BAT.jl: A Julia-Based Tool for Bayesian Inference

Oliver Schulz¹ · Frederik Beaujean² · Allen Caldwell¹ · Cornelius Grunwald³ · Vasyl Hafych¹ · Kevin Kröninger³ · Salvatore La Cagnina³ · Lars Röhrig³ · Lolian Shtembari¹

Received: 18 January 2021 / Accepted: 29 March 2021 / Published online: 12 April 2021
© The Author(s) 2021

Abstract

We describe the development of a multi-purpose software for Bayesian statistical inference, BAT.jl, written in the Julia language. The major design considerations and implemented algorithms are summarized here, together with a test suite that ensures the proper functioning of the algorithms. We also give an extended example from the realm of physics that demonstrates the functionalities of BAT.jl.

Keywords Bayesian analysis · Parameter inference · Model comparison · MCMC · Julia language

Introduction

The analysis of data with means of statistical methods is a key aspect of scientific research. Depending on the field of research, the type of data and the size of the corresponding data sets can vary strongly, e.g., few event counts obtained in searches for rare radioactive decays, huge samples of

astronomical data or images from medical imaging. The common theme connecting these different types of applications is the statistical analysis of the data. One is typically interested in estimating the free parameters of a scientific model given a particular data set, and in comparing two or more models. Bayesian reasoning allows for this in a consistent and easy-to-interpret way. The key element is the equation by Bayes and Laplace, that is

$$p(\theta|\mathcal{D}, M) = \frac{p(\mathcal{D}|\theta, M)p(\theta|M)}{\int d\theta p(\mathcal{D}|\theta, M)p(\theta|M)}, \quad (1)$$

where the term on the left-hand side, $p(\theta|\mathcal{D}, M)$, is the posterior probability (density)¹ for the set of free parameters θ given a data set $\mathcal{D}$ and assuming a model M . It is proportional to the product of the likelihood, $p(\mathcal{D}|\theta, M)$, and the prior knowledge about the parameters, $p(\theta|M)$. The denominator is often referred to as the evidence Z ; it is the probability to have observed the data $\mathcal{D}$ given the model M :

$$Z = P(\mathcal{D}|M) = \int d\theta p(\mathcal{D}|\theta, M)p(\theta|M). \quad (2)$$

The evidence Z is required for model comparison.

Inference about individual parameters can be performed using the multi-dimensional posterior probability or the marginalized probabilities:

✉ Oliver Schulz
oschulz@mpp.mpg.de
Frederik Beaujean
beaujean@mytum.de
Allen Caldwell
caldwell@mpp.mpg.de
Cornelius Grunwald
cornelius.grunwald@tu-dortmund.de
Vasyl Hafych
hafychvasyl@gmail.com
Kevin Kröninger
kevin.kroeninger@tu-dortmund.de
Salvatore La Cagnina
salvatore.lacagnina@tu-dortmund.de
Lars Röhrig
lars.roehrig@tu-dortmund.de
Lolian Shtembari
lolian@mpp.mpg.de

¹ Max Planck Institute for Physics, Munich, Germany

² C2PAP, Excellence Cluster Universe, LMU Munich, Munich, Germany

³ TU Dortmund University, Dortmund, Germany

¹ For better readability, we use the terms probability and probability density synonymously in the following.

$$p(\theta_i|\mathcal{D}, M) = \int p(\theta|\mathcal{D}, M) \prod_{i \neq j} d\theta_j. \quad (3)$$

We refer to commonly available textbooks for a general introduction to Bayesian inference as well as for the techniques and measures typically used, see, e.g., Refs. [1–6].

In most scientific applications, the model M results in a non-trivial form of the likelihood, such that assumptions that allow using common approximations do not hold (e.g., a Gaussian shape of the likelihood or a linear connection between the predictions and the model parameters). In such cases, it is often necessary to calculate integrals of the type appearing in Eq. 3 numerically. Efficient and reliable algorithms are an important aspect of such an evaluation, in particular for models with many parameters, or, more technically, many dimensions of integration. Similar arguments hold for the optimization problem of finding the best-fit parameters associated with the global or marginal modes of the posterior probability. A variety of automated tools are available, usually tailored to the needs of a particular field of research or a class of statistical models, such as STAN [7], PYMC [8], R [9] or OpenBUGS [10]. An important criterion to choose one tool over the others is its compatibility with the rest of the infrastructure used in a research field, typical data bases or programs used for processing the results obtained.

Due to the lack of such a tool in the field of particle physics, we originally developed the Bayesian Analysis Toolkit (BAT) [11], as a C++ library under the open-source LGPL license. It features several numerical algorithms for optimization, integration and marginalization with a strong focus on the use of Markov Chain Monte Carlo (MCMC) algorithms. BAT has been widely used in our field of research and examples of advanced applications in particle physics are global fits of complex models [12–18] and kinematic fitting [19]. Over time, BAT-C++ gained traction outside of particle physics as well. It has also been used in many other fields of research; for example in cosmology [20], astrophysics [21], and nuclear physics [22]. The sampling methods implemented in BAT-C++ have also been used to develop more advanced sampling algorithms [23, 24].

Given the wide range of possible applications, we began to develop a more easily portable version of BAT that does not come with the heavy dependencies on particle-physics software stacks and that also allows for smart parallelization. This development resulted in BAT.jl [25], a completely re-designed BAT implemented in Julia [26].

Here, we describe the design, features and numerical performance of version 2.0 of BAT.jl. It is available at <https://github.com/bat/BAT.jl> under the MIT open-source license [27], and documented at <https://bat.github.io/BAT.jl/stable/>.

The documentation also includes tutorials that new users can run and modify to quickly familiarize themselves with BAT.jl.

This paper is organized as follows: “[Design Considerations and Software Design](#)” describes the considerations that went into the design of the software and the code. “[Numerical Algorithms](#)” summarizes the numerical algorithms available in BAT.jl and “[Output and Visualization of Results](#)” the options provided to output and visualize the numerical results. Tests on the numerical performance of the algorithms is reported on in “[Numerical Test Suite](#)” and an extended example demonstrating the strength of BAT.jl is introduced in “[An Extended Example](#)”. “[Summary and Outlook](#)” provides a summary.

Design Considerations and Software Design

Design Considerations

BAT.jl aims to help solve a wide range of complex and computationally demanding problems. The design of the implementation is guided by the requirement to support multi-threaded and distributed code and offers a choice of sampling, optimization and integration algorithms. At the same time, we want to offer a user-facing interface that makes it easy to quickly solve comparatively simple problems, while offering the direct access to lower level functionality and tuning parameters that an expert may need to solve very hard problems. Finally, we wanted to make it very easy for the user to interact with and visualize results of BAT.jl’s algorithms.

We chose to implement BAT.jl in Julia due to Julia’s unique advantages for statistical and other numerical applications that require high numerical performance and easy composability of different algorithms.

Julia allows for writing code in an easy fashion, similar to Python, but at the same time enables that code to run with very high performance, such as code written in C, C++ or FORTRAN. In addition, Julia is one of the few languages based on multiple dispatch—this solves the expression problem [28], and therefore, results in a level of code-composability superior to object-oriented (i.e. single-dispatch) languages. This is complemented by Julia’s state-of-the-art package manager that makes it very easy for the user to install third-party packages.

