Hamiltonian Monte Carlo Sampling for Bayesian Hierarchical Regression in Prognostics

Lachlan Astfalck[1] and Melinda Hodkiewicz[2]

[1,2] *System Health Laboratory, The University of Western Australia, Perth, WA, 6009, Australia*
lachlan.astfalck@uwa.edu.au
melinda.hodkiewicz@uwa.edu.au

ABSTRACT

Advances in computational speed have enabled the development of many Bayesian probabilistic models due to Markov-Chain-Monte-Carlo (MCMC) posterior sampling methods. These models includes Bayesian hierarchical regression methods, which use group level information to inform individual asset predictions. Hierarchical models are increasingly used for prognostics as they recognise that the parameter estimates for an individual asset may be rationally influenced by data from other similar assets. Larger and high dimensional datasets require more efficient sampling methods for calculations, than traditional MCMC techniques. Hamiltonian Monte Carlo (HMC) has been used across many fields to address high dimensional, sparse, or non-conjugate data. Due to the need to find the posterior derivative and the flexibility in the tuning parameters, HMC is often difficult to hand code. We investigate a probabilistic programming language, Stan, which allows the implementation of HMC sampling, with particular focus on Bayesian hierarchical models in prognostics. The benefits and limitations for HMC using Stan are explored and compared to the widely used Gibbs Sampler and Metropolis-Hastings (MH) algorithm. Results are demonstrated using three case studies on lithium-ion batteries. Stan reduced coding complexity and sampled from posterior distributions more efficiently than parameters sampled with the Metropolis-Hastings algorithm. HMC sampling became less efficient with increasing data-size and hierarchical complexity, due to high curvature in the posterior distribution. Stan was shown to be a robust language which allows for easier inference to be made in the Bayesian paradigm.

1. INTRODUCTION

Bayesian inference has been widely utilised in prognostics, with methods including, particle filters (Orchard, Kacprzynski, Goebel, Saha, & Vachtsevanos, 2008), regression models (Coble & Hines, 2011), and hierarchical models (Zaidan, Harrison, Mills, & Fleming, 2015). A good overview of prognostic methods in a bayesian framework is presented in Saha, Goebel, Poll, and Christophersen (2009). The Bayesian paradigm performs inference about the parameters of interest by examining the posterior distribution, $p(\theta|y)$. This is formulated by taking information from both the prior distribution, $p(\theta)$, and the likelihood of the observed data $p(y|\theta)$. A comprehensive instruction on Bayesian statistics can be found in Gelman, Carlin, Stern, and Rubin (2014).

Until the mainstream arrival of numerical methods such as Markov-chain-Monte-Carlo (MCMC) in the early 90s, the state-of-the-art for Bayesian inference saw analytically tractable combinations of priors and likelihoods used to calculate the posterior distribution. This limited the range of uses of Bayesian statistics to situations where the posterior could be analytically derived. This is often not possible for models that describe realistically complex data, rather than simplified or idealistic data. With numerical method development, potential applications in the Bayesian paradigm, as well as the flexibility of Bayesian models has increased (Gilks, Richardson, & Spiegelhalter, 1995). However, many MCMC techniques can be complicated and confusing to the new practitioner, which hence makes Bayesian inference unapproachable.

Details of MCMC sampling schemes vary, however they are all similar in the sense that they generate samples from the posterior. This results in a finite number of autocorrelated samples which may be used for inference (Brooks, Gelman, Jones, & Meng, 2011). Due to the autocorrelation between the samples, many iterations may be required to draw enough samples to be representative of the posterior. MCMC algorithms are judged on how long they take to generate a single sample, but how correlated the samples are from iteration to iteration (Brooks & Roberts, 1998).

Due to the complexities of MCMC sampling, hand coding samplers runs the risk of hidden coding errors, model mis-

specification and run-time inefficiencies. A series of software platforms known as the BUGS family (WinBUGS, OpenBUGS, BUGS, and JAGS), allows practitioners to use Bayesian inference, without having to concern themselves of the intricacies of MCMC sampling (Lunn, Thomas, Best, & Spiegelhalter, 2000). The BUGS family primarily generates samples via Gibbs sampling and the Metropolis-Hastings algorithm, both members of the MCMC family described further in sections 3.1 and 3.2. Both of these methods rely on random-walk behaviour to explore the posterior. This can lead to sampling inefficiencies in highly correlated or high dimensional data. As such, with increasing model size and complexity, the implementation of many models using BUGS leads to a prohibitively long run-time for sampling.

A new family of MCMC algorithms called Hamiltonian Monte Carlo (HMC) (Neal et al., 2011) has promise to increase the sampling efficiency over the algorithms used by BUGS. But it has been slow to be adopted. This may be attributed to two reasons. The first is the implementation difficulty of the sampler. HMC must be tuned in three different locations, which for complex models requires an experienced user to program. The second is that HMC requires precise gradients. However, analytical formulas are rare and numerical techniques are imprecise, especially in high dimensions. These hurdles have been overcome with the No-U-Tuns-Sampler (NUTS) and automatic differentiation. These advances have been packaged into a number of open source, generic and flexible probabilistic programming languages such as Stan (Carpenter et al., 2016), pyMC3 (Salvatier, Wiecki, & Fonnesbeck, 2016) and Edward (Tran et al., 2016).

Newer probabilistic programming languages aim to supersede both the need for hand coding of MCMC samplers, and the existing BUGS platform, by offering more flexible and faster samplers. Two popular frameworks at present are Stan and pyMC3. Stan uses a custom modelling language, and can be called from a variety of other modelling languages, including R and Python. PyMC3 builds a representation of the model in Python[1].

We explore the principles that underlie both traditional MCMC and HMC sampling, and implement three case studies in the Stan language, all of which use Bayesian hierarchical modelling. Two of these case studies have been previously published in prognostics literature. Bayesian hierarchical modelling has been widely utilised in many scientific fields, and more recently in prognostics, as seen in Zaidan et al. (2015) and Xu, Li, and Chen (2016). Hierarchical modelling utilises information available at multiple levels of an observation, and is particularly powerful in the Bayesian paradigm (its frequentist cousin is known as empirical hierarchical modelling).

We explore how HMC performance scales with the size and complexity of the model, and its suitability for hierarchical models in prognostics. Our goal is to explore the benefits of probabilistic programming with Stan, and to provide guidance for practitioners looking for methods that lead to faster and more robust bayesian inference.

The remainder of this paper is organised as follows; section 2 presents a short overview of Bayesian modelling, section 3 discusses MCMC sampling techniques, section 4 reports on probabilistic programming techniques, section 5 explains the data and models that we looks at, section 6 shows the statistical models and results of the model implementations, and sections 7 and 8 present a discussion and concluding remarks, respectively. The supporting documentation for this paper can be found at `http://uwa.engineering/papers-and-presentations/`.

2. BAYESIAN MODELLING

Bayesian statistics views probability as a degree of belief, and not as a relative frequency over time, as frequentist statistics does. Bayesian statistics allows for a rich class of models to be used for a wide range of applications. It combines information in the prior knowledge $p(\theta)$ and the model likelihood $p(y|\theta)$ to inform a belief about the posterior distribution $p(\theta|y)$. Bayes' theorem states that

$$p(\theta|y) \propto p(y|\theta) \times p(\theta).$$

Note that in Bayesian statistics, our interest is in the uncertainty of the values of the parameters θ, and not in the uncertainty in the data y. Bayesian statistics derives the most credible parameter values θ, for a chosen descriptive model. Bayesian methods allow for uncertainty to be propagated through all stages of modelling, presenting a comprehensive description of possible parameter values.

