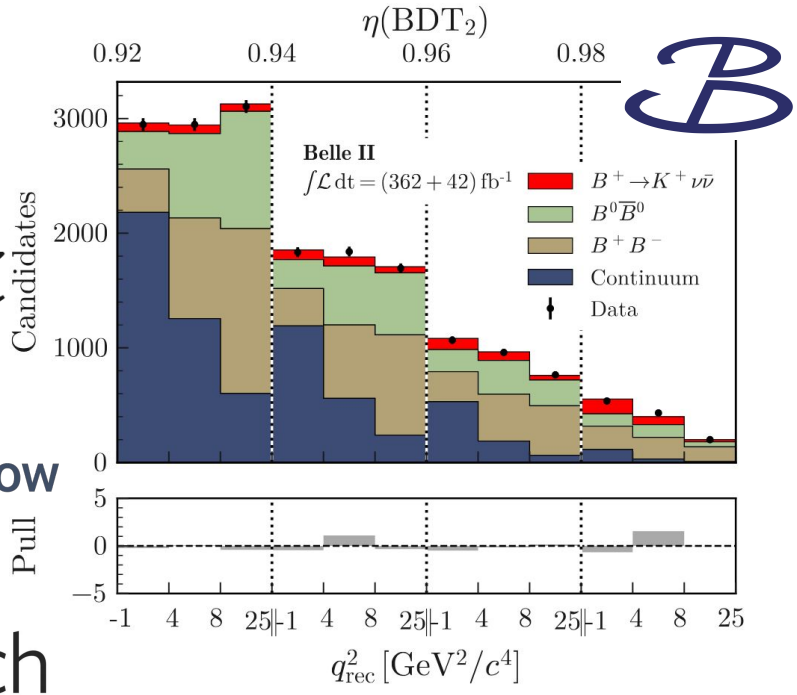


iminuit

TensorFlow

NumPy

PyTorch



PUBLISHING LIKELIHOODS

A single plain-text, human-readable file (JSON) specifies the entire HistFactory workspace!

This enables:

- Statistical combinations
- Reinterpretation (w/ signal patching)
- Propagating updated theory predictions
- …

pyhf
∂ifferentiable
ℒikelihoods

Publishing statistical models: Getting the most out of particle physics experiments

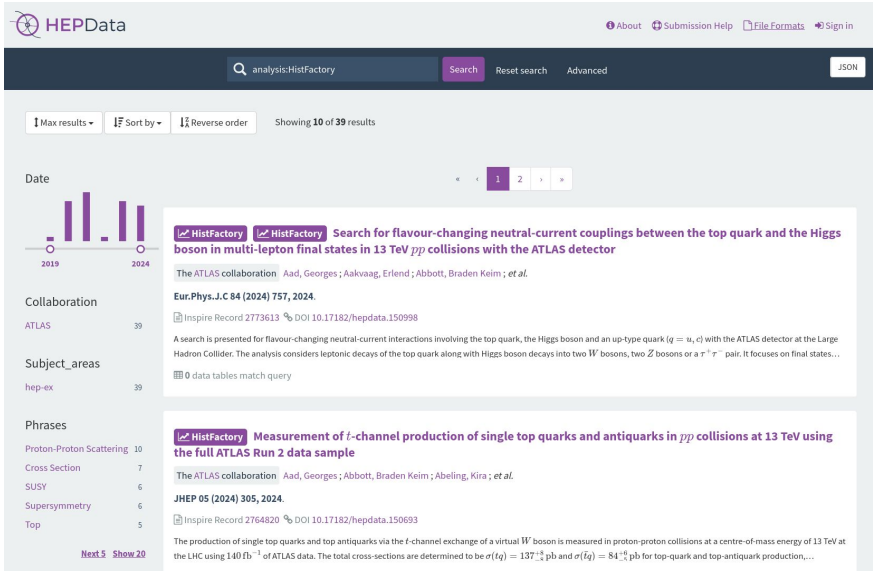
Kyle Cranmer[1*], Sabine Kraml[2‡], Harrison B. Prosper[3§] (editors), Philip Bechtle[4], Florian U. Bernlochner[4], Itay M. Bloch[5], Enzo Canonero[6], Marcin Chrzaszcz[7], Andrea Coccaro[8], Jan Conrad[9], Glen Cowan[10], Matthew Feickert[11], Nahuel Ferreiro Iachellini[12,13], Andrew Fowlie[14], Lukas Heinrich[15], Alexander Held[1], Thomas Kuhr[13,16], Anders Kvellestad[17], Maeve Madigan[18], Farvah Mahmoudi[15,19], Knut Dundas Morå[20], Mark S. Neubauer[11], Maurizio Pierini[15], Juan Rojo[8], Sezen Sekmen[22], Luca Silvestrini[23], Veronica Sanz[24,25], Giordon Stark[26], Riccardo Torre[8], Robert Thorne[27], Wolfgang Waltenberger[28], Nicholas Wardle[29], Jonas Wittbrodt[30]

```
{
  "channels": [
    {
      "name": "singlechannel",
      "samples": [
        {
          "name": "signal",
          "data": [12.0, 11.0],
          "modifiers": [
            {
              "name": "mu",
              "type": "normfactor",
              "data": null
            }
          ]
        },
        {
          "name": "background",
          "data": [50.0, 52.0],
          "modifiers": [
            {
              "name": "uncorr_bkguncrt",
              "type": "shapesys",
              "data": [3.0, 7.0]
            }
          ]
        }
      ]
    }
  ]
}
```

