Bayesian Methodologies with pyhf

Matthew Feickert[1], Lukas Heinrich[2], and Malin Horstmann[2]

[1]University of Wisconsin-Madison, Madison, Wisconsin, USA
[2]Technical University of Munich, Munich, Germany

Abstract. bayesian_pyhf is a Python package that allows for the parallel Bayesian and frequentist evaluation of multi-channel binned statistical models. The Python library pyhf is used to build such models according to the HistFactory framework and already includes many frequentist inference methodologies. The pyhf-built models are then used as data-generating model for Bayesian inference and evaluated with the Python library PyMC. Based on Monte Carlo Chain Methods, PyMC allows for Bayesian modelling and together with the arviz library offers a wide range of Bayesian analysis tools.

1 Introduction

to construct HistFactory models within Julia are not yet readily available [1].
pyhf is a Python library that implements the HistFactory template and already allows for frequentist inference [4, 5]. It is the aim of this work to utilize the Python library PyMC and the automatic differentiation capabilities of pyhf to enable advanced Bayesian analysis for HistFactory models.

2 Bayesian Statistics for HistFactory Models

2.1 HistFactory Models and pyhf

In the HistFactory template, the expected event rates are dependent on two sets of parameters, free parameters and constraint parameters . In contrast to the free parameters, the are constrained by external data, whose impact has to be considered when building the statistical model. This can be done by assuming auxiliary measurements with observations a for each parameter . Each auxiliary measurement then corresponds to a constraint term c which is added to the likelihood and controls the compatibility of the value of the constraint parameter with their corresponding auxiliary measurements a [4–6]. For simplicity, the model for these auxiliary measurements are either Gaussian or Poisson distributions.
Taking the constraints into account, the resulting statistical model for event rates n and auxiliary measurements a is then given by [4, 5]:

$$\begin{aligned} p(x|\,) &= p_{\text{main}}(x_{\text{main}}|\,)\,p_{\text{aux}}(x_{\text{aux}}|\,) \\ &= p(n,a|\,;\,) = \prod_{c\in\text{channels}} \prod_{b\in\text{bins}} \text{Poiss}(n_{cb}|\,_{cb}(\,;\,)) \prod_{c} c\,(a\,|\,); \end{aligned} \tag{1}$$

where p_{main} (p_{aux}) and x_{main} (x_{main}) describe the actual and auxiliary statistical model and observations for parameters .

The pure-Python library pyhf implements the HistFactory formalism for the analysis of multi-channel binned statistical models [4, 5]. Statistical models can be stored as pure JSON files, allowing for integration with other statistics libraries. The numeric backend that pyhf uses is flexible, allowing for auto-differentiable tensor-backends.

2.2 Bayes' Theorem

Bayesian inference is governed by Bayes' theorem [7]:

$$p(\,;\,|x,a) = \frac{p(x,a|\,;\,)\,p(\,;\,)}{p(x,a)}. \tag{2}$$

It describes the updating of a prior probability distribution $p(\,;\,)$ to a posterior distribution $p(\,;\,|x)$ by multiplication with a data-generating model $p(x|\,;\,)$. $\,;\,$ are parameters of interest (POI) and constraint parameters respectively and x,a observations and auxiliary measurements and evidence $p(x,a)$. While a closed form solution of Eq. (2) is intractable due to the evidence, approximate solutions are viable via sampling methods (such as MCMC [8]) as only a tractable joint likelihood $p(x,a|\,;\,)p(\,;\,)$ is required.

[1] The LiteHF.jl [3] package is working towards HistFactory within a Julia context.

2.3 PyMC

PyMC is a Python library for building Bayesian models and already includes a wide range of cross-checks and plotting functions through its arviz -backend [8, 9].
The statistical models are evaluated using Monte Carlo chain methods (MCMC), where the posterior is represented by sampling from the prior distribution steered by the likelihood. PyMC also allows for the implementation of external models, which makes it suitable for performing Bayesian inference with pyhf -based HistFactory models.
Within PyMC a whole set of MCMC techniques is available, e.g. prior and posterior sampling or predictive sampling. The returned objects are arviz.InferenceData containers, for which again the whole set of analytic tools provided by the arviz library are available [9].