2.1. Bayesian hierarchical modelling

A strength of Bayesian modelling is the ease with which hierarchical models can be expressed and fit. These models make intuitive use of the multiple levels of information that are often available for an observation. For instance, consider a longitudinal measurement of an asset $y_{t,j}$ for a population of J assets at time t. Each asset is dependent on a set of operational conditions θ_j. It is often a reasonable assumption that the operational conditions within a site share some characteristics and therefore come from a common distribution of site averages μ and site variance Σ. This is a hierarchical model. This example can be described by the three stages below.

[1]The reader is referred to http://probabilistic-programming.org/wiki/Home for a wider overview of probabilistic programming languages.

1. $y_{t,j} \sim p(y_{t,j}|\theta_j)$
   The measurement $y_{t,j}$ for the jth asset at time t is modelled as directly dependent on its operating conditions. The dependence of $y_{t,j}$ on the parameters higher up in the hierarchy, μ and Σ, is absorbed in its dependency on θ_j.
2. $\theta_j \sim p(\theta_j|\mu, \Sigma)$
   Operating conditions of the jth asset can be modelled as being drawn from a common distribution of site averages μ and Σ.
3. $\mu \sim p(\mu), \Sigma \sim p(\Sigma)$
   Site averages and variances are given a distribution from which values may be drawn.

Taking this even further, another level can be added to the hierarchy: each site may be modelled as coming from a common distribution of sites. This chain of dependencies exemplifies a hierarchical model.

A consequence of hierarchical modelling is shrinkage. Shrinkage is the tendency of parameter estimation towards the overall central mean. It is a advantage of hierarchical modelling as it can both mitigate false alarms and erroneous parameter classification until significant data is presented, and can lend information to new data points with limited data by sharing data across the distribution. For example take the model described above. Should a new asset $J + 1$ come on-line at site i and we wish to estimate its failure time $y_{J+1,i}$. Without seeing any data on it's operating conditions we may infer a range of possibilities from the average operating conditions of the site ϕ_i.

To find the distribution of any quantity of interest we marginalise the joint distribution by integrating over all other variables. For example, should we be interested in the average operating conditions, M, across all sites, we may find its posterior distribution as

$$p(M|Y) = \int\int\int p(\theta_{1:J}, \phi_{1:I}, M, \Sigma|Y) \quad d\Sigma \; d\phi \; d\theta$$

With increasing computational power, and increasing volumes of sensor and failure data being monitored, modelling data according to a hierarchy is becoming more popular in prognostics research. Zaidan et al. (2015), Xu et al. (2016), and Cripps and Pecht (2017) all use a hierarchical framework to model degradation data.

3. MARKOV CHAIN MONTE CARLO SAMPLING

Markov chain Monte Carlo (MCMC) development can be attributed to Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller (1953), and was used amongst the physics community. Despite its inception in 1953, it was not until the 1990s that MCMC methods were widely used by the Bayesian community when computational advances made it more accessible. MCMC algorithms generate a series of dependant and autocorrelated samples from the posterior distribution which is used for inference. MCMC methods are well suited to Bayesian hierarchical models as the conditional distributions at the different levels of the hierarchy are generally well behaved, however often analytically intractable.

The two most generic and widely used realised methods of MCMC sampling is the Metropolis-Hastings (MH) algorithm (Hastings, 1970) and the Gibbs sampler (Geman & Geman, 1984). The majority of MCMC development builds upon these two sampling methods, for example reversible jump MCMC (Green, 1995) and adaptive MCMC (Andrieu & Thoms, 2008). Hamiltonian Monte Carlo (HMC) simulation (Neal et al., 2011) is a recent sampling development, and has been successfully implemented to address high dimensional, sparse, or non-conjugate data. Each MCMC method uses a unique jumping distribution to sample the next sample from the most recent sample. Important to all MCMC methods is that the generation of each subsequent sample is Markovian, which means that each sample only depends on the sample immediately before it.

Due to the autocorrelation in the samples the practitioner has to be confident that the MCMC sampler has effectively explored the posterior distribution, regardless of how many MCMC samples have been generated. This is known as mixing. As both the MH algorithm and the Gibbs sampler rely on random-walk behaviour, their algorithms struggle to propose samples in regions of the posterior distant from their current location. Due to this, multiple transitions are needed to move between regions. This results in higher autocorrelation, and slow mixing. Conversely, HMC relies on Hamiltonian dynamics to propose a sample. For certain models, this allows well constructed HMC algorithms to propose values almost anywhere in the posterior from any given location. Figure 1 shows the samples from a MH algorithm and HMC sampling. As a result HMC can explore the posterior much more efficiently than the MH algorithm (Neal et al., 2011).

We briefly explain the MH algorithm and the Gibbs sampler below, followed by a more in depth explanation of HMC.

3.1. The Metropolis-Hastings algorithm

The Metropolis algorithm was developed in (Metropolis et al., 1953). Hastings (1970) later generalised the Metropolis algorithm to allow for non-symmetrical jumping distributions. It has since been referred to as the Metropolis-Hastings algorithm. The MH algorithm utilises a random walk procedure to explore the sample space. The jumping distribution $J(\theta^*|\theta^{(s-1)})$ proposes a new sample, given the location of the current sample. The proposed sample is accepted according to an accept/reject rule. The rate of acceptance of a new

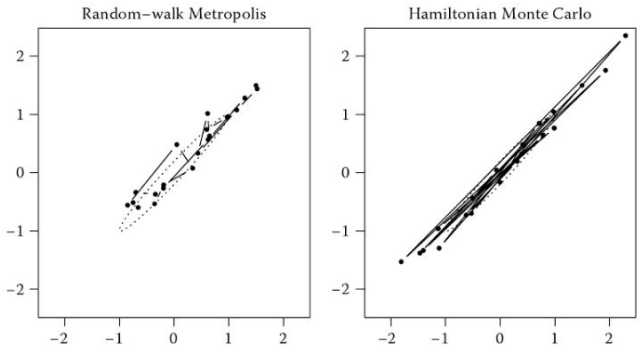

Figure 1. MH samples and HMC sample for a 2-dimensional Gaussian distribution with marginal standard deviations of one and correlation 0.98. The ellipses are drawn one standard deviation away from the mean. Adapted from Neal et al. (2011).

proposal is known as the acceptance rate. The algorithm for these steps is shown in algorithm 1.

Appropriate selection of jumping distribution $J_s(\theta^*|\theta^{(s-1)})$ may in itself be difficult, and the performance of the MCMC sampler relies on good selection (Brooks et al., 2011). A commonly implemented jumping distribution is the multivariate normal distribution, with the mean centered on the current sample location.

Algorithm 1: The Metropolis-Hastings Algorithm

Sample a starting point $\theta^0 \sim p_0(\theta)$.
for *s = 1,2,...,S* **do**
Sample $\theta^* \sim J_s(\theta^*|\theta^{(s-1)})$
Calculate the ratio of densities r as:

$$r = \frac{p(\theta^*|y)/J_s(\theta^*|\theta^{(s-1)})}{p(\theta^{(s-1)}|y)/J_s(\theta^{(s-1)}|\theta^*)}$$

Set

$$\theta^{(s)} = \begin{cases} \theta^* & \text{with probability min(r,1)} \\ \theta^{(s-1)} & \text{otherwise} \end{cases}$$

3.2. The Gibbs sampler

The Gibbs sample was first described in (Geman & Geman, 1984). A Gibbs sampler proposal is generated from the distribution of either a single parameter, or a vector of parameters, conditioned on all other components in the model. Conditional distributions of this form are called full conditional distributions. Full conditional distributions must be able to be sampled from. Commonly this is achieved by selecting conjugate prior distributions, where the family of the prior distribution leads to a posterior distribution of the same family. Whilst this reduces the families of distributions that may be selected to represent the model, it ensures that the sampler has an acceptance rate of 100%.

Suppose we define a vector of all model parameters θ which consists of d components, $\theta = (\theta_1, ..., \theta_d)$. The Gibbs sampler samples a new value for each component given the most recent locations of all other components. It requires d steps to sample a new value of θ. The jumping distribution here may be thought of as a d-step jump of the full conditional distributions. The algorithm for the Gibbs sampler is seen in algorithm 2.

