Template fits with `pyhf` & `cabinetry`

Alexander Held[1]

[1] University of Wisconsin-Madison

US ATLAS / IRIS-HEP Analysis Software Training Event 2024
https://indico.cern.ch/event/1376945/
July 19, 2024

This work was supported by the U.S. National Science Foundation (NSF) under Cooperative Agreements OAC-1836650 and PHY-2323298.

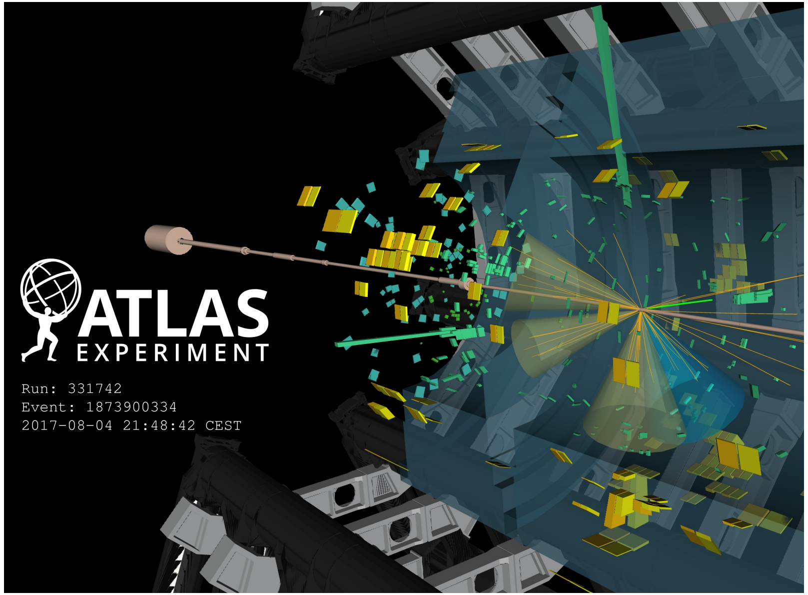
Big picture: turning collisions into publications

- **What we want:** statements about physical parameters θ, given data x collected by an experiment
    - connection: the likelihood $L_x(\theta) = p(x \mid \theta)$ — key ingredient for all subsequent statistical inference

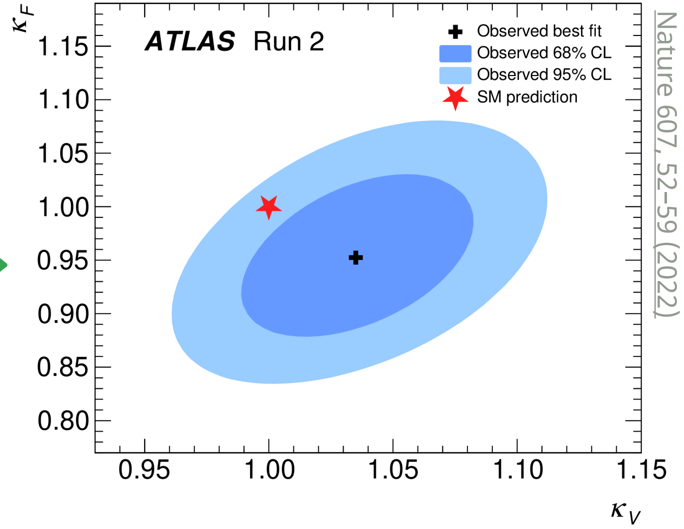
observations x

statements about parameters θ

Phys. Lett. B 784 (2018) 173



$p(x \mid \theta)$

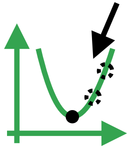


Nature 607, 52–59 (2022)

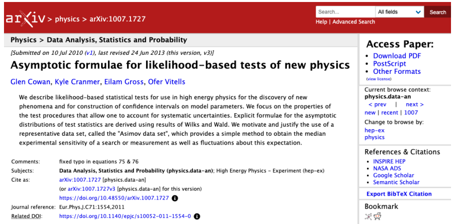
Statistical inference based on the likelihood (ratio)

- **Likelihood function** $L_x(\theta) = p(x \mid \theta)$ is the **key ingredient** for statistical inference
  - usually a function of many parameters: those we want to measure and nuisance parameters (NPs)
  - we typically use the *profile likelihood*: NP values are chosen to maximize the likelihood

- **Measure a parameter:**
  - minimize $-\log L_x(\theta)$



- **Discovery significance, parameter limits, …**
  - use test statistics based on profile likelihood ratio $\lambda(\mu) = \dfrac{L(\mu, \hat{\hat{\theta}})}{L(\hat{\mu}, \hat{\theta})}$
  - and/or Neyman construction


arXiv > physics > arXiv:1007.1727

Search... All fields Search

Help | Advanced Search

Physics > Data Analysis, Statistics and Probability

[Submitted on 10 Jul 2010 (v1), last revised 24 Jun 2013 (this version, v3)]

Asymptotic formulae for likelihood-based tests of new physics

Glen Cowan, Kyle Cranmer, Eilam Gross, Ofer Vitells

We describe likelihood-based statistical tests for use in high energy physics for the discovery of new phenomena and for construction of confidence intervals on model parameters. We focus on the properties of the test procedures that allow one to account for systematic uncertainties. Explicit formulae for the asymptotic distributions of test statistics are derived using results of Wilks and Wald. We motivate and justify the use of a representative data set, called the "Asimov data set", which provides a simple method to obtain the median experimental sensitivity of a search or measurement as well as fluctuations about this expectation.

