Accelerating Pseudo-Marginal MCMC using Gaussian Processes

C. C. Drovandi*, M. T. Moores† and R. J. Boys‡

*Australian Centre of Excellence for Mathematical and Statistical Frontiers and Queensland University of Technology, Brisbane, Australia 4001

† University of Warwick, Coventry CV4 7AL, United Kingdom

‡ Newcastle University, Newcastle upon Tyne NE1 7RU, United Kingdom

email: `c.drovandi@qut.edu.au`

October 3, 2017

Abstract

The grouped independence Metropolis-Hastings (GIMH) and Markov chain within Metropolis (MCWM) algorithms are pseudo-marginal methods used to perform Bayesian inference in latent variable models. These methods replace intractable likelihood calculations with unbiased estimates within Markov chain Monte Carlo algorithms. The GIMH method has the posterior of interest as its limiting distribution, but suffers from poor mixing if it is too computationally intensive to obtain high-precision likelihood estimates. The MCWM algorithm has better mixing properties, but tends to give conservative approximations of the posterior and is still expensive. A new method is developed to accelerate the GIMH method by using a Gaussian process (GP) approximation to the log-likelihood and train this GP using a short pilot run of the MCWM algorithm. This new method called GP-GIMH is illustrated on simulated data from a stochastic volatility and a gene network model. The new approach produces reasonable posterior approximations in these examples with at least an order of magnitude improvement in computing time. Code to implement the method for the gene network example can be found at `http://www.runmycode.org/companion/view/2663`.

Keywords: Gaussian processes, likelihood-free methods, Markov processes, particle Markov chain Monte Carlo, pseudo-marginal methods, state space models

1 Introduction

Bayesian inference for high-dimensional latent variable models is currently challenging. In particular Markov chain Monte Carlo (MCMC) samplers can suffer from poor mixing due to correlation between the parameter of interest and the latent variables. Beaumont (2003) and Andrieu and Roberts (2009) have introduced pseudo-marginal methods to improve the statistical efficiency of MCMC. These methods work by replacing the actual likelihood with an unbiased likelihood estimate in the Metropolis-Hastings ratio. This allows proposals for the Markov chain to be made directly on the space of the parameter of interest, rather than conditional on the value of a set of the latent variables.

One of these methods, the grouped independence Metropolis-Hastings (GIMH) method by Beaumont (2003), recycles the likelihood estimate for the current value of the chain to the next iteration. Andrieu and Roberts (2009) have shown that the GIMH method has the desired posterior as its limiting distribution, which is why it has received considerable attention in the literature (Andrieu et al. (2010); Doucet et al. (2015)). However, a drawback of the GIMH method is that it can suffer from poor mixing if it is too computationally expensive to estimate the likelihood with high precision.

The other method, the Markov chain within Metropolis (MCWM, Beaumont (2003)) algorithm, estimates the likelihood at both the current and proposed values of the Markov chain at every iteration. This method generally possesses better mixing properties as it is able to escape an overestimated likelihood value by re-estimating it at the next MCMC iteration. However, the MCWM method does not have the posterior distribution of the parameter of interest as its limiting distribution. Because of this, MCWM has received comparatively less attention.

In this paper we make use of a Gaussian process (GP) to accelerate the GIMH method while at the same time accepting some approximation to the posterior distribution.

Wilkinson (2014) proposes that GPs be used to accelerate approximate Bayesian computation (ABC) methods where the likelihood is approximated by generating many model simulations from each proposed parameter value, and measuring the distance between observed and simulated data through a careful choice of summary statistics. Here GPs are used to emulate the actual (ABC) log-likelihood surface based on noisy estimates obtained through simulation. The method iteratively uses the GP to discard implausible parts of the parameter space, retrains the GP in the updated not-implausible part of the parameter space and continues this process until the GP fit has been deemed as satisfactory. The final GP is then used within an MCMC method to predict the log-likelihood surface at all proposed values of the parameter of interest. GPs have also been used for ABC by Meeds and Welling (2014), Gutmann and Corander (2016) and Järvenpää et al. (2016).

We follow a similar approach to Wilkinson (2014) to accelerate pseudo-marginal methods. However, one key difference is that we take advantage of the pseudo-marginal literature. In particular, we use a short run of the MCWM method as a natural approach to obtain training samples for the GP in non-negligible regions of the posterior support. The MCWM method is ideal for training the GP as it has better mixing properties and is less prone to sticky periods than the GIMH method. Medina-Aguayo et al. (2016) develop sufficient conditions for the geometric ergodicity and hence the existence of an invariant distribution of MCWM.

Our experience with MCWM is that it is generally conservative (inflated posterior variance), allowing the tails of the posterior to be explored. The fitted GP is used instead of expensive likelihood estimates within the GIMH method. We introduce further novelties into our method to make it practically useful.

The paper has the following outline. In Sections 2.1 and 2.2 we provide a brief overview of pseudo-marginal methods and GPs, respectively. In Section 2.3 we present our new method, GP-GIMH, which uses the MCWM algorithm to train the GP and subsequently uses the GP to accelerate the GIMH method. Finally, in Section 4, we conclude with a discussion.