The Gibbs sampler is one of the most commonly used sampling technique used in Bayesian modelling. This is due to the simplicity of sampling, as well as the high acceptance rate. In Bayesian statistics the choice of a prior distribution should accurately reflect ones prior belief of the system. Assigning a prior distribution for computational ease and convenience that does not reflect ones prior beliefs is not ideal.

Algorithm 2: The Gibbs sampler

Sample a starting point $\theta^0 \sim p_0(\theta)$.
for *s = 1,2,...,S* **do**
Generate $\theta^{(s)}$ from $\theta^{(s-1)}$ with d sub-iterations:
$\theta_1^{(s)} \sim p(\theta_1|\theta_2^{(s-1)}, \theta_3^{(s-1)}, ..., \theta_d^{(s-1)})$
$\theta_2^{(s)} \sim p(\theta_2|\theta_1^{(s)}, \theta_3^{(s-1)}, ..., \theta_d^{(s-1)})$
$\vdots$
$\theta_d^{(s)} \sim p(\theta_d|\theta_1^{(s)}, \theta_2^{(s)}, ..., \theta_{d-1}^{(s)})$

3.3. Hamiltonian Monte Carlo sampling

An inherent difficulty with the Gibbs sampler and the Metropolis-Hastings algorithm is their random walk behaviour. Random walk behaviour can take a long time to converge for high-dimensional, sparse, or highly correlated target distributions. Further, the Gibbs sampler is only usable with a limited set of priors. HMC does not have either of these limitations.

HMC borrows the concept of Hamiltonian movement from physics. Hamiltonian movement ensures that a dynamic system maintains constant energy. For example should a ball be rolling around the inside of a frictionless bowl, it's total energy will always remain constant. The potential energy of the ball at any given point is related to the height or position of the ball, and the kinetic energy of the ball is related to the ball's momentum. In HMC we consider the position of the sampler to be the parameters θ in the target space, with an augmented momentum variable ϕ_j. It follows that the posterior distribution $p(\theta|y)$ is augmented with the distribution $p(\phi)$. We specify the momenta $p(\phi)$ to be independent of $p(\theta|y)$, hence we may define the joint posterior distribution as $p(\theta, \phi|y) = p(\phi)p(\theta|y)$. Note that we are only interested in the simulations of θ and samples of ϕ may be discarded.

ϕ is introduced as an auxiliary variable to allow for efficient exploration of the sample space.

Standard practice sees ϕ specified as a multivariate normal distribution, with the same dimensions as θ, mean 0 and covariance set to a specified mass matrix M. To reduce complexity, we commonly use a diagonal mass matrix, M. If so, the components of ϕ are independent, with $\phi_j \sim N(0, M_{jj})$ for each dimension $j = 1, ..., d$. It can be useful for M to roughly scale with the inverse covariance matrix of the posterior distribution, $(\text{var}(\theta|y))^{-1}$, but the algorithm works in any case; better scaling of M will merely make HMC more efficient.

In addition to the posterior density constant, HMC requires the gradient of the log-posterior density. If θ has d dimensions, this gradient is $\frac{d \log p(\theta|y)}{d\theta} = \left(\frac{d \log p(\theta|y)}{d\theta_1}, ..., \frac{d \log p(\theta|y)}{d\theta_d}\right)$. Gradients are often not analytically tractable, so paths are approximated with a numerical method known as the leapfrog integrator. The leapfrog integrator depends on the step-size ϵ and the number of steps L. Two main issues complicate the tuning of these parameters. Firstly, if ϵ is too big approximation errors may cause divergence from the real distribution resulting in low acceptance rates, however if it's too small it increases computational cost. Secondly, if the optimum trajectory ϵL is too short the simulator is inefficient at exploring the distribution and if it is too long the trajectory retraces its steps.

HMC is tuned in three places; (1) the probability distribution for the momentum variables ϕ, (2) the scaling factor ϵ of the leapfrog steps, and (3) the number of leapfrog steps L per iteration. It is clear that implementation of HMC sampling is significantly more involved than more traditional MCMC techniques. This may be overcome with use of an appropriate probabilistic programming language, discussed further in Section 4. The full algorithm for the HMC sampler is shown in algorithm 3.

HMC sampling makes the jump from the current sampling location to the next by simulating the ball at its current location θ with random momenta ϕ, and for a finite time interval proposing the position at the end of its path with probability $\min(r, 1)$. r is the ratio of densities as defined in algorithm 3. It may help to think of HMC sampling as MH sampling, with a more advanced jumping distribution.

As mentioned, HMC sampling cannot sample from models with discrete parameters. If possible, the discrete parameters can be marginalised out analytically, however for some cases this is not trivial and for others it is not possible at all. In prognostics this is not as great a concern as for many other statistical models as parameters generally exist continuously, especially where longitudinal data is available. Whilst not common, examples of the use of discrete parameters in prognostics are; the representation of latent states (Wei, Huang, & Chen, 2009) and classification (Tobon-Mejia, Medjaher, & Zerhouni, 2012).

Algorithm 3: Hamiltonian Monte Carlo sampling

Sample a starting point $\theta^0 \sim p_0(\theta)$.
for *s = 1,2,...,S* **do**
  Update $\phi \sim N(0, M)$
  for *l=1,2,...,L* **do**
    Use the gradient (the vector derivative) of the log-posterior density of θ to make a half-step of ϕ:

$$\phi \leftarrow \phi + \frac{1}{2}\epsilon \frac{d \log p(\theta|y)}{d\theta}$$

    Use the momentum vector ϕ to update the position vector θ:

$$\theta \leftarrow \theta + \epsilon M^{-1} \phi$$

    Again use the gradient of θ to half-update ϕ:

$$\phi \leftarrow \phi + \frac{1}{2}\epsilon \frac{d \log p(\theta|y)}{d\theta}$$

  Calculate the ratio of densities r as:

$$r = \frac{p(\theta^*|y)p(\phi^*)}{p(\theta^{(s-1)}|y)p(\phi^{(s-1)})}$$

  Set

$$\theta^{(s)} = \begin{cases} \theta^* & \text{with probability min(r,1)} \\ \theta^{(s-1)} & \text{otherwise} \end{cases}$$

3.4. Combinations of sampling methods

HMC may be implemented in various combinations with the Gibbs sampler or the MH algorithm, to sample from complicated distributions. There are three main reasons why this may be done. They are;

1. It may make sense to partition independent variables into blocks to simplify the computation, or to speed convergence.
2. Should discrete variables be present, HMC will not be able to update them, due to its Hamiltonian dynamics. In this case, the space may be partitioned into discrete and continuous parameters. All continuous parameters may then be updated using HMC, and the remaining discrete parameters may be updated using either a Gibbs sampler or MH.
3. Should a model contain a mix of conjugate and non-conjugate distributions, then a Gibbs sampler may be implemented where a applicable to maximise the acceptance ratio, and HMC may be used to sample from the remaining distribution

```
data {
  vector[2] x;
}
parameters {
  real mu;
}
model {
  mu ~ normal(0, 1);
  x ~ normal(mu, 1);
}
```

Figure 2. Example of simple model written in Stan

4. PROBABILISTIC PROGRAMMING LANGUAGES

Probabilistic programming languages (PPLs) are a class of programming languages designed to describe probabilistic models, and to perform inference on them. When used in a Bayesian context, what is typically described is the posterior distribution of interest with inclusion of the observed data, the parameters and their priors, and the model likelihood. What is desired is inference about parameters of the posterior, for instance their posterior expectations and standard deviations. An example of a probabilistic program, written in the language Stan, and demonstrating these components is shown in Figure 2.