Comments:	fixed typo in equations 75 & 76
Subjects:	**Data Analysis, Statistics and Probability (physics.data-an)**; High Energy Physics - Experiment (hep-ex)
Cite as:	arXiv:1007.1727 [physics.data-an]
	(or arXiv:1007.1727v3 [physics.data-an] for this version)
	https://doi.org/10.48550/arXiv.1007.1727
Journal reference:	Eur.Phys.J.C71:1554,2011
Related DOI:	https://doi.org/10.1140/epjc/s10052-011-1554-0

Access Paper:
- Download PDF
- PostScript
- Other Formats

(view license)

Current browse context:
physics.data-an
< prev | next >
new | recent | 1007
Change to browse by:
hep-ex
physics

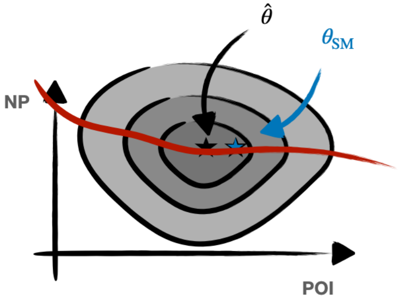
References & Citations
- INSPIRE HEP
- NASA ADS
- Google Scholar
- Semantic Scholar

Export BibTeX Citation

Bookmark

arXiv.1007.1727

full likelihood



profile NPs

profile likelihood

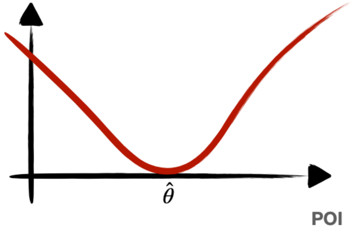


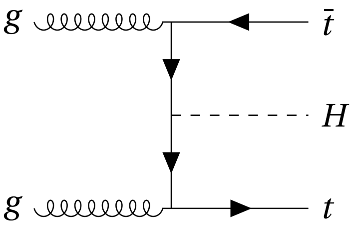
figure credit: L. Heinrich

An intractable likelihood function

- We **need $p(x \mid \theta)$** — unfortunately this very high-dimensional **integral** is ***intractable***, **cannot evaluate** this

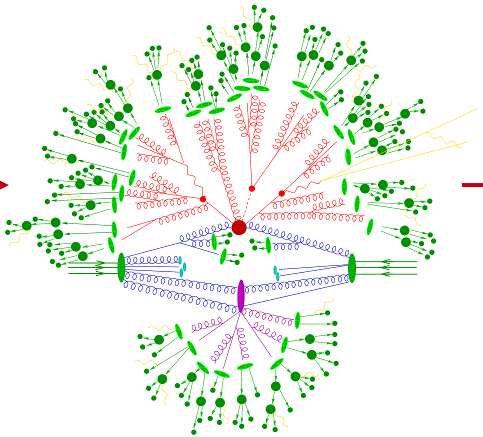
$$p\left(x \mid \theta\right) = \int \mathrm{d}z_D \mathrm{d}z_S \mathrm{d}z_P \; p\left(x \mid z_D\right) p\left(z_D \mid z_S\right) p\left(z_S \mid z_P\right) p\left(z_P \mid \theta\right)$$

parton level z_P



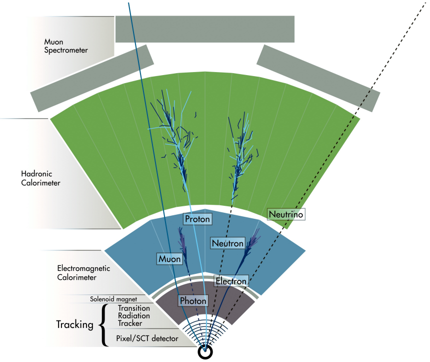
The dependence on parameters θ is here.

parton shower z_S



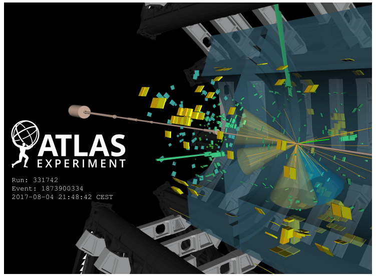
JHEP 0902 (2009) 007

detector interaction z_D



CERN-EX-1301009

observables x



Phys. Lett. B 784 (2018) 173

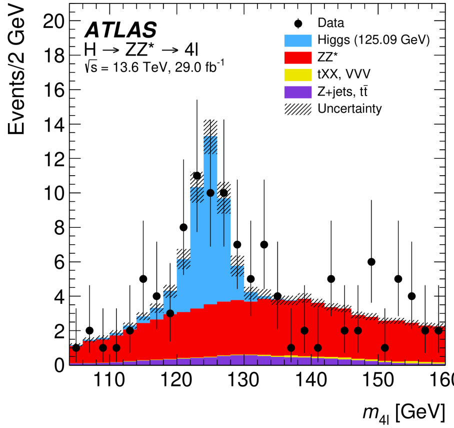
Density estimation & summary statistics

- There is one thing we *can* do: **simulate samples** $x_i \sim p(x \mid \theta)$
  - use MC samples to estimate the density $p(x \mid \theta)$, e.g. by filling histograms with the samples x_i

- Histograms are hit by the **curse of dimensionality**
  - number of samples x_i needed scales exponentially with dimension of observation

- We use **summary statistics** to reduce dimensionality of our measurements
  - operate on objects like jets instead of detector channel responses
  - use physicists & machine learning to efficiently compress information

- **Challenge:** finding the right low-dimensional summary statistic — crucial for sensitivity



Eur. Phys. J. C 84 (2024) 78

HistFactory & pyhf

The `HistFactory` model: overview

- **`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
  - models can be serialized to *workspaces*

unconstrained parameters, e.g. POI

observed data

prediction (summed over samples)

constraint term (e.g. Gaussian)

