To appear in Statistical Science

Past, Present, and Future of Software for Bayesian Inference

Erik Štrumbelj and Alexandre Bouchard-Côté and Jukka Corander and Andrew Gelman and Håvard Rue and Lawrence Murray and Henri Pesonen and Martyn Plummer and Aki Vehtari

Abstract. Software tools for Bayesian inference have undergone rapid evolution in the past three decades, following popularisation of the first generation MCMC-sampler implementations. More recently, exponential growth in the number of users has been stimulated both by the active development of new packages by the machine learning community and popularity of specialist software for particular applications. This review aims to summarize the most popular software and provide a useful map for a reader to navigate the world of Bayesian computation. We anticipate a vigorous continued development of algorithms and corresponding software in multiple research fields, such as probabilistic programming, likelihood-free inference, and Bayesian neural networks, which will further broaden the possibilities for employing the Bayesian paradigm in exciting applications.

Key words and phrases: statistics, data analysis, MCMC, computation, probabilistic programming.

1. INTRODUCTION

In the past three decades, Bayesian inference has established itself as a viable alternative to more classical approaches to statistical inference and is now a must-have tool for every statistician's toolbox. Many theoretical and methodological developments have contributed to the success of Bayesian statistics. However, no development has been as important for mass adoption as was the emergence of accessible and robust software.

Our goal with this paper is to introduce the reader to the history, the state of the art, and the future of software for Bayesian inference. We aim to provide the reader with a comprehensive survey of popular software, key developments in statistics and computing that enabled the software, and the limitations and challenges faced. The paper is aimed both at the Bayesian statistics practitioner and those that are less familiar with the field and would like to learn more about the Bayesian inference tasks and the tools used to solve them.

Before we proceed, we briefly discuss the background and introduce some basic terminology that we use throughout the paper.

1.1 Bayesian Inference

The essence of the Bayesian approach to inference is combining the chosen *likelihood* $p(y|\theta)$ and *prior* distribution $p(\theta)$ of the parameters θ (or the joint distribution $p(\theta, y)$) with the *data* y to compute the posterior distribution of the parameters $p(\theta|y)$. We do it with Bayes' rule:

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)} = \frac{p(y|\theta)p(\theta)}{\int_\Omega p(y|\theta)p(\theta)d\theta} \propto p(y|\theta)p(\theta).$$

The most common quantities of interest in Bayesian inference are posterior properties of parameters or functions thereof, which can be expressed in terms of expectations

Erik Štrumbelj is a Professor at the Faculty of Computer and Information Science, University of Ljubljana, Slovenia (e-mail: erik.strumbelj@fri.uni-lj.si). Alexandre Bouchard-Côté is a Professor of statistics at the University of British Columbia (e-mail: bouchard@stat.ubc.ca). Jukka Corander is a Professor at the Faculty of Medicine, University of Oslo, at the Faculty of Science, University of Helsinki, and Associate Faculty Member at Wellcome Sanger Institute (e-mail: jukka.corander@medisin.uio.no). Andrew Gelman is a Professor of statistics and political science at Columbia University (e-mail: gelman@stat.columbia.edu). Håvard Rue is a Professor of statistics at King Abdullah University of Science and Technology (e-mail: haavard.rue@kaust.edu.sa). Lawrence Murray has contributed to this work as an independent researcher (e-mail: lawrence@indii.org). Henri Pesonen is a Researcher at Oslo Centre for Biostatistics and Epidemiology, Oslo University Hospital (e-mail: henri.pesonen@medisin.uio.no). Martyn Plummer is a Professor of statistics at the University of Warwick, (e-mail: martyn.plummer@warwick.ac.uk). Aki Vehtari is a Professor in computational Bayesian modeling at Aalto University (e-mail: aki.vehtari@aalto.fi).

over the posterior distribution $p(\theta|y)$:

$$E(g(\theta)|y) = \int_{\Omega} g(\theta)p(\theta|y)d\theta.$$

Prior and posterior predictions, model selection, and other quantities of interest follow a similar pattern. Thus, the main computational problem of Bayesian inference is computing integrals.

Our choice of likelihood and prior rarely lead to a closed-form solution for $p(\theta|y)$, which in most cases can only be evaluated up to a multiplicative constant, and even less often to a closed-form solution for the integral. Therefore, computing the quantities of interest is a numerical problem and is a challenge in itself.

1.2 Software for Bayesian Inference

For our discussion of software for Bayesian inference, we divide the software components into three groups: the modeling language, the computation methods, and the utilities.

1.2.1 Modeling Language We use the term modeling language in the broadest sense of a component that allows the user to specify the likelihood, prior, and data (from now on, we use *model* to refer to all of these combined). Alternatively, Bayesian inference can be done by specifying a generative model for $p(\theta, y)$ instead (see Section 4.3) and some languages support specifying both. See Appendix for an illustrative example in different modeling languages.

Every modeling language represents some kind of trade-off between generality and accessibility. On one end of the spectrum are expressive languages, such universal probabilistic programming languages (PPLs), and general-purpose programming languages like Python. On the other end we have software that allows for a single model or a limited number of options. And in between we have Bayesian inference-specific declarative (for example, WinBUGS [74]), imperative (for example, Stan [21]), or formula-based languages (for example, R-INLA [71] and rstanarm [6]) that use syntax similar to the formula object used by generalized linear models (GLMs) in the core R stats package [121]), etc.

The choice of modeling language more so than any other component determines the target user. Or, when the software is designed with a target user in mind, no component is more influenced by the requirements of the target user than the modeling language. And, as demonstrated by the variety of different modeling languages, Bayesian inference users are a heterogeneous group and there is no one-size-fits-all approach.

1.2.2 Computation Methods Once the model is specified, the next step is to perform the computation of the posterior and other quantities of interest. Therefore, a complete software for Bayesian inference must implement one or more Bayesian computation methods.

There is no method that is able to perform practically feasible Bayesian computation for every model. Therefore, many different computation methods have been developed, and each method represents a trade-off between generality and efficiency. The computation method determines the class of models that can be computed and usually limits the software more than the modeling language. That is, it is not uncommon that the modeling language allows for the specification of models that the computation method is not able to compute, not even in theory. And, as a rule, there always exist models that a computation method will not be able to deal with in practice, even though it is able to do so in theory.

In this paper, our treatment of Bayesian computation is from a Bayesian software perspective: we limit ourselves to discussing methods that were key for the development of software for Bayesian inference and listing the methods implemented in the software. For details about the history and the state of the art of Bayesian computation, we refer the reader to [81].

1.2.3 Utilities With utilities, we refer to all software components that do not fall in to the previous two groups, but are still common in Bayesian software and convenient if not essential to the Bayesian inference workflow (for a detailed treatment of the Bayesian workflow, we refer the reader to [41, 46]):

- *Diagnosing Bayesian computation*: Bayesian computation methods can and often do fail to find the optimum solution or, in the case of Markov chain Monte Carlo (MCMC), properly explore the posterior distribution. Diagnostics tools are essential to identifying potential issues before proceeding with the interpretation of the results. Furthermore, most key methods are MCMC and therefore sampling-based and approximate. Approximation error must also be quantified and included in the interpretation of the results. Common diagnostics are traceplots, Monte Carlo standard errors, effective sample size (ESS), $\hat{R}$, and simulation-based calibration [115].
- *Model validation and comparison*: Prior, posterior, and model visualization, prior and posterior predictive checks, (approximate) leave-one-out cross-validation and model evaluation criteria such as WAIC [134], and computing Bayes factors. The modeling language determines how easy or difficult it is to compute these [20]. For example, for prior and posterior predictive checks we have to draw samples from the prior $p(y)$ and posterior predictive distribution $p(y_{new}|y)$. For Bayes factors we have to evaluate the marginal $p(y)$ and for cross-validation we have to evaluate the posterior predictive $p(y_{new}|y)$.

- *Computation libraries*: Matrix algebra libraries, support for probability distributions and other statistical computation, support for high-performance computing, and automatic differentiation (AD) libraries.
- *Interfaces*: Often, Bayesian software provides the user with only low-level command-line interface to the computation, where the data and model are passed as files. For convenience, interfaces are then developed that allow the user to access the computation from a popular higher-level language such as Python and R.
- *Documentation*: This includes software documentation, language definition, examples, case studies, and other material that make it easier to use the software.

1.3 Scope and Organization

In part, this paper is a survey of the most popular and historically most relevant software for general-purpose Bayesian inference. We also include popular software that serve a more specific purpose. For example, software that provides only Bayesian computation of a utility or software that focuses on a more narrow class of models. When it comes to less commonly used and more specific software, this paper is biased towards Python and R, the two most popular languages for data analysis. See Tables 1 and 2 for an estimate of the relative popularity of Bayesian software packages for Python and R, respectively.

General-purpose Bayesian computation has had two distinct periods, each dominated by a certain type of Bayesian computation and software. From the early 1990s to the 2010s, it was Gibbs sampling and the quintessential representative of software is BUGS. From the 2010s up to now, it is Hamiltonian Monte Carlo (HMC) and the quintessential representative is Stan. The first part of the remainder of the paper roughly corresponds to these two periods. In Section 2 we describe Gibbs sampling, the typical structure of Gibbs sampling-based software, and the BUGS language. We also include software that might have been developed later but is related to, was inspired by, or is a continuation of BUGS. Similarly, Section 3 focuses on HMC and Stan.

We dedicate Section 4 to software that we were not able to meaningfully assign to either of the two periods. It features software that focuses on computation, software that targets a more specific class of models, and the latest developments in Bayesian software and universal PPLs.

We discuss the future of software for Bayesian inference in Section 5.