Julia also enables automatic differentiation of almost arbitrary code via both multiple dispatch [29] and via its LISP-like meta-programming capabilities [30]. This makes it possible to use gradient-based algorithms such as HMC-sampling [31–33] and L-BFGS optimization [34] with automatic differentiation, so the user is not required to provide a hand-written gradient for likelihood and prior densities. Julia

code can be run on both CPUs and GPUs [35]. The language also offers first-class support for writing multi-threaded and distributed code. These features significantly lower the effort required when tackling problems that require highly efficient code and massive computational resources.

Julia also has very good support for interfacing with code written in C, FORTRAN, C++, PYTHON and many other languages. Therefore, while BAT.jl itself is written in Julia, the user can easily access likelihood functions written in another language, typically with minimal or no impact on performance. This is important when the likelihood functions include complex existing (e.g., in physical or biological) models.

BAT.jl is designed to integrate well with related packages in the Julia software ecosystem. To further improve this integration and code reuse, we have released functionalities that may be also useful outside of BAT.jl's main scope as separate packages, e.g., `ArrayOfArray.jl`, `ValueShapes.jl` and `EmpiricalDistributions.jl`. As such, BAT.jl is modular, and we aim to improve this modularity in future releases.

Software Design

The software model of BAT.jl is centered on positive-definite densities. These may be normalized (and can then be viewed as probabilities) or not: likelihoods, priors and posteriors are all expressed as densities (represented by the type `AbstractDensity`). BAT.jl automatically converts user-provided density-like objects like log-likelihood functions, distributions and histograms to subtypes of `AbstractDensity`.

Julia's unique advantages as a multi-dispatch programming language allow us to provide a very compact user-facing API that still makes it possible to build complex statistical analysis chains based on fundamental operations like sampling, optimization and integration.

To operate on densities, BAT.jl offers functions like `bat_sample`, `bat_findmode` and `bat_integrate`. These can be combined in a very flexible and intuitive fashion: `bat_sample` will automatically try to sample from prior densities via iid (independent and identically distributed) sampling, from posterior densities via MCMC and from existing samples themselves via resampling. `bat_findmode` and `bat_integrate` will automatically sample, use optimization algorithms or analyse existing samples, depending on the given density.

BAT.jl has a unified mechanism to manage default behavior and algorithmic choices. The function `bat_default` lets the user query which algorithm with which settings would be used for a given task. BAT.jl also records the choice of algorithms and their configuration (whether explicit or implicit) in its results. In general, BAT.jl will always try to choose an appropriate default strategy for a given task, but will let the

user override default choices for algorithms and configuration or tuning parameters.

To take advantage of the parallel architecture of modern computer systems, BAT.jl uses Julia's advanced multithreading scheduler to parallelize operations automatically where possible. For example, MCMC chains automatically run on separate threads while the user can still use multi-threading within the implementation of the likelihood function to further load out the processors of the system, without over-subscription. MCMC sampling and integration can also be run on multiple remote hosts, using Julia's support for compute clusters. MPI message transport can be used when available, but a plain TCP/IP network is sufficient.

We take great care to ensure that results are reproducible, independent of the possibly multi-threaded and distributed computation strategy. BAT uses a hierarchical scheme to partition and distribute counter-based random number generators (RNGs). By default, BAT uses the Philox RNG [36] to generate random numbers. We automatically partition this counter space (using a safe upper limit for the possible amount of RNG generation in each separate computation). Each MCMC chain, and even each step of each MCMC chain, effectively uses its own independent RNG—no matter which resources that step is scheduled to be computed on. If computations are hierarchical, each partition of an RNG counter space can be partitioned again and again, following the graph of the computation. The counter space of generators like Philox typically consists of two or four 64-bit numbers. Therefore, even nested parallel computations, each with an ample reserve of random numbers, will not run out of counter space.

Numerical Algorithms

Several algorithms for marginalization, integration and optimization are implemented in BAT.jl, giving it a toolbox character that also allows for the future inclusion of further methods, algorithms and software packages. The central algorithms available in BAT.jl are summarized in the following. We do not go into detail on additional minor functionalities, like simple evaluation of the probability distribution on a grid for a small number of dimensions and the usage of quasirandom sequences.

Sampling Algorithms

BAT.jl currently provides a choice of two main MCMC sampling algorithms to the user, Metropolis–Hastings (MH) and Hamiltonian Monte Carlo (HMC). Different algorithms are more or less suited for different target densities—for example, HMC sampling cannot be used if the target is not differentiable.

Metropolis-Hastings

The Metropolis–Hastings algorithm [37] is the original MCMC algorithm to produce a random set of numbers θ or vectors $\boldsymbol{\theta}$ that have the properties of a Markov chain and that converge towards a target distribution. In Bayesian analysis, the limiting distribution of this set $\pi(\boldsymbol{\theta})$ will be the posterior probability density $p(\boldsymbol{\theta}|M)$. The samples are generated as follows: starting from a state θ_i at iteration i , a new state $\boldsymbol{\theta}'$ is proposed according to a (often symmetric) proposal distribution $g(\boldsymbol{\theta}'|\boldsymbol{\theta})$. The proposal is accepted with a probability:

$$P_{\text{accept}} = \min\left(1, \frac{\pi(\boldsymbol{\theta}')}{\pi(\boldsymbol{\theta}_i)} \frac{g(\boldsymbol{\theta}_i|\boldsymbol{\theta}')}{g(\boldsymbol{\theta}'|\boldsymbol{\theta}_i)}\right), \quad (4)$$

resulting in $\theta_{i+1} = \boldsymbol{\theta}'$, or $\theta_{i+1} = \boldsymbol{\theta}_i$ if the proposal is rejected. We run several Markov chains in parallel and repeatedly test for convergence during a burn-in phase (see “MCMC Burn-In Process”).

By default, BAT.jl uses a multivariate Student’s t distribution as the proposal distribution. The scale and correlation of the proposal are adapted automatically to efficiently generate samples from essentially any smooth, unimodal distribution. Another important characteristic of Markov chains is the acceptance rate α , the ratio of accepted proposal points to the total number of samples in the chain. For any given target and proposal distribution, there is an optimal α that will allow the best exploration and performance of the chain.

To achieve a desired acceptance ratio the proposal distribution is tuned to adapt it to the target. After each tuning cycle (see “MCMC Burn-In Process”), the covariance matrix of the proposal function, Σ , is updated based on the sample covariance of the last iterations and it is then multiplied with a scale factor c that governs the range of the proposal. c is tuned to force the acceptance rate to lie in a region of $\alpha_{\min} \leq \alpha \leq \alpha_{\max}$ and is restricted to the region $c_{\min} \leq c \leq c_{\max}$. The adjustment of the scale factor is described in Algorithm 1 of [38]. The default values in BAT.jl for the acceptance rate and scale factor ranges are $\alpha_{\min} = 0.15$, $\alpha_{\max} = 0.35$ [39] and $c_{\min} = 10^{-4}$, $c_{\max} = 100$ respectively.

Hamiltonian Monte Carlo

One of the most sophisticated MCMC sampling methods is Hamiltonian Monte Carlo (HMC) [31–33]. Using a proposal function that is adjusted to the shape of the target distribution, HMC algorithms can yield higher acceptance rates and less correlated samples than other sampling algorithms based on random walks, thus reducing the number of samples required to fully explore the target distribution.