2.4 Prior Constraints from Auxiliary Measurements and Ur-Priors

In order to get a sampling representation of the posterior using MCMC methods the prior distribution and the HistFactory models are needed, i.e following Eq. 2:

$$p(; \ jx; a) \quad p(x; aj ; \ )p(; \ ): \tag{3}$$

While $p(x; aj ; \ )$ can be build using pyhf , the prior beliefs of the value of the parameters still have to be quanti ed. It would be possible to treat and equally, i.e. to determine some ur-prior $p(; \ )$ by hand and update with x and a in parallel. Ur-priors describe the belief about the parameter value before taking the observations into account.
The approach followed in this work however relies on a di erent treatment of the constraint priors $p( )$. It is based on separating the auxiliary observations a from the main inference step and using it instead in an initial inference step. In this step, Bayes' theorem is used to update ur-priors $p_{ur}( )$ with the auxiliary measurements a, thus incorporating the information gained from the auxiliary measurements a, see Eq. (4).

$$p(\ ja) \quad p(aj \ )p_{ur}(\ ) \tag{4}$$

The posteriors $p(\ ja)$ from Eq. (4) can then be used as prior belief in the main inference step.
This approach is useful, as the number of constraint measurements can get arbitrarily high which would imply high computational cost when updating $p( )$ and $p( )$ together. Splitting the auxiliary update of the constraint parameters into this initial step solves this issue based on the concept of conjugate priors. This concept dictates that for given sets of distributions for the priors and the data-generating model, the posterior distribution is of the same distribution family as the prior and can be given in closed form the priors and posteriors are then conjugate. Due to the limited possibilities for the auxiliary measurements (which can be either Gaussian or Poisson distributed), this concept is viable for the constraint parameters with corresponding Gaussian and Gamma distributed ur-priors, see Table 1 [7]. The freedom to choose these compatible ur-priors is justi ed as the priors will be dominated by the auxiliary measurements. Indeed, in the limit of very vague ur-priors the priors are completely dominated by the auxiliary measurements a:

$$\begin{aligned} &_{ur} \quad _{aux} \ ! \quad {}^{0}! \ a; \quad {}^{0}! \quad _{aux}; \\ &_{ur}; \ _{ur} ! \ 0 \ ! \quad {}^{0}; \ {}^{0}! \ a: \end{aligned} \tag{5}$$

As the impact of the auxiliary measurements can now be implemented in closed form, no sampling is needed for the implementation of the knowledge gained from the auxiliary measurements a.

Posterior	Data-Gen. Model	Ur-Prior
$N(\ \mid {}^{0}, {}^{0})$	$N(a\mid\ , {}_{aux})$	$N(\ \mid {}_{ur}, {}_{ur})$
$(\ \mid {}^{0}, {}^{0})$	$\mathrm{Poiss}(a\mid\ )$	$(\ \mid {}_{ur}, {}_{ur})$

Table 1: Conjugate priors for Poisson and Normal distributed auxiliary measurements a [7, 10].

While Table 1 fixes the general structure of the posterior distribution (to be used as priors in the main inference), the hyperparameters governing these still have to be determined.
For a single auxiliary measurement a, the hyperparameters for the Gaussian posteriors of $$ for some given ur-hyperparameters ${}_{ur}, {}_{ur}$ follow [10]:

$$ {}^{0} = \frac{{}^{2}_{aux}\ {}^{2}_{ur}}{{}^{2}_{aux} + {}^{2}_{ur}} \left(\frac{{}_{ur}}{{}^{2}_{ur}} + \frac{a}{{}^{2}_{aux}} \right); \qquad {}^{0} = \frac{{}^{2}_{aux}\ {}^{2}_{ur}}{{}^{2}_{aux} + {}^{2}_{ur}}; \tag{6} $$

The hyperparameters describing the posterior Gamma distribution for a single auxiliary observation a can for some given ur-hyperparameters ${}_{ur}, {}_{ur}$ be derived as:

$$ {}^{0} = {}_{ur} + a; \qquad {}^{0} = {}_{ur} + 1; \tag{7} $$

In contrast to the Gaussian constraints, the implementation of the Poisson constraints in pyhf comes with the following change of variable:

$$ \ ! \ = \frac{}{a}: \tag{8} $$

Using:

$$ p(\ )d\ = p(\ )d \tag{9} $$

the posterior Gamma distribution over $$ is then derived as:

$$ \begin{aligned} p(\ \mid a) \quad (\ \mid {}^{0}, {}^{0})\frac{d}{d} &= a\frac{^{1}}{(\ {}^{0})}e^{\ a} \\ (\ \mid {}^{0}, a\ {}^{0}) &= (\ \mid {}_{ur} + a; a({}_{ur} + 1)): \end{aligned} \tag{10} $$

Applying the methods described above to Eq. (2), the final form of Bayes' theorem used in this work then reads:

$$ p(\ ; \ \mid x; a) \propto p(x\mid\ ; \ )p(\ )p(\ \mid a): \tag{11} $$

2.5 Hamiltonian Monte Carlo Sampling

pyhf supports auto-differentiation via its jax, torch and tensorflow backends [4, 5, 11–13]. Therefore, the Hamiltonian Monte Carlo (HMC) step method provided by PyMC is viable for analysing pyhf HistFactory models. Details on the