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

auxiliary data, e.g. from calibration measurement

constrained nuisance parameters

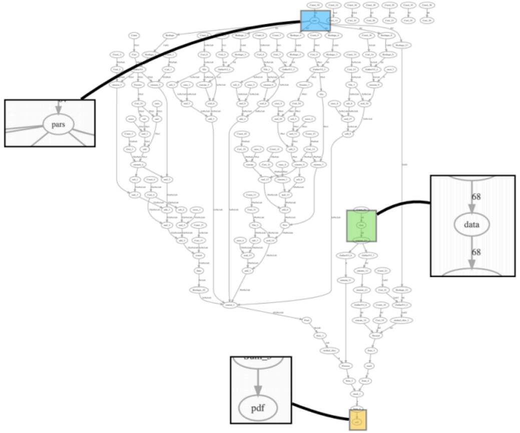
product over all bins in all channels

HistFactory: implementations

- **Until 2018**, the **HistFactory** model had only been implemented in **ROOT**
  - using RooFit, with RooStats available for statistical inference

- **pyhf** implements the **HistFactory** model in **pure Python** (`pip install pyhf`)
  - leverages tensor backends: efficient vectorized calculations & hardware acceleration
    - can automatically differentiate through statistical model (computational graph)
      - exact gradients for minimizers
      - enables end-to-end analysis optimization: **neos**
    - backend-agnostic API (and CLI)

computational graph
for HistFactory



example: autodiff through model yield prediction (e.g. for uncertainty propagation) *it just works!*

```
from jax import jacfwd
import pyhf

pyhf.set_backend("jax")

model = pyhf.simplemodels.correlated_background(
    signal=[12.0, 11.0],
    bkg=[50.0, 52.0],
    bkg_up=[45.0, 53.0],
    bkg_down=[55.0, 51.0],
)

bin_yield = lambda val: model.expected_data([val, 0.0], include_auxdata=False)

par_val = 0.3
print("yield", bin_yield(par_val))
# derivative is -5, -1 which makes sense given the templates + linear interpolation
print("jacfwd", jacfwd(bin_yield)(par_val))
```

A HistFactory JSON workspace with pyhf

- JSON structure maps directly to workspace structure
  - highly human-readable!

```json
{
    "channels": [
        {
            "name": "SR",
            "samples": [
                {
                    "data": [10.0, 15.0],
                    "modifiers": [
                        {"data": null, "name": "mu", "type": "normfactor"}
                    ],
                    "name": "Signal"
                },
                {
                    "data": [50.0, 45.0],
                    "modifiers": [
                        {"data": {"hi": 1.1, "lo": 0.9}, "name": "Modeling_unc", "type": "normsys"}
                    ],
                    "name": "Background"
                }
            ]
        }
    ],
    "measurements": [
        {
            "config": {"parameters": [], "poi": "mu"},
            "name": "minimal_example"
        }
    ],
    "observations": [{"data": [60.0, 60.0], "name": "SR"}],
    "version": "1.0.0"
}
```

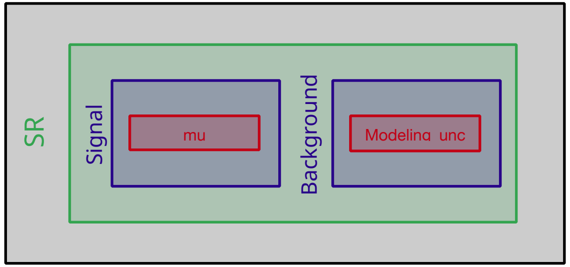
single channel

two samples

modifiers

measurement configuration

observed data



pyhf: summary

- **pyhf** provides
  - a declarative JSON schema for workspaces, used for statistical model publication and reinterpretation
  - a HistFactory implementation in Python that leverages tensor backends

- **pyhf** is a **library** exposing an API providing relevant functionality also found in RooFit, HistFactory and RooStats
  - it does not provide high-level functionality which applications like HistFitter, TRExFitter, WSMaker focus on
  - examples of things the pyhf API provides:
    - model yield prediction & NLL given parameters, details about model structure, MLE, workspace pruning
  - examples of things not in scope for pyhf:
    - post-fit model prediction plots, nuisance parameter ranking

Model construction & use with `cabinetry`

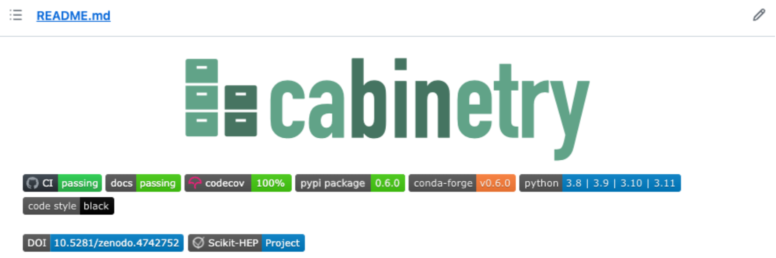
Intro: constructing and using statistical models

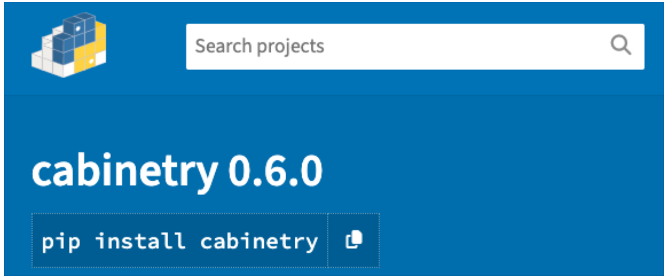
- **Binned template fits** are widely used for **statistical inference**

- **Statistical models** used in particle physics are often **rather complex**
  - lots of book-keeping to handle $\mathcal{O}(10\mathrm{k})$ histograms for typical ATLAS applications
  - frequent model modifications needed for tests & debugging

- A set of **tools** emerged over time to aid with **model construction** and **inference**
  - In ATLAS: `HistFitter` and many more internal tools, `Combine` for CMS
  - (some of) these tools also provide utilities to visualize inference result & simplify debugging

