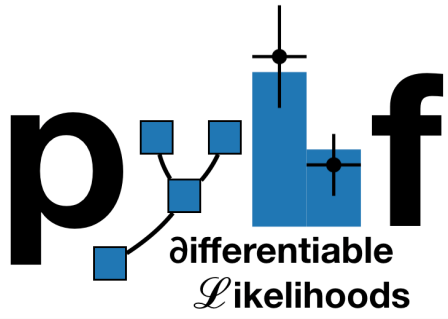
Pyhf and Combine Converter

Peter Ridolfi

Mentored by Alexander Held and Matthew Feickert

GitHub repository: https://github.com/peterridolfi/Pyhf-to-Combine-converter.git



Motivation

- Analyses done on ATLAS and CMS experiments are very similar
  - In many cases, an analysis done by pyhf on ATLAS data can be done by Combine on the same data
- pyhf is much more “lightweight” than Combine, so it may be easier, faster, or more efficient for an analysis of CMS data
- Combine offers many more options for statistical inference and analysis, so it may be advantageous for a more involved analysis of ATLAS data
- However, people in the CMS experiment are not trained to use pyhf, and vise versa
- To take advantage of these nuances, a conversion is necessary

Project overview

- Bidirectional translation between pyhf and CMS Combine
- Different formats for model specifications (HistFactory JSON specifications vs Datacard text specifications)
- Specifications in both models require many histograms and counts to be given for accurate inferences
- Different “building blocks” that need to be converted
- Some parts are not supported, must be handled in the translation

Workspace examples

```
imax 1
jmax 3
kmax 1

---------------------------------
bin b1
observation 12
---------------------------------
bin     b1    b1    b1    b1
process     sig    bkg1    bkg2   bkg3
process     0     1    2     3
rate     5     8      1     2
---------------------------------
lumi lnN - 1.01 1.01 1.01
---------------------------------
```

Figure 1 (left): A simple Combine datacard specification


```
{
    "channels": [
      {
        "name": "b1",
        "samples": [
          {
            "name": "sig",
            "data": [
              5
            ],
            "modifiers": [
              {
                "name": "mu",
                "type": "normfactor",
                "data": null
              }
            ]
          },
          {
            "name": "bkg1",
            "data": [
              8
            ],
            "modifiers": [
              {
                "name": "lumi",
                "type": "normsys",
                "data": {"hi": 1.01, "lo": 0.99009901}
              }
            ]
          },
```


Figure 2 (right): An excerpt from pyhf json specifications for the same model

How to run the translation

- Designed to be run in an environment with both Combine and pyhf supported
- A docker container is recommended, instructions on the GitHub page
- Run from the CLI, only the JSON file or the datacard.txt file is necessary to input
- If using shapes (multiple bins for Combine->pyhf, the shapes root file is also required to be in the same directory

Pyhf->Combine

- Parse through JSON model and translate to a Datacard object
- After Datacard object is populated, manually write datacard file
- Generate histograms and put them in a shapes root file for Combine
  - I used Hist and uproot for this
- Identify samples that are signal based on the presence of certain modifiers

Combine->pyhf

- Generate Datacard object with DatacardParser
- Build JSON spec from scratch using this object
- Must read histograms in the shapes file to put binned data in the JSON
  - Using uproot
- Generate signal strength modifiers to act on signal in the JSON spec, add as measurements
- Combine systematics like lnU are not supported, must raise an error to the user