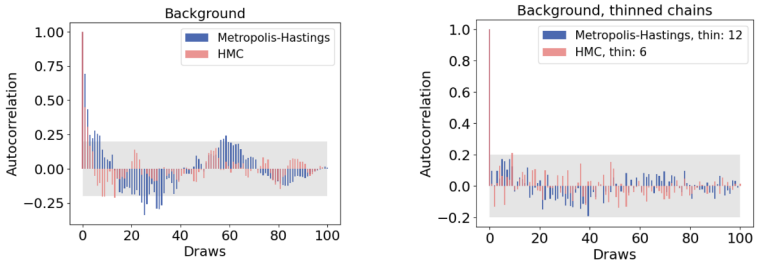


Figure 1: Autocorrelation length for the background parameter for HMC and Metropolis-Hastings steps. The left plot shows the original chains, the right the thinned chains. The shaded band indicates the acceptable length.

HMC sampling method can be found in [14]. For this work, it is sufficient to point out that HMC relies on the derivatives of the model and while this comes with a computational cost, the quality of the drawn samples should be higher compared to other step methods, such as Metropolis-Hastings [15].

The quality of MCMC chains can be measured using the autocorrelation length, i.e. the correlation between subsequent samples. Independent samples are necessary to fully express the parameter space. In order to reduce the autocorrelation length, thinning can be applied. In thinned chains, only every nth sample is kept in the final chains [16]. Fig. 1 visualises how the Metropolis-Hastings chains have to be thinned twice as much ($n = 12$) compared to the HMC chains ($n = 6$) in order to keep the autocorrelation within an acceptable range (see Sec. 3 for the model used). Accordingly, in order to produce sampling chains of the same magnitude, the computational cost of the gradient calculation can seen as substituting the cost for drawing twice as many samples for Metropolis-Hastings steps.

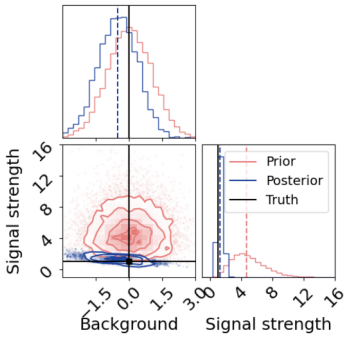
3 A Bayesian Workflow

For testing and evaluating the Bayesian inference model we follow Ref. [17] and present the selected steps below.

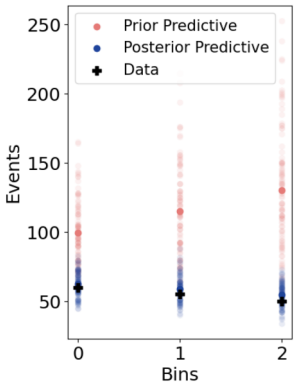
We demonstrate the inference methods derived in Sec. 2, i.e. building a statistical model using `pyhf` and then evaluating it using the whole range of inference techniques provided by PyMC and `arviz`, using a simple model. This model has three bins and one signal strength parameter (the POI of the model) and one correlated background parameter (constrained by a Normal-distributed auxiliary measurement) with ur-hyperparameters $_{ur}$; $_{ur} = 0; 2$. The event counts n_i for each bin i and a signal s_i and background b_i can be calculated as:

$$n_i = s_i + b_i: \tag{12}$$

The main result of Bayesian inference are the posterior parameter distributions, which can be used to predict observations (predictives). These results are visualized in Fig. 2. A possible next check is a calibration check, which tests the computational faithfulness of the inference, i.e. whether the distribution of posterior samples can capture a distribution of pseudo-data observations. In particular, if a set of pseudo-observations



(a) Correlation for signal and background distribution. Indicated in black is the underlying truth.



(b) Prior and posterior predictive. Indicated in black is the observed data.

Figure 2: Comparing prior and posterior distribution for the parameters (2a) and predictions for the observations (2b).

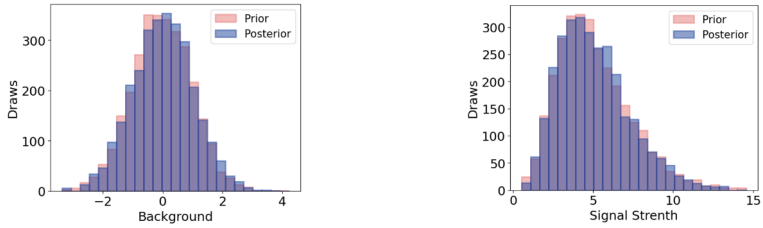


Figure 3: Calibration check for the signal POI and the background using 3000 pseudo-observations drawn from the prior predictive.

x is sampled from the prior predictive, the resulting distribution of posteriors should approximate the prior distribution, see Eq. 13.

$$p(\ ;\ ) \overset{!}{=} \int \mathrm{d}x\,\mathrm{d}\ ^{0}\,\mathrm{d}\ ^{0}\, p(\ ;\ \ jx)\ p(yj\ ^{0};\ ^{0}) \tag{13}$$

In Fig. 3 this is visualised for the simple model introduced above.

