MCMC for Bayesian Uncertainty Quantification from Time-Series Data

Philip Maybank[1]([✉]) , Patrick Peltzer[2], Uwe Naumann[2], and Ingo Bojak[3]

[1] Numerical Algorithms Group Ltd (NAG), Oxford, UK
philip.maybank@nag.co.uk
[2] Software and Tools for Computational Engineering (STCE), RWTH Aachen
University, Aachen, Germany
info@stce.rwth-aachen.de
[3] School of Psychology and Clinical Language Sciences, University of Reading,
Reading, UK
i.bojak@reading.ac.uk

Abstract. In computational neuroscience, Neural Population Models (NPMs) are mechanistic models that describe brain physiology in a range of different states. Within computational neuroscience there is growing interest in the inverse problem of inferring NPM parameters from recordings such as the EEG (Electroencephalogram). Uncertainty quantification is essential in this application area in order to infer the mechanistic effect of interventions such as anaesthesia.

This paper presents `C++` software for Bayesian uncertainty quantification in the parameters of NPMs from approximately stationary data using Markov Chain Monte Carlo (MCMC). Modern MCMC methods require first order (and in some cases higher order) derivatives of the posterior density. The software presented offers two distinct methods of evaluating derivatives: finite differences and exact derivatives obtained through Algorithmic Differentiation (AD). For AD, two different implementations are used: the open source Stan Math Library and the commercially licenced `dco/c++` tool distributed by NAG (Numerical Algorithms Group). The use of derivative information in MCMC sampling is demonstrated through a simple example, the noise-driven harmonic oscillator. And different methods for computing derivatives are compared. The software is written in a modular object-oriented way such that it can be extended to derivative based MCMC for other scientific domains.

Keywords: Uncertainty quantification · Algorithmic Differentiation · Computational neuroscience

1 Introduction

Bayesian uncertainty quantification is useful for calibrating physical models to observed data. As well as inferring the parameters that produce the best fit between a model and observed data, Bayesian methods can also identify the

range of parameters that are consistent with observations and allow for prior beliefs to be incorporated into inferences. This means that not only can predictions be made but also their uncertainty can be quantified. In demography, Bayesian analysis is used to forecast the global population [7]. In defence systems, Bayesian analysis is used to track objects from radar signals [1]. And in computational neuroscience Bayesian analysis is used to compare different models of brain connectivity and to estimate physiological parameters in mechanistic models [13]. Many more examples can be found in the references of [2,6,15].

We focus here on problems which require the use of Markov Chain Monte Carlo (MCMC), a widely applicable methodology for generating samples approximately drawn from the posterior distribution of model parameters given observed data. MCMC is useful for problems where a parametric closed form solution for the posterior distribution cannot be found. MCMC became popular in the statistical community with the re-discovery of Gibbs sampling [26], and the development of the BUGS software [15]. More recently it has been found that methods which use derivatives of the posterior distribution with respect to model parameters, such as the Metropolis Adjusted Langevin Algorithm (MALA) and Hamiltonian Monte Carlo (HMC) tend to generate samples more efficiently than methods which do not require derivatives [12]. HMC is used in the popular Stan software [4]. From the perspective of a C++ programmer, the limitations of Stan are as follows: it may take a significant investment of effort to get started. Either the C++ code has to be translated into the Stan modelling language. Or, alternatively, C++ code can be called from Stan, but it may be challenging to (efficiently) obtain the derivatives that Stan needs in order to sample efficiently.

The software that we present includes (i) our own implementation of a derivative-based MCMC sampler called simplified manifold MALA (smMALA). This sampler can be easily be used in conjunction with C++ codes for Bayesian data analysis, (ii) Stan's MCMC sampler with derivatives computed using `dco/c++`, an industrial standard tool for efficient derivative computation.

An alternative approach to the one presented in this paper would be simply to use Stan as a stand-alone tool without the smMALA sampler and without `dco/c++`. Determining the most efficient MCMC sampler for a given problem is still an active area of research, but at least within computational neuroscience, it has been found the smMALA performs better than HMC methods on certain problems [25]. Determining the most appropriate method for computing derivatives will depend on both the user and the problem at hand. In many applications Algorithmic Differentiation (AD) is needed for the reasons given in Sect. 2.3. The Stan Math Library includes an open-source AD tool. Commercial AD tools such as `dco/c++` offer a richer set of features than open-source tools, and these features may be needed in order to optimize derivative computations. For example, the computations done using the Eigen linear algebra library [10] can be differentiated either using the Stan Math Library or using `dco/c++` but there are cases where `dco/c++` computes derivatives more efficiently than the Stan Math Library [22]. The aim of the software we present is to offer a range of options that both make it easy to get started and to tune performance.