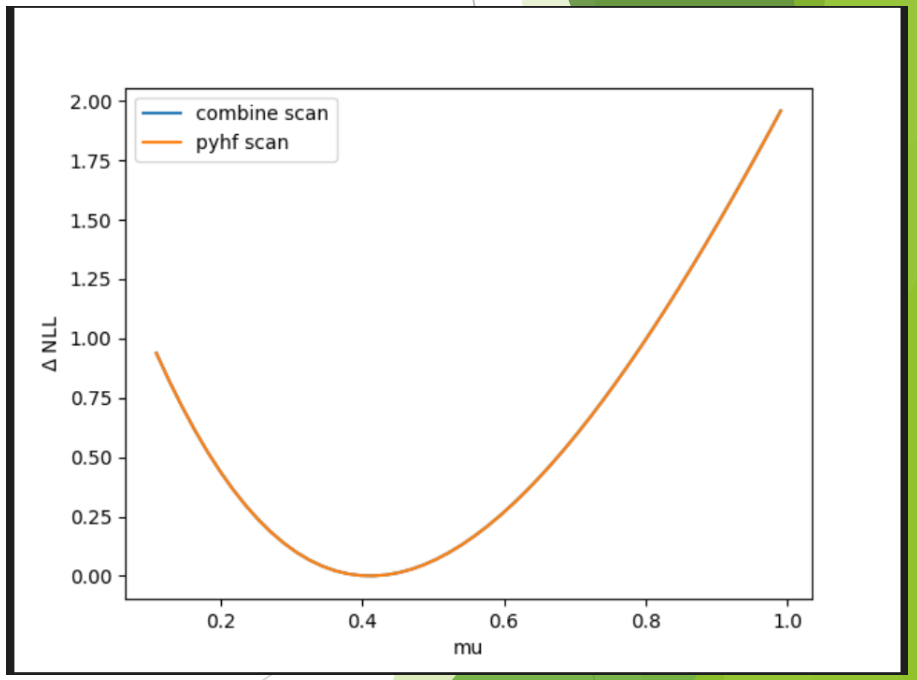
Important differences

- Uncertainty modifiers ‘staterror’ and ‘shapesys’ are handled differently in Combine (autoMCStats) – fits must be converted before comparing them
  - Pyhf values centered at 1, Combine centered at 0
- Combine implements a “shape/norm split”, whereas pyhf does not
  - By default, Combine adds a normalization systematic to any shape systematic (unless this is directly turned off)
  - In pyhf, a normalization modifier must be added by the user
  - Combine does not allow separate normalization and shape uncertainties-> normalizations must be specified with ‘shape?’ and the split must be turned off

Validation of the translation

- NLL plots generated for converted models to compare inferences
  - Many speedbumps
- Found perfect agreement for these scans (except for autoMCStats parameters, which is attributed to the difference in how it is implemented)
- Expected yields were calculated in converted models and shown to agree
- “Round trip” translation was done and the specs match

Figure 3: NLL scan for pyhf and Combine with “profiled” parameters (refits at each point)



Validation cont.

- Translation and fits were done on a realistically sized model, agreement persisted
- Pull plots were generated on the fitted parameter values to further prove model agreement
  - ‘Staterror’ and ‘shapesys’ have been omitted due to fundamental differences in value
  - Note: The miniscule pulls that were created for the Combine spec seems to be due to the prefit parameters that Combine created, as they were very close to the actual fit value, as opposed to the base prefit values (0 or 1) used by pyhf

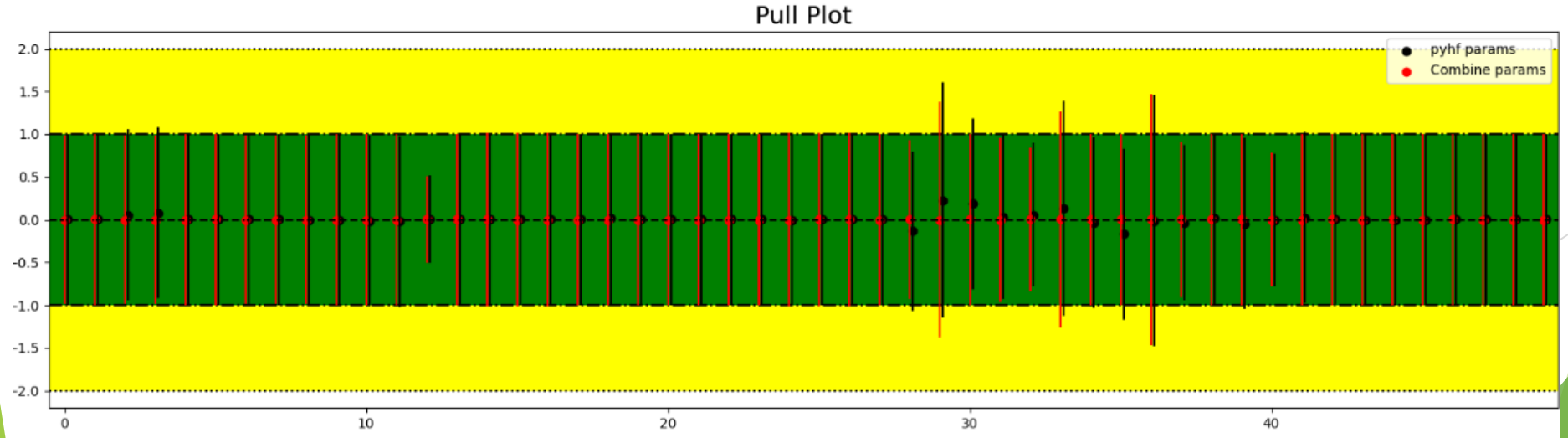


Figure 4: Pull plot of modifiers for a large model, with the pulls calculated from the pyhf spec in black and the Combine spec in red

Closing remarks

- The success of this project is big for the HEP community
  - There are many more options for how to analyze and display data, with no additional time spent learning new model specifications
- The script is currently available for public use on GitHub
  - https://github.com/peterridolfi/Pyhf-to-Combine-converter
- The project is also available as a python package on PyPI, so it can be easily installed using pip
  - pip install pyhf-combine-converter

Questions?

- Thank you for listening!