MAIN TAKEAWAYS

- Simplicity is key for serialization
  - Simple construction
  - Simple I/O
- Backward compatibility essential
  - pyhf version as metadata
- Conversion (from other formats)
  - eg. pyhf <> HistFactory XML (RooWorkspace)
- ‘Closed world’ model simplifies serializability and scope of responsibilities

```
>>> with open('workspace.json') as ws:
...     spec = json.load(ws)
...
>>> workspace = pyhf.Workspace(spec)
>>> pdf = workspace.model()
>>> data = workspace.data(model)
>>> pyhf.infer.mle.fit(data, pdf)
array([0.95260667, 0.99635345, 1.02140172])
```

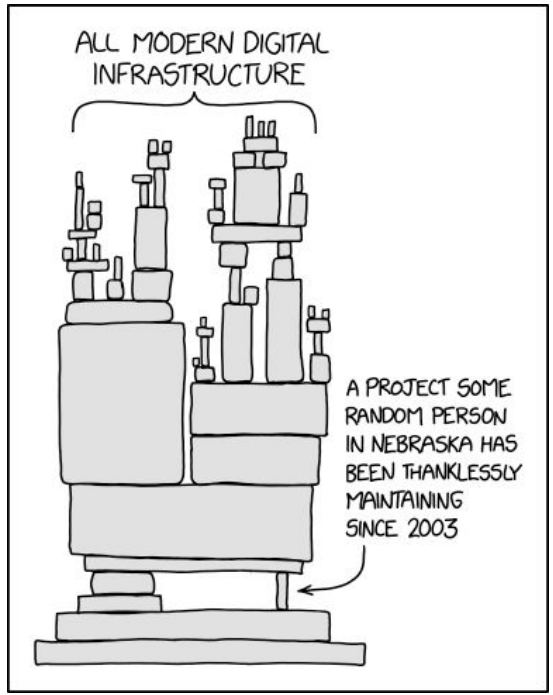


39 HistFactory models on HEPData

A NEW COMMON CORE …

… REQUIRES A SOLID FOUNDATION

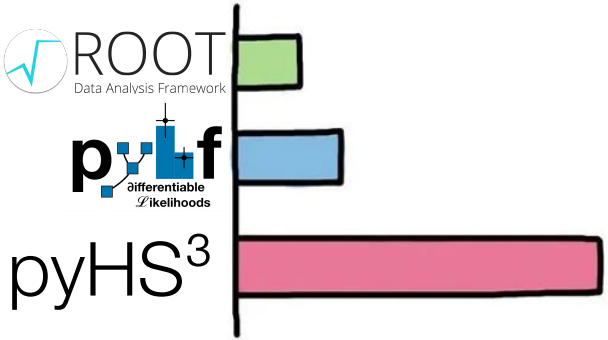
- Built with HS^3 in mind from the start
- Well maintainable
- Backwards-compatible
  - Tools change and so will HS^3
  - Results should remain reproducible
- Limited scope
  - Set clear boundaries of the tool
- Maximized efficiency
  - Fast evaluation of likelihood and gradients
  - eg. automatic differentiation
- Well interfaced with already existing tools