2. FIRST GENERATION - GIBBS SAMPLING-BASED

In the period between the early 90s and early 2010s, the most popular software for general-purpose Bayesian inference was based on graphical models and Gibbs sampling as the method of Bayesian computation.

The main assumption of this approach is that the conditional independence between variables in our joint distribution $p(\mathcal{V}) = p(\theta, y)$ can be represented by a directed acyclic graph (DAG), where each variable is represented by a node and every node is conditionally independent of all other nodes, given its Markov blanket.

A model that admits such a representation is called a Bayesian network and is a class of probabilistic graphical models (see Appendix for an example). The joint distribution can then be factored as

$$p(\mathcal{V}) = \prod_{v \in \mathcal{V}} p(v|\mathcal{P}(v))$$

and the full conditional of a node is

$$\begin{aligned} p(v|\mathcal{V}/v) &\propto p(\mathcal{V}) \\ &\propto p(v|\mathcal{P}(v)) \prod_{u \in \mathcal{V}: v \in \mathcal{P}(u)} p(u|\mathcal{P}(u)), \end{aligned} \tag{1}$$

where $\mathcal{P}(v)$ are parent nodes of v.

A Markov chain that updates one node at a time using its full conditional will converge to the posterior distribution under weak conditions. From a practical perspective, this means that we only have to be able to iteratively sample from the full conditionals. Algorithm 1 is a summary of the Gibbs sampling algorithm. A major appeal of the algorithm is that there are no algorithm parameters that need to be tuned, which is a useful property for automated inference. For the purpose of sampling from a full conditional, a hierarchy of samplers is typically used. Because the model is stated in a symbolic way, it is straightforward to check the properties of the full conditional. In most cases, the more specific the distribution, the more efficient the sampling algorithm that we can use.

Algorithm 1
k - number of nodes,
$p(x_j|x_{-j})$ - k full conditionals,
x_0 - starting value,
m - number of samples

procedure GIBBS SAMPLING( )
  for $i \leftarrow 1 : m$ **do**
    for $j \leftarrow 1 : k$ **do**
      $x_j^{(i)} \sim p(x_j|x_1^{(i)}, \ldots, x_{j-1}^{(i)}, x_{j-1}^{(i-1)}, \ldots, x_k^{(i-1)})$
    end for
    $i-$th sample $\leftarrow x^{(i)}$
  end for
end procedure

2.1 BUGS

The quintessential representative of this approach is BUGS (Bayesian inference Using Gibbs Sampling) [110]. The BUGS project started at the Medical Research Council Biostatistics Unit in Cambridge in 1989. The BUGS

TABLE 1

Total Python Package index (PyPI) downloads for Bayesian inference-related Python packages referenced in this paper for the period between Jan 1 2022 and Dec 31 2022. We obtained the information from the PyPI dataset (bigquery-public-data.pypi). We include matplotlib [63], the most popular Python package for statistical graphics, as a baseline for comparison. While these counts should in most cases be a reasonable proxy for relative popularity, we have to keep in mind that users can also download these packages from other sources. Inclusion in other packages and automated downloads can also bias the results.)

package	download count	description
matplotlib	339834089	Python plotting package
pystan	30416188	Python interface to Stan, a package for Bayesian inference
cmdstanpy	30061999	Python interface to CmdStan
prophet	12279924	Automatic Forecasting Procedure
tensorflow-probability	10666937	Probabilistic modeling and statistical inference in TensorFlow
arviz	7388303	Exploratory analysis of Bayesian models
pymc3	4278991	Probabilistic Programming in Python: Bayesian Modeling and Probabilistic ML with Theano
pyro-ppl	3383969	A Python library for probabilistic modeling and inference
httpstan	1854832	HTTP-based interface to Stan, a package for Bayesian inference.
emcee	1143412	The Python ensemble sampling toolkit for MCMC
pymc	682274	Probabilistic Programming in Python: Bayesian Modeling and Probabilistic ML with PyTensor
numpyro	566378	Pyro PPL on NumPy
dynesty	277271	A dynamic nested sampling package for computing Bayesian posteriors and evidences.
bambi	125998	BAyesian Model Building Interface in Python
elfi	75236	Engine for Likelihood-free Inference
edward	61403	A library for probabilistic modeling, inference, and criticism
blackjax	42843	Flexible and fast inference in Python
pyjags	20140	Python interface to JAGS library for Bayesian data analysis.
oryx	15550	Probabilistic programming and deep learning in JAX
edward2	11457	Edward2

TABLE 2

Total RStudio [120] CRAN mirror downloads for Bayesian inference-related R packages referenced in this paper for the period between Jan 1 2022 and Dec 31 2022. We used the cranlogs package [27]. We include ggplot2 [135], the most popular R package for statistical graphics, as a baseline for comparison. While these counts should in most cases be a good proxy for relative popularity, we have to keep in mind that users can also download these packages from other CRAN mirrors or directly from code repositories. Also, some popular R packages are not available on CRAN, for example, R-INLA, cmdstanr, the R interface to Stan, or R2MultiBUGS, the R interface to MultiBUGS.)

package	download count	description
ggplot2	31457872	Create Elegant Data Visualisations Using the Grammar of Graphics
mgcv	1523237	Mixed GAM Computation Vehicle with Automatic Smoothness Estimation
coda	1190640	Output Analysis and Diagnostics for MCMC
rstan	993086	R Interface to Stan
loo	738325	Efficient Leave-One-Out Cross-Validation and WAIC for Bayesian Models
bayestestR	599283	Understand and Describe Bayesian Models and Posterior Distributions
prophet	338276	Automatic Forecasting Procedure
posterior	314669	Tools for Working with Posterior Distributions
bayesplot	308747	Plotting for Bayesian Models
bnlearn	286003	Bayesian Network Structure Learning, Parameter Learning and Inference
shinystan	272855	Interactive Visual and Numerical Diagnostics and Posterior Analysis for Bayesian Models
BayesFactor	239538	Computation of Bayes Factors for Common Designs
rjags	228433	Bayesian Graphical Models using MCMC
brms	215302	Bayesian Regression Models using Stan
MCMCpack	186124	Markov Chain Monte Carlo (MCMC) Package
rstanarm	164469	Bayesian Applied Regression Modeling via Stan
bridgesampling	155278	Bridge Sampling for Marginal Likelihoods and Bayes Factors
R2WinBUGS	61926	Running WinBUGS and OpenBUGS from R SPLUS
nimble	36471	MCMC, Particle Filtering, and Programmable Hierarchical Modeling
abc	36251	Tools for Approximate Bayesian Computation (ABC)
R2OpenBUGS	27284	Running OpenBUGS from R
greta	8453	Simple and Scalable Statistical Modeling in R
abctools	6404	Tools for ABC Analyses
EasyABC	5344	Efficient Approximate Bayesian Computation Sampling Schemes

software evolved into WinBUGS [74, 76, 112], which updated the BUGS language and the sampling algorithms, and OpenBUGS, a GNU General Public License release of WinBUGS that also runs on Linux (with some limitations) [111]. BUGS, WinBUGS, and OpenBUGS are no longer developed, but the BUGS project has inspired other software, which we discuss at the end of this section. A detailed history of BUGS is provided by Lunn et al. [75].

The factorization from Eq. (1) is the basis for both the BUGS language and the Gibbs sampling-based computation. The BUGS language is a declarative language in which the user states all the parent-child relationships $\mathcal{P}(v)$ between the variables in the model. See Appendix for an example of a Bayesian network model in JAGS, a language which is very similar to WinBUGS.

For sampling from the full conditionals, WinBUGS implements a different approach for each of the following contingencies [74, Chapter 12.1]:

- Discrete distribution: The inverse CDF method.
- Standard distribution: Standard algorithm for that distribution.
- Log-concave: Derivative free adaptive rejection sampling [47]. Many standard distributions are log-concave, including the exponential family. The product of log-concave is also log-concave, so it is common for the full-conditional to be log-concave.
- Restricted range: Slice sampling [91].
- Unrestricted range: Current point Metropolis.

OpenBUGS includes block sampling methods that jointly sample from groups of nodes that are likely to be correlated based on the structure of the model. Block updating solves one of the disadvantages of Gibbs sampling: it is strongly dependent on the parameterization of the model. If two variables have a high posterior correlation but are updated independently using Gibbs sampling, then the Markov chain will exhibit high autocorrelation for both variables. Block updating of correlated nodes solves this problem, which otherwise falls to the user to solve by reparameterizing the model.

A strong point of the BUGS PPL is that the distinction between data and parameters is made at run time, based on provided observations. Vectors can also be partially observed, by leaving the unobserved elements unknown (NA). This simplifies the simulation of draws for posterior checks. Although WinBUGS focuses on Bayesian networks, there is some limited support for undirected graphs (factor models) as long as the entire subset of variables is represented as a single multivariate node so that their values are sampled jointly. WinBUGS also supports graphical model specification in plate notation with the DoodleBUGS editor.

MultiBUGS [54] is a continuation of the BUGS project. The major contribution of MultiBUGS is that it provides a more efficient implementation and several parallelization techniques. In a multi-core environment, MultiBUGS can be several orders of magnitude more efficient than OpenBUGS.

R interfaces are available for WinBUGS, OpenBUGS, and MultiBUGS: R2WinBUGS [113], R2OpenBUGS [126], and R2MultiBUGS.[1]

2.2 JAGS

JAGS (Just Another Gibbs Sampler) [97] is similar to WinBUGS in its language and computation (see [74, Chapter 12.6] for differences). Unlike WinBUGS and OpenBUGS, which are written in Component Pascal, JAGS is written in C++ and portable. This has contributed to its popularity and the fact that is still being actively developed. See Appendix for an example of a model written in JAGS.