In HMC, the D -dimensional parameter space is expanded to $2D$ dimensions by introducing so-called momenta $\mathbf{p}$ as hyperparameters, moving from the original phase space to the canonical phase space $\mathbf{q} \rightarrow (\mathbf{q}, \mathbf{p})$. To conform to standard notation when discussing HMC, we here use $\mathbf{q}$ to represent the parameters of the model in place of $\boldsymbol{\theta}$.

In the HMC formalism, the target distribution $\pi(\mathbf{q})$ is lifted to the canonical phase space using a joint probability distribution:

$$\pi(\mathbf{q}, \mathbf{p}) = \pi(\mathbf{p}|\mathbf{q})\pi(\mathbf{q}) = e^{-H(\mathbf{q}, \mathbf{p})}, \quad (5)$$

where the probability distribution of the momenta $\pi(\mathbf{p}|\mathbf{q})$ is chosen to be conditional. The last equality in Eq. (5) comes from defining the so-called Hamiltonian as

$$H(\mathbf{q}, \mathbf{p}) = -\log \pi(\mathbf{q}, \mathbf{p}) = -\log \pi(\mathbf{p}|\mathbf{q}) - \log \pi(\mathbf{q}). \quad (6)$$

The differential equations

$$\frac{dq_i}{dt} = \frac{\partial H}{\partial p_i}, \quad \frac{dp_i}{dt} = -\frac{\partial H}{\partial q_i}, \quad (7)$$

are well known from classical mechanics and referred to as the Hamilton’s equations of motion. Solving the equations of motion for a certain time T allows moving along trajectories ϕ and gives a transition in the canonical phase space

$$(\mathbf{q}, \mathbf{p}) \rightarrow \phi_T(\mathbf{q}, \mathbf{p}) = (\mathbf{q}^*, \mathbf{p}^*), \quad (8)$$

resulting in the new point $(\mathbf{q}^*, \mathbf{p}^*)$. By marginalizing over the momenta $\mathbf{p}$, we obtain a new proposal point $\mathbf{q}^*$ in the original parameter space. This proposal is then either accepted as a new sampling point or rejected by calculating an acceptance ratio, similar to the MH algorithm. Since the proposal points are generated using information of the target distribution, their acceptance rates are higher than samples using non-problem-specific proposal distributions.

Since HMC requires gradient information and introduces multiple hyperparameters (such as momenta and integration times) into the sampling process, performing Bayesian analyses with HMC samplers is usually not as straight-forward as using the MH algorithm as it requires additional computational steps such as the numerical integration of the equations of motions and the selection and tuning of the hyperparameters. BAT.jl uses the AdvancedHMC.jl package [40] for the single HMC sampling steps. AdvancedHMC.jl provides several flavours of HMC, including multiple versions of the No-U-Turn Sampler (NUTS) [41]. Higher level operations and the burn-in process are handled by BAT.jl itself, like for MH sampling. Due to the efficient support of automatic differentiation in Julia, e.g., through the package ForwardDiff.jl [29], the gradient of the target, required for HMC, can

often be derived automatically. This makes it quite easy to use HMC within BAT.

MCMC Burn-In Process

Different MCMC sampling algorithms have different tuning parameters, e.g., the scale and shape of the proposal function for MH. However, a common requirement for the generation of samples that faithfully follow the target density is a suitable burn-in process: starting with an initial sample, each MCMC chain must be allowed to run until it has converged to its stationary distribution. Several MCMC chains must be compared to ensure that they share the same stationary distribution and are not, for example, limited to different modes of the posterior.

BAT.jl will by default use four MCMC chains, which are iterated in parallel on multiple threads (and in the future, also on multiple compute nodes). We initialize each MCMC chain with a random sample drawn from the prior, and we require that efficient sampling is possible for all priors. Typically, priors will be composed from common distributions provided by the Julia package Distributions.jl, which supports iid sampling for all of its distributions.

Once the MCMC chains are initialized, burn-in, MCMC tuning and convergence testing are performed in cycles. The user specifies the desired number of samples after burn-in, the length of each tuning/burn-in-cycle is by default 10% of desired number of final samples. During each cycle, each MCMC chain is iterated and tuning parameters are adjusted in an algorithm-specific fashion. At the end of each cycle, we check for convergence of all MCMC chains. Tuning and burn-in are complete when all chains are tuned (according to algorithm-specific criteria) and have converged (see below). MCMC samples produced until the point are discarded by default, then chains are run for the desired number of steps (the user can also set limits like maximum wall-clock time) without further modification of tuning parameters. If tuning and convergence are not successful within a (user adjustable) maximum number of cycles, the user has the option between receiving a warning message or the sampling to terminate with an error exception.

Convergence Tests

To determine if the Markov chains have converged and the burn-in phase can stop, we adopt the Gelman–Rubin convergence test [42] and the Brooks–Gelman test [42] (our default).

We consider first a single parameter θ and running N chains in parallel, where each chain produces M samples: $\theta_{1i}, \dots, \theta_{Ni}$ (where $i = 1, \dots, M$). The Gelman–Rubin test

relies on two estimators of the variance of θ : the within-chain variance estimate:

$$W = \sum_{i=1}^M \sum_{j=1}^N \frac{(\theta_{ij} - \bar{\theta}_i)^2}{M(N-1)}, \quad (9)$$

and the pooled variance estimate

$$\hat{V} = \frac{(N-1)W}{N} + \sum_{i=1}^M \frac{(\bar{\theta}_i - \bar{\theta})^2}{M-1}, \quad (10)$$

where $\bar{\theta}_i$ is the i th chain mean and $\bar{\theta}$ is the overall mean. Using these estimators, we construct the potential scale reduction factor (PSRF) denoted by $\hat{R}$:

$$\hat{R} = \frac{\hat{V}}{W}. \quad (11)$$

Since the N chains are randomly initiated from an over-dispersed initial distribution, within a finite number of samples per chain, $\hat{V}$ overestimates the target variance while W underestimates it. This implies that $\hat{R}$ will have a value larger than 1 and the degree of convergence of the chains is measured by the closeness of $\hat{R}$ to the value 1.

Therefore, we construct the multivariate PSRF (MPSRF) denoted by $\hat{R}_p$,

$$\hat{R}_p = \frac{N-1}{N} + \left(\frac{M+1}{M} \right) A_1, \quad (12)$$

where the variance estimates are

$$W^* = \sum_{i=1}^M \sum_{j=1}^N \frac{(\theta_{ij} - \bar{\theta}_i)(\theta_{ij} - \bar{\theta}_i)^T}{M(N-1)}, \quad (13)$$

$$\frac{B^*}{N} = \sum_{i=1}^M \frac{(\bar{\theta}_i - \bar{\theta})(\bar{\theta}_i - \bar{\theta})^T}{M-1}, \quad (14)$$

$$\hat{V}^* = \frac{(N-1)W}{N} + \frac{B^*}{N}, \quad (15)$$

and A_1 is the largest eigenvalue of the matrix $\frac{W^{*-1}B^*}{N}$. The default cut-off we use to declare convergence in the burn-in phase is $\hat{R}, \hat{R}_p \leq 1.1$.

