Binned template fits with cabinetry

Kyle Cranmer[1], **Alexander Held[1]**

[1] New York University

PyHEP 2021
https://indico.cern.ch/event/1019958/
July 6, 2021

This work was supported by the U.S. National Science Foundation (NSF) Cooperative Agreement OAC-1836650 (IRIS-HEP).

Introduction

- **Binned template fits** are widely used for **statistical inference** at the LHC and beyond

- **`HistFactory`** is a statistical model for **binned template fits**
  - prescription for constructing probability density functions (pdfs) from small set of building blocks
  - covers wide range of use cases, extensively used in ATLAS
  - models can be serialized to *workspaces*

the **`HistFactory`** pdf (pyhf docs)

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

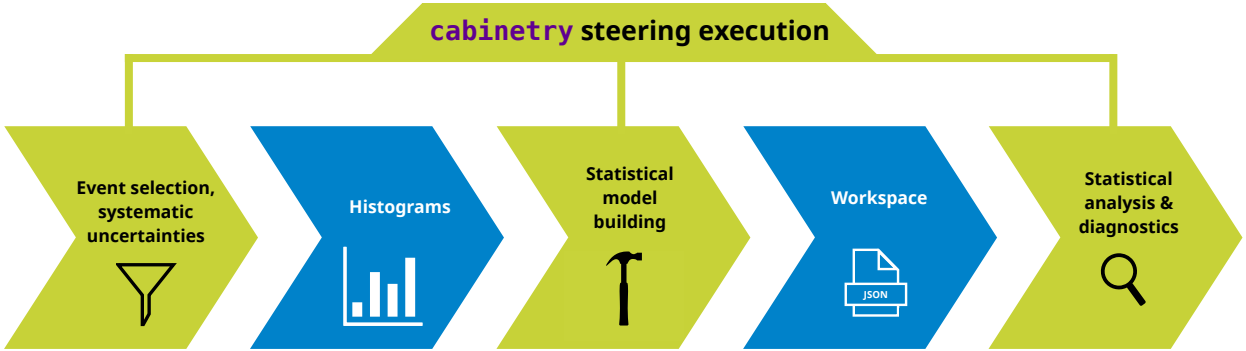
- **`cabinetry`** is a **Python library** for constructing and operating **`HistFactory`** models

```
>pip install cabinetry
```

  - uses **`pyhf`** (**`HistFactory`** model in Python)
  - integrates seamlessly with the flourishing Python HEP ecosystem
  - modular design: drop in and out of **`cabinetry`** whenever needed

Working with `cabinetry`

- **`cabinetry`** is used to
  - **design** and **construct statistical models** (workspaces) from instructions in **declarative configuration**
    - analyzers specify selections for signal/control regions, (Monte Carlo) samples, systematic uncertainties
    - **`cabinetry`** steers creation of template histograms (region ⊗ sample ⊗ systematic)
    - **`cabinetry`** produces **`HistFactory`** workspaces (serialized fit model)
  - perform **statistical inference**
    - including diagnostics and visualization tools to study and disseminate results



Designing a statistical model

- **declarative configuration** (JSON/YAML/dictionary) specifies everything needed to build a workspace
  - can concisely capture complex region ⊗ sample ⊗ systematic structure

general settings

list of phase space regions (channels)