JAGS (Just Another Gibbs Sampler) is a clone of BUGS that has a completely independent code base but aims for similar functionality, although it notably lacks a graphical user interface (see [74, Chapter 12.6] for a summary of differences). JAGS is written in C++ and runs on Windows, MacOS, and Linux. It is published under the GNU General Public License, version 2. JAGS incorporates a copy of the R math library, which provides high-quality algorithms for random number generation and calculation of quantities associated with probability distributions. The workhorse sampling method for JAGS is slice sampling [91], which can be applied to both continuous- and discrete-valued nodes. The "glm" module of JAGS incorporates efficient samplers for generalized linear mixed models (GLMMs). These samplers are based on the principle of data augmentation, a commonly used technique to simplify sampling from a graphical model by adding new nodes [59]. In this case, data augmentation reduces GLMMs with binary outcomes [1, 57, 98] or binary and Poisson outcomes [39] to a linear model with normal outcomes. This reduction to a normal linear model allows block updating of all the parameters in the linear predictor, which is much more efficient than Gibbs sampling. The underlying engine for the linear model uses sparse matrix algebra [29], which handles fixed and random effects simultaneously.

2.3 Nimble

Nimble [30], similar to BUGS, focuses on graphical models. It is an extension of the BUGS language but also implements a modeling language embedded in R, both of which are compiled to C++. Several Bayesian computation methods are implemented, including Metropolis-Hastings (MH), Gibbs sampling, and sequential Monte Carlo (SMC), and the user has the flexibility of assigning different sampling methods to different nodes. Recently they have also added support for AD and HMC. See Appendix for an example of a model written in Nimble's R-embedded language.

[1] https://github.com/MultiBUGS/R2MultiBUGS

3. SECOND GENERATION - HMC-BASED

The two main drawbacks of the BUGS-like approach are the limited expressiveness of the language (imperative language, no local variables, conditional statements...) and the inefficiency of computation. The single-node exploration of Gibbs sampling is inefficient when the nodes are highly correlated in the posterior, in particular, when the dimensionality in terms of parameters is high, it reverts to random walk behavior [61, 90].

The only MCMC algorithm that theoretically scales to high dimensions on a broad class of models is HMC. Introductions to HMC have been provided, for example, by Neal [89] and Betancourt [10], and a more detailed mathematical treatment by Betancourt [11].

HMC is a physics-inspired approach to proposing the next state that uses the gradient of the target density for a better understanding of its geometry. Hamiltonian dynamics consist of a d-dimensional position vector q and a d-dimensional momentum vector p. The evolution of the system as a function of "algorithmic time," t, is determined by the function $H(q,p)$ (the Hamiltonian) and the ordinary differential equations:

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

To simulate Hamiltonian dynamics, we need to discretize time with some step size ϵ. The most commonly used method is the leapfrog symplectic integrator. Hamiltonian dynamics have several properties, which are important for HMC to work: they preserve the Hamiltonian, they are reversible, and they are symplectic and thus volume preserving.

For HMC the Hamiltonian H is typically chosen so that it is separable: $H(q,p) = U(q) + K(p)$, where $U(q)$ is the potential energy and $K(p)$ the kinetic energy of the system. The main idea of HMC is to use the Hamiltonian to define a joint density of position and momentum:

$$p(q,p) \propto e^{-H(q,p)} = e^{-U(q)} e^{-K(p)}.$$

Substituting $U(q) = -\log f(q)$, where f is proportional to the density we want to sample from, and use standard kinetic energy, we get:

$$p(q,p) \propto f(q) e^{-\frac{1}{2} p^T M^{-1} p}.$$

The joint density $p(q,p)$ can be seen as the target density over the position vector q augmented by an independent multivariate Gaussian for the momentum vector p, with mean 0 and covariance M.

Hamiltonian dynamics conserves the Hamiltonian, so all states on a trajectory will have the same density $p(\cdot,\cdot)$. That makes Hamiltonian dynamics suitable for proposing the next state in an MCMC algorithm, because a trajectory can propose a state far away in position q from the current state, but still with acceptance probability 1. To reach every possible state, we have to sample a new momentum. Because the kinetic and potential energy parts of the joint density are independent and we are sampling from the actual distribution of momentum p, this sampling leaves the target distribution invariant. That is, $p(q,p)$ remains the stationary distribution of the Markov chain. In practice, however, the leapfrog method, while being a stable simulation of Hamiltonian dynamics, will not conserve the Hamiltonian exactly, and there will be relatively small fluctuations. That is why we still have to apply a Metropolis correction. Putting it all together, Algorithm 2 summarizes the basic HMC algorithm.

Algorithm 2

f - a function proportional to our target density,
q_0 - starting value,
ϵ - step size,
L - number of steps,
M mass matrix,
m - number of samples

procedure HMC
  for $i \leftarrow 1 : m$ **do**
    $p \sim N(0, M)$ ▷ resample momentum
    get (q^*, p^*) with L leapfrog steps of size ϵ from (q_{i-1}, p)
    $\alpha \leftarrow \min\left\{1, e^{-H(q^*,p^*)+H(q_{i-1},p)}\right\}$
    sample $u \sim U(0,1)$
    if $u \leq \alpha$ **then** ▷ Metropolis correction
      $q_i \leftarrow q^*$ ▷ accept transition
    else
      $q_i \leftarrow q_{i-1}$
    end if
  end for
  $i-$ th sample $\leftarrow q_i$
end procedure

The main ideas behind HMC had been known for more than 20 years before HMC featured in popular Bayesian software [33]. The key enabler of more automatic use of HMC was the development of automatic differentiation (AD; see [5] for an introduction and survey). Simulating the Hamiltonian dynamics of HMC requires the gradient of the density and in order for the software to be general-purpose, we must be able to compute the gradient for any program the user can code. Of the four general approaches to computing derivatives, three will not work: manually deriving them is not practical, numerical differentiation via finite differences is too unstable due to rounding and truncation errors and also is slow in high dimensions, and symbolic differentiation suffers from expression swell and leads to inefficient code. AD instead exploits the fact that every program is a composition of elementary operations and, as long as each elementary operation also implements a derivative, we can apply the chain rule to derive the gradient of the composition. This leads to machine level

precision of gradients. Most modern inference software implements or imports an AD library. An important limitation of HMC is that it can only be used on smooth spaces.

A challenge in making HMC useful in general-purpose Bayesian inference is automatically tuning its parameters (mass matrix, step size, number of steps). HMC-based software typically implements one or more warmup phases for parameter tuning. Software then proceeds with sampling and the warmup samples are discarded. The key development was the no-U-turn sampler (NUTS) [61], which is, with some modifications, still the core Bayesian computation method in Stan. The basic idea of NUTS is to have a dynamic number of steps by simulating the Hamiltonian trajectory until we detect a turn back towards the starting state (or reach the maximum number of steps). While promising alternatives for tuning the number of steps, including more GPU computation friendly variants [60, 62], NUTS is still the most common implementation of HMC and is also available in most modern software for general-purpose Bayesian inference.

HMC/NUTS admits several specific MCMC diagnostics [10]: when the step size is too large to capture a feature of the target density (which can lead to non-negligible bias), this is likely to manifest as a diverging simulation which can be detected and we can use a smaller step size; reaching the maximum number of steps before terminating the trajectory is an indication of inefficient exploration; The Bayesian fraction of missing information (BMFI) [9] quantifies how well momentum resampling matches the marginal energy distribution and can be used to detect poor adaptation during warmup or inefficient exploration.

3.1 Stan

Stan [21] is by far the most popular software for general-purpose Bayesian inference. Stan is implemented in C++, has a standalone command line interface, but also has mature interfaces for Python and R (RStan [122], PyStan [101]) and lightweigt wrappers for Python and R (CmdStanPy [123], CmdStanR [42], BridgeStan [2]). There are also interfaces for most languages that are traditionally used for data analysis: Matlab (Matlabstan), Julia (Stan.jl), Stata (StataStan), Mathematica (MathematicaStan), Scala (ScalaStan), and http request-based interface (httpstan).

While Stan implements black-box variational inference [69], Laplace approximation, and standard optimization methods, the core Bayesian computation method is NUTS, a variant of HMC. Stan has a rich mathematics library with AD [22], and OpenCL-based GPU support with kernel fusion [23, 24].

The Stan PPL is an imperative language with which the user specifies the computation of the (log-)posterior. A program is divided into blocks, the most important of which are data, parameters, and model. See Appendix for an example of a model written in Stan. The distinction between data and parameters is made at compile time, so changing a variable from data to a parameter (or vice versa) requires moving it from the data to the parameter block (or vice versa) and recompiling. Notable work on the Stan language includes SlicStan [51, 52, 53], which contains several improvements, and translating Stan to Pyro [4].

Because of HMC-based computation, the class of models that can be fit by Stan are models with a smooth density. An important omission are models with discrete parameters, which currently have to be manually marginalized out. This means that Stan does not subsume what can be fit with BUGS and that HMC does not make Gibbs sampling-based software obsolete. However, empirical evidence suggests that, when applicable, Stan is currently the go-to software for general-purpose Bayesian inference [7].

The majority of Stan users are not writing the models directly in the Stan language. There are several popular packages that provide a simplified formula or options-based modeling language for a mode specific class of models and use a Stan backend for modeling and computation: the R package brms [16] for modeling with hierarchical models; Prophet [117] implemented in Python [119] and R [118] for nonlinear time series forecasting with trend, seasonality, and holiday effects; and the R package rstanarm [6] for a Bayesian analogue to R lm, glm, aov, etc. Overall, there are more than 140 R packages built on top Stan, providing easy to use interfaces for various types of models common in different applications. The success of these packages is not only due to Stan, but also due to increasing number of useful utilities in R, Python, and Julia.