This describes the simple model of $x_1, x_2 \sim N(\mu, 1)$, where x_1 and x_2 are observations, and the mean μ is unknown with a prior of $N(0, 1)$. PPLs allow this model representation to be used directly for inference on the parameter space, typically by generating samples from the posterior via an MCMC algorithm such as HMC or Gibbs sampling. These algorithms are included as part of the language and require no further programming on the behalf of the practitioner. In this respect, the inference is 'automatic' once the model is correctly specified, precluding the possibility of programming errors in the inference.

Historically, the most successful and well-known PPL for Bayesian analysis has been the BUGS (Bayesian inference Using Gibbs Sampling) project introduced in 1989 and its subsequent family of tools, including WinBUGS, more recently OpenBUGS, and JAGS (Just Another Gibbs Sampler), all of which share the same underlying probabilistic language (Lunn et al., 2000; Plummer, 2003). As their names suggest, these tools share the property that they construct a sampler for a model by updating each parameter one at a time via Gibbs sampling. They may use a Metropolis-Hastings step, but often exploit the conjugate structure of models in order to use a full Gibbs step when possible. Thus, while these samplers have been highly successful for many models, they have trouble for highly correlated parameter spaces, as described in Section 3.

Stan is a more recent probabilistic programming language that addresses this shortcoming by dispensing with Gibbs sampling entirely, instead providing a highly optimised and self-tuning implementation of a Hamiltonian Monte Carlo sampler (Carpenter et al., 2016). The language used by Stan is similar to that of the BUGS family. The variant of HMC used by Stan is the No-U-Turn Sampler, which adaptively tunes parameters in order to create an efficient sampler. In addition to posterior sampling via HMC, Stan has the capacity to perform other types of inference on the posterior distribution, including variational inference (Blei, Kucukelbir, & McAuliffe, 2016), a form of approximate inference, and maximum a-posteriori inference, which finds the mode of the posterior. One major shortcoming of Stan compared to the BUGS family of tools is its inability to sample from discrete parameter spaces, which arises due to its reliance on HMC as its only sampling method. This limits its utility in implementing many appealing classes of models, such as change-point detection models. None-the-less, many models can be implemented in continuous parameter space, and Stan has become popular since its release. Its website contains a partial list of papers that have used it at `http://mc-stan.org/citations/`.

5. DATA

The aim of this paper is explore the implementation of HMC sampling for Bayesian hierarchical regression models used in prognostics. We present three case studies, two of which have been previously published and use a Gibbs sampler with MH step in the implementation. All case studies focus on the modelling of lithium-ion battery degradation. Lithium-ion battery degradation is selected for three reasons; (1) it is popular within prognostics research, (2) many data-sets are publicly available, and (3) it naturally lends itself to being modelled with Bayesian hierarchical regression.

The first case study models the capacity fade inside of a lithium ion battery as a simple high order polynomial random effects model. We use this simple linear model to illustrate the implementation differences between Stan, and hand coding a Gibbs sampler. The second case study models the capacity fade inside of a lithium ion battery as a double-exponential random effects model as presented in Cripps and Pecht (2017). The last case study models the voltage drops within each discharge cycle as a single knot spline as presented in Xu et al. (2016). These models are herein referred to as Model 1, Model 2, and Model 3, respectively.

For each model the data, and the statistical model is described in Section 6. Models 2 and 3 were originally implemented with a Gibbs sampler, with a MH step where necessary. The full conditional distributions of the original models is presented in the Appendix, as well as the full conditional distributions required to implement Model 1 with a Gibbs sampler. The same priors have been selected for HMC sampling as

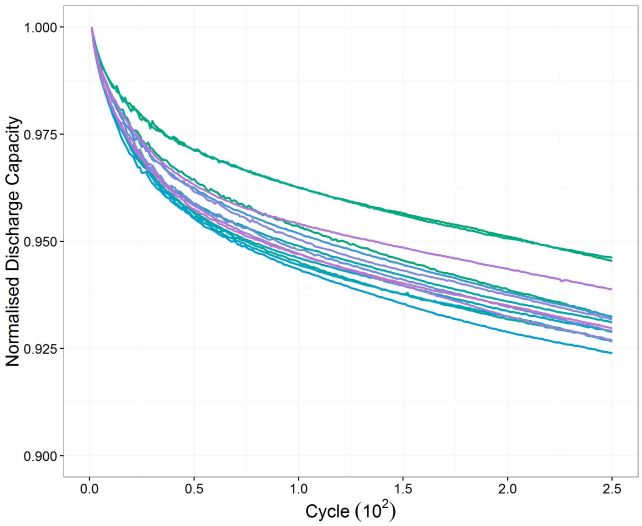

Figure 3. Li-ion battery normalised discharge capacity fade data for healthy batteries, and the suspected anomalous batteries, as presented in Cripps and Pecht (2017).

to not affect the posterior distributions between implementations. Further discussion on the appropriateness of prior selection is available in Section 7.1.

Model 1 and Model 2 use data from The Centre of Advanced Life-Cycle Engineering (CALCE). Model 3 uses data from the NASA Ames Data Repository (Saha & Goebel, 2007).

Each model is sampled with 100000 iterations, with the first 50000 counted as the warm-up period and not used for subsequent analysis. To compare the sampling efficiency of each model, effective sample size is used. The effective sample size takes into account the autocorrelation of the samples and calculates what would be an equivalently sized sample of independent values. The closer in value the effective sample size is to the number of iterations the better. For comparison of implementation and sampling efficiency, the posterior distributions of Models 1 and 2 are found with both HMC sampling and Gibbs/MH sampling. Model 3 is implemented in Stan and is used as a proof that HMC can effectively sample from complex, high dimensional distributions.

6. CASE STUDIES

6.1. Model 1 - Linear random effects model

Battery cells were tested to 250 charge-discharge cycles from the manufacturer, with the aim of detecting anomalous behaviour between manufacturing batches (Cripps & Pecht, 2017). Data from a single manufacturing batch is shown in Figure 3. The first case study simplifies the original analysis to provide contrast between HMC sampling with Stan and the more traditional Gibbs sampling.

We model the discharge capacity measurements, y_j, of the jth battery as a 4th order polynomial with Gaussian error, where y_j is a vector of all measurements for battery j. This is expressed as

$$y_j \sim N(X\beta_j, \sigma^2)$$
$$\beta_j \sim N(\mu, \Sigma)$$

for $j = 1, 2, ..., J$ batteries. β_j is a vector of coefficients for each battery, and X is a design matrix that holds exponents of the cycles $t \in \{1 : T\}$ such that

$$X = \begin{bmatrix} 1 & 1 & 1^2 & 1^3 & 1^4 \\ 1 & 2 & 2^2 & 2^3 & 2^4 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & T & T^2 & T^3 & T^4 \end{bmatrix}.$$

For this data, X is the same for all batteries. The battery specific coefficients β_j are assumed to be drawn from a normal distribution of overall means μ and group level variance Σ.

β_j, μ, Σ and σ^2 are all considered unknown. The joint posterior distribution is found as

$$p(\beta_{1:J}, \mu, \Sigma, \sigma^2|Y) \propto p(Y|\beta_{1:J}, \sigma^2) \times p(\beta_{1:J}|\mu, \Sigma) \times p(\sigma^2, \mu, \Sigma).$$

HMC and Gibbs sampling of this model is implemented in Stan and from first principles in R, respectively. The code for the Stan implementation is shown in Figures 4. Stan implementation requires only the description of the statistical model, conversely, hand coding a sampler in R requires in depth knowledge of the MCMC sampling method. Differences in code complexity is seen between the two sets of code, hence chances for hidden coding errors is a larger issue in the first principles R implementation. The full code for both implementations is available in the supporting documentation.

The HMC sampler had an effective sample size approximately ranging from 10000 to 26000 (the ideal effective sample size is 50000). Similarly, the Gibbs sampler had an effective sample size approximately ranging from 23000 to 50000. Both sampling methods have similar run-times, and the Gibbs sampler had higher effective sample sizes which is to be expected in more simplistic linear models with low covariance between parameters.