list of samples (MC/data)

```yaml
General:
  Measurement: "Example"
  InputPath: "input/{SamplePaths}"
  HistogramFolder: "histograms/"
  POI: "Signal_norm"

Regions:
  - Name: "Signal_region"
    Filter: "nJets >= 8"
    Variable: "jet_pt"
    Binning: [200, 300, 400, 500]

Samples:
  - Name: "Data"
    SamplePaths: "data.root"
    Tree: "events"
    Data: True

  - Name: "Signal"
    SamplePaths: "signal.root"
    Tree: "events"
    Weight: "weight_nominal"

  - Name: "Background"
    SamplePaths: "background.root"
    Tree: "events"
    Weight: "weight_nominal"
```

```yaml
Systematics:
  - Name: "Luminosity"
    Up:
      Normalization: 0.05
    Down:
      Normalization: -0.05
    Samples: ["Signal", "Background"]
    Type: "Normalization"

  - Name: "ModelingVariation"
    Up:
      Tree: "events_up"
      Weight: "weight_modeling"
    Down:
      Tree: "events_down"
      Weight: "weight_modeling"
    Smoothing:
      Algorithm: "353QH, twice"
    Samples: "Background"
    Type: "NormPlusShape"

NormFactors:
  - Name: "Signal_norm"
    Samples: "Signal"
    Nominal: 1
    Bounds: [0, 10]
```

list of systematic uncertainties

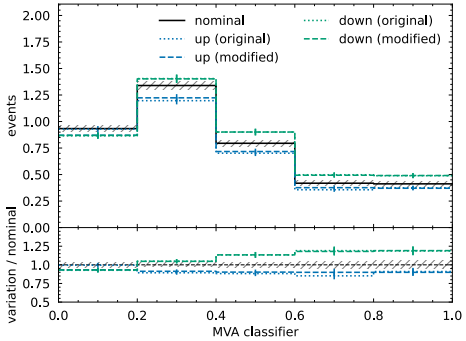
list of normalization factors

Template histograms and workspace building

- **workspaces construction** happens in three steps:
  1) create template histograms from columnar data following config instructions
     - backends execute instructions (default: **uproot**, experimental: **coffea**)
  2) optional: apply post-processing to templates (e.g. smoothing)
  3) assemble temples into workspace (JSON file)

- utilities provided to **visualize and debug** fit model
- possible to provide **custom code** for template creation

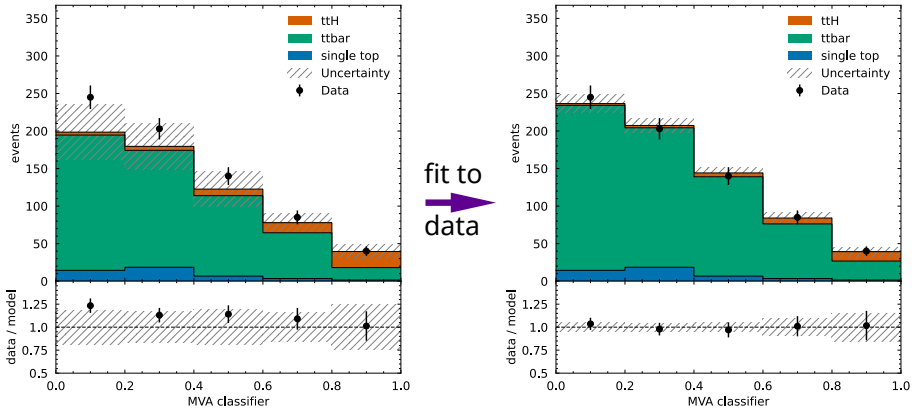
visualization of individual template histograms



event yield table

sample	Control region	Signal region
single top	44.74	0.35
ttbar	635.98	13.28
z_ttH	30.90	1.80
total	711.61 ± 28.28	15.43 ± 2.69
data	713.00	14.00

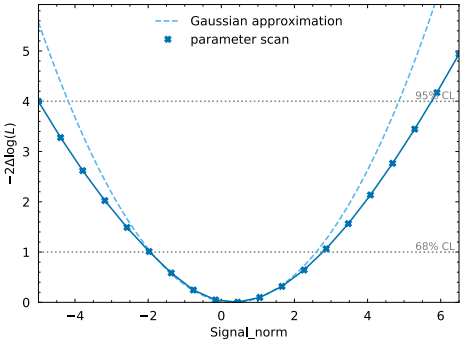
fit model visualization



Statistical inference

- implementations for all **common inference tasks** exist
  - includes associated visualizations
  - results validated against **ROOT**-based implementation