Effective Sample Size

A drawback of MCMC is that the samples we obtain are correlated. BAT.jl provides an effective sample size (ESS) estimator to calculate what number of iid samples would be equivalent to N given MCMC samples, in respect to the variance of sample-mean estimates. It is also a valuable

indicator on whether a sufficient number of MCMC samples has been produced.

The effective sample size is estimated as

$$\text{ESS} = \frac{N}{\hat{\tau}}, \quad (16)$$

where $\hat{\tau}$ is the integrated autocorrelation time. $\hat{\tau}$ is estimated from the normalized autocorrelation function $\hat{\rho}(\tau)$:

$$\hat{\tau}_k = 1 + 2 \sum_{\tau=1}^{\infty} \hat{\rho}_k(\tau), \quad (17)$$

$$\hat{\rho}_k(\tau) = \frac{\hat{c}_k(\tau)}{\hat{c}_k(0)}, \quad (18)$$

$$\hat{c}_k(\tau) = \frac{1}{N - \tau} \sum_{n=1}^{N-\tau} (\theta_{k,i} - \hat{\theta}_k)(\theta_{k,i+\tau} - \hat{\theta}_k), \quad (19)$$

where k refers dimension index of the multivariate sample $\theta_i = \{\theta_{1,i}, \dots, \theta_{D,i}\}$. Here, all samples for a given parameter k are used, independently of whether multiple chains have been run. The first index refers now to the variable under discussion (as opposed to the chain number, above). These quantities allow us to calculate an effective sample size for each dimension $\text{ESS}_k = \frac{N}{\hat{\tau}_k}$.

When evaluating Eq. 17, we cannot, in practice, actually sum over all lags τ : while $\hat{c}_k(\tau)$ theoretically decays to zero for high lags τ , in practice it exhibits a noisy behavior that makes the sum over $\hat{c}_k(\tau)$ unstable. Therefore, we need to truncate the sum using a heuristic cut-off. The default cut-off in BAT.jl is Geyer's initial monotone sequence estimator [43], optionally Sokal's method [44] can be chosen.

Algorithms for Point Estimates

The global mode of a posterior distribution is often a quantity of interest. While the MCMC sample with the largest value of the target density may come close to the true mode, it is sometimes not as close as required. It is, however, an ideal starting point for a local optimization algorithm than can then further refine the mode estimation. BAT.jl offers automatic mode-estimation refinement using the Nelder-Mead [45] and LBFGS [46] optimization algorithms, by building on the Optim.jl [34] package. When using LBFGS, a gradient of the posterior distribution is required. Again, we utilize the Julia automatic-differentiation package ecosystem to automatically compute that gradient.

Another quantity that is often computed from samples is a marginal mode. To construct marginals, a binning of

the samples is performed. The optimal number of bins can be determined by using Square-root choice, Sturges' formula, Rice Rule, Scott's normal reference rule, or the Freedman–Diaconis rule. The latter is the default.

BAT.jl also provides functionality to estimate other quantities such as the median, the mean, quantiles and standard deviations, and to propagate errors on a fit function.

Integration Algorithms

Evidence Estimation using AHMI

In many applications, it is desirable or even necessary to compute the evidence or marginal likelihood Z (see Eq. 2). An example for the use of Z is the calculation of a Bayes factor for the comparison of two models M_A and M_B :

$$\text{BF} \equiv \frac{p(\mathcal{D}|M_A)}{p(\mathcal{D}|M_B)} = \frac{Z_A}{Z_B}.$$

BAT.jl includes the Adaptive Harmonic Mean Integration (AHMI) algorithm [47] to compute Z given the samples $\{\theta\}$.

AHMI can integrate samples from any sampling algorithm as long as the samples come in the form of `BAT.DensitySampleVector`. It's use of hyper-rectangles, however, limits the applicability to a moderate number of dimensions (≈ 20 in the case of a multivariate normal distribution).

Evidence Calculation Using an Interface to CUBA

In addition to integration via AHMI, BAT offers evidence calculation using the Cuba [48] integration library. Cuba implements multiple integration algorithms that cover a range of (Monte Carlo and deterministic) importance sampling, stratified sampling and adaptive subdivision integration strategies. These will typically not scale to high-dimensional spaces, but can provide quick and robust results for low-dimensional problems.

Parameter Space Transformations

Different algorithms have different requirements on the structure and domain of the densities they operate on. HMC, for example (like other gradient-based algorithms), requires a continuous target density. As a result, it does not perform well if the density is bounded. CUBA, on the other hand, can only operate on the unit hypercube, and so requires a bounded density. It is therefore often necessary to perform a change of variables, transforming the original density into one more suitable for the chosen algorithm. The prior distribution will typically contain sufficient information on the structure and domain of the posterior distribution to choose a suitable transformation.

BAT.jl will, by default, automatically try to internally transform posterior densities, so that the prior becomes equivalent to a standard multivariate-normal or multivariate-uniform distribution in the transformed space, depending on the requirements of the algorithm chosen to operate on the posterior. Prior distributions are often Product distributions and these are simply transformed elementwise. For univariate distributions, BAT.jl will transform according to their (inverse) CDF, multivariate normal distributions are transformed according to their covariance matrix, and hierarchical distributions are transformed iteratively. The mechanism can be extended by specialized transformations for complex or custom prior distributions. Automatic transformations for additional relevant multivariate prior distributions, e.g. Dirichlet distributions, will be added in future versions.

It is of course also possible for the the user to transform target densities manually.

Output and Visualization of Results

The results of running the numerical algorithms in BAT.jl are presented in text and also in graphical form. In addition, user-defined interfaces can be written to bring the results into any other format.

Graphical Summary of the Results

As a key element of all statistical analyses is the graphical representation of outcomes, BAT.jl includes functionalities to create visualizations of the analyses results in a user-friendly way. By providing a collection of plot recipes to be used with the Plots.jl² package, several plotting styles for 1D and 2D representations of (marginalized) distributions of samples and priors are available through simple commands. Properties of the distributions, such as highest density regions or point estimates like mean and mode values, can be automatically highlighted in the plots. Further recipes to visualize the results of common applications, such as function fitting, are provided. While the plot recipes provide convenient default options, the details of the plotting styles can be quickly modified and customized. Since all information about the posterior samples and the priors are available to the user, completely custom visualizations are of course also possible. Examples of plots created with the included plot recipes are shown in “[An Extended Example](#)”.

Table 1 Listing of the analytical form of two-dimensional test functions used for performance testing

Name	Function
Normal	$f(x) = \frac{\exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)}{\sqrt{(2\pi)^d \Sigma }}$
Multi cauchy	$f(\lambda) = \prod_{i=1}^2 \frac{1}{2} [\text{Cauchy}(\lambda_i \mu, \sigma) + \text{Cauchy}(\lambda_i -\mu, \sigma)]$
Funnel	$f(\lambda) = \mathcal{N}(\lambda_1 0, a^2) \prod_{i=2}^n \mathcal{N}(\lambda_i 0, \exp(2b\lambda_1))$

Written Summary of the Results

BAT.jl can display a written summary containing information about the sampling process and the results of the parameter estimation. The summary includes a list of all parameters with their mean, standard deviation, global and marginal mode. If the number of parameters is not too high, the written summary also includes a table with the covariance of all parameters. In addition, the effective number of samples is reported.