3.2 ADMB

AD Model Builder (ADMB) [38] was the first PPL based on AD. It is similar to Stan in that data, parameters, and the likelihood and priors are described separately and that a distinction is made between data and parameters. ADMB is tailored more to the optimization-based inference, but also implements MH, Laplace approximation, and HMC with manual tuning. A third-party implementation of NUTS for ADMB is available [84].

3.3 PyMC

PyMC [105] is a Python library for Bayesian inference. It includes HMC, SMC, and black-box variational inference. It is based on PyTensor,[3] a Python mathematics library that is a fork of Aesara, and continuation of the no-longer-developed Theano [8], which was the PyMC3 backend up to the current major release 4.0 and the renaming to PyMC. The computational graphs in PyMC

[2] github.com/roualdes/bridgestan

[3] www.github.com/pymc-devs/pytensor

are transpiled to C-code, Numba, or JAX [15] (a high-performance AD library for running Pyton/NumPy code on CPU, GPU, and TPU), which allows for highly optimized code. PyMC syntax is similar to other Bayesian software PPLs. See Appendix for an example of a model written in PyMC. The Bambi (BAyesian Model-Building Interface) package [19] is built on PyMC and designed to simplify the use of hierachical GLMs.

4. OTHER SOFTWARE

4.1 R-INLA

R-INLA [71, 130] is a popular R package for Bayesian inference with latent Gaussian models. This class of models does not require a PPL; instead, the models are specified with a standard R formula, similar to lm/glm in the core R stats package [121] and extended formula syntax for smooths and hierarchical ("random effect") terms similar as, for example, in the R package mgcv [137]. Latent Gaussian models, when the number of hyperparameters is moderate and some additional assumptions, allow for efficient computation using integrated nested Laplace approximation (INLA) [82, 103, 104], an approximate Bayesian computation method. For models that meet these criteria, R-INLA is a very efficient alternative to MCMC methods and would be difficult to replace. A key feature of R-INLA is the support for continuous spatial models using the stochastic partial differential equation (SPDE) approach [3, 68, 72]. Recently, the model representation has been improved and the inner Laplace approximation have been replaced with a variational Bayes correction layer, to facilitate better scaling properties with respect to data size, model size, and number of computing cores [44, 131, 132]. This can be expanded to variance, skewness, and to correcting marginals for hyperparameters.

4.2 Universal PPLs

No agreed upon definition of what a *universal* PPL is exists. But it is generally accepted that a universal PPL program can have probabilistic operations anywhere. For example, that not even the number of random variables can be determined statically. We will use the notion of inverting simulators used in [100]; that is, that the user codes a stochastic simulation and the PPL framework is able to infer the properties of the simulation given the observed data.

In this sense, BUGS, Stan, and other languages mentioned so far are not universal PPLs; they can be viewed as Bayesian inference-driven systems that streamline the Bayesian inference workflow within classes of models for which inference can easily be automated. Designing a universal PPL primarily focuses on having a general-purpose language and then an inference framework that is able to handle all the algorithms that can be specified. In theory a universal PPL subsumes Bayesian inference and it is arguably easier to code a stochastic simulator than it is to design an appropriate statistical inference. However, it is not clear if inference can be automated and be efficient enough for such a broad class of algorithms.

From a Bayesian statistics practitioner's perspective, universal PPLs are still more an object of research than of general practical use. However, there have been many promising developments. In the remainder of this section we highlight some of the more popular or recent universal PPLs. Other relevant related works include early universal PPL languages Church [50], Venture [79], and Anglican [128], Julia-based Gen [28], Turing.jl [45] (and its more recent frontend DynamicPPL [116]), and Python-embedded Edward/Edward2 [129].

4.2.1 Bean Machine [124] is Bayesian software and a declarative universal PPL embedded in Python with a PyTorch backend. In essence, Bean Machine allows for a specification of a distribution over Bayesian networks with possibly different numbers of variables. While imperative languages are becoming more common, including the currently most popular Stan, the authors argue for declarative PPLs over imperative ones. In particular, that the (possibly dynamic) dependency structure between variables is more easily recovered from a declarative model description and that inference can more easily be adapted to individual blocks of variables, including second-order methods that are usually infeasible in higher dimensions. Bean Machine implements several single-site samplers, NUTS, Newtonian Monte Carlo, and black box VI. It allows for blocking of variables and custom proposers. See Appendix for an example of a model written in Bean Machine.

4.2.2 Birch [88] is a universal PPL that transpiles to C++, with GPU support. Users implement the joint distribution of their model in a generative manner, with a preference for generic and object-oriented programming paradigms. Inference methods are based on SMC with gradient-based kernels. A defining feature of Birch is support for *automatic marginalization* and *automatic conditioning*. Much like AD, these recognize known forms, such as conjugacies and discrete enumerations, to marginalize out random variables where possible, and condition them on later simulations where necessary. The implementation of these is based on a heuristic known as *delayed sampling* [87], which reveals these opportunities during program execution by deferring the simulation of random variables for as long as possible. The result is the automatic enhancement of inference methods with features such as Rao-Blackwellization [87] and variable elimination [136]. Birch has been demonstrated on problems where the number of random variables is unknown, such as multi-object tracking [88] (where the number of objects is unknown), and statistical phylogenetics [102] (where the number of extinct side branches of a phylogeny is unknown). See Appendix for an example of a model written in Birch.

4.2.3 Pyro [12] is a Python PPL built on the PyTorch [93] backend. The main computation method in Pyro is stochastic variational inference, so the software is aimed at scalable probabilistic machine learning. NumPyro [94] is a NumPy-based backend for the Pyro PPL that uses JAX for AD and compilation to CPU/GPU.

4.2.4 Blang [14] is an open source package for approximating posterior distributions over arbitrary spaces, i.e. Bayesian models containing not only integer and real variables but also user-defined datatypes such as phylogenetic trees, random graphs and sequence alignments. The Blang project includes a standard library of common datatypes and distributions, written in the Blang language, and extension points to create new datatypes and associated distributions. Users can publish versioned Blang packages containing new datatypes and distributions and import contributed packages and their transitive dependencies. The Blang language's scoping rules are used to automatically detect sparsity patterns and construct a type of graphical model known as a factor graph. Based on this factor graph, the posterior distribution is approximated via an adaptive non-reversible parallel tempering algorithm [114], which by default is parallelized over the user's CPU cores, but can also be distributed over MPI (Message Passing Interface) thanks to Blang's integration with the Pigeons distributed Parallel Tempering package.[4] See Appendix for an example of a model written in Blang.

4.3 Likelihood-free Bayesian inference

Likelihood-free inference (LFI) methods such as approximate Bayesian computation (ABC) [108], Bayesian synthetic likelihood (BSL) [99], machine learning-based posterior approximations and surrogate likelihood methods [56, 25] refer to (mostly) Bayesian computation methods that can be used when it is impossible or infeasible to evaluate the likelihood function, but a generative simulator model exists. Such methods are popular for example in astrostatistics, genetics, ecology, systems biology and human cognition modeling. Engine for Likelihood-Free Inference (ELFI) [73] is a Python package for LFI that covers all the main approaches (ABC, BSL and ML-based methods). ELFI has a modular design that consists of a DAG-based modeling API and a separate API for inference, allowing a user to choose flexibly from a selection of algorithms that generate a sample from the approximate posterior distribution. Sampling can be done using adaptive Importance Sampling or MCMC/HMC, and with or without the use of a surrogate model for the likelihood function approximation. The surrogate model emulates a target function using Gaussian processes (GPs) and active learning (Bayesian optimization). The active learning approach has been demonstrated to accelerate likelihood-free inference up to several orders of magnitude. Other general-purpose ABC packages are Python packages pyABC [106] and ABCpy [34], and R packages abc [26], EasyABC [64], and abctools [92] and ABCreg [127]. Neural network based surrogate models are accessible via Python package sbi [125] and R package [2] provides a toolbox for BSL. More detailed surveys of ABC software are provided by Nunes and Prangle[92] and Kousathanas et al. [67].

4.4 Software that Focuses on Computation

Blackjax[5] is a Python library of MCMC methods for JAX. It works on CPU and GPU, is robust, efficient, and easily integrates with PPLs that provide densities compatible with JAX (TFP, Oryx, NumPyro, Aesara, PyTensor/PyMC).

emcee [36, 37] is a Python implementation of the Affine Invariant MCMC ensemble Bayesian computation method [49]. This derivative-free approach is suitable for low-dimensional problems with black-box likelihoods, which are common in astrophysics. Another Python package that is popular in astrophysics is dynesty [109], which implements dynamic nested sampling [58]

Mamba.jl[6] is a Julia package aimed at users who want to use and develop MCMC methods. It implements several MCMC methods (HMC, NUTS, Metropolis-within-Gibbs, etc.) and MCMC diagnostics. Another popular Julia package that implements state-of-the-art Bayesian computation methods is DynamicHMC.jl.[7]

4.5 Other general-purpose software

Other general purpose software includes Infer.NET [83] is a machine learning library written in C# for the .NET framework. It facilitates automatic approximate inference for Bayesian networks and Markov random fields. Bayesian computation is mostly limited to message passing. TensorFlow Probability (TFP) [31] is a Python library built on TensorFlow [35]. An example of a PPL with a very compact syntax is greta [48], a PPL embedded in R but based on TensorFlow and TFP. While limited in the Bayesian computation methods provided, it is extensible. See Appendix for an example of a model written in greta. Oryx[8] is a PPL built on top of JAX. Journal of Statistical Software also recently published a special issue on Bayesian software [18], which includes some software covered by this paper and other specialized software.