2 Methods for Spectral Time-Series Analysis

2.1 Whittle Likelihood

The software presented in this paper is targeted at differential equation models with a stable equilibrium point and stochastic input. We refer to such models as stable SDEs. The methods that are implemented assume that the system is operating in a regime where we can approximate the dynamics through linearization around the stable fixed point. If the time-series data is stationary this is a reasonable assumption. Note that the underlying model may be capable of operating in nonlinear regimes such as limit cycles or chaos in addition to approximately linear dynamics. However, parameter estimation using data in nonlinear regimes quickly becomes intractable - see Chapter 2 of [16]. The stability and linearity assumptions are commonly made in the computational neuroscience literature, see for example [18].

In order to simplify the presentation we illustrate the software using a linear state-space model, which is of the form,

$$d\mathbf{X}(t) = A\,\mathbf{X}(t)dt + \mathbf{P}(t), \tag{1}$$

where the term $A\,\mathbf{X}(t)$ represents the deterministic evolution of the system and $\mathbf{P}(t)$ represents the noisy input. The example we analyze in this paper is the noise-driven harmonic oscillator, which is a linear state-space model with

$$A = \begin{pmatrix} 0 & 1 \\ -\omega_0^2 & -2\zeta\omega_0 \end{pmatrix}, \quad P(t) = \begin{pmatrix} 0 \\ dW(t) \end{pmatrix}, \tag{2}$$

and where $dW(t)$ represents a white noise process with variance σ_{in}^2. The observations are modelled as $Y_k = X_0(k \cdot \Delta t) + \epsilon_k$ with $\epsilon_k \sim N(0, \sigma_{obs}^2)$.

Our aim is to infer model parameters $(\omega_0, \zeta, \sigma_{in})$ from time-series data. This could be done in the time-domain using a Kalman filter, but it is often more computationally efficient to do inference in the frequency domain [3,17]. In the case where we only have a single output (indexed by i) and a single input (indexed by j), we can compute a likelihood of the parameters θ in the frequency domain through the following steps.

1. Compute the (left and right) eigendecomposition of A, such that,

$$A\mathcal{R} = \Lambda\mathcal{R}, \quad \mathcal{L}A = \mathcal{L}\Lambda, \quad \mathcal{L}\mathcal{R} = \text{diag}(c) \tag{3}$$

   where $\text{diag}(c)$ is a diagonal matrix, such that c_i is the dot product of the ith left eigenvector with the ith right eigenvector.
2. Compute ijth element of the transfer matrix for frequencies $\omega_1, \ldots, \omega_K$,

$$\mathcal{T}(\omega) = \mathcal{R}\,\text{diag}\left[\frac{1}{c_k(i\omega - \lambda_k)}\right]\mathcal{L}. \tag{4}$$

3. Evaluate the spectral density for component i of $\mathbf{X}(t)$, $f_{X_i}(\omega)$, and the spectral density for the observed time-series, $f_Y(\omega)$,

$$f_{X_i}(\omega) = |\mathcal{T}_{ij}(\omega)|^2 \, f_{P_j}(\omega), \tag{5}$$

$$f_Y(\omega) = f_{X_i}(\omega) + \sigma_{obs}^2 \Delta t, \tag{6}$$

where $f_{P_j}(\omega)$ is the spectral density for component j of $\mathbf{P}(t)$.
4. Evaluate the Whittle likelihood,

$$p(y_0, \ldots, y_{n-1}|\theta) = p(S_0, \ldots, S_{n-1}|\theta) \approx \prod_{k=1}^{n/2-1} \frac{1}{f_Y(\omega_k)} \exp\left[-\frac{S_k}{f_Y(\omega_k)}\right], \tag{7}$$

where $\{S_k\}$ is the Discrete Fourier Transform of $\{y_k\}$. Note that θ represents a parameter set (e.g. specific values of $\omega_0, \zeta, \sigma_{in}$) that determines the spectral density.