As both implementations are sampling the same model, it is expected that the posterior distributions for each parameter be the same. This can be seen by the two sampled posterior distributions for the noise parameter σ, shown in Figure 5.

```
data {
 int<lower=1> J;
 int<lower=1> N;
 int<lower=1> K;
 int<lower=1, upper=J> jj[N];
 matrix[N,K] x_melt;
 vector[N] y_melt;
}

parameters {
 corr_matrix[K] Omega;
 vector<lower=0>[K] tau;
 vector[K] mu;
 matrix[J,K] beta;
 real<lower=0> sigma;
}

transformed parameters {
 matrix[K,K] Sigma_beta;
 Sigma_beta = quad_form_diag(Omega, tau);
}

model {
 for (j in 1:J) {
  beta[j] ~ multi_normal(mu, Sigma_beta);
 }
 for (n in 1:N) {
  y_melt[n] ~ normal(
  dot_product(beta[jj[n]], x_long[n]), sigma);
 }
}
```

Figure 4. Stan code for Model 1 implementation with HMC sampling

The minor difference in the centre of the distribution can be attributed the stochastic variance between the two samplers.

6.2. Model 2 - Non-linear random effects model

We extend Model 1 to replicate the non-linear random effects models shown in Cripps and Pecht (2017). The same data is modelled, as seen in Figure 3.

Let $y_{j,t}$ be the discharge capacity measurements, where $y_{j,t}$ is the individual measurement of the jth battery at time t. The non-linear random effects model, with Gaussian error, is modelled as

$$y_{j,t} \sim N(f(t;\alpha_j), \sigma^2),$$
$$f(t;\alpha_j) = \alpha_{1,j} e^{\alpha_{2,j} t} + \alpha_{3,j} e^{\alpha_{4,j} t}$$
$$\alpha_j = \begin{pmatrix} \alpha_{1,j} \\ \alpha_{2,j} \\ \alpha_{3,j} \\ \alpha_{4,j} \end{pmatrix} \sim N\left(\begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_4 \end{pmatrix}, \begin{pmatrix} \tau_1^2 & 0 & 0 & 0 \\ 0 & \tau_2^2 & 0 & 0 \\ 0 & 0 & \tau_3^2 & 0 \\ 0 & 0 & 0 & \tau_4^2 \end{pmatrix} \right)$$

for $j = 1, 2, ..., J$ batteries and $t = 1, 2, ..., T$ charge-discharge cycles. Note here that the parameters contained within α_j are individual to the jth battery discharge profile, and are assumed to be themselves drawn from a distribution of the overall means μ, and across group variance τ.

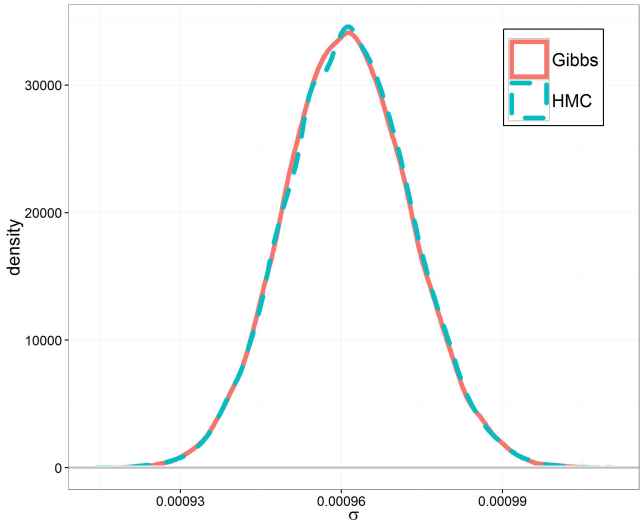

Figure 5. Sampled densities of the noise parameter σ.

α_j, μ, τ and σ^2 are all considered unknown. The joint posterior distribution is found as

$$p(\alpha_{1:J}, \mu, \tau^2, \sigma^2|Y) \propto p(Y|\alpha_{1:J}, \sigma^2) \times p(\alpha_{1:J}|\mu, \tau) \times p(\sigma^2, \mu, \tau^2).$$

The code for this model has been omitted for the sake of brevity. Both the Stan and R implementations are available in the supporting documentation.

The effective sample size of the Stan implementation ranged between 34382 and 50000 samples. Conversely the effective sample size of the R implementation ranged between 214 and 26548. Whilst, the parameters that were sampled with a Gibbs sampler had high effective sample sizes, the α parameters that were sampled with the MH algorithm had much lower effective sizes, ranging from 214 to 520 samples. The lowest effective sample size amongst the α parameters sampled by the HMC sampler was 48646. Further, the MH step had a high rejection rate of 62%, aiding autocorrelation in the samples. Again, run-times for both implementations is similar and the coding of the Stan model is much more simplistic. As expected, the parameter samples from both implementations converged on the same distributions.

6.3. Model 3 - Volatage discharge

Xu et al. (2016) model lithium-ion battery degradation of the publicly available data-set from the NASA Data Repository (Saha & Goebel, 2007). An important difference to note between this model and the previous two is that each voltage discharge trajectory is modelled as an individual group inside of a hierarchical model, rather than each battery. Each tra-

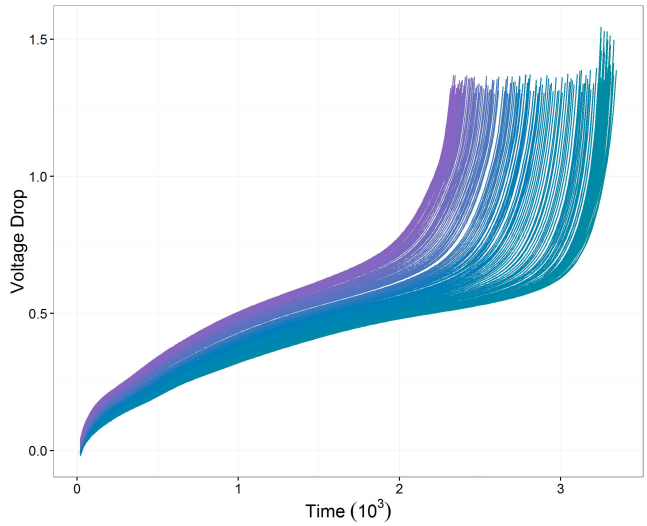

Figure 6. Li-ion battery voltage discharge profile for 168 charge-discharge cycles

jectory is roughly normalised as to be independent of charge-discharge cycle. The discharge profiles for each of the 168 cycles is shown in Figure 6.

Let $U_{i,t}$ be the output voltage measurements at time t for the ith discharging cycle, where $i \in \{1, ..., 168\}$. $U_{i,t}$ is specified as

$$U_{i,t} \sim N(f(t;\theta_i), \sigma^2).$$

$f(t;\theta_i)$ is a non-linear function with parameters θ_i dependent on each cycle i and is modelled by the single knot spline

$$f(t;\theta) = \begin{cases} a_0 + a_1 t + a_2 t^2 + a_3 t^3, & t \leq \tau \\ b_0 + b_1(t-\tau) + b_2(t-\tau)^2 + b_3(t-\tau)^3, & t > \tau \end{cases},$$

where τ is the location of the knot in the spline model and is itself considered to be a continuous parameter of interest. To ensure the smoothness of $f(t;\theta)$, its imposed that it is second order continuous for each point t. Therefore, the parameters must satisfy

$$\begin{cases} b_0 = a_0 + a_1\tau + a_2\tau^2 + a_3\tau^3 \\ b_1 = a_1 + 2a_2\tau + 3a_3\tau^2 \\ b_2 = a_2 + 3a_3\tau \end{cases}.$$

The parameters $\beta = [a_0, a_1, a_2, a_3, b_3]$ and $\theta = [\beta, \tau]$ are defined as freely changing parameters. It is clear that θ depends on the cycle index. To account for this, θ_i can be represented by the linear Gaussian model

$$\theta_i \sim N(C_i\gamma, \Sigma)$$

where C_i is a design matrix dependant on the cycle index i, γ is the parameter vector independent of the cycle and Σ is the variance-covariance matrix. Derivations of C_i and γ is omitted. The reader may refer to Xu et al. (2016) for further information.