[4] https://github.com/Julia-Tempering/Pigeons.jl

[5] https://github.com/blackjax-devs/blackjax

[6] https://github.com/brian-j-smith/Mamba.jl

[7] https://github.com/tpapp/DynamicHMC.jl

[8] https://github.com/jax-ml/oryx/

4.6 Popular utilities and specialized software

CODA [96] is a still popular R package for post-hoc diagnostics and analysis of MCMC output. ArviZ [70] is a Python package which provides MCMC diagnostics, model evaluation, and model validation tools. ArviZ is backend-agnostic and currently the most popular such tool in Python. The R package bridgesampling [55] estimates marginal likelihoods, Bayes factors, posterior model probabilities, and normalizing constants. The R package posterior [17] subsets, binds, mutates, and converts between formats of MCMC samples and includes lightweight implementations of state-of-the-art posterior inference diagnostics. BayesFactor [85] is an R package for computing Bayes factors for contingency tables, one- and two-sample designs, one-way designs, general ANOVA designs, and linear regression. The R package shinystan [43] provides a graphical user interface for interactive Markov chain Monte Carlo (MCMC) diagnostics and other tools for analyzing a posterior sample. The procedures in shinystan are agnostic to what generated the MCMC samples but with some added functionality for models fit with RStan. The R package loo [133] performs efficient approximate leave-one-out cross-validation for Bayesian models fit using MCMC methods. The R package bayestestR [78] has tools for dealing with uncertainty and effects in a Bayesian statistics framework. It is agnostic of the software that generated the posterior samples and includes MAP estimates, measures of dispersion, ROPE, and Bayes factors. The R package bayesplot [40] has graphing functions for Bayesian models, including posterior draws, visual MCMC diagnostics, and graphical predictive checking. The R package projpred [95] performs projection predictive variable selection for Bayesian generalized linear and additive multilevel models fit using MCMC methods. The R package priorsense [66] performs efficient prior and likelihood sensitivity analysis for Bayesian models fit using MCMC methods. MCMCpack [80] is an R package that implements MCMC-based computation for several statistical methods. Tools for structural learning and parameter estimation of Bayesian networks include bnlearn [107], Bayes Net for Matlab [86], HUGIN [77], VIBES [13], MSBNx [65], along with commercial tools GEnIe/SMILE [32] and Netica.[9]

5. CHALLENGES AND FUTURE PERSPECTIVES

The field of software for Bayesian inference has never been more active or varied. There are developments in all directions, providing better tools that allow for more accessible, robust, or efficient treatment of typical modeling as well as pushing the boundaries of what can be done.

[9] https://www.norsys.com/netica.html

Similar to programming languages, where one might prefer Python for general-purpose programming, R for data wrangling and visualization, or the emerging Julia for high-performance data analytics, there is no one-size-fits-all approach to software for Bayesian inference. Stan is the typical choice for Bayesian model building and inference, Pyro or TFP for Bayesian machine learning, and numerous other tools for more specialized tasks. Such diversity is understandable, because limiting the tool simplifies it and allows for more efficient computation. While there has been some encouraging progress in universal PPLs and underlying Bayesian computation, it is not yet clear if a novel trade-off between expressivity and efficiency can be struck, leading to a third generation of tools.

As a result, users have to either accept the limitations of their tool of choice to learn how to work with multiple tools and languages. A natural solution would be to automatically translate between languages or from statistical notation into code, as illustrated in Appendix A. This is a difficult problem, because languages differ in expressivity and even when they exist, automatic translations could result in inefficient code. Regardless, there appears to be a relative lack of incentive in this area.

A new PPL is most often learned from model examples with code and data, or from translations from a language we are already familiar with. So it is not a surprise that popular PPLs such as BUGS and Stan have extensive documentation, including user's manuals, case studies, and in the case of Stan, examples of translations from BUGS to Stan. Popular PPLs are also accessible on all popular platforms and through major programming languages (typically standalone with interfaces), have open governance and an active community that facilitates communication between users and developers, typically from their genesis.

As the standards for statistical analysis rise, support for the statistical workflow is also becoming more important. To an extent, this is already addressed by some excellent standalone utilities. However, certain parts of the workflow are more difficult to encapsulate because they rely on the underlying language or computation. Another issue is that sometimes it may be difficult to pinpoint from a software point of view if the analysis is not working as expected, especially when black-box components are used.

The practical importance of scalability is already acknowledged in modern software for Bayesian inference, and all popular languages have at least some support for it, either through third-party or native libraries. Scalability with respect to data size is today primarily handled with optimized matrix algebra computation on massively parallel devices such as GPUs. We anticipate that this will further improve with developments in hardware and computation libraries. Scalability with respect to model size depends more on the Bayesian computation used. HMC is currently the state of the art for general-purpose fully-Bayesian computation. Some limitations of HMC, as is the case with

most other types of general Bayesian computation, can be overcome with careful reparametrization or by assigning different computation to different blocks of parameters. However, there is currently no automated approach to this. It remains to be seen if and how general-purpose software (Stan, JAGS, MultiBUGS) will be superseded and what role will universal PPLs, approximate Bayesian computation, and SMC play.

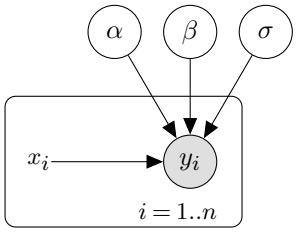
APPENDIX A: MODELING LANGUAGE EXAMPLES

In this Appendix, we illustrate several modeling languages with this example of Bayesian linear regression:

$$
\begin{aligned}
y_i|\beta,\alpha,\sigma,x_i &\sim N(\beta x_i + \alpha,\, \sigma^2),\, i = 1 \ldots n \\
\alpha &\sim N(0, 5^2) \\
\beta &\sim N(0, 5^2) \\
\sigma &\sim U(0, 10),
\end{aligned}
$$

where y_i is the dependent variable and x_i is the predictor.

This model is in the class of Bayesian networks. Its representation as a graphical model in plate notation is:



JAGS

```
model {
  for (i in 1:n) {
    y[i] ~ dnorm(beta * x[i] + alpha,  1 / (sigma * sigma))
  }

  alpha ~ dnorm(0, 1 / 25)
  beta  ~ dnorm(0, 1 / 25)
  sigma ~ dunif(0, 10)
}
```

Nimble

```
nimbleCode({
  for(i in 1:n) {
    y[i] ~ dnorm(beta * x[i] + alpha, sd = sigma)
  }

  alpha ~ dnorm(0, sd = 5)
  beta  ~ dnorm(0, sd = 5)
  sigma ~ dunif(0, 10)
})
```

PyMC

```
with Model() as model:
    sigma = Uniform("sigma", lower = 0, upper = 10)
    alpha = Normal("alpha", 0, sigma = 5)
    beta = Normal("beta", 0, sigma = 5)

    likelihood = Normal("y", mu = beta * x + alpha,
    sigma = sigma, observed = y)
```

Stan

```
data {
  int<lower=0> n;
  vector[n] x;
  vector[n] y;
}
parameters {
  real alpha;
  real beta;
  real<lower=0, upper=10> sigma;
}
model {
  y ~ normal(beta * x + alpha, sigma);
  alpha ~ normal(0, 5);
  beta ~ normal(0, 5);
}
```

Bean Machine

```
@bm.random_variable
def alpha():
  return Normal(0, 5)
@bm.random_variable
def beta():
  return Normal(0, 5)
@bm.random_variable
def sigma():
  return Uniform(0, 1)
@bm.random_variable
def x(i):
  return Normal(0, sigma())
@bm.random_variable
def y():
  return Normal(logit = beta() * x + alpha(), sigma())
```

Birch

```
alpha ~ Normal(0.0, 25.0);
beta ~ Normal(0.0, 25.0);
sigma ~ Uniform(0.0, 10.0);
y ~ Normal(beta*x + alpha, sigma*sigma);
```

Greta

```
alpha <- normal(0, 5)
beta  <- normal(0, 5)
sigma <- uniform(0, 10)

distribution(y) <- normal(beta * x + alpha, sigma)
```

ELFI

```
def linear_regression(alpha,
                      beta,
                      sigma,
                      x,
                      batch_size=1,
                      random_state=None):

    x = x.reshape(1,-1)
    n = x.shape[1]
    random_state = random_state or numpy.random
    alpha = numpy.repeat(alpha.reshape(-1,1), n, axis=1)
    beta_x = numpy.matmul(beta.reshape(-1, 1), x)
    noise = numpy.matmul(
        random_state.randn(n, batch_size),
```

```
        numpy.diag(sigma)).T
    y = alpha + beta_x + noise

    return y

m = elfi.ElfiModel()
elfi.Prior('normal', 0, 5, model=m, name='alpha')
elfi.Prior('normal', 0, 5, model=m, name='beta')
elfi.Prior('uniform', 0, 10, model=m, name='sigma')
elfi.Simulator(linear_regression,
               m['alpha'],
               m['beta'],
               m['sigma'],
               x,
               name='linreg')
```

Blang

```
model LinRegression {
  param GlobalDataSource data
  param Plate<Integer> observationPlate
  param Plated<RealVar> x

  random RealVar alpha, beta, sigma
  random Plated<RealVar> y

  laws {
    alpha ~ Normal(0, 25)
    beta ~ Normal(0, 25)
    sigma ~ ContinuousUniform(0, 10)
    for (Index<Integer> i : observationPlate.indices) {
      y.get(i) | beta, alpha, sigma, RealVar x_i = x.get(i)
        ~ Normal(beta * x_i + alpha, sigma * sigma)
    }
  }
}
```


ACKNOWLEDGMENTS

Special thanks to Christian Robert for the initiative and encouragement for this work.