The matrix A that parameterizes a linear state-space model is typically non-symmetric, which means that eigenvectors and eigenvalues will be complex-valued. We use Eigen-AD [22], a fork of the C++ linear algebra library Eigen [10]. Eigen is templated which facilitates the application of AD by overloading tools and Eigen-AD provides further optimizations for such tools. The operations above require an AD tool that supports differentiation of complex variables. AD of complex variables is considered in [24]. It is currently available in the `feature/0123-complex-var` branch of the Stan Math Library and in `dco/c++` from release 3.4.3.

2.2   Markov Chain Monte Carlo

In the context of Bayesian uncertainty quantification, we are interested in generating samples from the following probability distribution,

$$p(\theta|y_0, \ldots, y_{n-1}) \propto p(y|\theta)p(\theta), \tag{8}$$

where $p(y|\theta)$ is the likelihood of the parameter set θ given observed data $y_0, \ldots, y_{n-1}$, and $p(\theta)$ is the prior distribution of the parameters. In many application where the likelihood $p(y|\theta)$ is based on some physical model we cannot derive a closed form expression for the posterior density $p(\theta|y)$. Markov Chain Monte Carlo (MCMC) has emerged over the last 30 years as one of the most generally applicable and widely used framework for generating samples from the posterior distribution [6,9]. The software used in this paper makes use of two MCMC algorithms: the No U-Turn Sampler (NUTS) [12] and the simplified manifold Metropolis Adjusted Langevin Algorithm (smMALA) [8]. NUTS is called via the Stan environment [4]. It is a variant of Hamiltonian Monte Carlo (HMC), which uses the gradient (first derivative) of the posterior density,

whereas smMALA uses the gradient and Hessian (first and second derivatives) of the posterior density, smMALA is described in Algorithm 1.

The error in estimates obtained from MCMC is approximately $C/\sqrt{N}$, where N is the number of MCMC iterations and C is some problem-dependent constant. In general it is not possible to demonstrate that MCMC has converged, but there are several diagnostics that can indicate non-convergence, see Section 11.4 of [6] for more detail. Briefly, there are two phases of MCMC sampling: burn-in and the stationary phase. Burn-in is finished when we are in the region of the parameter space containing the true parameters. In this paper we restrict ourselves to synthetic data examples. In this case it is straight-forward to assess whether the sampler is burnt in by checking whether the true parameters used to simulate the data are contained in the credible intervals obtained from the generated samples. In real data applications, it is good practice to test MCMC sampling on a synthetic data problem that is analogous to the real data problem. During the stationary phase we assess convergence rate using the effective sample size, *N Eff*. If we were able to generate independent samples from the posterior then the constant C is $\mathcal{O}(1)$. MCMC methods generate correlated samples, in which case C may be $\gg 1$. A small *N Eff* (relative to N) indicates that this is the case.

If we are sampling from a multivariate target distribution, *N Eff* for the ith component is equal to,

$$\frac{S}{1 + 2\sum_k \hat{\rho}_i(k)}, \tag{9}$$

where S is the number of samples obtained during the stationary period, and $\hat{\rho}_i(k)$, is an estimate of the autocorrelation at lag k for the ith component of the samples. This expression can be derived from writing down the variance of the average of a correlated sequence (Chapter 11 of [6]). The key point to note is that if the autocorrelation decays slowly *N Eff* will be relatively small.

Algorithm 1: smMALA

Input: Data, y; Initial value for $\theta = (\theta_1, \ldots, \theta_N)$;
         Likelihood $l(y|\theta)$; Prior distribution $p(\theta)$.
Parameters : Step size, h; Number of iterations, I
for $i = 2, \ldots, I$ **do**
     Evaluate gradient and Hessian of the unnormalized log posterior,
     $g_\theta = \nabla\left[\log[\ l(y|\theta)\ p(\theta)\ ]\right]$ and $\mathbf{H}_\theta$;
     Set $C = h^2\mathbf{H}_\theta^{-1}$, and $m = \theta + \frac{1}{2}C\,g_\theta$;
     Propose $\theta^* \sim q(\theta^*|\theta) = N(m, C)$;
     Evaluate acceptance ratio, $\alpha(\theta, \theta^*) = \min\left[1, \dfrac{l(y|\theta^*)p(\theta^*)q(\theta|\theta^*)}{l(y|\theta)p(\theta)q(\theta^*|\theta)}\right]$;
     Draw $u \sim \text{Uniform}(0, 1)$;
     if $u < \alpha(\theta, \theta^*)$ **then** Set $\theta = \theta^*$
end

