EPJ Web of Conferences **245**, 06001 (2020) https://doi.org/10.1051/epjconf/202024506001
CHEP 2019

BAT.jl - Upgrading the Bayesian Analysis Toolkit

Allen Caldwell[1], *Cornelius* Grunwald[2,*], *Vasyl* Hafych[1], *Kevin* Kröninger[2], *Salvatore* La Cagnina[2], *Oliver* Schulz[1], and *Lolian* Shtembari[1]

[1]Max Planck Institute for Physics, Munich
[2]TU Dortmund University, Dortmund

Abstract. In all but the simplest cases, performing data analysis based on Bayesian reasoning requires the use of advanced algorithms. The Bayesian Analysis Toolkit (BAT) provides a collection of algorithms and methods that facilitate the application of Bayesian statistics to user-defined problems of arbitrary complexity. With BAT.jl, we present a modern rewrite of BAT in the Julia programming language. Through the use of a modular software design that is capable of running parallel and distributed, and by extending the tool with new sampling and integration algorithms, BAT.jl is a high-performance framework for Bayesian inference, meeting the requirements of modern data analysis.

1 Introduction

Statistical inference is a key element in nearly all fields of scientific research, with the goal of gaining knowledge about models from observed data. Typical tasks of inference involve the estimation of unknown parameters, model comparisons and hypothesis testing. Various statistical methods have been developed for conducting statistical inference. Following the Bayesian reasoning, by interpreting probabilities as a *degree-of-belief*, it is possible to update ones current knowledge about the parameters $\vec{\lambda}$ of a model M with regard to new data $\vec{D}$ using Bayes' theorem

$$P(\vec{\lambda}|\vec{D}, M) = \frac{P(\vec{D}|\vec{\lambda}, M) \cdot P_0(\vec{\lambda}, M)}{\int P(\vec{D}|\vec{\lambda}, M) \cdot P_0(\vec{\lambda}|M)\, \mathrm{d}\vec{\lambda}}\,, \tag{1}$$

where $P(\vec{D}|\vec{\lambda}, M)$ is the *likelihood* and $P_0(\vec{\lambda}|M)$ expresses *prior* knowledge about the distribution of the parameters. The *posterior* probability distribution of the parameters, $P(\vec{\lambda}|\vec{D})$, contains the updated knowledge when considering the data $\vec{D}$. The denominator in Eq. (1) describes the probability to obtain the observed data when assuming the model M, which can also be written as $P(\vec{D}|M)$ and is sometimes referred to as the *evidence*.
Only in the simplest cases, however, it is possible to evaluate the posterior distribution analytically. In most real-world problems this is not feasible due to the complexity of the likelihood and prior distribution. In such cases, a numerical evaluation of the posterior density is needed. As the phase space grows exponentially with the dimensionality of the problem (*curse of dimensionality*), efficient algorithms are needed for a numerical evaluation of Eq. (1) in problems with multiple parameters.

*e-mail: cornelius.grunwald@tu-dortmund.de

A technique that allows the efficient sampling from arbitrary distributions in a multi-dimensional parameter space, and that therefore revolutionized Bayesian inference, is Markov Chain Monte Carlo (MCMC). Markov chains provide a systematic method to draw random samples that follow a target distribution by generating a sequence of points in the parameter space. Using MCMC methods therefore allows the evaluation of complex and high-dimensional posterior distributions in Bayesian inference.

2 From the Bayesian Analysis Toolkit to BAT.jl

2.1 BAT - The C++ original

The Bayesian Analysis Toolkit (BAT) [1] is a software package providing a collection of algorithms and methods for performing Bayesian inference, particularly focusing on the use of MCMC techniques. It offers the infrastructure for implementing user-defined problems in a general-purpose language, allowing to specify likelihoods and prior distributions of arbitrary complexity without requiring the use of a tool-specific modeling language. Through the use of the Metropolis-Hastings MCMC algorithm [2, 3], BAT permits the sampling of posterior probability distributions in a multi-dimensional parameter space. Included optimization and integration algorithms allow marginalization and the estimation of parameters as well as their uncertainties and correlations. Methods for performing goodness-of-fit tests, error propagation and model comparisons are included as well as features for default outputs in the form of numerical results and plots. By offering templates for common analysis tasks, e.g. for histogram fitting, the usage of Bayesian methods is facilitated also for non-experienced users. Originally designed to be used in high-energy physics, BAT is written as a C++ library, depending on the ROOT framework [4].