In each cycle, m_i output voltages $U_{i,t_{i,j}}$ were measured at times $t_{i,j}$, $j = 1, 2, ..., m_i$. Let U_i denote the observations of the ith discharging profile. The model can be expressed as a pseudo linear model $U_i \sim N(X_i(\tau_i)\beta_i, \sigma^2 I_{m_i})$, $i = 1, ..., 168$, where

$$U_i = \begin{bmatrix} U_i(t_{i,1}) \\ \vdots \\ U_i(t_{i,k}) \\ U_i(t_{i,k+1}) \\ \vdots \\ U_i(t_{i,m_i}) \end{bmatrix},$$

$$X_i(\tau_i) = \left[\begin{array}{ccccc} 1 & t_{i,1} & t_{i,1}^2 & t_{i,1}^3 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & t_{i,k} & t_{i,k}^2 & t_{i,k}^3 & 0 \\ \hline 1 & t_{i,k+1} & t_{i,k+1}^2 & t_{i,k+1}^3 - (t_{i,k+1}-\tau_i)^3 & (t_{i,k+1}-\tau_i)^3 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & t_{i,m_i} & t_{i,m_i}^2 & t_{i,m_i}^3 - (t_{i,m_i}-\tau_i)^3 & (t_{i,m_i}-\tau_i)^3 \end{array}\right],$$

and $t_{i,k} \leq \tau_i \leq t_{i,k+1}$. As in the previous models, all of the unknowns can be expressed as random variables. The joint posterior is found as

$$p(\theta_{1:n}, \gamma, \Sigma, \sigma^2 | U_{1:n}) \propto p(U_{1:n}|\theta_{1:n}, \sigma^2) \times p(\theta_{1:n}|\gamma, \Sigma) \times p(\sigma^2, \gamma, \Sigma).$$

Code for this model implementation in Stan is available in the supporting documentation. The effective independent sample size ranged between 251 and 875 samples.

7. DISCUSSION

The following sections discuss the advantages fo HMC modelling.

7.1. Prior Selection

As mentioned in Section 2, Bayesian inference requires a prior distribution over each parameter. It is important that this accurately represents the modeller's prior belief of the parameter distribution. In the absence of any belief of the

value of the parameters, non-informative priors are set such that they are diffuse enough to capture a wide array of possibilities. The rational for using non-informative priors is to let the data drive all of the analysis, so that inference is unaffected by information external to the current data (Gelman et al., 2014).

By using Gibbs sampling as the primary MCMC sampling method, the distributions that may be set as priors are limited, as Gibbs sampling relies on being able to sample from the full conditional distribution. An example, and perhaps the biggest instance of this, arises in the specification of priors for variance terms. Perhaps the most common selection of prior for variance terms is the inverse-gamma distribution for univariate data and the inverse-Wishart distribution for multivariate data. Issues arise, however, when these distributions are constructed to be non-informative. Consider for example the inverse-gamma(ϵ, ϵ) distribution. As $\epsilon \to 0$, as to make the distribution non-informative, it yields an improper posterior density, and thus must be set to a reasonable value (Gelman et al., 2006). For datasets in which low values of variance is possible, inference becomes sensitive to ϵ. Carpenter et al. (2016) recommends the use of Half Cauchy distribution to model variance parameters.

Implementation of the models in Stan did not require the explicit specification of conjugate priors. Non-informative non-conjugate priors can be selected. Although sampling efficiency may be increased with certain prior selection, HMC may still sample from any combination of likelihood and prior, provided both are continuous. This allows the practitioner to select the most appropriate model, rather than something convenient for modelling.

A pertinent application of this for prognostics research comes from the modelling of lithium-ion battery capacity degradation. As an example, past research has seen battery degradation modelled with Gaussian noise, despite clearly positively skewed data due to the self healing nature of lithium-ion batteries. This can be seen in (He, Williard, Osterman, & Pecht, 2011).

7.2. Sampling autocorrelation

The autocorrelation of the MCMC samples generated scaled with the dimensionality of the statistical models. For the low dimensional hierarchical models shown by Case Studies 1 and 2, the HMC sampler was able to efficiently generate near independent samples from the posterior distribution. For Case Study 3, where there was an increase in sampling dimensions, there was a marked increase in autocorrelation between samples. The traceplots for a single parameter from each study is shown in Figure 7.

For the linear random effects model shown in Case Study 1 the Gibbs sampler outperformed the HMC sampler in terms

Figure 7. Traceplots for parameters in the HMC implementation of Models 1, 2 and 3 with highest autocorrelation. The top image shows the samples of the noise parameter σ from Model 1. The middle image shows the hyper-variance parameter τ_3^2 from Model 2. The bottom image shows the sample from the hyper-mean parameter γ_{10} from Model 3.

of effective sample size. This is due to the small covariance amongst the parameters, meaning that the Gibbs sampler was able to efficiently move through the sample space at each iteration. Conversely, in the non-linear random effects model shown in Case Study 2 the HMC sampler generated near perfect samples whereas the Gibbs/MH sampling scheme had high amounts of autocorrelation. This is seen in the traceplots for the $\alpha_{1,1}$ parameter shown in Figure 8. The top image shows the samples of $\alpha_{1,1}$ generated by HMC sampling, and the bottom image shows the samples of $\alpha_{1,1}$ generated via a MH sampling scheme.

7.3. Programming implementation

By implementing models in Stan, practitioners can focus on the statistical model, rather than spend time coding the samplers. This has three main advantages, (1) Bayesian models are more accessible as the modeller does not have to understand the code for complex MCMC dynamics, (2) the potential for coding errors in the model is reduced, and (3) time is saved on implementation. Note these advantages are not unique to Stan, and are shared by many other probabilistic

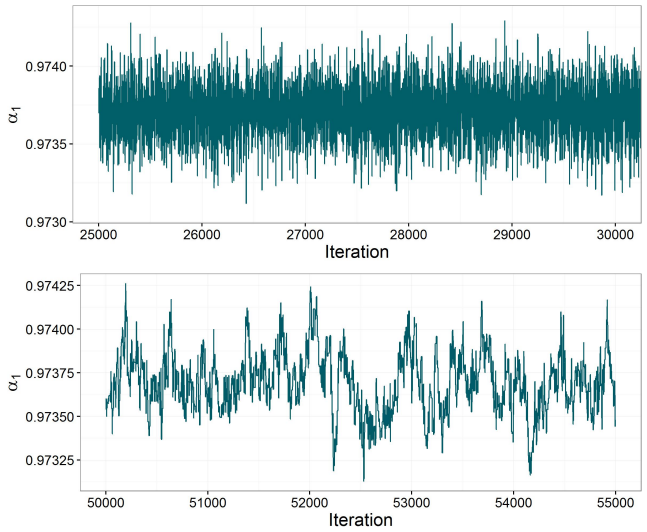

Figure 8. Traceplots for the $\alpha_{1,1}$ parameter from Model 2. The top image is sampled with the HMC sampler, the bottom is sampled with a MH algorithm

programming languages such as BUGS or pyMC3. Whilst the implementation may be simplified, analysis of the generated MCMC samples is still required to assess whether it is representative of the posterior distribution. Stan does not identify sampling issues such a lack of convergence, or poorly mixed samples. This is still the responsibility of the practitioner.