FUNDING

Erik Štrumbelj's work is partially funded by the Slovenian Research Agency (research core funding No. P2-0442). Andrew Gelman's work is partially funded by the U.S. Office of Naval Research.

REFERENCES

[1] ALBERT, J. H. and CHIB, S. (1993). Bayesian analysis of binary and polychotomous response data. *Journal of the American Statistical Association* **88** 669–679.

[2] AN, Z., SOUTH, L. F. and DROVANDI, C. (2022). BSL: An R package for efficient parameter estimation for simulation-based models via Bayesian synthetic likelihood. *Journal of Statistical Software* **101** 1–33.

[3] BAKKA, H., RUE, H., FUGLSTAD, G.-A., RIEBLER, A., BOLIN, D., ILLIAN, J., KRAINSKI, E., SIMPSON, D. and LINDGREN, F. (2018). Spatial modeling with R-INLA: A review. *Wiley Interdisciplinary Reviews: Computational Statistics* **10** e1443.

[4] BAUDART, G., BURRONI, J., HIRZEL, M., MANDEL, L. and SHINNAR, A. (2021). Compiling Stan to generative probabilistic languages and extension to deep probabilistic programming. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation* 497–510.

[5] BAYDIN, A. G., PEARLMUTTER, B. A., RADUL, A. A. and SISKIND, J. M. (2018). Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research* **18** 1–43.

[6] BEN GOODRICH, I. A. JONAH GABRY and BRILLEMAN, S. (2021). rstanarm: Bayesian applied regression modeling via Stan R package version 2.21.3, https://CRAN.R-project.org/package=rstanarm.

[7] BERAHA, M., FALCO, D. and GUGLIELMI, A. (2021). JAGS, NIMBLE, Stan: A detailed comparison among Bayesian MCMC software. *arXiv:2107.09357*.

[8] BERGSTRA, J., BREULEUX, O., BASTIEN, F., LAMBLIN, P., PASCANU, R., DESJARDINS, G., TURIAN, J., WARDE-FARLEY, D. and BENGIO, Y. (2010). Theano: A CPU and GPU math expression compiler. In *Proceedings of the Python for scientific computing conference (SciPy)* **4** 1–7. Austin, TX.

[9] BETANCOURT, M. (2016). Diagnosing suboptimal cotangent disintegrations in Hamiltonian Monte Carlo. *arXiv:1604.00695*.

[10] BETANCOURT, M. (2017). A conceptual introduction to Hamiltonian Monte Carlo. *arXiv:1701.02434*.

[11] BETANCOURT, M., BYRNE, S., LIVINGSTONE, S. and GIROLAMI, M. (2017). The geometric foundations of Hamiltonian Monte Carlo. *Bernoulli* **23** 2257–2298.

[12] BINGHAM, E., CHEN, J. P., JANKOWIAK, M., OBERMEYER, F., PRADHAN, N., KARALETSOS, T., SINGH, R., SZERLIP, P., HORSFALL, P. and GOODMAN, N. D. (2019). Pyro: Deep universal probabilistic programming. *Journal of Machine Learning Research* **20** 973–978.

[13] BISHOP, C., SPIEGELHALTER, D. and WINN, J. (2002). VIBES: A variational inference engine for Bayesian networks. *Advances in Neural Information Processing Systems* **15**.

[14] BOUCHARD-CÔTÉ, A., CHERN, K., CUBRANIC, D., HOSSEINI, S., HUME, J., LEPUR, M., OUYANG, Z. and SGARBI, G. (2022). Blang: Bayesian declarative modeling of general data structures and inference via algorithms based on distribution continua. *Journal of Statistical Software* **103** 1–98.

[15] BRADBURY, J., FROSTIG, R., HAWKINS, P., JOHNSON, M. J., LEARY, C., MACLAURIN, D., NECULA, G., PASZKE, A., VANDERPLAS, J., WANDERMAN-MILNE, S. and ZHANG, Q. (2018). JAX: Composable transformations of Python+NumPy programs. version 0.3.13, http://github.com/google/jax/.

[16] BÜRKNER, P.-C. (2017). brms: An R package for Bayesian multilevel models using Stan. *Journal of Statistical Software* **80** 1–28.

[17] BÜRKNER, P.-C., GABRY, J., KAY, M. and VEHTARI, A. (2022). posterior: Tools for working with posterior distributions R package version 1.3.1, https://mc-stan.org/posterior/.

[18] CAMELETTI, M. and GÓMEZ-RUBIO, V. (2021). Software for Bayesian statistics. *Journal of Statistical Software* **100** 1–7.

[19] CAPRETTO, T., PIHO, C., KUMAR, R., WESTFALL, J., YARKONI, T. and MARTIN, O. A. (2022). Bambi: A simple interface for fitting Bayesian linear models in Python. *Journal of Statistical Software* **103** 1–29.

[20] CARPENTER, B. (2021). What do we need from a probabilistic programming language to support Bayesian workflow? In *International Conference on Probabilistic Programming (PROBPROG)* 46.

[21] CARPENTER, B., GELMAN, A., HOFFMAN, M. D., LEE, D., GOODRICH, B., BETANCOURT, M., BRUBAKER, M., GUO, J., LI, P. and RIDDELL, A. (2017). Stan: A probabilistic programming language. *Journal of Statistical Software* **76**.

[22] CARPENTER, B., HOFFMAN, M. D., BRUBAKER, M., LEE, D., LI, P. and BETANCOURT, M. (2015). The Stan math library: Reverse-mode automatic differentiation in C++. *arXiv:1509.07164*.

[23] CEŠNOVAR, R. (2022). Parallel computation in the Stan probabilistic programming language, PhD thesis, Univerza v Ljubljani, Fakulteta za računalništvo in informatiko.
[24] CIGLARIČ, T., ČEŠNOVAR, R. and ŠTRUMBELJ, E. (2020). Automated OpenCL GPU kernel fusion for Stan Math. In *Proceedings of the International Workshop on OpenCL* 1–6.
[25] CRANMER, K., BREHMER, J. and LOUPPE, G. (2020). The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences* **117** 30055–30062.
[26] CSILLÉRY, K., FRANÇOIS, O. and BLUM, M. G. (2012). abc: an R package for approximate Bayesian computation (ABC). *Methods in Ecology and Evolution* **3** 475–479.
[27] CSÁRDI, G. (2019). cranlogs: Download logs from the 'RStudio' 'CRAN' mirror R package version 2.1.1.
[28] CUSUMANO-TOWNER, M. F., SAAD, F. A., LEW, A. K. and MANSINGHKA, V. K. (2019). Gen: A general-purpose probabilistic programming system with programmable inference. In *Proceedings of the 40th ACM Sigplan conference on Programming Language Design and Implementation* 221–236.
[29] DAVIS, T. A. (2006). *Direct Methods for Sparse Linear Systems*. SIAM.
[30] DE VALPINE, P., TUREK, D., PACIOREK, C. J., ANDERSON-BERGMAN, C., LANG, D. T. and BODIK, R. (2017). Programming with models: writing statistical algorithms for general model structures with NIMBLE. *Journal of Computational and Graphical Statistics* **26** 403–413.
[31] DILLON, J. V., LANGMORE, I., TRAN, D., BREVDO, E., VASUDEVAN, S., MOORE, D., PATTON, B., ALEMI, A., HOFFMAN, M. and SAUROUS, R. A. (2017). Tensorflow distributions. *arXiv:1711.10604*.
[32] DRUZDZEL, M. J. (1999). SMILE: Structural Modeling, Inference, and Learning Engine and GeNIe: A development environment for graphical decision-theoretic models. In *American Association for Artificial Intelligence Proceedings* 902–903.
[33] DUANE, S., KENNEDY, A. D., PENDLETON, B. J. and ROWETH, D. (1987). Hybrid Monte Carlo. *Physics Letters B* **195** 216–222.
[34] DUTTA, R., SCHOENGENS, M., PACCHIARDI, L., UMMADISINGU, A., WIDMER, N., KÜNZLI, P., ONNELA, J.-P. and MIRA, A. (2021). ABCpy: A high-performance computing perspective to approximate Bayesian computation. *Journal of Statistical Software* **100** 1–38.
[35] ET AL., M. A. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org.
[36] FOREMAN-MACKEY, D., FARR, W. M., SINHA, M., ARCHIBALD, A. M., HOGG, D. W., SANDERS, J. S., ZUNTZ, J., WILLIAMS, P. K., NELSON, A. R., DE VALBORRO, M. et al. (2019). emcee v3: A Python ensemble sampling toolkit for affine-invariant MCMC. *arXiv:1911.07688*.
[37] FOREMAN-MACKEY, D., HOGG, D. W., LANG, D. and GOODMAN, J. (2013). emcee: The MCMC hammer. *Publications of the Astronomical Society of the Pacific* **125** 306.
[38] FOURNIER, D. A., SKAUG, H. J., ANCHETA, J., IANELLI, J., MAGNUSSON, A., MAUNDER, M. N., NIELSEN, A. and SIBERT, J. (2012). AD Model Builder: Using automatic differentiation for statistical inference of highly parameterized complex nonlinear models. *Optimization Methods and Software* **27** 233–249.
[39] FRÜHWIRTH-SCHNATTER, S., FRÜHWIRTH, R., HELD, L. and RUE, H. (2009). Improved auxiliary mixture sampling for hierarchical models of non-Gaussian data. *Statistics and Computing* **19** 479–492.
[40] GABRY, J. and MAHR, T. (2022). bayesplot: Plotting for Bayesian models. R package version 1.10.0.
[41] GABRY, J., SIMPSON, D., VEHTARI, A., BETANCOURT, M. and GELMAN, A. (2019). Visualization in Bayesian workflow. *J. R. Stat. Soc. A* **182** 389-402.
[42] GABRY, J. and ČEŠNOVAR, R. (2022). A lightweight R interface to CmdStan R package version 0.5.3, https://github.com/stan-dev/cmdstanr/.
[43] GABRY, J. and VEEN, D. (2022). shinystan: Interactive visual and numerical diagnostics and posterior analysis for Bayesian models R package version 2.6.0, https://CRAN.R-project.org/package=shinystan.
[44] GAEDKE-MERZHÄUSER, L., VAN NIEKERK, J., SCHENK, O. and RUE, H. (2023). Parallelized integrated nested Laplace approximations for fast Bayesian inference. *Statistics and Computing* **33** 1–16.
[45] GE, H., XU, K. and GHAHRAMANI, Z. (2018). Turing: A language for flexible probabilistic inference. In *International Conference on Artificial Intelligence and Statistics* 1682–1690. PMLR.
[46] GELMAN, A., VEHTARI, A., SIMPSON, D., MARGOSSIAN, C. C., CARPENTER, B., YAO, Y., KENNEDY, L., GABRY, J., BÜRKNER, P.-C. and MODRÁK, M. (2020). Bayesian workflow. *arXiv:2011.01808*.
[47] GILKS, W. R. and WILD, P. (1992). Adaptive rejection sampling for Gibbs sampling. *Journal of the Royal Statistical Society, Series C* **41** 337–348.
[48] GOLDING, N. (2019). greta: Simple and scalable statistical modelling in R. *Journal of Open Source Software* **4** 1601.
[49] GOODMAN, J. and WEARE, J. (2010). Ensemble samplers with affine invariance. *Communications in Applied Mathematics and Computational Science* **5** 65–80.
[50] GOODMAN, N., MANSINGHKA, V., ROY, D. M., BONAWITZ, K. and TENENBAUM, J. B. (2012). Church: a language for generative models. *arXiv:1206.3255*.
[51] GORINOVA, M., MOORE, D. and HOFFMAN, M. (2020). Automatic reparameterisation of probabilistic programs. In *International Conference on Machine Learning* 3648–3657. PMLR.
[52] GORINOVA, M. I. (2022). Program analysis of probabilistic programs. *arXiv:2204.06868*.
[53] GORINOVA, M. I., GORDON, A. D. and SUTTON, C. (2019). Probabilistic programming with densities in SlicStan: Efficient, flexible, and deterministic. *Proceedings of the ACM on Programming Languages* **3** 1–30.
[54] GOUDIE, R. J., TURNER, R. M., DE ANGELIS, D. and THOMAS, A. (2020). MultiBUGS: A parallel implementation of the BUGS modelling framework for faster Bayesian inference. *Journal of Statistical Software* **95**.
[55] GRONAU, Q. F., SINGMANN, H. and WAGENMAKERS, E.-J. (2020). bridgesampling: An R package for estimating normalizing constants. *Journal of Statistical Software* **92** 1–29.
[56] GUTMANN, M. U. and CORANDER, J. (2016). Bayesian optimization for likelihood-free inference of simulator-based statistical models. *Journal of Machine Learning Research* **17** 1–47.
[57] HELD, L. and HOLMES, C. C. (2006). Bayesian auxiliary variable models for binary and multinomial regression. *Bayesian Analysis* **1** 145–168.
[58] HIGSON, E., HANDLEY, W., HOBSON, M. and LASENBY, A. (2019). Dynamic nested sampling: An improved algorithm for parameter estimation and evidence calculation. *Statistics and Computing* **29** 891–913.
[59] HOBERT, J. P. (2011). The data augmentation algorithm: Theory and methodology. In *Handbook of Markov Chain Monte Carlo* 279–320. CRC Press.