The cabinetry library



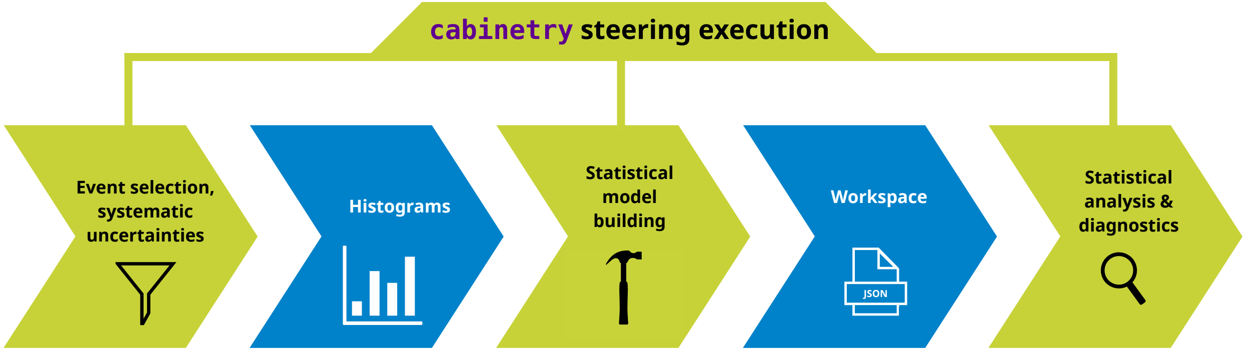


- **cabinetry** is a modern **Python library** for constructing and/or operating **HistFactory** models
  - ❯`pip install cabinetry`
  - uses pyhf, integrates seamlessly with the Python HEP ecosystem
  - modular design: use the pieces of `cabinetry` you need
  - part of the Scikit-HEP project

- **cabinetry** ↔ **pyhf** is roughly like **TRExFitter** ↔ **ROOT (RooFit, HistFactory, RooStats)**

Working with cabinetry

- **cabinetry** is a **Python library** for creating and operating HistFactory models
  - **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 or collects provided 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"

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]
```

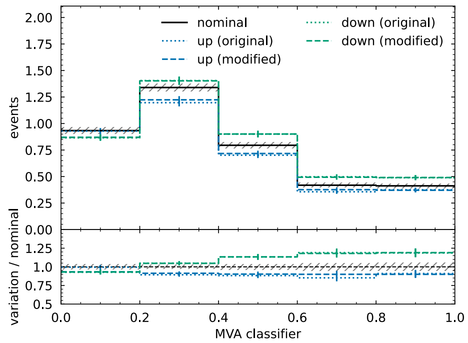
list of systematic uncertainties

list of normalization factors

Template histograms and workspace building

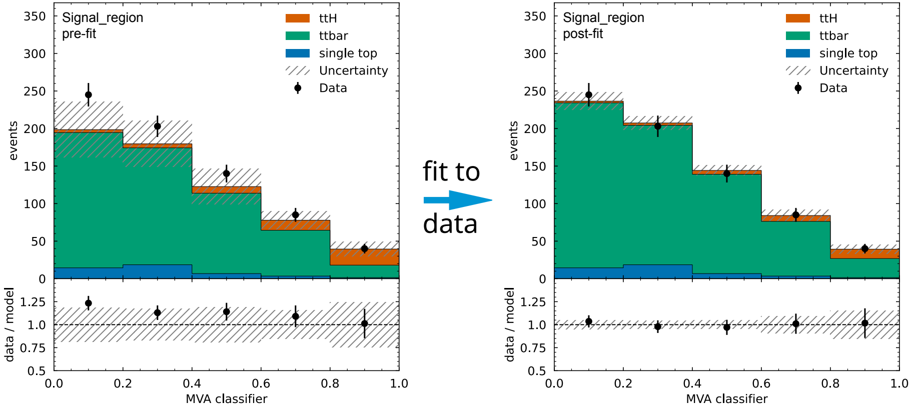
- **Workspaces construction** happens in three steps:
  1) create template histograms from columnar data following config instructions
     - backends execute instructions (default: `uproot`, experimental: coffea)
     - alternatively: collect existing user-provided histograms
  2) optional: apply post-processing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

visualization of individual template histograms



- Utilities provided to **visualize and debug** fit model

fit model visualization



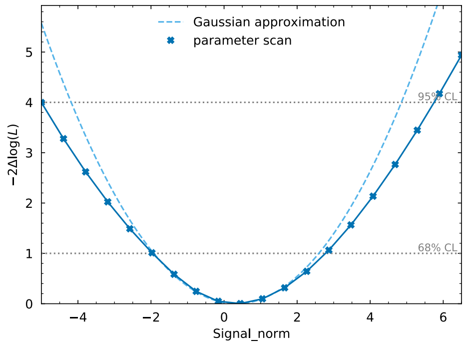
event yield table

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

Statistical inference

- Implementations for **common inference tasks** exist
  - includes associated visualizations

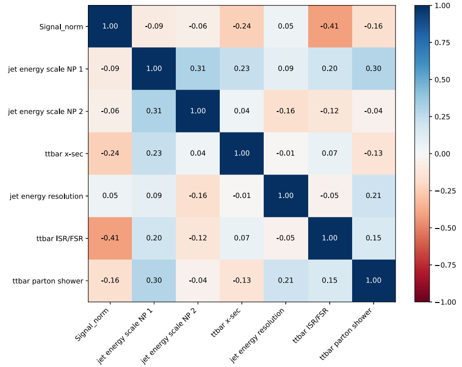
likelihood scans



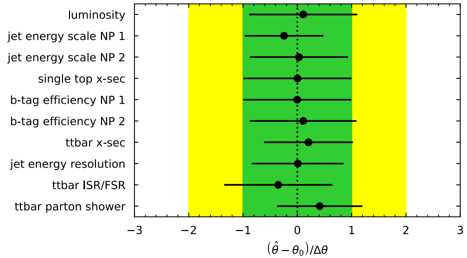
discovery significance