7.4. Disadvantages of HMC

HMC has some clear disadvantages. As mentioned in Section 3.3 HMC is sensitive to parameter tuning, and even though Stan provides a significant step toward automation of the sampler, tuning may still require thought. Stan uses the initial warm-up stage to tune the parameters to provide the desired acceptance rate (this defaults to 80%). Further, Stan implements the No-U-Turns Sampler (NUTS) to efficiently explore the sample space. More information on this is available in Hoffman and Gelman (2014).

A further crucial disadvantage is that, due to it's reliance on continuous gradients, it cannot sample from models with discrete parameters. Hence, models with categorical variables, population numbers or latent variables may not be able to be directly implemented in Stan. Should these parameters be able to be marginalised out analytically, the model may still be able to run in Stan, however for some cases this is not trivial and for others it is not possible at all. A practitioner implementing the sampler directly would have many options, most prominently block sampling. The discrete states may be sampled with Metropolis Hastings or Gibbs, while the continuous section of the model is sampled with HMC. Stan presently does not allow for this to be implemented, and direct coding of inferential algorithms is time consuming and error prone. The BUGS family of samplers do allow discrete parameters, but contain other limitations as described in Section 4, in particular that they do not allow the use of HMC. Other probabilistic programming languages, such as PyMC3 or Edward (both built on top of Python), allow the full block sampling scheme to be described and used without requiring direct coding, but in the authors' opinion they are not yet as mature as Stan (Salvatier et al., 2016; Tran et al., 2016). We are hopeful that there will be improvement in this space over the next few years.

Whilst HMC proposes a new and efficient way of MCMC sampling, it is clear that certain pathologies exist, and the practitioner should be aware of them. We implemented a HMC sampling scheme on three hierarchical models with differing degrees of complexity. As HMC sampling uses gradient approximation to sample from the posterior distribution, high curvature posteriors can lead to divergence from the target distribution. In hierarchical models this is apparent with small values in the hyper-variance parameters. Papaspiliopoulos, Roberts, and Sköld (2007) proposes that this may be overcome by re-parametrising the random effect term in the hierarchical model. Briefly, the centred form, which has been used in this research, models the random effects directly. Using the univariate nomenclature from Model 2, this is represented as $\alpha \sim N(\mu, \tau^2)$. The non-centred reparametrisation models the random effects indirectly by letting $\alpha = \mu + \tau Z$ where $Z \sim N(0, 1)$. This re-parametrisation smooths the posterior of the centred model, leading to less divergences and a more efficient sampler. Non-centred reparametrisation is currently being explored and hopes to alleviate some pathologies that occur with sampling hierarchical models with HMC (Betancourt & Girolami, 2015; Monnahan, Thorson, & Branch, 2016).

8. CONCLUSION

Large, complex and high dimensional data-sets are increasingly becoming an issue for data analysis and prognostics. However, when processed with appropriately powered software tools they can lead to insight and analysis that was previously infeasible. We demonstrate that HMC sampling, using Stan is a simpler and more effective way to generate MCMC samples in Bayesian analysis. We advocate for Stan to be increasingly used amongst the prognostics community to address large and complex Bayesian models, as it has the potential to extend the boundaries of feasible models for applied problems, and lead to better analysis and prediction in prognostics.

REFERENCES

Andrieu, C., & Thoms, J. (2008). A tutorial on adaptive mcmc. *Statistics and Computing*, *18*(4), 343–373.

Betancourt, M., & Girolami, M. (2015). Hamiltonian monte carlo for hierarchical models. *Current trends in Bayesian methodology with applications*, *79*, 30.

Blei, D. M., Kucukelbir, A., & McAuliffe, J. D. (2016). Variational Inference: A Review for Statisticians.

Brooks, S., Gelman, A., Jones, G. L., & Meng, X.-L. (2011). *Handbook of markov chain monte carlo*. Chapman and Hall/CRC.

Brooks, S., & Roberts, G. O. (1998). Convergence assessment techniques for markov chain monte carlo. *Statistics and Computing*, *8*(4), 319–335.

Carpenter, B., Gelman, A., Hoffman, M., Lee, D., Goodrich, B., Betancourt, M., ... Riddell, A. (2016). Stan: A probabilistic programming language. *Journal of Statistical Software*, *20*.

Coble, J., & Hines, J. W. (2011). Applying the general path model to estimation of remaining useful life. *International Journal of Prognostics and Health Management*, *2*(1), 71–82.

Cripps, E., & Pecht, M. (2017). A bayesian nonlinear random effects model for identification of defective batteries from lot samples. *Journal of Power Sources*, *342*, 342–350.

Gelman, A., Carlin, J. B., Stern, H. S., & Rubin, D. B. (2014). *Bayesian data analysis* (Vol. 2). Chapman & Hall/CRC Boca Raton, FL, USA.

Gelman, A., et al. (2006). Prior distributions for variance parameters in hierarchical models (comment on article by browne and draper). *Bayesian analysis*, *1*(3), 515–534.

Geman, S., & Geman, D. (1984). Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*(6), 721–741.

Gilks, W. R., Richardson, S., & Spiegelhalter, D. (1995). *Markov chain monte carlo in practice*. CRC press.

Green, P. J. (1995). Reversible jump markov chain monte carlo computation and bayesian model determination. *Biometrika*, 711–732.

Hastings, W. K. (1970). Monte carlo sampling methods using markov chains and their applications. *Biometrika*, *57*(1), 97–109.

He, W., Williard, N., Osterman, M., & Pecht, M. (2011). Prognostics of lithium-ion batteries based on dempster–shafer theory and the bayesian monte carlo method. *Journal of Power Sources*, *196*(23), 10314–10321.

Hoffman, M. D., & Gelman, A. (2014, January). The No-U-turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. *J. Mach. Learn. Res.*, *15*(1), 1593–1623.

Lunn, D. J., Thomas, A., Best, N., & Spiegelhalter, D. (2000). Winbugs - a bayesian modelling framework: Concepts, structure, and extensibility. *Statistics and Computing*, *10*(4), 325–337.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of state calculations by fast computing machines. *The journal of chemical physics*, *21*(6), 1087–1092.

Monnahan, C. C., Thorson, J. T., & Branch, T. A. (2016). Faster estimation of bayesian models in ecology using hamiltonian monte carlo. *Methods in Ecology and Evolution*.

Neal, R. M., et al. (2011). Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, *2*, 113–162.

Orchard, M., Kacprzynski, G., Goebel, K., Saha, B., & Vachtsevanos, G. (2008). Advances in uncertainty representation and management for particle filtering applied to prognostics. In *Prognostics and health management, 2008. phm 2008. international conference on* (pp. 1–6).

Papaspiliopoulos, O., Roberts, G. O., & Sköld, M. (2007). A general framework for the parametrization of hierarchical models. *Statistical Science*, 59–73.

Plummer, M. (2003). *Jags: A program for analysis of bayesian graphical models using gibbs sampling*.

Saha, B., & Goebel, K. (2007). Battery data set. *NASA Ames Prognostics Data Repository*.

Saha, B., Goebel, K., Poll, S., & Christophersen, J. (2009). Prognostics methods for battery health monitoring using a bayesian framework. *IEEE Transactions on instrumentation and measurement*, *58*(2), 291–296.

Salvatier, J., Wiecki, T., & Fonnesbeck, C. (2016). Probabilistic programming in Python using PyMC3. *PeerJ Computer Science*, *2:e55*. Retrieved from `https://doi.org/10.7717/peerj-cs.55`

Tobon-Mejia, D., Medjaher, K., & Zerhouni, N. (2012). Cnc machine tool's wear diagnostic and prognostic by using dynamic bayesian networks. *Mechanical Systems and Signal Processing*, *28*, 167–182.

Tran, D., Kucukelbir, A., Dieng, A. B., Rudolph, M., Liang, D., & Blei, D. M. (2016). Edward: A library for probabilistic modeling, inference, and criticism. *arXiv preprint arXiv:1610.09787*.

Wei, T., Huang, Y., & Chen, C. P. (2009). Adaptive sensor fault detection and identification using particle filter algorithms. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *39*(2), 201–213.