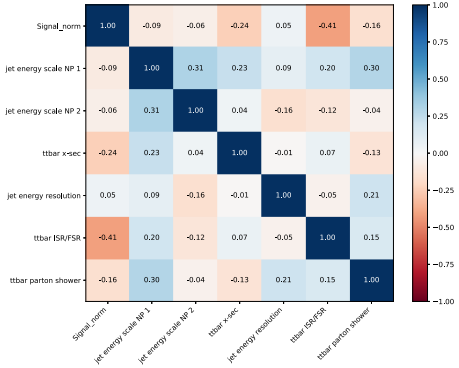
likelihood scans



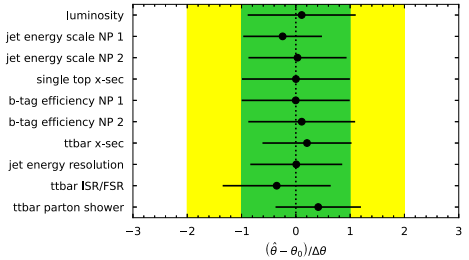
discovery significance

```
$ cabinetry significance workspaces/example_workspace.json
INFO - cabinetry.fit - calculating discovery significance
INFO - cabinetry.fit - observed p-value: 1.13053295%
INFO - cabinetry.fit - observed significance: 2.280
INFO - cabinetry.fit - expected p-value: 0.42110716%
INFO - cabinetry.fit - expected significance: 2.635
```

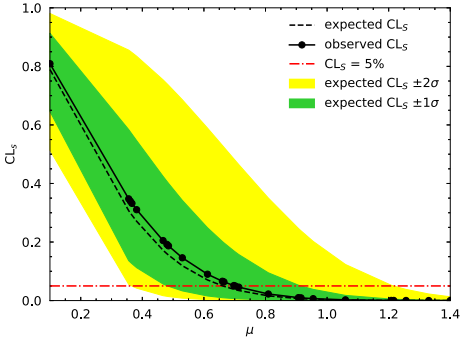
parameter correlations



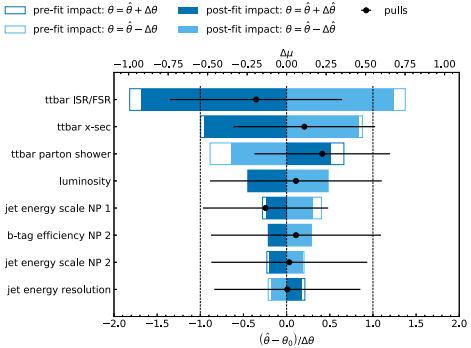
nuisance parameter pulls



upper parameter limits



nuisance parameter impacts



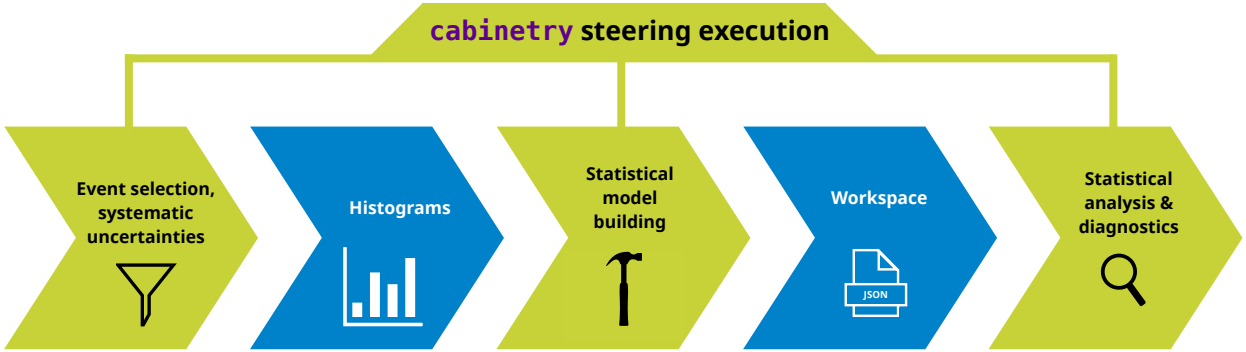
Future directions

- **`cabinetry`** is being **actively developed**
  - everything shown here today is available in **`cabinetry`** version 0.2.3