2.2 BAT.jl - The new version in Julia

As we recognized that there is a broader range of users interested in performing Bayesian analyses with BAT, not only from the high-energy physics (HEP) sector, we intended to improve BAT and make it more easily applicable in further fields of research. A major aspect for this was to remove its dependencies on HEP-specific software. Therefore, we aimed for a new software design that is adaptable and simple to use but at the same time also performant and tailored to the needs of modern scientific computation, such as running in parallel and distributed. In addition, we planned to enhance the toolkit character of BAT by including new features and extending its collection of algorithms to support Bayesian inference with BAT in more fields of application. For these reasons, we are currently developing BAT.jl [5], a successor of BAT, written in the Julia programming language [6].

Julia is a modern general-purpose programming language that is particularly suited for high-performance numerical and scientific computations. The first stable version (v.1.0) of Julia has been released in August 2018. Through its optional typing system and a multiple-dispatch paradigm, just-in-time compilation and numerous other features, Julia is a powerful language that offers a range of capabilities that support the design goals we pursue with the redevelopment of BAT. Julia's built-in package management system allows for a modular software design and the uncomplicated installation and use of packages. The fast growing Julia ecosystem provides a large collection of packages for various applications. Julia also offers great flexibility through its native interfaces to other languages, that allow to call code written in C/C++, python, FORTRAN and other languages. As Julia is designed for parallel and distributed computing, running code on computing clusters is inherently supported,

meeting the requirements of modern data science software. All these (and further) aspects therefore make Julia well suited for our rewrite of BAT.

3 Current status & future prospects of BAT.jl

The first stable version of BAT.jl (v.1.0.1) has been released in December 2019. With this release we introduced the general infrastructure and user interface of the new tool and the main functionality for performing Bayesian inference on user-defined problems. A weighted Metropolis-Hastings MCMC algorithm is currently implemented as the default algorithm for sampling posterior distributions. Methods providing numerical estimates of the best-fit parameters, their uncertainties and correlations are included as well as functionality supporting an uncomplicated visualization of the outputs. Through the implementation of a novel algorithm for estimating integrals based on the samples, called Adaptive Harmonic Mean Integration (AHMI) [7], BAT.jl features the estimation of high-dimensional integrals of the posterior distribution. More detailed information on the structure and current features of BAT.jl is given in Sec. 4, where a simple example is presented.

With the current release of BAT.jl, a first step towards a modern framework for user-friendly Bayesian inference has been accomplished. We are now working towards the next upgrade of the software, planning to extend its features and collection of methods. For example, we are in the process of including new sampling algorithms into BAT.jl, such as Hamiltonian Monte Carlo (HMC) [8] and an affine invariant ensemble sampler [9]. At the same time, we are aiming to speed up the sampling by supporting the parallelization of computations at different levels. A partitioning of the parameter space into several subspaces, for example, will allow to run independent Markov chains in each of them, making the sampling of multimodal distributions more efficient. In BAT.jl, this approach will become feasible as the AHMI algorithm permits to calculate the integrals in each of the subspaces, thus providing a proper reweighting when finally joining the samples of all individual chains. With these developments in progress and further to come, BAT.jl is going to be a performant toolkit that offers a variety of state-of-the-art algorithms for Bayesian inference, allowing the users to choose the approach that fits their problems best.

4 Using BAT.jl - An Example

The BAT.jl API facilitates a straight-forward implementation of user-defined problems. While the basic setup for performing Bayesian inference with BAT.jl requires only a minimal number of commands, options for advanced configurations are accessible and allow experienced users a detailed control over the algorithms. In the following, we will demonstrate the basic steps of using BAT.jl and highlight its current features by conducting a simple example.