2.3   Derivative Computation

A finite difference approximation to the first and second derivatives of the function $F(\theta) : \mathbb{R}^N \to \mathbb{R}^M$ can be computed as,

$$\frac{\partial F}{\partial \theta_i} = \frac{F(\theta + he_i) - F(\theta)}{h}, \qquad \frac{\partial^2 F}{\partial \theta_i \partial \theta_j} = \frac{\frac{\partial F}{\partial \theta_i}(\theta + he_j) - \frac{\partial F}{\partial \theta_i}(\theta)}{h},$$

where e_i is the ith Cartesian basis vector, and h is a user-defined step-size. For first-order derivatives the default value we use is $\sqrt{\epsilon}|\theta_i|$, where ϵ is machine epsilon for double types, i.e., if θ_i is $\mathcal{O}(1)$, $h \approx 10^{-8}$. For second derivatives the default value is $\epsilon^{1/3}|\theta_i|$, so that $h \approx 5 \cdot 10^{-6}$. More details on the optimal step size in finite difference approximations can be found in [20]. First derivatives computed using finite differences require $\mathcal{O}(N)$ function evaluations, where N is the number of input variables. Second derivatives require $\mathcal{O}(N^2)$ function evaluations.

Derivatives that are accurate to machine precision can be computed using AD tools. AD tools generally use one of two modes: tangent (forward) or adjoint (reverse). For a function with N inputs and M outputs, tangent mode requires $\mathcal{O}(N)$ function evaluations and adjoint mode requires $\mathcal{O}(M)$ function evaluations. In statistical applications our output is often a scalar probability density, so adjoint AD will scale better with N than either tangent mode or finite differences in terms of total computation time. Adjoint AD is implemented as follows using the Stan Math Library, [5].

```
// Define and init matrix with Stan's adjoint type
Matrix<stan::math::var,Dynamic,1> theta = input_values;

// Compute primal and gradient
stan::math::var lp = computation(theta);
lp.grad();

// Extract gradient
Matrix<double,Dynamic,1> grad_vec(theta.size());
for(int j=0; j<theta.size(); j++) grad_vec(j) = theta(j).adj();
```

In the code above the function is evaluated using the `stan::math:var` scalar type rather than `double`. During function evaluation, derivative information is recorded. When the `grad()` function is called this derivative information is interpreted and derivative information is then accessed by calling `adj()` on the input variables. Similarly to the Stan Math Library, exact derivatives can be computed using the NAG `dco/c++` tool developed in collaborations with RWTH Aachen University's STCE group, [14].

```
// Define and init matrix with dco's adjoint type
typedef dco::ga1s<double> DCO_MODE;
```

```
typedef DCO_MODE::type Scalar;
Matrix<Scalar,Dynamic,1> theta = input_values;

// Enable dynamic activity analysis
for(int j=0; j<theta.size(); j++)
  DCO_MODE::global_tape->register_variable(theta(j));

// Compute primal and gradient
Scalar lp = computation(theta);
dco::derivative(lp) = 1.0;
DCO_MODE::global_tape->interpret_adjoint();

// Extract gradient
Matrix<double,Dynamic,1> grad_vec(theta.size());
for(int j=0; j<theta.size(); j++)
  grad_vec(j) = dco::derivative(theta(j));
```

dco/c++ provides its adjoint type using the `dco::ga1s<T>::type` typedef, where T s the corresponding primal type (e.g. `double`). Higher order adjoints can be achieved by recursively nesting the adjoint type. Using this type, the program is first executed in the augmented forward run, where derivative information is stored in the `global_tape` data structure. The `register_variable` function initialises recording of the tape and facilitates dynamic varied analysis [11]. Derivative information recorded during the augmented primal run is then interpreted using the `interpret_adjoint()` function and `dco::derivative` is used to access the propagated adjoints.