- **next steps and goals:**
  - short term: improved visualization API for simplified figure handling and customization (#113, #142)
  - support histogram inputs (#219)
  - more generic handling of templates related to interpolations (#51)
  - longer term: support end-to-end automatic differentiation (#233)
    - optimize analysis selection and design via gradient descent, see **`neos`** (PyHEP 2020 talk)
  - *your ideas?*
    - your thoughts and contributions are welcome!

Summary

- **`cabinetry`** is
  - a modular, Python-based library to create and operate statistical models for inference with template fits
  - leveraging the power of many libraries in a growing Python HEP ecosystem
  - openly developed on GitHub
  - available to try it out yourself on Binder!

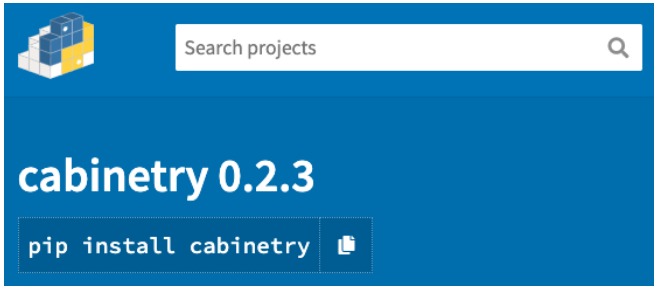


Backup

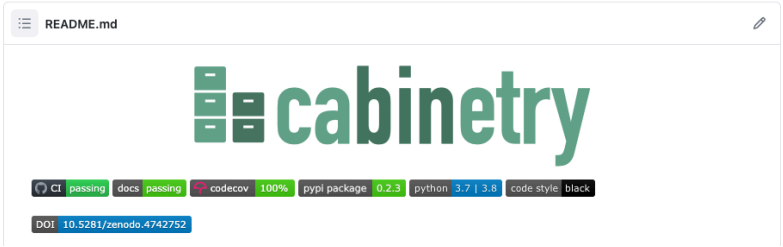
Links to `cabinetry`

- **`cabinetry`:**
  - can be installed via `$ pip install cabinetry`
    - `cabinetry[contrib]` for extra features
  - is open source and
    - developed on GitHub
    - published on PyPI
    - documented on Read the Docs
    - part of IRIS-HEP