4.1 Installation & activation

Before using BAT.jl for the first time, it needs to be installed. By being a registered Julia package, the installation is performed using Julia's built-in package manager:

```
# installation via package manager (only once before first use)
using Pkg
pkg"add BAT"
```

After the installation, BAT.jl can be used in Julia by loading the package. For our example, we also include some more packages providing helpful functionalities:

```
# activate BAT.jl (and other useful packages)
using BAT
using IntervalSets, Distributions, Plots
```

4.2 Model definition

The most important task for the user is the implementation of the statistical model in terms of a (log)-likelihood function. In our example, we consider two parameters λ_1 and λ_2 with the likelihood following a linear combination of normal distributions:

```
likelihood = let μ = [15, 10], σ = [1.5, 2.5]
    params -> begin

        # custom implementation of normal distribution
        log_normal = ( -0.5 * log(2π*σ[1]^2)
                    - (params.λ1 - μ[1])^2 / (2σ[1]^2) )

        # normal distribution from Distributions.jl
        n1 = pdf(Normal(μ[2]-5, σ[2]), params.λ2)
        n2 = pdf(Normal(μ[2]+5, σ[2]), params.λ2)

        return log_normal + log(n1 + 3*n2)
    end
end
```

As demonstrated above, when defining the likelihood in BAT.jl, it is possible to implement custom functions as well as to refer to pre-implemented functions from other packages (e.g. from *Distributions.jl* [10]). Currently, the implementation needs to return the logarithm of the likelihood value. With the next upgrade, however, we will introduce a mechanism that allows to return either the logarithmic or the non-logarithmic value of the likelihood. When formulating the likelihood, the user is not restricted to a certain modeling language or to the use of differentiable functions, but can define models of any complexity. Since in many real-world applications, the likelihood itself is the result of sophisticated calculations and might even be distributed over several source codes, with BAT.jl it is straightforward to call external likelihoods via Julia's native interfaces to other programming languages, like python, C/C++, Fortran, R, Mathematica. It is also possible to use likelihoods from any other software running in separate processes using BAT.jl's lightweight binary communication protocol.

In BAT.jl, the model parameters (in this example λ_1 and λ_2) are defined when specifying the prior distributions. In this step, it is possible to assign names to the parameters and formulate prior knowledge about their distributions:

```
prior = NamedTupleDist(
   λ1 = Normal(6, 2.5), # normal-distributed prior for the first parameter
   λ2 = -30.0..30.0 # uniform prior in a given range for the second parameter
)
```

It is again possible to use pre-defined distributions (e.g. from additional packages) or to provide custom implementations of distributions. BAT.jl also enables to use histograms as prior distributions.

Following Bayes' theorem in Eq. (1), the posterior density is defined by the likelihood and a prior distribution for the corresponding parameters:

```
posterior = PosteriorDensity(likelihood, prior)
```

4.3 Sampling

The algorithm to be used for generating samples of the posterior distribution needs to be chosen and the number of Markov chains, as well as the number of samples that should be generated per chain, need to be set:

```
# choose sampling algorithm
algorithm = MetropolisHastings()

nchains = 8      # number of Markov chains
nsamples = 10^6  # number of samples per chain
```

In the current release of BAT.jl, the default sampling algorithm is a weighted version of the Metropolis-Hastings MCMC algorithm. In this implementation, proposed steps of the Markov chain that would have been rejected by the original Metropolis-Hastings algorithm, are kept for the samples, getting a weight that is proportional to their acceptance probability. With this minimal set of definitions it is then possible to sample the posterior distribution:

```
samples, stats = bat_sample(posterior, (nsamples, nchains), algorithm)
```

For advanced users, a fine-grained control over the settings of the MCMC algorithms allows to modify the default choices of the initialisation, burn-in and tuning strategies of the chains, the selection of the proposal function and the convergence criteria.

4.4 Results & Output

As a result of running the Markov chains with the selected algorithm, weighted samples of the posterior distribution are obtained and statistical estimates such as mode, mean and covariance matrix of the parameters are provided:

```
println("Mode:", stats.mode, "Mean:", stats.mean, "Covariance:", stats.cov)
```

For visualizing the sampling results, BAT.jl includes a collection of plotting recipes to be used with the Plots.jl package [11]. The recipes allow a user-friendly visualization of the (marginalized) prior and posterior distributions. Different types of representations and settings for customizing the plots are provided. Alternatively, custom visualizations of the results are possible, as the full information of the samples is available to the user. Plotting the samples of the posterior distribution together with the prior distributions enables to visualize the knowledge-update gained through the inference:

```
plot(samples, globalmode=true)
plot!(prior)
```

The resulting plots for this example are shown in Fig. 1.

Based on the samples, the integral of the posterior distribution (and a corresponding uncertainty estimate) can be computed using the AHMI algorithm:

```
evidence = bat_integrate(samples)
```

Among further use cases, this facilitates the calculation of Bayes factors for model comparisons.