3   Results for Noise Driven Harmonic Oscillator

We now present the results of MCMC sampling using smMALA to estimate parameters of the noise driven harmonic oscillator. Then we demonstrate that, for this particular model, MCMC sampling can be accelerated by using NUTS. We also compare run times and sampling efficiency when AD is used to evaluate derivatives. Here we estimate parameters using synthetic data (i.e. data generated from the model). This is a useful check that the MCMC sampler is working correctly: we should be able to recover the parameters that were used to simulate the data. We generate a pair of datasets representing different conditions (which we label c_1 and c_2). In Table 1 we show that the 95% credible intervals for each parameter include the actual parameter values for all 5 parameters. These results came from one MCMC run of 10,000 iterations. Figure 1 shows 95% confidence intervals for the spectral density obtained using the Welch method alongside 95% credible intervals for the spectral density estimated from 10,000 MCMC iterations. This is another useful check: the spectral density predictions generated by sampled parameter sets are consistent with non-parametric estimates of the spectral density.

Table 1. Estimated quantiles of posterior distribution for noise driven harmonic oscillator. Synthetic data was generated by simulating from the model: duration = 20.0, time-step = 0.01. Two datasets (c_1 and c_2) were generated with different parameter values for ω_0 and σ_{in} in each dataset (values are given in 'actual' column). The quantiles are estimated from a sequence of 10, 000 MCMC samples from the joint (posterior) distribution of all the parameters given all of the data. For example, 2.5% of the MCMC samples had $\omega_0(c_1)$ values less than 77.3.

	Actual	Quantile		
		0.025	0.50	0.975
$\omega_0(c_1)$	80	77.3	80.3	81.9
$\omega_0(c_2)$	40	36.8	38.8	41.3
$\sigma_{in}(c_1)$	100	92	101	111
$\sigma_{in}(c_2)$	10	9.81	10.8	12.4
ζ	0.2	0.164	0.193	0.223

Table 2 uses the same model and the same datasets as above to compare smMALA with NUTS, and finite differences with AD. The AD implementation used in smMALA was dco's tangent over adjoint mode (i.e. `dco::gt1s` combined with `dco::ga1s`). In NUTS we used the dco's tangent mode for computing the derivatives of the spectral density and Stan's adjoint mode (`stan::math::var`) for the rest of the computation. Given the most recent developments in the `feature/0123-complex-var` branch of the Stan Math Library it would likely be possible to use Stan's adjoint mode for the whole computation. However this was not the case when we started writing the code. In general users may find that they need the more advanced functionality of `dco/c++` for part of their computation in order to obtain good performance.

The MCMC samplers were each run for 1,000 iterations. The results were analyzed using Stan's `stansummary` command-line tool. To account for correlation between MCMC samples we use the *N Eff* diagnostic defined in Sect. 2.2 to measure sampling efficiency, min *N Eff* is the minimum over the 5 model parameters. Table 2 shows that, for the noise driven harmonic oscillator, we can accelerate MCMC sampling by a factor of around 3–5 by using NUTS rather than smMALA. We also see that the *N Eff/s* is higher for finite differences than for AD, because of the small number of input variables. However, even for this simple model the NUTS min *N Eff* is higher for AD than for finite differences suggesting that the extra accuracy in the derivatives results in more efficient sampling per MCMC iteration.

In the context of Bayesian uncertainty quantification what we are interested in is whether the MCMC samples are an accurate representation of the true posterior distribution. A necessary condition for posterior accuracy is that the true parameters are (on average) contained in the credible intervals derived from the MCMC samples. This is the case for all the different variants of MCMC that we tested. The main differences we found between the different variants was
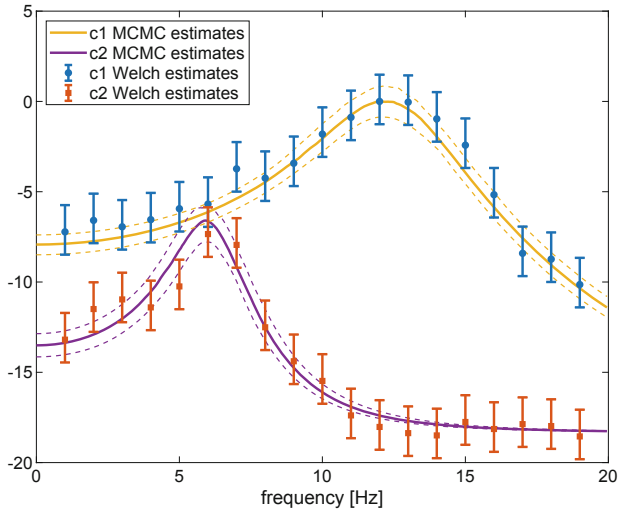
Fig. 1. Spectral density estimates for noise driven harmonic oscillator for the datasets described in Table 1.