```
$ 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
```

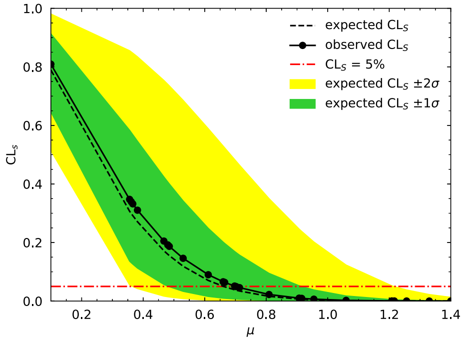
parameter correlations



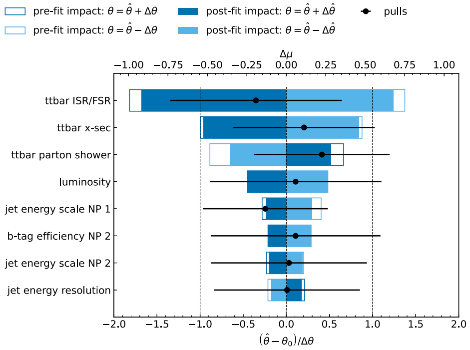
nuisance parameter pulls



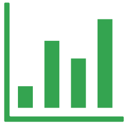
upper parameter limits



nuisance parameter impacts



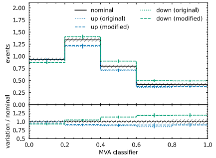
Full workflow example



generate histograms for all samples: nominal & variations

can read histograms or ntuples *with* `cabinetry`

cabinetry



→

workspace: serialized statistical model

built by `cabinetry`
or from *external source*

cabinetry


```json
{
    "channels": [
        {
            "name": "SR",
            "samples": [
                {
                    "data": [10.0, 15.0],
                    "modifiers": [
                        {"data": null, "name": "mu", "type": "normfactor"}
                    ],
                    "name": "Signal"
                },
                {
                    "data": [50.0, 45.0],
                    "modifiers": [
                        {"data": {"hi": 1.1, "lo": 0.9}, "name": "Modeling_unc", "type": "normsys"}
                    ],
                    "name": "Background"
                }
            ]
        }
    ],
    "measurements": [
        {
            "config": {"parameters": [], "poi": "mu"},
            "name": "minimal_example"
        }
    ],
    "observations": [{"data": [60.0, 60.0], "name": "SR"}],
    "version": "1.0.0"
}
```


→

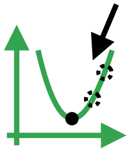
$$L(\vec{k},\vec{\theta}) = p(\vec{n},\vec{a} \mid \vec{k},\vec{\theta})$$

turn into likelihood function

a function, given (observed + auxiliary) data $\vec{n},\vec{a}$
done via `pyhf`

pyhf ∂ifferentiable ℒikelihoods

→



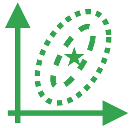
minimize NLL

using function minimizer, e.g. MINUIT (MIGRAD algorithm)
via `iminuit`

iminuit

can wrap these steps with `cabinetry`

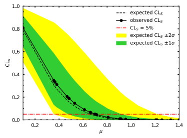
→



inference results

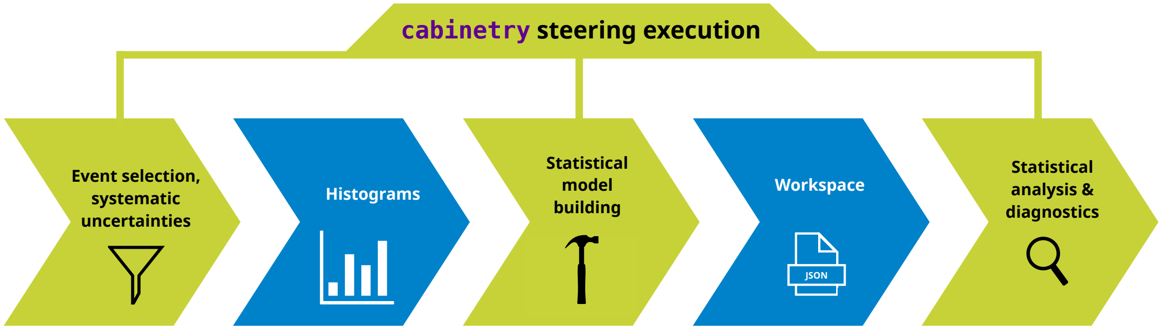
plots & tables
produced by `cabinetry`

cabinetry



`cabinetry`: summary

- **`cabinetry`** is
  - a modular Python library to create and/or operate statistical models for inference with template fits
  - built upon the powerful and growing Python HEP ecosystem
  - using a slightly different design approach to other tools: more library, less framework
    - analyzers will generally need to write some code: hopefully less "black box" and more flexible, but more work



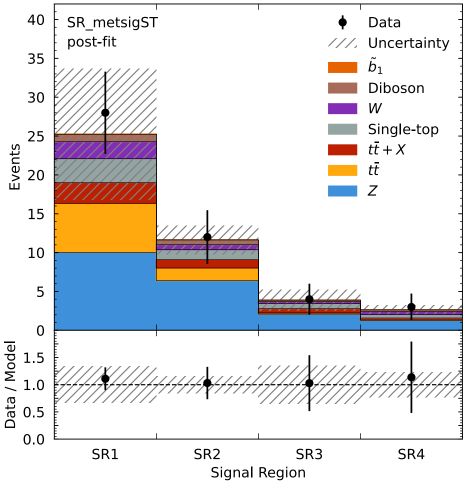
Backup

Working with an unknown workspace

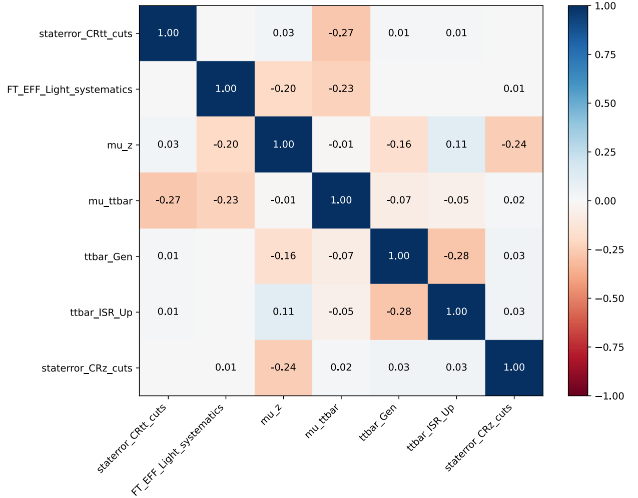
- Pick a **workspace** from **HEPData**: 10.17182/hepdata.89408.v3 (analysis: JHEP 12 (2019) 060)
  - download workspace with `pyhf`
  - perform inference and visualize results with `cabinetry`
  - can use inference features regardless of how a workspace was built, functionality factorizes!
- See arXiv:2109.04981 and try it on Binder

Search for bottom-squark pair production with the ATLAS detector in final states containing Higgs bosons, *b*-jets and missing transverse momentum