iminuit BAT.jl Bayesian Analysis Toolkit zfit pyhf differentiable likelihoods Scikit HEP

TENTATIVE PLAN

- Start with representative analyses from different HEP experiments providing
  - HS^3-compatible serialization of their probability models
  - Public data and statistical inference results for cross-checks
- Pick the right (modern) tools for the job
  - TensorFlow-probability, scipy.stats, pytorch, JAX
  - Minimize dependencies as much as possible
- Documentation, Tutorials, and Testing
  - Documentation describes a "Model Interface" / "Inference Interface"
  - Translate existing statistics tutorials and lectures to show how to use python-HS^3
- Maintenance
  - A robust CI/CD to maintain cross-checks with the representative analyses
  - Enforce code styles and lint with tools like ruff and pylint
  - Open-source to allow for community contributions



OPEN QUESTIONS

SERIALIZE THE OPEN WORLD

- How open world do we need to be?
- How to handle custom mathematical functions in a user-friendly way?
- How to allow users to provide their own custom functions as plugins?

COMPUTATIONAL EFFICIENCY

- How to build a computational graph that can be exported and used as part of end-to-end differentiation?

```
>>> import numexpr as ne
>>> import numpy as np
>>> a = np.random.rand(3)
>>> b = np.random.rand(3)
>>> ne.evaluate("sin(a) + arcsinh(a/b)")
array([0.24508376, 2.512371  , 4.03901998])
```

pyhf
∂ifferentiable
𝒫ikelihoods

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

simpler, tensor-based, with automatic differentiation

python-HistFactory computational tree

ERuM Data project proposal

Lorenz Gärtner, Thomas Kuhr, Giordon Stark

Pythonic HS^3 inference

Project summary

In experimental high energy physics, the ability to communicate and publish results in a computationally-accessible format is important for allowing theorists and phenomenologists to (re)use analysis data. In recent years, efforts have made progress on various fronts of the principles of FAIR (Findability, Accessibility, Interoperability, and Reusability) data; including SLHA files for parameter configurations used in Monte Carlo generators to a popular plain-text specification for HistFactory (HiFa) probability models and its corresponding implementation: pyhf. This project

COMMENTS WELCOME

BACKUP

MORE OPEN QUESTIONS

- Only python? Julia? Rust?
- What is the best way to serialize probability models following the HS3 schema? JSON? YAML? BSON? Some other plain-text encoding that's more highly-compressible, or a binary encoding?
- How to deal with plotting?
- How to deal with debugging?
- What gets pulled out of pyhf, and how to design in such a way that it is mostly compatible with pyhf's API?

HISTFACTORY - A mathematical prescription

$$f(\boldsymbol{n}, \boldsymbol{a} \,|\, \boldsymbol{\eta}, \boldsymbol{\chi}) = \underbrace{\prod_{c \in \text{ channels}} \prod_{b \in \text{ bins}_c} \text{Pois}\left(n_{cb} \,|\, \nu_{cb}\left(\boldsymbol{\eta}, \boldsymbol{\chi}\right)\right)}_{\text{Simultaneous measurement of multiple channels}} \underbrace{\prod_{\chi \in \boldsymbol{\chi}} c_\chi(a_\chi|\,\chi)}_{\text{constraint terms for “auxiliary measurements”}}$$

Multiple, disjoint channels of binned distributions with multiple samples contributing to each with additional (shared) systematics between sample estimates

- Poisson p.d.f. for bins observed in all channels
- Constraint p.d.f. (and data) for auxiliary measurements (systematics: normalization, shape,etc)

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