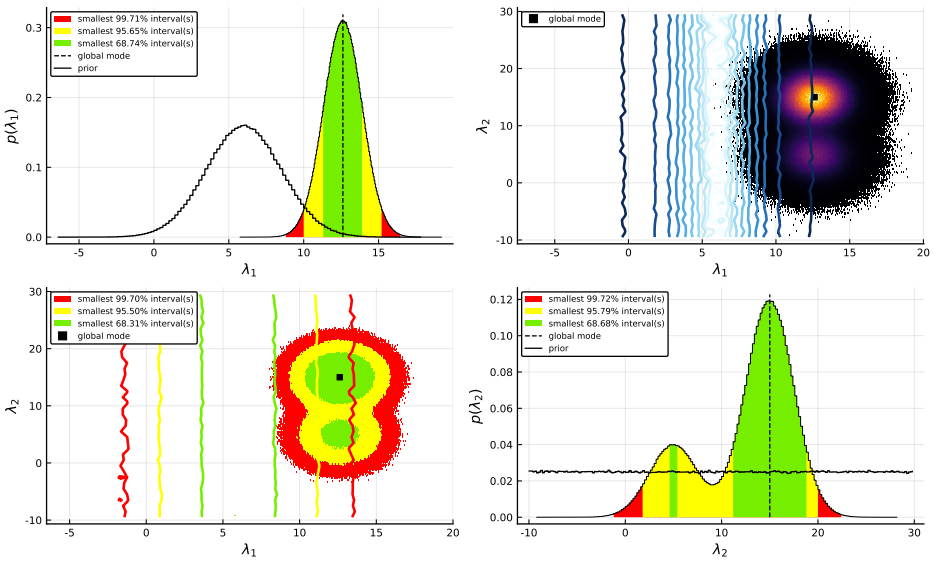

Figure 1. Plots visualizing the posterior samples together with the prior distribution as marginalized 1D and 2D histograms. The smallest intervals containing 68.3 %, 95.5 % and 99.7 % are highlighted. The global mode is indicated.

5 Conclusions

Bayesian inference is a powerful technique for data analysis. In most cases, however, it requires the use of sophisticated algorithms, such as Markov Chain Monte Carlo. With the current release of BAT.jl v.1.0.1, we present a rewrite of the Bayesian Analysis Toolkit (BAT) that provides the infrastructure and algorithms for performing Bayesian inference within a contemporary and user-friendly tool. By developing and implementing new algorithms for efficient sampling and integration, we are currently in the progress of extending the functionalities of BAT.jl. Together with novel approaches for parallelization, these developments will make BAT.jl a modern high-performance tool providing a collection of sophisticated algorithms facilitating Bayesian inference on user-defined problems.

References

[1] A. Caldwell, D. Kollar, K. Kröninger, *BAT - The Bayesian Analysis Toolkit*, Comput. Phys. Commun. **180** (2009) 2197.

[2] N. Metropolis et al., *Equation of State Calculations by Fast Computing Machines*, J. Chem. Phys. **21** (1953) 1087.

[3] W. K. Hastings, *Monte Carlo Sampling Methods Using Markov Chains and Their Applications*, Biometrika **57** (1970) 97.

[4] R. Brun and F. Rademakers, *ROOT - An Object Oriented Data Analysis Framework*, Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A **389** (1997) 81.

[5] A. Caldwell et al., *BAT.jl - A Bayesian Analysis Toolkit in Julia*, doi:10.5281/zenodo.2605312.

[6] J. Bezanson, A. Edelman, S. Karpinski and V. Shah, *Julia: A fresh approach to numerical computing*, SIAM review **59** (2017) 65.

[7] A. Caldwell, R. C. Schick, O. Schulz and M. Szalay, *Integration with an Adaptive Harmonic Mean Algorithm*, arXiv:1808.08051.

[8] S. Duane, A. D. Kennedy, B. J. Pendleton and D. Roweth, *Hybrid Monte Carlo*, Phys. Lett. B **195** (1987) 216

[9] J. Goodman and J. Weare, *Ensemble samplers with affine invariance*, Comm. App. Math. Comp. Sci. **5** (2010) 65.

[10] M. Besançon et al., *Distributions.jl: Definition and Modeling of Probability Distributions in the JuliaStats Ecosystem*, arXiv:1907.08611.

[11] Plots.jl package: https://github.com/JuliaPlots/Plots.jl.