All of these summary results are also accessible in a programmatic fashion using BAT.jl functions, so the user can utilize them in further automated analysis.

File I/O

It is important that users have means to easily preserve the results of the MCMC sampling process, which often is computationally expensive. We provide several ways to do this:

BAT.jl is compatible with the Julia package JLD2. JLD2 provides a mechanism to store almost arbitrary Julia data structures to disk in an HDF5-based format. JLD2-output may not be readable with other software versions later on though, since almost everything is preserved, including the density functions themselves. Therefore, the combination of BAT.jl and JLD2 allows for complete persistence, but is intended for short-term storage.

To allow for long-term data preservation, BAT.jl also provides a function to store posterior sample variates, weights and log-density values to HDF5 files in a more basic fashion in plain HDF5 data sets (i.e. flat arrays). Any version of BAT.jl will always be able to read sampling output stored this way by older versions. The output is also easy to read using any programming language basic HDF5-support.

Furthermore, samples can also be easily written to ASCII/CSV-files using standard Julia packages, since BAT's sampling output is compatible with standard Julia table formats.

² <https://github.com/JuliaPlots/Plots.jl>.

Fig. 1 BAT default plots for the multi-modal Cauchy distribution. The plots on the upper left and lower right show the marginalized distribution for each dimension. The other two plots show the full 2D distribution with the lower left plot focusing on illustrating probabil-

ity intervals and the upper right one the general shape of the 2D sample. The dashed line indicates the global mode of the sample, while the green, yellow and red colored samples are defined by the 68.57%, 99.5% and 99.7% quantiles

Numerical Test Suite

A test suite to evaluate the numerical performance of the sampling algorithms is included in BAT.jl, and must be passed before each release of a new version. Samples are MCMC-generated from, and then compared to, a set of test distributions. A list of these distributions is given in Table 1. We compare the mean values, variances, and the global modes of the samples with those of the test distributions. We also calculate the p values of Kolmogorov–Smirnov (KS) tests for each parameter, by comparing the marginal distributions from the sampling algorithm with marginal distributions from samples generated by iid sampling. Small p values lead to further investigations to ensure that the sampling algorithm is functioning properly.

Additionally, the integral of the target distributions is calculated from the samples using AHMI. Since AHMI relies on an accurate sampling of the target distribution, the AHMI integral value provides a very sensitive test of the sampling algorithm.

As an example, Figs. 1 and 2 show the distributions of the samples generated for a multi-modal Cauchy and for the funnel distribution, respectively.

Assuming iid sampling and a large number of samples, the differences, in units of standard deviations, between the observed distributions and the true distributions are expected to follow a unit normal distribution. This should also be the case if the number of MCMC samples is large enough. For our tests, we compare the expectations in intervals (bins) of the function arguments. The standard deviation is estimated for each bin as the square root of the expected number of entries from the test function. For each bin with an expectation larger than ten, the observed number of entries is divided by that standard deviation. A histogram of these values, also referred to as pull plot, can be seen in Fig. 3. It is compatible with expectations.

Table 2 summarizes the expected and observed mean values, variances and global modes for the different two-dimensional test functions, together with the corresponding KS test p values and AHMI integral values. Very good agreement is observed in all distributions with a maximal deviation of 4% in the mode, 4% in the variance. The AHMI integrals are all very close to the true values, and are typically within the reported uncertainty. The smallest KS test p value of 0.093. We note that the ESS defined in “Effective Sample Size” is used in calculating the p value for the KS

Fig. 2 BAT default plot for the funnel distribution. The plots on the upper left and lower right show the marginalized distribution for each dimension. The other two plots show the full 2D distribution with the lower left plot focusing on illustrating probability intervals and

the upper right one the general shape of the 2D sample. The dashed line indicates the mode of the sample whilst the green, yellow and red colored areas represent the 68.57%, 99.5% and 99.7% intervals of the sample respectively. Both distributions are normalized to unity

Fig. 3 Pull plot of the difference between the analytical function (red curve) and the distribution of the samples (blue histogram) for the multi-modal Cauchy (left) and funnel distribution (right)

test. For the Cauchy distributions, the p values close to 1 indicate that the ESS values may be underestimated. We have noticed that this can occur when the samples become highly correlated.

For the two-dimensional test cases, we use we use 8 Markov chains with 10^6 steps per chain. For the

n -dimensional test cases, we use 4 Markov chains and some additional non-default settings, detailed in Table 3. Note that while we relax the Brooks–Gelman convergence threshold in the test suite, to be able to probe some higher dimensional cases more easily, we use more sensitive tests to actually

Table 2 Performance test results for two-dimensional functions

Name	Normal		Multi cauchy		Funnel	
	Target	Test	Target	Test	Target	Test
Mode	[15, 10]	[14.999, 10.004]	[− 5, 5]	[− 4.81, 4.786]	[− 1, 0]	[− 0.998, − 0.001]
Mean target	[15, 10]	[14.999, 10.003]	[0, 0]	[− 0.874, 0.352]	[0, 0]	[− 0.001, 0.005]
Var	[2.25, 6.25]	[2.243, 6.246]	[−, −]	[6808.7, 12802.4]	[1, 7.407]	[0.998, 7.095]
AHMI integral	1	0.997 ± 0.002	1	1.003 ± 0.002	1	0.999 ± 0.002
KS test p value	[0.093, 0.939]		[1.0, 1.0]		[0.809, 0.999]	

Table 3 Non-default MCMC settings used for tests in n -dimensions

Parameters	Values
Brooks Gelman Convergence threshold	1.6
Brooks Gelman Convergence corrected	False
Init tries per chain	64..1024
Number of steps per chain	10,000
Number of steps per cycle	100,000
Maximum number of cycles	300

verify correct sampling (see above). BAT v2.0.0 defaults were used for all other settings (like hyper-parameters). For the sampling tests, we do not use BAT's space transformation capabilities, so that the tests measure the performance of the sampler on the actual target distributions.

The AHMI integral and KS test p values are calculated for the test functions from 2 up to 20 dimensions. Figure 4 shows the integral values and their uncertainties.

The integral of the multi-modal Cauchy and funnel distribution are calculated for up to 10 and 16 dimensions, respectively, using AHMI, whereas the integral for the normal distribution is calculated for up to 20 dimensions³. In all cases where the AHMI algorithm is able to report an integral value, the result is compatible within the quoted uncertainty with the expected value. The distribution of the KS test p values for the test functions from 2 to 20 dimensions is shown in Fig. 5. The distributions of the p values for the normal and funnel distribution are compatible with the expectation. The p values for the Cauchy distribution are, similar to the 2 dimensional performance measures, closer to one due to higher correlation of the samples. We have also executed the test suite for the HMC sampling algorithm. Here, we present results for the funnel distribution in 20 up to 35 dimensions.

The KS p values, shown in Fig. 6 follow an approximately flat distribution between 0 and 1, indicating that both sampling algorithms perform well.

Fig. 4 Integral calculated using AHMI for the normal, multi-modal Cauchy and funnel distribution between 2 and 20 dimensions. The colored areas represent the uncertainty provided by AHMI

³ For a higher number of dimensions, the AHMI algorithm cannot determine appropriate integration subvolumes and reports its inability to perform the integral.