```python
import json
import cabinetry
import pyhf
from cabinetry.model_utils import prediction
from pyhf.contrib.utils import download

# download the ATLAS bottom-squarks analysis probability models from HEPData
download("https://www.hepdata.net/record/resource/1935437?view=true", "bottom-squarks")

# construct a workspace from a background-only model and a signal hypothesis
bkg_only_workspace = pyhf.Workspace(json.load(open("bottom-squarks/RegionC/BkgOnly.json")))
patchset = pyhf.PatchSet(json.load(open("bottom-squarks/RegionC/patchset.json")))
workspace = patchset.apply(bkg_only_workspace, "sbottom_600_280_150")

# construct the probability model and observations
model, data = cabinetry.model_utils.model_and_data(workspace)

# produce visualizations of the pre-fit model and observed data
prefit_model = prediction(model)
cabinetry.visualize.data_mc(prefit_model, data)

# fit the model to the observed data
fit_results = cabinetry.fit.fit(model, data)

# produce visualizations of the post-fit model and observed data
postfit_model = prediction(model, fit_results=fit_results)
cabinetry.visualize.data_mc(postfit_model, data)
```



(workspace contains additional channels not shown here)



pyhf tutorial material

- **`pyhf`** tutorial: https://pyhf.github.io/pyhf-tutorial/
  - especially recommended:
    - auxiliary data (helpful to understand beyond just `pyhf`)
    - HistFactory and modifier sections (including interactive model exploration!)

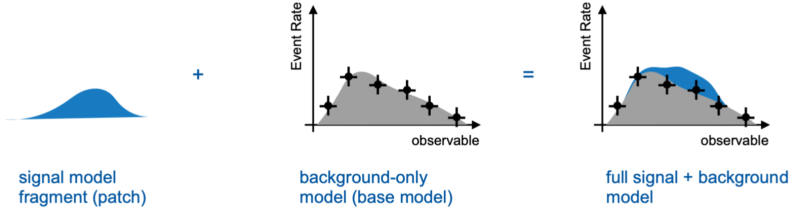
- **`cabinetry`** tutorial: https://github.com/cabinetry/cabinetry-tutorials
  - Binder links in README