[60] HOFFMAN, M. and SOUNTSOV, P. (2022). Tuning-free generalized Hamiltonian Monte Carlo. *Proceedings of Machine Learning Research* **151** 7799–7813.

[61] HOFFMAN, M. D., GELMAN, A. et al. (2014). The No-U-Turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research* **15** 1593–1623.

[62] HOFFMAN, M. D., RADUL, A. and SOUNTSOV, P. (2021). An adaptive MCMC scheme for setting trajectory lengths in Hamiltonian Monte Carlo. *International Conference on Artificial Intelligence and Statistics*.

[63] HUNTER, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* **9** 90–95.

[64] JABOT, F., FAURE, T. and DUMOULIN, N. (2013). Easy ABC: Performing efficient approximate Bayesian computation sampling schemes using R. *Methods in Ecology and Evolution* **4** 684–687.

[65] KADIE, C. M., HOVEL, D. and HORVITZ, E. (2001). MSBNx: A component-centric toolkit for modeling and inference with Bayesian networks. *Microsoft Research, Richmond, WA, Technical Report MSR-TR-2001-67* **28**.

[66] KALLIOINEN, N., PAANANEN, T., BÜRKNER, P.-C. and VEHTARI, A. (2021). Detecting and diagnosing prior and likelihood sensitivity with power-scaling. *arXiv:2107.14054*.

[67] KOUSATHANAS, A., DUCHEN, P. and WEGMANN, D. (2018). A guide to general-purpose ABC software. In *Handbook of Approximate Bayesian Computation* 369–413. CRC Press.

[68] KRAINSKI, E., GÓMEZ-RUBIO, V., BAKKA, H., LENZI, A., CASTRO-CAMILO, D., SIMPSON, D., LINDGREN, F. and RUE, H. (2018). *Advanced Spatial Modeling With Stochastic Partial Differential Equations Using R and INLA*. CRC Press.

[69] KUCUKELBIR, A., TRAN, D., RANGANATH, R., GELMAN, A. and BLEI, D. M. (2017). Automatic differentiation variational inference. *Journal of Machine Learning Research*.

[70] KUMAR, R., CARROLL, C., HARTIKAINEN, A. and MARTÍN, O. A. (2019). ArviZ a unified library for exploratory analysis of Bayesian models in Python. *Journal of Open Source Software*.

[71] LINDGREN, F. and RUE, H. (2015). Bayesian spatial modelling with R-INLA. *Journal of Statistical Software* **63** 1–25.

[72] LINDGREN, F., RUE, H. and LINDSTRÖM, J. (2011). An explicit link between Gaussian fields and Gaussian Markov random fields: The stochastic partial differential equation approach. *Journal of the Royal Statistical Society, Series B* **73** 423–498.

[73] LINTUSAARI, J., VUOLLEKOSKI, H., KANGASRAASIO, A., SKYTÉN, K., JARVENPAA, M., MARTTINEN, P., GUTMANN, M. U., VEHTARI, A., CORANDER, J. and KASKI, S. (2018). Elfi: Engine for likelihood-free inference. *Journal of Machine Learning Research* **19** 1–7.

[74] LUNN, D., JACKSON, C., BEST, N., THOMAS, A. and SPIEGELHALTER, D. (2012). *The BUGS Book: A Practical Introduction to Bayesian Analysis*. CRC Press.

[75] LUNN, D., SPIEGELHALTER, D., THOMAS, A. and BEST, N. (2009). The BUGS project: Evolution, critique and future directions. *Statistics in medicine* **28** 3049–3067.

[76] LUNN, D. J., THOMAS, A., BEST, N. and SPIEGELHALTER, D. (2000). WinBUGS - A Bayesian modelling framework: Concepts, structure, and extensibility. *Statistics and Computing* **10** 325–337.

[77] MADSEN, A. L., LANG, M., KJÆRULFF, U. B. and JENSEN, F. (2003). The Hugin tool for learning Bayesian networks. In *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty* 594–605. Springer.

[78] MAKOWSKI, D., BEN-SHACHAR, M. S. and LÜDECKE, D. (2019). bayestestR: Describing effects and their uncertainty, existence and significance within the Bayesian framework. *Journal of Open Source Software* **4** 1541.

[79] MANSINGHKA, V., SELSAM, D. and PEROV, Y. (2014). Venture: A higher-order probabilistic programming platform with programmable inference. *arXiv:1404.0099*.

[80] MARTIN, A. D., QUINN, K. M. and PARK, J. H. (2011). MCMCpack: Markov chain Monte Carlo in R. *Journal of Statistical Software* **42** 22.

[81] MARTIN, G. M., FRAZIER, D. T. and ROBERT, C. P. (2022). Computing Bayes: From then 'til now'. *https://arxiv.org/abs/2208.00646*.

[82] MARTINS, T. G., SIMPSON, D., LINDGREN, F. and RUE, H. (2013). Bayesian computing with INLA: New features. *Computational Statistics & Data Analysis* **67** 68–83.

[83] MINKA, T., WINN, J. M., GUIVER, J. P., ZAYKOV, Y., FABIAN, D. and BRONSKILL, J. (2018). /Infer.NET 0.3. Microsoft Research Cambridge. http://dotnet.github.io/infer.

[84] MONNAHAN, C. C. and KRISTENSEN, K. (2018). No-U-turn sampling for fast Bayesian inference in ADMB and TMB: Introducing the adnuts and tmbstan R packages. *PloS One* **13** e0197954.

[85] MOREY, R. D. and ROUDER, J. N. (2022). BayesFactor: Computation of Bayes factors for common designs R package version 0.9.12-4.4, https://CRAN.R-project.org/package=BayesFactor.

[86] MURPHY, K. (2001). The Bayes Net toolbox for Matlab. *Computing science and statistics* **33** 1024–1034.