cabinetry on PyPI



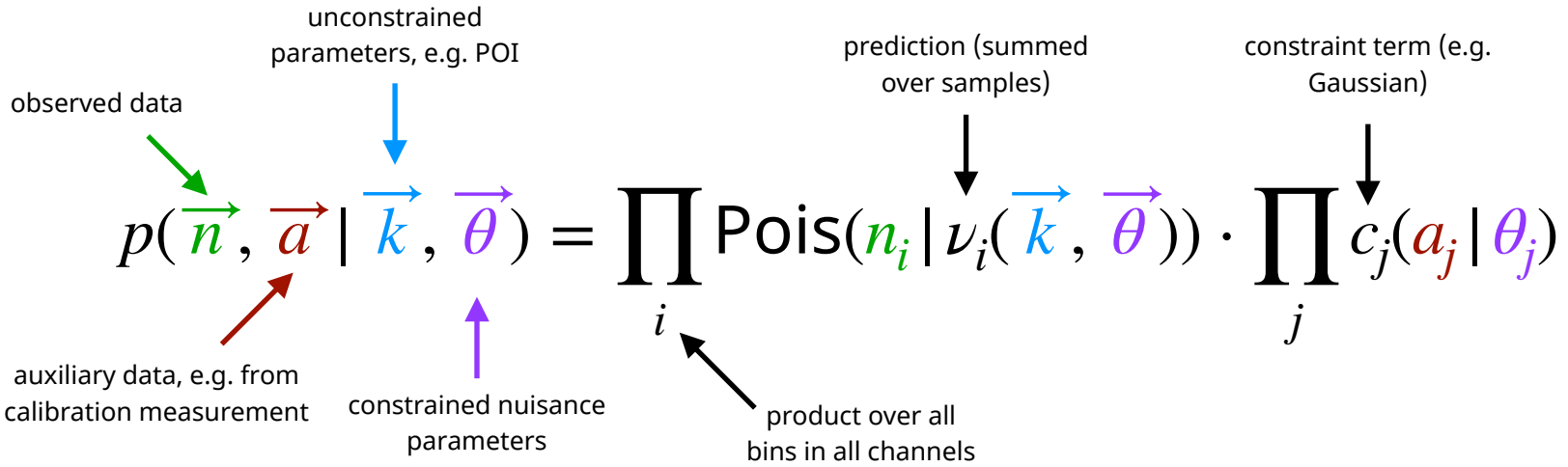
cabinetry on GitHub



documentation on Read the Docs

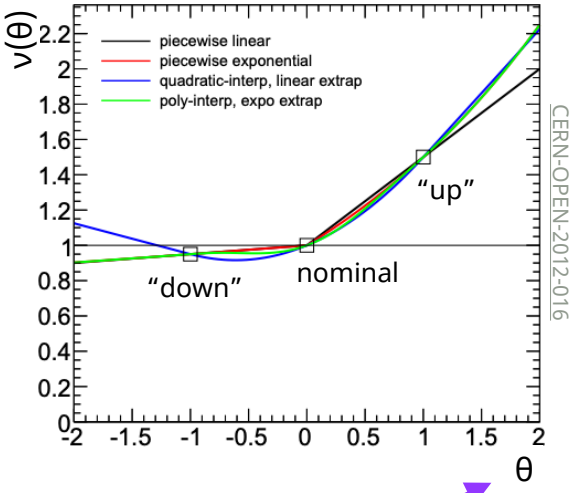
Statistical analysis: the `HistFactory` model

- **`HistFactory`** is the standard model used in ATLAS for **binned statistical analysis**
  - **`pyhf`** is a python implementation of this model
  - the **`HistFactory`** model specifies how to construct the likelihood function
  - **`cabinetry`** turns a declarative specification about cuts, systematics etc. into a statistical model
  - **`pyhf`** turns that model into a likelihood function



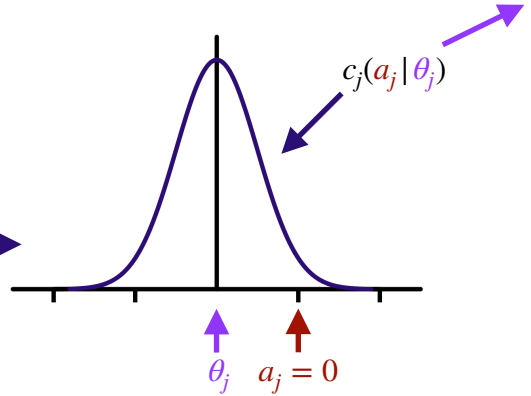
Systematic uncertainties with HistFactory

- common **systematic uncertainties** specified with **two template histograms**
  - "up variation": model prediction for $\theta = +1$
  - "down variation": model prediction for $\theta = -1$
  - interpolation & extrapolation provides model predictions ν for *any* $\vec{\theta}$