- Happy to go into more detail regarding any points & work through examples with code! Feel free to ask any questions.

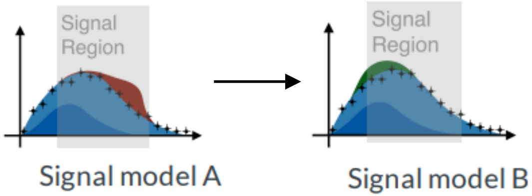
Model patching

- Especially in searches, it is common to use **many different models that slightly differ**
  - same background model but many different signal hypotheses (e.g. different resonance masses)

- It is possible to **edit and swap out pieces of a workspace** via **[JSON Patch]**
  - e.g. add a new component to your model



  - or replace your signal model



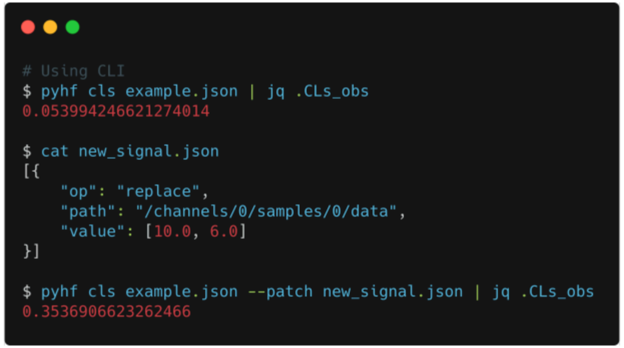
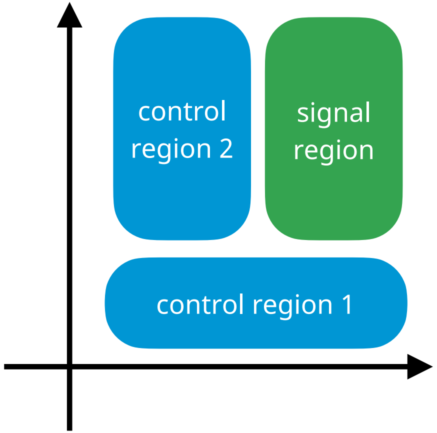


figure credit: Lukas Heinrich

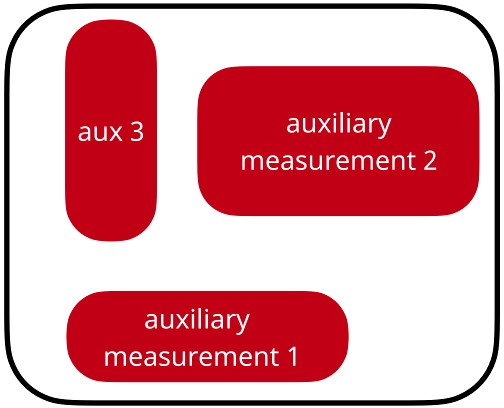
A measurement: primary and auxiliary observables

primary observables $\vec{x}$



data in our analysis

auxiliary observables $\vec{a}$



calibration measurements + theory
(assumed to be statistically independent)

- Our models are a **combination of primary and auxiliary measurements** $p_{primary}(\vec{x} \mid \vec{\nu}) \cdot p_{aux}(\vec{a})$
  - auxiliary: both experimental (e.g. detector calibration) and theory (e.g. changes in simulation)

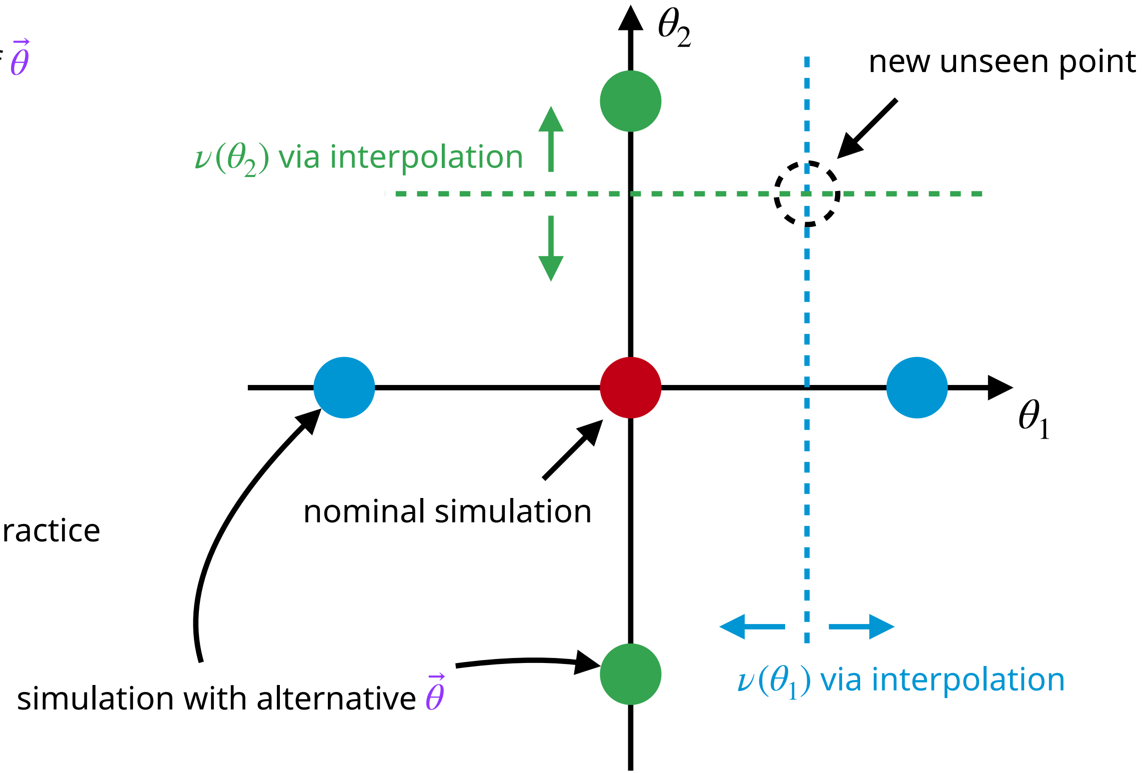
Systematic variations

- Need to model $\nu(\vec{k}, \vec{\theta})$ for any value of nuisance parameters $\vec{\theta}$ encoding systematic uncertainties

- **Ideal case:** just run simulator for any value of $\vec{\theta}$
  - not computationally feasible in practice

- **Instead:** pick some values & **interpolate**
  - in practice we use on-axis variations
  - variations typically are "one at a time"

- Lots of **assumptions** here that we rely on in practice
  - where to simulate
  - interpolation choice
  - effects factorize