[87] MURRAY, L. M., LUNDÉN, D., KUDLICKA, J., BROMAN, D. and SCHÖN, T. B. (2018). Delayed sampling and automatic Rao-Blackwellization of probabilistic programs. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS)*.

[88] MURRAY, L. M. and SCHÖN, T. B. (2018). Automated learning with a probabilistic programming language: Birch. *Annual Reviews in Control* **46** 29–43.

[89] NEAL, R. M. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*.

[90] NEAL, R. M. (1993). *Probabilistic inference using Markov chain Monte Carlo methods*. Department of Computer Science, University of Toronto.

[91] NEAL, R. M. (2003). Slice sampling. *Annals of Statistics* **31** 705–767.

[92] NUNES, M. A. and PRANGLE, D. (2015). abctools: An R package for tuning approximate Bayesian computation analyses. *R Journal* **7** 189–205.

[93] PASZKE, A., GROSS, S., MASSA, F., LERER, A., BRADBURY, J., CHANAN, G., KILLEEN, T., LIN, Z., GIMELSHEIN, N., ANTIGA, L. et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems* **32**.

[94] PHAN, D., PRADHAN, N. and JANKOWIAK, M. (2019). Composable effects for flexible and accelerated probabilistic programming in NumPyro. *arXiv:1912.11554*.

[95] PIIRONEN, J., PAASINIEMI, M., CATALINA, A., WEBER, F. and VEHTARI, A. (2023). projpred: Projection predictive feature selection. R package version 2.4.0 https://mc-stan.org/projpred/.

[96] PLUMMER, M., BEST, N., COWLES, K. and VINES, K. (2006). CODA: Convergence diagnosis and output analysis for MCMC. *R News* **6** 7–11.

[97] PLUMMER, M. et al. (2003). JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling. In *Proceedings of the 3rd International Workshop on Distributed Statistical Computing* **124** 1-10.

[98] POLSON, N. G., SCOTT, J. G. and WINDLE, J. (2013). Bayesian inference for logistic models using Pólya–Gamma latent variables. *Journal of the American Statistical Association* **108** 1339–1349.

[99] PRICE, L. F., DROVANDI, C. C., LEE, A. and NOTT, D. J. (2018). Bayesian synthetic likelihood. *Journal of Computational and Graphical Statistics* **27** 1–11.

[100] RAINFORTH, T. W. G. (2017). Automating inference, learning, and design using probabilistic programming, PhD thesis, University of Oxford.

[101] RIDELL, A. (2022). Python interface to Stan Python package version 3.6.0, https://pypi.org/project/pystan/.

[102] RONQUIST, F., KUDLICKA, J., SENDEROV, V., BORGSTRÖM, J., LARTILLOT, N., LUNDÉN, D., MURRAY, L., SCHÖN, T. B. and BROMAN, D. (2021). Universal probabilistic programming offers a powerful approach to statistical phylogenetics. *Communications Biology* **4**.

[103] RUE, H., MARTINO, S. and CHOPIN, N. (2009). Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. *Journal of the Royal Statistical Society, Series B* **71** 319–392.

[104] RUE, H., RIEBLER, A., SØRBYE, S. H., ILLIAN, J. B., SIMPSON, D. P. and LINDGREN, F. K. (2017). Bayesian computing with INLA: A review. *Annual Review of Statistics and Its Application* **4** 395–421.

[105] SALVATIER, J., WIECKI, T. V. and FONNESBECK, C. (2016). Probabilistic programming in Python using PyMC3. *PeerJ Computer Science* **2** e55.

[106] SCHÄLTE, Y., KLINGER, E., ALAMOUDI, E. and HASENAUER, J. (2022). pyABC: Efficient and robust easy-to-use approximate Bayesian computation. *https://arxiv.org/abs/2203.13043*.

[107] SCUTARI, M. (2010). Learning Bayesian networks with the bnlearn R Package. *Journal of Statistical Software* **35**.

[108] SISSON, S. A., FAN, Y. and BEAUMONT, M. (2018). *Handbook of Approximate Bayesian Computation*. CRC Press.

[109] SPEAGLE, J. S. (2020). dynesty: a dynamic nested sampling package for estimating Bayesian posteriors and evidences. *Monthly Notices of the Royal Astronomical Society* **493** 3132–3158.

[110] SPIEGELHALTER, D., THOMAS, A., BEST, N. and GILKS, W. (1996). BUGS 0.5: Bayesian inference using Gibbs sampling manual (version ii). *MRC Biostatistics Unit, Institute of Public Health, Cambridge, UK* 1–59.

[111] SPIEGELHALTER, D., THOMAS, A., BEST, N. and LUNN, D. (2014). OpenBUGS user manual. *Version 3.2.3*.

[112] SPIEGELHALTER, D. J., THOMAS, A., BEST, N. and LUNN, D. (2003). WinBUGS version 1.4 user manual. *MRC Biostatistics Unit, Cambridge. URL http://www. mrc-bsu. cam. ac. uk/bugs*.

[113] STURTZ, S., LIGGES, U. and GELMAN, A. (2005). R2WinBUGS: A Package for Running WinBUGS from R. *Journal of Statistical Software* **12** 1–16.

[114] SYED, S., BOUCHARD-CÔTÉ, A., DELIGIANNIDIS, G. and DOUCET, A. (2021). Non-reversible parallel tempering: A scalable highly parallel MCMC scheme. *Journal of Royal Statistical Society, Series B* **84** 321–350.

[115] TALTS, S., BETANCOURT, M., SIMPSON, D., VEHTARI, A. and GELMAN, A. (2018). Validating Bayesian inference algorithms with simulation-based calibration. *arXiv:1804.06788*.

[116] TAREK, M., XU, K., TRAPP, M., GE, H. and GHAHRAMANI, Z. (2020). DynamicPPL: Stan-like speed for dynamic probabilistic models. *arXiv:2002.02702*.

[117] TAYLOR, S. J. and LETHAM, B. (2018). Forecasting at scale. *American Statistician* **72** 37–45.

[118] TAYLOR, S. J. and LETHAM, B. (2021). prophet: Automatic Forecasting Procedure R package version 1.0, https://CRAN.R-project.org/package=prophet.

[119] TAYLOR, S. J. and LETHAM, B. (2022). Prophet: Automatic Forecasting Procedure Python package version 1.1.1, https://pypi.org/project/prophet/.

[120] RSTUDIO TEAM (2021). RStudio: Integrated Development Environment for R RStudio, PBC, Boston, MA.

[121] R CORE TEAM (2022). R: A Language and Environment for Statistical Computing R Foundation for Statistical Computing, Vienna, Austria.

[122] STAN DEVELOPMENT TEAM (2022). RStan: the R interface to Stan R package version 2.21.7, https://mc-stan.org/.

[123] TEAM, S. D. (2022). A lightweight Python interface to CmdStan Python package version 1.0.8, https://pypi.org/project/cmdstanpy/.

[124] TEHRANI, N., ARORA, N. S., LI, Y. L., SHAH, K. D., NOURSI, D., TINGLEY, M., TORABI, N., LIPPERT, E., MEIJER, E. et al. (2020). Bean machine: A declarative probabilistic programming language for efficient programmable inference. In *International Conference on Probabilistic Graphical Models* 485–496. PMLR.

[125] TEJERO-CANTERO, A., BOELTS, J., DEISTLER, M., LUECKMANN, J.-M., DURKAN, C., GONÇALVES, P. J., GREENBERG, D. S. and MACKE, J. H. (2020). sbi: A toolkit for simulation-based inference. *Journal of Open Source Software* **5** 2505.

[126] THOMAS, N. (2020). R2OpenBUGS: Running OpenBUGS from R R package version 3.2-3.2.1, https://CRAN.R-project.org/package=R2OpenBUGS.

[127] THORNTON, K. R. (2009). Automating approximate Bayesian computation by local linear regression. *BMC genetics* **10** 1–5.

[128] TOLPIN, D., VAN DE MEENT, J.-W., YANG, H. and WOOD, F. (2016). Design and implementation of probabilistic programming language Anglican. In *Proceedings of the 28th Symposium on the Implementation and Application of Functional programming Languages* 1–12.

[129] TRAN, D., HOFFMAN, M. W., MOORE, D., SUTER, C., VASUDEVAN, S. and RADUL, A. (2018). Simple, distributed, and accelerated probabilistic programming. *Advances in Neural Information Processing Systems* **31**.

[130] VAN NIEKERK, J., BAKKA, H., RUE, H. and SCHENK, O. (2021). New frontiers in Bayesian modeling using the INLA package in R. *Journal of Statistical Software* **100** 1–28.

[131] VAN NIEKERK, J., KRAINSKI, E., RUSTAND, D. and RUE, H. (2023). A new avenue for Bayesian inference with INLA. *Computational Statistics & Data Analysis* 107692.

[132] VAN NIEKERK, J. and RUE, H. (2021). Correcting the Laplace Method with Variational Bayes. *arXiv:2111.12945*.

[133] VEHTARI, A., GABRY, J., MAGNUSSON, M., YAO, Y., BÜRKNER, P.-C., PAANANEN, T. and GELMAN, A. (2022). loo: Efficient leave-one-out cross-validation and WAIC for Bayesian models R package version 2.5.1.

[134] VEHTARI, A., GELMAN, A. and GABRY, J. (2017). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing* **27** 1413–1432.

[135] WICKHAM, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer.

[136] WIGREN, A., RISULEO, R. S., MURRAY, L. M. and LINDSTEN, F. (2019). Parameter elimination in particle Gibbs sampling. *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*.

[137] WOOD, S. N. (2017). *Generalized Additive Models: An Introduction with R, Second Edition*. CRC Press.