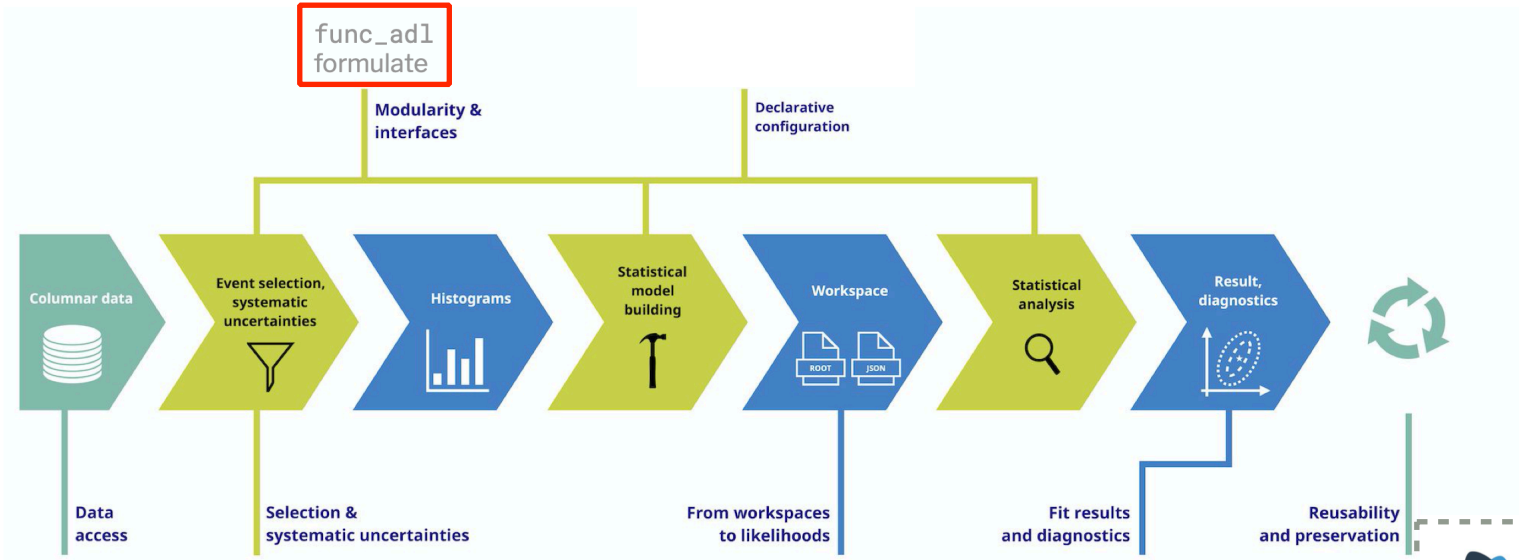
- **Gaussian constraint terms** used to model auxiliary measurements (in most cases)
  - centered around nuisance parameter (NP)
  - normalized width ($\sigma = 1$) and mean (auxiliary data $a_j = 0$)
  - penalty for pulling NP away from best-fit auxiliary measurement value

$$p(\vec{n}, \vec{a} \mid \vec{k}, \vec{\theta}) = \prod_i \text{Pois}(n_i \mid \nu_i(\vec{k}, \vec{\theta})) \cdot \prod_j c_j(a_j \mid \theta_j)$$



`cabinetry` within the broader ecosystem

possibilities for specifying cuts / translating between languages



Why `cabinetry`?

- **why `cabinetry`?**
  - pure Python and no ROOT dependency, fills gap in Python ecosystem
  - modular approach: avoid lock-in
    - benefit from growing columnar analysis ecosystem (**`coffea`** etc.)
  - openly developed, fully available to broader community beyond a specific experiment
  - follow good practices with extensive automated testing (see coverage)
  - chance to take different design decisions informed by years of experience with existing tools
    - decouple fit model specification and implementation
    - declarative approach, but allow custom code injection at core steps in the workflow

- **why the name?**
  - a workspace is like a cabinet: it organizes data into many bins (like drawers in a cabinet)
  - the building of these “workspace cabinets” is **`cabinetry`**