the sampling efficiency. As discussed in Sect. 2.2, a sampling efficiency that is 3–5 time greater means a reduction in the value of C in the Monte Carlo error $(C/\sqrt{N})$ by that factor. Another way of interpreting this is that we could reduce the number of MCMC iterations by a factor of 3–5 and expect to obtain the same level of accuracy in the estimated posterior distribution.

Table 2. Noise-driven harmonic oscillator benchmarking results.

MCMC sampler	Derivative implementation	CPU time (s)	min $N\ Eff$	min $N\ Eff$/s
smMALA	Finite differences	3.3	152	46
smMALA	`dco/c++`	5.2	152	29
NUTS	Finite differences	3.2	485	153
NUTS	Stan and `dco/c++`	3.9	506	130

4   Discussion

The Whittle likelihood function is written to be polymorphic over classes derived from the a base class called `Stable_sde`. The function signature includes a reference to the base class.

```
template<typename SCALAR_T>
SCALAR_T log_whittle_like(Stable_sde<SCALAR_T> & m, ...);
```

Classes that are derived from the `Stable_sde` base class must define the following pure virtual function.

```
virtual Matrix<SCALAR_T, Dynamic, Dynamic> sde_jacobian(...) = 0;
```

The spectral density evaluation (Eq. (5)) is implemented in the base class, reducing the effort required to do spectral data analysis for other stable SDEs. The smMALA sampler is written to be polymorphic over classes derived from a base class called `Computation`. Classes that are derived from the `Computation` base class must define the following pure virtual function.

```
virtual SCALAR_T eval(Matrix<SCALAR_T,Dynamic,1>& theta,...) = 0;
```

This function should evaluate the posterior density in the user's model. The gradient and Hessian of the posterior density are implemented in the `Computation` class. This reduces the effort required to use NUTS or smMALA for other computational models. Classes derived from `Stable_sde` or from `Computation` can be instantiated with several different scalar types including `double`, `stan::math::var`, and a number of `dco/c++` types. This enables automatic evaluation of first and second derivatives. The gradient and Hessian functions need a template specialization to be defined in the base class for each different scalar type that is instantiated.

We conclude with some comments regarding the choice of MCMC sampler and the method for evaluating derivatives. Our software only implements derivative-based MCMC as this tends to be more computationally efficient than other MCMC methods [12,23]. Derivative-based MCMC samplers can be further subdivided into methods that use higher-order derivatives, such as smMALA (sometimes referred to as Riemannian) and methods that only require first-order derivatives (such as NUTS). Riemannian methods tend to be more robust to complexity in the posterior distribution, but first-order methods tend to be more computationally efficient for problems where the posterior geometry is relatively simple. We recommend using smMALA in the first instance, then NUTS as a method that may provide acceleration in MCMC sampling, in terms of effective samples per second, *N Eff/s*. Regarding the derivative method, finite differences often results in adequate performance for problems with moderate input dimension (e.g. 10–20), at least with smMALA. But for higher-dimensional problems (e.g. Partial Differential Equations or Monte Carlo simulation) we recommend accelerating derivative computation using adjoint AD [19,21]. The software presented enables users to implement and benchmark all these alternatives so that the most appropriate methods for a given problem can be chosen.