Fig. 5 KS test p values calculated for each marginal for the normal, multi-modal Cauchy and funnel distribution between 2 and 20 dimensions. The horizontal axis indicates the number of dimensions, while the p value is given on the vertical axis

Fig. 6 KS p values for all marginals of the sampled funnel distribution with 20 up to 35 dimensions, using both the HMC and the MH sampling algorithm. The horizontal axis indicates the number of dimensions, the p value is given on the vertical axis

An Extended Example

In the following, we demonstrate the potential of BAT.jl by solving a realistic problem of a type often encountered in particle and astroparticle physics experiments, namely, fitting a model to a set of data and determining if a specific signal process is present in the data.

In the example, we imagine that we are searching for a rare phenomenon: e.g., a particular nuclear decay, which leaves a specific and well-defined signature in the

experiment. The experiment itself comprises several different detectors that can measure the energy of an event and which are all sensitive to the signal in a limited energy window, for example from 0 to 200 keV. The data we collect will come from two different sources, signal and background. We assume that while shielding measures are present that limit the detection of background events, we are not able to suppress them completely.

To claim a discovery of the signal in this kind of experiment, it is not sufficient to detect events close to the energies predicted by the theory of the desired signal.

Instead, the task is to make a statement on the probability of having detected signal events in the presence of background. This implies the comparison of two different models, namely the background-only (BKG) model, where we assume that no signal is present in the data and all events are due to background sources, and the signal-plus-background (S+BKG) model, where we assume that we detected events from both sources. We fit both of these models to the data and then compare them using a Bayes Factor.

Data Model

As the experimental observable is set of energy values E , we will formulate the model for signal and background processes in this quantity. We assume that the probability distribution for background events follows an exponential function characterized by a decay constant λ , that is

$$p_B(E|\lambda) = \lambda e^{-\lambda \cdot E}. \quad (20)$$

The probability distribution for signal events follows a normal distribution with known mean value μ_S and also known standard deviation σ_S , that is

$$p_S(E|\mu_S, \sigma_S) = \frac{1}{\sqrt{2\pi} \cdot \sigma_S} e^{-\frac{1}{2} \left(\frac{E - \mu_S}{\sigma_S} \right)^2}. \quad (21)$$

In the current example, we chose $\mu_S = 100$ keV and $\sigma_S = 2.5$ keV. Each detector will operate for a finite amount of time, T_i , also referred to as exposure. The total number of expected background events for detector i is then

$$\mu_i^B = T_i \cdot B_i, \quad (22)$$

where B_i [counts/year] estimates the background rate, i.e. the number of events per year of operation, assuming that the rate of background events does not change.

Similarly, we can estimate the expected number of signal events in detector i as

$$\mu_i^S = T_i \cdot \epsilon_i \cdot S, \quad (23)$$

where ϵ_i is the efficiency of the detector to recognize the signal event and S [events/year⁻¹] is the signal rate and is representative of the signal strength.

Apart from modeling the data collected in the experiment, we might also need to model the detectors themselves. Suppose we use a total of five detectors in our experiment, but given that it takes time to build them, we start operating the detectors at different times resulting in different exposures. Since the detectors are produced one at a time and the manufacturer has time to refine the production process, the

Table 4 Exposure and efficiency of the fictional detectors

Detector i	Exposure T_i [yr]	Efficiency ϵ_i
1	1.6	0.5
2	1.3	0.6
3	1.0	0.7
4	0.7	0.8
5	0.4	0.9

detection efficiency might be better for detector produced at a later stage. We can also assume that the background rates will not be exactly the same but they will be close to each other, since this quantity mostly depends on the properties of the detector material and the production process. In order to account for the correlation between the background rates of the different detectors, we assume that the individual background rates B_i are randomly distributed according to a log-normal distribution, that is

$$p(B_i) \sim \text{log-normal}(\mu_B, \sigma_B). \quad (24)$$

The log-normal distribution is a commonly used prior for non-negative parameters.

Since $p(B_i)$ depends on μ_B and σ_B , our prior will have a hierarchical, resp. layered structure. BAT.jl allows the user to express hierarchical priors in straightforward fashion, the prior distribution of some model parameters can be expressed as a function of other model parameters.

Given the parameters μ_B and σ_B , the mean of the log-normal distribution is $m_B = e^{\mu_B + \frac{\sigma_B^2}{2}}$. In the following, we found it more intuitive to work with m_B and then set $\mu_B = f(m_B, \sigma_B)$. In our example, we assume five detectors with an exposure and efficiency given in Table 4.

Statistical Model

Since we have five detectors with different exposures and detection efficiencies, we split our data into five different data sets $\mathcal{D}_i$. The likelihood for the S+BKG model and a single data set is then

$$\begin{aligned} \mathcal{L}_i(\mathcal{D}_i|S, B_i) &= \prod_{j=1}^{N_i^{\text{obs}}} \frac{1}{\mu_i^B + \mu_i^S} \\ &\quad \left[\mu_i^B \lambda e^{-\lambda E_j} + \mu_i^S \frac{1}{\sigma_S \sqrt{2\pi}} e^{-\frac{1}{2} \left(\frac{E_j - \mu_S}{\sigma_S} \right)^2} \right], \end{aligned} \quad (25)$$

where N_i^{obs} is the number of events in data set $\mathcal{D}_i$. The total likelihood is constructed as the product of all $\mathcal{L}_i$ weighted with the Poisson terms [49]:

Fig. 7 Binned generated data

$$\mathcal{L}(\{\mathcal{D}_i\} | S, \{B_i\}) = \prod_{i=1}^5 \left[\frac{e^{-(\mu_i^B + \mu_i^S)} (\mu_i^B + \mu_i^S)^{N_i^{\text{obs}}}}{N_i^{\text{obs}}!} \cdot \mathcal{L}_i(\mathcal{D}_i | S, B_i) \right]. \quad (26)$$

We use the same likelihood for the BKG model, but with all μ_i^S set to zero.

Apart from the likelihood, we also specify the priors for the free parameters of the model. These are the signal rate $S \sim \text{Uniform}(0, 10) \text{ year}^{-1}$, the background rate parameters $m_B \sim \text{Uniform}(0, 5 \cdot 10^{-2}) \text{ counts/year}$ and $\sigma_B \sim \text{Uniform}(0.1, 1.0) \text{ counts/year}$ as well as the decay constant $\lambda \sim \text{Uniform}(0, 100)$.

Data and Results

The data for the analysis are generated synthetically. We choose a decay constant of $\lambda_{\text{true}} = 50$ with background rate parameters $m_B = 4.7$ and $\sigma_b = 0.5$. In addition, we include three signal events, i.e., $S = 0.9375$. The exact data used in this example are available in Tables 5 and 6 in Appendix “Example Data” and Fig. 7 shows the binned histogram of these data. As can be seen, without knowing that there are three signal events at 100 keV, it would be very difficult to recognise them just by looking at the data.

In the following, BAT.jl v2.0.0 default settings (e.g., hyper-parameters) are used, unless noted otherwise.

We sample the posterior densities of both models using HMC with 4 MCMC chains and 10^5 steps per chain. BAT.jl’s automatic internal parameter space transformation allows us to do this even though the prior is hierarchical and contains bounded distributions.