Xu, X., Li, Z., & Chen, N. (2016). A hierarchical model for lithium-ion battery degradation prediction. *IEEE Transactions on Reliability*, *65*(1), 310–325.

Zaidan, M. A., Harrison, R. F., Mills, A. R., & Fleming, P. J. (2015). Bayesian hierarchical models for aerospace gas turbine engine prognostics. *Expert Systems with Applications*, *42*(1), 539–553.

BIOGRAPHIES

Lachlan Astfalck is a PhD candidate from the University of Western Australia working in the System Health Laboratory. He recieved his B.Eng (1st class Hons) degree from The University of Western Australia in 2014 and graduated as valedictorian of his class. In 2014 he was awarded the Chevron Chair prize in Gas Process Engineering, and in 2012 the Convocation of UWA Graduates Undergraduate Prize. At the beginning of 2017 he joined the ARC Research Hub for Offshore Floating Systems. His current research focuses on the industrial implementation of prognostic systems, sensitivity analysis, and meta-ocean prediction.

Melinda R. Hodkiewicz is the BHP Billiton Fellow for Engineering for Remote Operations at the University of Western Australia (UWA). She has a BA (Hons) in Metallurgy and Science of Materials from Oxford University in 1985, and a Ph.D. in Mechanical Engineering from the University of Western Australia in 2004. Prior to her PhD, she worked in industry in operations and maintenance roles. She now leads the System Health Laboratory at UWA and works in the areas of asset health, maintenance and safety.

APPENDIX A

8.1. Model 1 - Full conditional distributions

Conjugate priors are selected for μ, Σ, σ^2, where $\mu \sim N(\mu_0, \Lambda_0)$, $\Sigma \sim IW(\eta_0, S_0^{-1})$, and $\sigma^2 \sim IG(\nu_0/2, \nu_0\sigma_0^2/2)$. The full conditional distributions are

$$\begin{aligned}
\beta_j &\sim N(\hat{\mu_\beta}, \hat{\Sigma}_\beta) \\
\mu &\sim N(\hat{\mu}_n, \hat{\Lambda}_n) \\
\Sigma &\sim IW(\eta_0 + n, (S_0 + S_\theta)^{-1}) \\
\sigma^2 &\sim IG([\nu_0 + \sum n_j]/2, [\nu_0\sigma_0^2 + SSR]/2)
\end{aligned}$$

where

$$\begin{aligned}
\hat{\Sigma}_\beta &= (\Sigma^{-1} + X'X/\sigma^2)^{-1} \\
\hat{\mu_\beta} &= \hat{\Sigma}_\beta(\Sigma^{-1}\mu + X'y_j/\sigma^2) \\
\hat{\Lambda}_n &= (\Lambda_0^{-1} + J\Sigma^{-1})^{-1} \\
\hat{\mu}_n &= \hat{\Lambda}_n(\Lambda_0^{-1}\mu_0 + J\Sigma^{-1}\bar{\beta}) \\
S_\theta &= \sum_{j=1}^{J}(\beta_j - \mu)(\beta_j - \mu)' \\
SSR &= \sum_{j=1}^{J}\sum_{i=1}^{n_j}(y_{i,j} - \beta_j' x_{i,j})^2
\end{aligned}$$

8.2. Model 2 - Full conditional distributions

The full conditional distributions, as originally presented, are listed below. Uninformative priors are set, where $\mu_i \sim U(-1,1)$ for $i = 1,2,3,4$, $\tau_i^2 \sim U(0,1)$ for $i = 1,2,3,4$, and $\sigma^2 \sim U(0,1)$. $U(a,b)$ denotes a uniform distribution over the interval (a,b).

$$\begin{aligned}
\sigma^2|Y,\alpha &\sim IG\left(\frac{JT}{2} - 1, \frac{1}{2}\sum_{j=1}^{J}(y_j - f_j)'(y_j - f_j)\right)\mathbb{1}\{\sigma^2 < 1\} \\
\mu_i|\alpha_{1:J,i},\tau_i^2 &\sim N\left(\bar{\alpha}_i, \frac{\tau_i^2}{J}\right)\mathbb{1}\{-1 < \mu < 1\} \\
\tau_i^2|\alpha_{1:J,i},\mu_i &\sim IG\left(\frac{J}{2} - 1, \frac{1}{2}\sum_{j=1}^{J}(\alpha_{i,j} - \mu_i)^2\right)\mathbb{1}\{\tau_i^2 < 1\}.
\end{aligned}$$

The conditional distribution $p(\alpha_j|Y,\sigma^2,\mu,\tau)$ is unknown, and MH sampling is performed with a symmetrical Gaussian random walk proposal distribution, where

$$\begin{aligned}
p(\alpha_j|Y,\sigma^2,\mu,\tau) \propto \left(2\pi\sigma^2\right)^{-T/2}\exp\left\{-\frac{1}{2\sigma^2}(y_j - f_i)'(y_j - f_i)\right\} \\
\times|2\pi\Sigma|^{-1/2}\exp\left\{-\frac{1}{2}(\alpha_j - \mu)'\Sigma(\alpha_j - \mu)\right\}
\end{aligned}$$

8.3. Model 3 - Full conditional distributions

Conjugate priors are chosen for γ, Σ and σ_ϵ^2, where $\gamma \sim N(\mu,\Lambda)$, $\Sigma \sim IW(\nu,\eta)$ and $\sigma_\epsilon^2 \sim IG(\alpha,\omega)$. The full conjugate distributions of these parameters is

$$\begin{aligned}
\gamma|\Sigma,\theta_{1:n} &\sim N(\hat{\mu},\hat{\Lambda}) \\
\Sigma|\gamma,\theta_{1:n} &\sim IW(\hat{\nu},\hat{\eta}) \\
\sigma_\epsilon^2|\theta_{1:n},U_{1:n} &\sim IG(\hat{\alpha},\hat{\omega})
\end{aligned}$$

where

$$\hat{\Lambda} = \left(\sum_{i=1}^{n} C_i' \Sigma^{-1} C_i + \Lambda^{-1} \right)^{-1}$$

$$\hat{\mu} = \hat{\Lambda} \left(\sum_{i=1}^{n} C_i' \Sigma^{-1} \theta_i + \Lambda^{-1} \mu \right)$$

$$\hat{\nu} = \nu + \sum_{i=1}^{n} (\theta_i - C_i \gamma)(\theta_i - C_i \gamma)'$$

$$\hat{\eta} = \eta + n$$

$$\hat{\alpha} = \alpha + \sum_{i=1}^{n} \frac{m_i}{2}$$

$$\hat{\omega} = \omega + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{m_i} \left[U_i(t_{i,j}) - f(t_{i,j}; \theta_i) \right]^2$$

As the parameters $\theta_i, i = 1, 2, ..., n$ are considered conditionally statistically independent, they are sampled separately. θ_i is decomposed into it's two parts β_i and τ_i. The full conditional distribution of β_i is $\beta_i | \tau_i, \gamma, \Sigma, \sigma_\epsilon^2, U_i \sim N(\hat{\mu}_\beta, \hat{\Sigma}_\beta)$, where

$$\hat{\Sigma}_\beta = \left(\frac{X_i' X_i}{\sigma_\epsilon^2} + \Sigma_\beta^{-1} \right)^{-1}$$

$$\hat{\mu}_\beta = \hat{\Sigma}_\beta \left(\frac{X_i' U_i}{\sigma_\epsilon^2} + \Sigma_\beta^{-1} \mu_\beta \right).$$

The full conditional distribution of τ_i cannot be expressed in analytic form, and is sampled using a MH algorithm. Its posterior is proportional to

$$p(\tau_i | \beta_i, \gamma, \Sigma) \propto p(\tau_i | \beta_i, \gamma, \Sigma) \times p(U_i | \sigma_\epsilon^2, \theta_i)$$