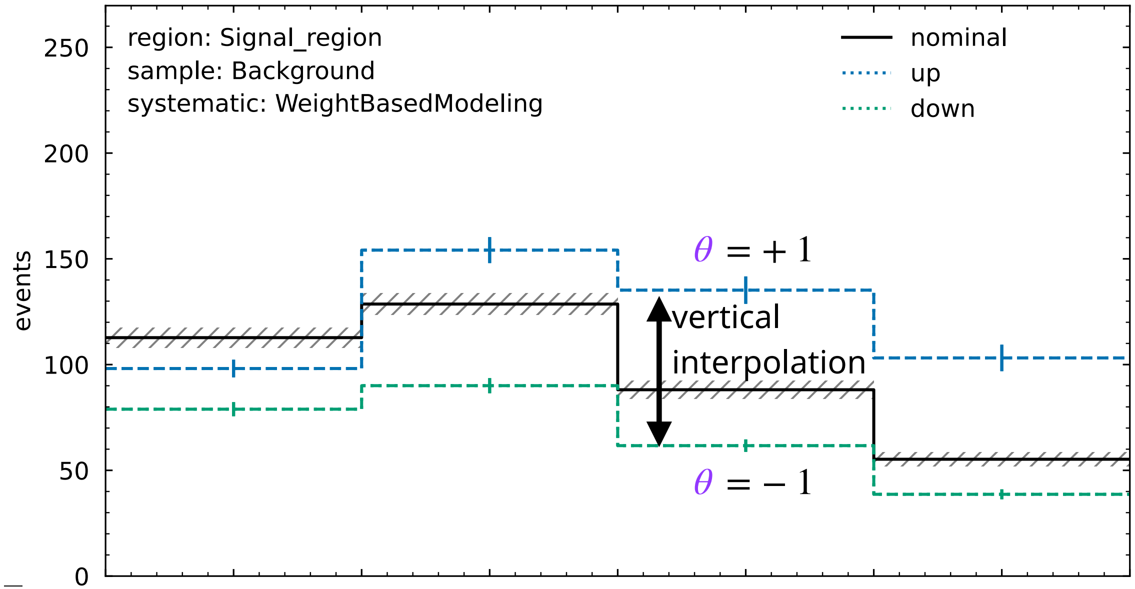


Interpolating between points

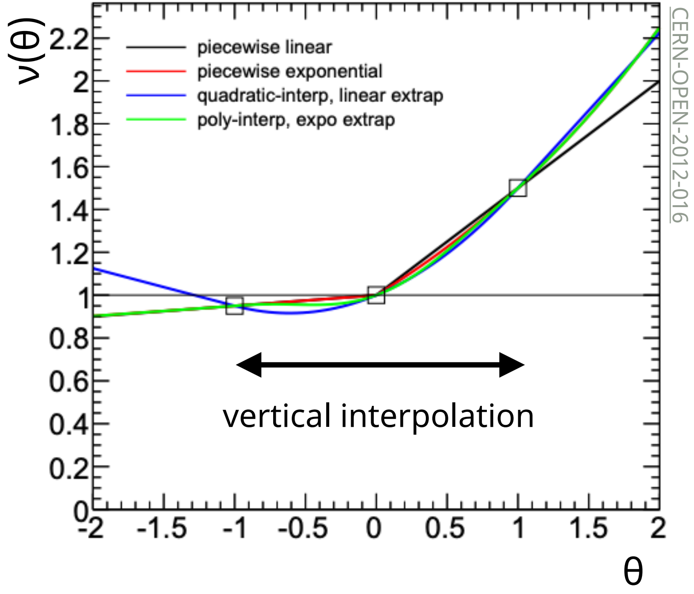
interpolation approach is technically relatively simple
➡ limit risk of surprises

- Use model prediction $\nu_i(\vec{k}, \vec{\theta})$ for three points θ, **interpolate to generalize**
  - interpolation is typically "vertical", other approaches exist (but more specialized)
  - note: information about statistical uncertainties in varied templates is lost here (arXiv:1809.05778)

toy example: distributions for $\theta = -1, 0, +1$

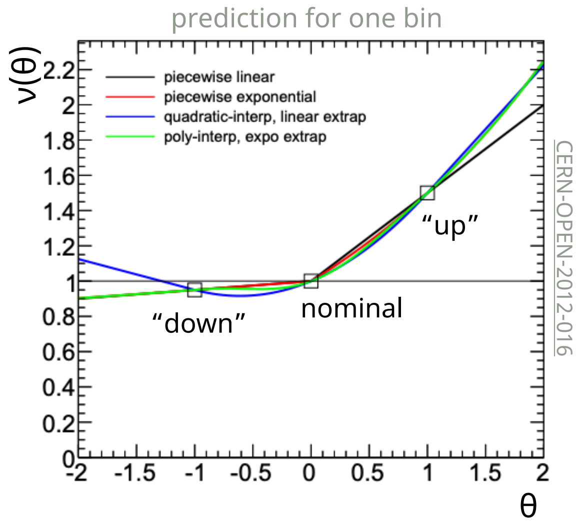


interpolation in one bin



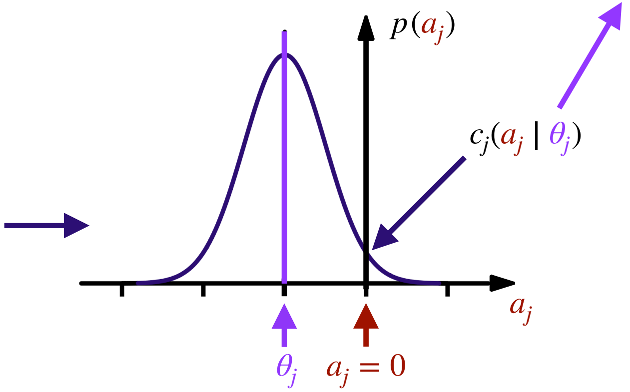
Systematic uncertainties with `HistFactory`

- 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) θ_j
  - normalized width ($\sigma = 1$) and mean (auxiliary data $a_j = 0$)
  - penalty for pulling NP away from best-fit auxiliary measurement value

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

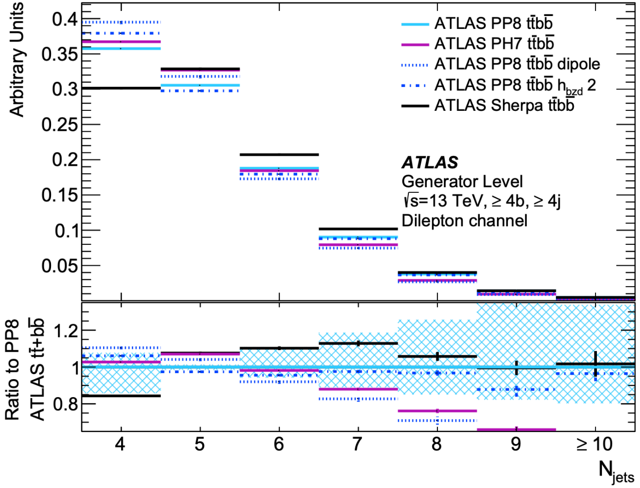


Complication: two-point systematics

two-point systematics are inherently problematic and deserve special attention

- Sometimes have cases where **variations in simulator chain are discrete**
  - e.g. choice of one simulator vs alternative
- Typical treatment: **interpolate to treat as continuous, symmetrize**
  - lots of assumptions here, but need to make a choice to profile
- Especially **tricky to deal with** when these play a large role
  - concerns about overly constraining uncertainty of nuisance parameter
  - best-fit model prediction may lie away from both choices

modeling choices for main background of ttH(bb)



LHCHWG-2022-003