References

1. Arulampalam, M.S., Maskell, S., Gordon, N., Clapp, T.: A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. IEEE Trans. Signal Process. **50**(2), 174–188 (2002)
2. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer, New York (2006)
3. Bojak, I., Liley, D.: Modeling the effects of anesthesia on the electroencephalogram. Phys. Rev. E **71**(4), 041902 (2005)
4. Carpenter, B., et al.: Stan: A probabilistic programming language. J. Stat. Softw. **76**(1), 1–32 (2017)
5. Carpenter, B., Hoffman, M.D., Brubaker, M., Lee, D., Li, P., Betancourt, M.: The Stan math library: reverse-mode automatic differentiation in C++. arXiv preprint arXiv:1509.07164 (2015)
6. Gelman, A., Carlin, J.B., Stern, H.S., Dunson, D.B., Vehtari, A., Rubin, D.B.: Bayesian Data Analysis. CRC Press, Boca Raton (2013)
7. Gerland, P., et al.: World population stabilization unlikely this century. Science **346**(6206), 234–237 (2014)
8. Girolami, M., Calderhead, B.: Riemann manifold Langevin and Hamiltonian Monte Carlo methods. J. Roy. Stat. Soc.: Ser. B (Stat. Methodol.) **73**(2), 123–214 (2011)
9. Green, P.J., Łatuszyński, K., Pereyra, M., Robert, C.P.: Bayesian computation: a summary of the current state, and samples backwards and forwards. Stat. Comput. **25**(4), 835–862 (2015)
10. Guennebaud, G., Jacob, B., et al.: Eigen v3 (2010). http://eigen.tuxfamily.org
11. Hascoët, L., Naumann, U., Pascual, V.: "To be recorded" analysis in reverse-mode automatic differentiation. Future Gen. Comput. Syst. **21**(8), 1401–1417 (2005)
12. Hoffman, M.D., Gelman, A.: The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. J. Mach. Learn. Res. **15**(1), 1593–1623 (2014)
13. Kiebel, S.J., Garrido, M.I., Moran, R.J., Friston, K.J.: Dynamic causal modelling for EEG and MEG. Cogn. Neurodyn. **2**(2), 121 (2008)
14. Leppkes, K., Lotz, J., Naumann, U.: Derivative code by overloading in C++ (dco/c++): introduction and summary of features. Technical report AIB-2016-08, RWTH Aachen University, September 2016. http://aib.informatik.rwth-aachen.de/2016/2016-08.pdf
15. Lunn, D., Jackson, C., Best, N., Thomas, A., Spiegelhalter, D.: The BUGS Book: A Practical Introduction to Bayesian Analysis. CRC Press, Boca Raton (2012)
16. Maybank, P.: Bayesian inference for stable differential equation models with applications in computational neuroscience. Ph.D. thesis, University of Reading (2019)
17. Maybank, P., Bojak, I., Everitt, R.G.: Fast approximate Bayesian inference for stable differential equation models. arXiv preprint arXiv:1706.00689 (2017)
18. Moran, R.J., Stephan, K.E., Seidenbecher, T., Pape, H.C., Dolan, R.J., Friston, K.J.: Dynamic causal models of steady-state responses. NeuroImage **44**(3), 796–811 (2009)
19. NAG: NAG algorithmic differentiation software. https://www.nag.com/content/algorithmic-differentiation-software. Accessed 27 Jan 2020
20. NAG: OptCorner: the price of derivatives - using finite differences. https://www.nag.co.uk/content/optcorner-price-derivatives-using-finite-differences. Accessed 27 Jan 2020
21. Naumann, U., du Toit, J.: Adjoint algorithmic differentiation tool support for typical numerical patterns in computational finance. J. Comput. Finan. **21**(4), 23–57 (2018)

22. Peltzer, P., Lotz, J., Naumann, U.: Eigen-AD: Algorithmic differentiation of the Eigen library. arXiv preprint arXiv:1911.12604 (2019)
23. Penny, W., Sengupta, B.: Annealed importance sampling for neural mass models. PLoS Comput. Biol. **12**(3), e1004797 (2016)
24. Pusch, G., Bischof, C., Carle, A.: On Automatic Differentiation of Codes with Complex Arithmetic with Respect to Real Variables, September 1995
25. Sengupta, B., Friston, K.J., Penny, W.D.: Gradient-based MCMC samplers for dynamic causal modelling. NeuroImage **125**, 1107–1118 (2016)
26. Smith, A.F., Roberts, G.O.: Bayesian computation via the Gibbs sampler and related Markov chain Monte Carlo methods. J. Roy. Stat. Soc.: Ser. B (Methodol.) **55**(1), 3–23 (1993)