To determine whether the model with or without signal should be preferred, we also compute the evidences of the BKG and S+BKG models by applying the AHMI algorithm to the posterior samples. We then calculate the Bayes factor under the assumption of the same prior probability for the two models, that is

$$\text{BF} = \frac{p(S+\text{BKG}|\mathcal{D})}{p(\text{BKG}|\mathcal{D})} = 3.4, \quad (27)$$

which supports the claim that the data contains both signal and background events.

Having determined that our data does indeed contain signal, we look at the marginal posterior distributions in order to check how well the fit reconstructs the parameters used to generate the synthetic data. With BAT.jl the user can easily plot the results just like in Fig. 8, which shows both the 1D and 2D marginal posterior distributions for the signal rate S and the background decay constant λ . In the marginalized distribution of a parameter, the mode is representative of the most likely scenario and inspecting the modes in Fig. 8 we notice that S peaks at 0.94 while λ peaks at 47. Both modes are very close to the nominal values that were used in data generation.

Since we assume that there was a correlation between the background rates of the individual detectors, we examine the posterior of the model parameters that control the distribution of the B_i -s in Fig. 9. We notice that a mean background rate m_B between 6 and 7 events per year is most likely. The spread of the posterior log-normal distribution will likely be small since the posterior of the parameter σ_B exhibits an exponential decay peaking at 0.

Fig. 8 Posterior distribution of the signal-rate S and the background-rate λ . The blue line (in the upper-left and lower-right plot) shows the prior

Fig. 9 Posterior distribution of the background rate mean m_B and its sigma σ_B . The blue line (in the upper-left and lower-right plot) shows the prior

Fig. 10 Distribution of all data events, compared with the S+BKG model using best fit parameters and central quantiles according to the full posterior distribution

Finally, we compare our S+BKG model with the data in Fig. 10.

Summary and Outlook

We have developed a platform-independent software package for Bayesian inference, BAT.jl. BAT.jl features a toolbox for numerical algorithms to perform calculations often encountered in Bayesian inference, in particular sampling, optimization and integration algorithms as well as flexible input/output routines. BAT.jl also allows for interfacing with arbitrary custom codes, e.g. for the evaluation of complex models. We use the Julia programming language to provide a lightweight but powerful interface, parallel processing and automatic differentiation. We intend for the package to appeal to a wide user base, not constrained to a specific realm of science. The main application of BAT.jl is to study models that are characterized by (numerically) complex likelihood functions. In this paper, we describe the design choices, the implemented algorithms, and the procedure to test the implementation. We also give a concrete physics example that demonstrates the capabilities of BAT.jl. BAT.jl has already seen first use in several scientific works [50–55].

For the future, we plan to extend the functionality available in BAT.jl further, adding more algorithms, novel sampling schemes and multi-level parallelization.

Example data

See Tables 5 and 6.

Table 5 The background events produced for the extended example

Detector	Energy [keV]
1	32.299
1	69.086
1	25.44
1	75.614
1	7.927
1	3.335
1	20.671
1	82.752
1	136.753
1	19.977
1	34.572
1	52.326
1	4.755
2	63.766
2	4.603
2	68.246
2	4.273
2	49.339
2	41.749
2	47.499
2	37.522
2	26.041
2	58.087
3	58.823
3	8.785
3	55.583
3	63.517
3	58.298
3	40.373
3	7.876
3	139.171
4	31.879
4	21.281
5	53.042

Table 6 Signal events produced for the extended example

Detector	Energy [keV]
3	100.794
3	98.933
5	98.559

Acknowledgements The authors would like to thank Tatyana Abramova for the fruitful discussions about Turchin’s method of regularization and Hamiltonian Monte Carlo, and to thank Scott Hayashi for contributing to the BAT.jl unit tests.

Funding Open Access funding enabled and organized by Projekt DEAL. C.G. is supported by the Studienstiftung des Deutschen Volkes.

This project is supported by the Deutsche Forschungsgemeinschaft (DFG), project KR 4060/7-1, and by the European Union's Framework Programme for Research and Innovation Horizon 2020 (2014-2020) under the Marie Skłodowska-Curie Grant Agreement No.765710.

Availability of Data and Material Not applicable.

Code Availability The source code of BAT.jl is available at <https://github.com/bat/BAT.jl> and under DOI 10.5281/zenodo.2587213, as well as via the Julia package management system. The example application presented in this work is included. BAT.jl is published under the MIT open-source license [27].