4 Conclusions

The methods presented above are implemented in the Python package bayesian_pyhf [18]. This software package enables the parallel Bayesian and frequentist analysis of multi-channel binned models within the single software framework pyhf. The current interface of the packagebayesian_pyhf is demonstrated in Listing 1. Further enhancements regarding the user interface and stability with respect to multi-chain sampling are ongoing. A full integration in the pyhf library is also planned.

```
with infer.model(model, unconstr_priors, data):
    post_data = pymc.sample(draws=10_000, chains=1)
    post_pred = pymc.sample_posterior_predictive(post_data)
    prior_pred = pymc.sample_prior_predictive(10_000)
```

Listing 1: Pseudo-code for evaluating HistFactory models (model) using PyMC given unconstrained parameters (unconstr_priors) and observations (data). post(prior)_pred are the posterior (prior) predictives and post_data are the samples from the posterior distribution. Following the PyMC syntax [8], the with statement opens a context, that initializes the inference in a way that all actions within the block are interpreted with respect to the given model, data and priors. In addition, the methodologies regarding conjugate priors from Sec 2.1 are applied under the hood, resulting in the constraint priors which are added to the model parameters for sampling.

5 Acknowledgements

MH and LH are supported by the Excellence Cluster ORIGINS, which is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC-2094-390783311. MF is supported by the U.S. National Science Foundation (NSF) under Cooperative Agreement OAC-1836650 (IRIS-HEP).

References

[1] R. Brun, F. Rademakers, Root - an object oriented data analysis framework, proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth., www:https://root.cern/download/lj.ps.gz

[2] O. Schulz, F. Beaujean, A. Caldwell, C. Grunwald, V. Hafych, K. Kröninger, S.L. Cagnina, L. Röhrig, L. Shtembari, SN Computer Science **2**, 210 (2021)

[3] LiteHF.jl , http://github.com/JuliaHEP/LiteHF.jl

[4] L. Heinrich, M. Feickert, G. Stark, pyhf: v0.7.2, https://github.com/scikit-hep/pyhf/releases/tag/v0.7.2, https://doi.org/10.5281/zenodo.1169739

[5] L. Heinrich, M. Feickert, G. Stark, K. Cranmer, Journal of Open Source Software **6**, 2823 (2021)

[6] K. Cranmer, G. Lewis, L. Moneta, A. Shibata, W. Verkerke (ROOT), Tech. rep., New York U., New York (2012), https://cds.cern.ch/record/1456844

[7] Accessed: 09.08.2023, https://people.eecs.berkeley.edu/~jordan/courses/260-spring10/other-readings/chapter9.pdf

[8] A.P. Oriol, A. Virgile, C. Colin, D. Larry, F.C. J., K. Maxim, K. Ravin, L. Jupeng, L.C. C., M.O. A. et al., PeerJ Computer Science **9**, e1516 (2023)

[9] R. Kumar, C. Carroll, A. Hartikainen, O. Martin, Journal of Open Source Software **4**, 1143 (2019)

[10] K. Murphy, Conjugate Bayesian analysis of the Gaussian distribution (2007), https://www.researchgate.net/publication/229000727_Conjugate_Bayesian_analysis_of_the_Gaussian_distribution

[11] J. Bradbury, R. Frostig, P. Hawkins, M.J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne et al., JAX: composable transformations of Python+NumPy programs (2018), http://github.com/google/jax

[12] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin et al., TensorFlow: Large-scale machine learning on heterogeneous systems (2015), software available from tensorflow.org, https://www.tensorflow.org/

[13] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer (2017)

[14] N.K. Vishnoi, An introduction to hamiltonian monte carlo method for sampling (2021), 2108.12107

[15] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, The Journal of Chemical Physics **21**, 1087 (1953)

[16] S. Hoyer, J. Hamman, Journal of Open Research Software **5** (2017)

[17] M. Betancourt, Towards a principled bayesian workflow, accessed: 09.08.2023, https://betanalpha.github.io/assets/case_studies/principled_bayesian_workflow.html

[18] M. Horstmann, bayesian_pyhf, https://github.com/malin-horstmann/bayesian_pyhf