Declarations

Conflict of Interest The authors declare that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. D'Agostini G. Bayesian reasoning in data analysis: a critical introduction. World Scientific; 2003.
2. Hartigan JA. Bayes theory. New York: Springer; 1983.
3. Jaynes ET, Larry Bretthorst G. Probability theory: the logic of science. Cambridge: Cambridge University Press; 2003.
4. Kendall MG, et al. Kendall's advanced theory of statistics: bayesian inference. Hodder Arnold; 1994.
5. MacKay D. Information theory, inference and learning algorithms. Cambridge University Press; 2003.
6. Sivia D, Skilling J. Data analysis: a Bayesian tutorial. Oxford University Press; 2006.
7. Carpenter B, et al. Stan: a probabilistic programming language. J Stat Softw Articles. 2017;76(1):1–32. <https://doi.org/10.18637/jss.v076.i01>. issn: 1548-7660. <https://www.jstatsoft.org/v076/i01>.
8. Wiecki Christopher Fonnesbeck John Salvatier Thomas. Probabilistic programming in Python using PyMC3. In: (). <https://doi.org/10.7717/peerj-cs.55>.
9. R Core Team. R: A language and environment for statistical computing. R foundation for statistical computing. Vienna, Austria, 2017. <https://www.R-project.org/>.
10. Lunn DJ, et al. WinBUGS—a bayesian modelling framework: concepts, structure, and extensibility. Stat Comput. 2000;10(4):325–37. <https://doi.org/10.1023/A:1008929526011> (issn: 1573-1375).
11. Caldwell A, Kollar D, Kröninger K. BAT: the bayesian analysis toolkit. Comput Phys Commun. 2009;180:2197–209. <https://doi.org/10.1016/j.cpc.2009.06.026>.
12. Bevan AJ, et al. The UTfit collaboration average of D meson mixing data: winter 2014. JHEP. 2014;03:123 (arXiv:1402.1664 [hep-ph]).
13. Ghosh D, Salvarezza M, Senia F. Extending the analysis of electroweak precision constraints in composite higgs models. 2015. arXiv:1511.08235 [hep-ph].
14. Ciuchini M, et al. Update of the electroweak precision fit, interplay with Higgs-boson signal strengths and model-independent constraints on new physics. In: International Conference on High Energy Physics 2014 (ICHEP 2014) Valencia, Spain, July 2–9, 2014. 2014. arXiv:1410.6940 [hep-ph].
15. Ciuchini M, et al. Electroweak precision observables, new physics and the nature of a 126 GeV Higgs boson. JHEP. 2013;08:106 (arXiv:1306.4644 [hep-ph]).
16. de Blas J, et al. Global Bayesian analysis of the Higgs-boson couplings. In: International Conference on High Energy Physics 2014 (ICHEP 2014) Valencia, Spain, July 2–9, 2014. 2014. arXiv:1410.4204 [hep-ph].
17. Agostini M, Benato G, Detwiler J. Discovery probability of next-generation neutrinoless double- β decay experiments. Phys Rev D. 2017;96(5):053001. <https://doi.org/10.1103/PhysRevD.96.053001> (arXiv:1705.02996 [hep-ex]).
18. Caldwell A, et al. Global Bayesian analysis of neutrino mass data. Phys Rev D. 2017;96(7):073001. <https://doi.org/10.1103/PhysRevD.96.073001> (arXiv:1705.01945 [hep-ph]).
19. Erdmann J, et al. A likelihood-based reconstruction algorithm for top-quark pairs and the KL Fitter framework. Nucl Instrum Meth. 2014;A748:18 (arXiv:1312.5595 [hep-ex]).
20. Luongo O, Pisani GB, Troisi A. Cosmological degeneracy versus cosmography: a cosmographic dark energy model. 2015. arXiv:1512.07076 [gr-qc].
21. Ullio P, Valli M. A critical reassessment of particle Dark Matter limits from dwarf satellites. 2016. arXiv:1603.07721 [astro-ph. GA].
22. Rappold C, et al. Hypernuclear production cross section in the reaction of $6\text{Li} + 12\text{C}$ at 2A GeV. Phys Lett B. 2015;747:129.
23. Caldwell A, Liu C. Target density normalization for Markov Chain Monte Carlo Algorithms. 2014. arXiv:1410.7149 [physics.data-an].
24. Kröninger K, Schumann S, Willenberg B. (MC)**3—a Multi-Channel Markov Chain Monte Carlo algorithm for phase-space sampling. Comput Phys Commun. 2015;186:1 (arXiv:1404.4328 [hep-ph]).
25. Schulz O, et al. BAT.jl. <https://doi.org/10.5281/zenodo.2587213>.
26. Bezanson J, et al. Julia: a fresh approach to numerical computing. In: CoRR abs/1411.1607. 2014. arXiv:1411.1607.
27. The MIT License. <https://opensource.org/licenses/MIT>. Accessed: 2020-07-23.
28. Zenger M, Odersky M. Independently extensible solutions to the expression problem. 2004. <http://infoscience.epfl.ch/record/52625>.
29. Revels J, Lubin M, Papamarkou T. Forward-Mode automatic differentiation in Julia. 2016. arXiv:1607.07892 [cs.MS].
30. Innes M. Don't Unroll Adjoint: differentiating SSA-form programs. In: CoRR abs/1810.07951. 2018. arXiv:1810.07951.
31. Duane S, et al. Hybrid Monte Carlo. Phys Lett B. 1987;195(2):216–22. [https://doi.org/10.1016/0370-2693\(87\)91197-X](https://doi.org/10.1016/0370-2693(87)91197-X). issn: 0370-2693. <http://www.sciencedirect.com/science/article/pii/037026938791197X>.
32. Neal RM. MCMC using Hamiltonian dynamics. In: Handbook of Markov Chain Monte Carlo. CRC Press; 2011. <https://doi.org/10.1201/b10905-7>. Chap. chapter5.
33. Betancourt M. A conceptual introduction to Hamiltonian Monte Carlo. In: arXiv e-prints, arXiv:1701.02434 (2017). arXiv:1701.02434 [stat.ME].

34. Mogensen PK, Riseth AN. Optim: a mathematical optimization package for Julia. *J Open Source Softw.* 2018;3(24):615. <https://doi.org/10.21105/joss.00615>.
35. Besard T, Foket C, De Sutter B. Effective Extensible programming: unleashing Julia on GPUs. *IEEE Trans Parallel Distrib Syst.* 2018. <https://doi.org/10.1109/TPDS.2018.2872064>. issn: 1045-9219. [arXiv:1712.03112](https://arxiv.org/abs/1712.03112) [cs.PL].
36. Salmon JK. Parallel random numbers: as easy as 1, 2, 3. In: *SC '11*. 2011. p. 1–12. <https://doi.org/10.1145/2063384.2063405>.
37. Metropolis N, et al. Equation of state calculations by fast computing machines. *J Chem Phys.* 1953;21:1087–92. <https://doi.org/10.1063/1.1699114>.
38. Beaujean F. A Bayesian analysis of rare B decays with advanced Monte Carlo methods. 2012.
39. Roberts GO, Gelman A, Gilks WR. Weak convergence and optimal scaling of random walk metropolis algorithms. *Ann Appl Probab.* 1997;7(1):110–20.
40. Ge H, Xu K, Ghahramani Z. Turing: a language for flexible probabilistic inference. In: *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9–11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*. 2018. p. 1682–1690. <http://proceedings.mlr.press/v84/ge18b.html>.
41. Hoffman MD, Gelman A. The No-U-Turn Sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *J Mach Learn Res.* 2014;15(47):1593–623. <http://jmlr.org/papers/v15/hoffman14a.html>.
42. Gelman A, Rubin DB. Inference from iterative simulation using multiple sequences. *Stat Sci.* 1992;07(04):503–11.
43. Geyer CJ. Practical Markov Chain Monte Carlo. *Stat Sci.* 1992;7(4):473–83.
44. Madras N, Sokal AD. The pivot algorithm: a highly efficient Monte Carlo method for the self-avoiding walk. *J Stat Phys.* 1988;50:109–86.
45. Nelder JA, Mead R. A simplex method for function minimization. *Comput J.* 1965;7:308–13.
46. Liu DC, Nocedal J. On the limited memory BFGS method for large scale optimization. *Math Program.* 1989;45(1–3):503–28. <https://doi.org/10.1007/BF01589116> (issn: 0025-5610).
47. Caldwell A, et al. Integration with an adaptive harmonic mean algorithm. *Int J Mod Phys A.* 2020;35(24):2050142. <https://doi.org/10.1142/S0217751X20501420>.
48. Hahn T. Cuba—a library for multidimensional numerical integration. *Comput Phys Commun.* 2005;168(2):78–95.
49. Beringer J. Review of particle physics. *Phys Rev D.* 2012;86(010001).
50. Bißmann S, et al. Constraining top-quark couplings combining topquark and B decay observables. *Eur Phys J C.* 2020;80(2):136. <https://doi.org/10.1140/epjc/s10052-020-7680-9> ([arXiv:1909.13632](https://arxiv.org/abs/1909.13632) [hep-ph]).
51. Bißmann S, et al. Correlating uncertainties in global analyses within SMEFT matters. 2019. [arXiv:1912.06090](https://arxiv.org/abs/1912.06090) [hep-ph].
52. Stadnichuk E, et al. Prototype of a segmented scintillator detector for particle flux measurements on spacecraft. 2020. eprint: [arXiv:2005.02620](https://arxiv.org/abs/2005.02620).
53. Caldwell A, et al. Infections and identified cases of COVID-19 from random testing data. 2020. eprint: [arXiv:2005.11277](https://arxiv.org/abs/2005.11277).
54. Agostini M, et al. Final results of GERDA on the search for neutrinoless double- β decay. *Phys Rev Lett.* 2020;125:252502. <https://doi.org/10.1103/PhysRevLett.125.252502>.
55. Bißmann S, et al. Top and Beauty synergies in SMEFT-fits at present and future colliders. 2020. eprint: [arXiv:2012.10456](https://arxiv.org/abs/2012.10456).

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.