2 Accelerated Pseudo-Marginal MCMC

In this section we give some background on pseudo-marginal MCMC methods and Gaussian processes before describing how, by emulating the log-likelihood using a GP, we can accelerate pseudo-marginal MCMC.

2.1 Pseudo-Marginal MCMC

Suppose we have observed data $\boldsymbol{y}$ in Y which is described by a statistical model with likelihood function $p(\boldsymbol{y}|\boldsymbol{\theta})$ and depends on an unknown parameter $\boldsymbol{\theta}$ in $\mathbb{R}^d$. Prior beliefs about the parameter are summarised by the prior density $p(\boldsymbol{\theta})$. We assume that the model requires, or is facilitated by, an auxiliary variable $\boldsymbol{x}$ in X, whose value is not of direct interest. In this scenario the complete data likelihood is $p(\boldsymbol{y},\boldsymbol{x}|\boldsymbol{\theta}) = p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\theta})p(\boldsymbol{x}|\boldsymbol{\theta})$ and leads to the observed data likelihood $p(\boldsymbol{y}|\boldsymbol{\theta}) = \int_{\mathsf{X}} p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\theta})p(\boldsymbol{x}|\boldsymbol{\theta})d\boldsymbol{x}$. Ideally this observed data likelihood is combined with the prior to make inferences about the parameters via the posterior density $p(\boldsymbol{\theta}|\boldsymbol{y}) \propto p(\boldsymbol{y}|\boldsymbol{\theta})p(\boldsymbol{\theta})$. However, in non-toy problems the observed data likelihood is an analytically intractable integral. Therefore the parameter posterior is accessed as the marginal of the joint posterior for all unknowns, that is, via $p(\boldsymbol{\theta}|\boldsymbol{y}) = \int_{\mathsf{X}} p(\boldsymbol{\theta},\boldsymbol{x}|\boldsymbol{y})d\boldsymbol{x}$.

A standard Bayesian approach for fitting such a latent variable model is to develop an MCMC algorithm that samples the joint posterior $p(\boldsymbol{\theta},\boldsymbol{x}|\boldsymbol{y})$ and marginalises by ignoring the $\boldsymbol{x}$ samples. A common approach is to develop an MCMC algorithm using two blocks, $\boldsymbol{\theta}$ and $\boldsymbol{x}$, that iteratively samples from the full conditionals $p(\boldsymbol{\theta}|\boldsymbol{x},\boldsymbol{y})$ and $p(\boldsymbol{x}|\boldsymbol{\theta},\boldsymbol{y})$. A key problem with such algorithms is that they can mix poorly because of high posterior correlation between the blocks $\boldsymbol{\theta}$ and $\boldsymbol{x}$. Further, for non-trivial state space models, $p(\boldsymbol{x}|\boldsymbol{\theta},\boldsymbol{y})$ cannot be sampled directly and is difficult to sample efficiently (see Andrieu et al. (2010) for a discussion). In an attempt to overcome the mixing issue, Beaumont (2003) develop algorithms that replace the computationally intractable likelihood $p(\boldsymbol{y}|\boldsymbol{\theta})$ with an unbiased estimate $\hat{p}(\boldsymbol{y}|\boldsymbol{\theta})$. The underpinning mathematics of these pseudo-marginal MCMC algorithms is studied in Andrieu and Roberts (2009) and they develop conditions under which they indeed have the correct posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{y})$ as their limiting distribution. A simple example of an unbiased likelihood estimate is one obtained through importance sampling, namely

$$\hat{p}(\boldsymbol{y}|\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\frac{p(\boldsymbol{y}|\boldsymbol{x}_i,\boldsymbol{\theta})p(\boldsymbol{x}_i|\boldsymbol{\theta})}{g(\boldsymbol{x}_i)},$$

where $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \overset{\text{iid}}{\sim} g(\boldsymbol{x})$ and g is an importance density defined on X. Alternative approaches to obtaining an unbiased likelihood estimate are available. For example, Andrieu et al. (2010) show that when the model of interest is a state-space model, the likelihood $p(\boldsymbol{y}|\boldsymbol{\theta})$ can be estimated unbiasedly using a particle filter with N particles. Such pseudo-marginal methods are referred to as particle Markov chain Monte Carlo (PMCMC). We consider models in the state-space form in Section 3 and use the bootstrap particle filter of Gordon et al. (1993) to obtain an unbiased likelihood estimator.

Bérard et al. (2014) establish a log-normal central limit theorem for the particle filter. In PMCMC, the likelihood is estimated with multiplicative noise, $\hat{p}(\boldsymbol{y}|\boldsymbol{\theta}) = W p(\boldsymbol{y}|\boldsymbol{\theta})$, where the random weight W is strictly positive with $\mathsf{E}[W] = 1$. The CLT defines a limiting distribution for the noise in which $\log W \xrightarrow{d} \mathcal{N}\left(-\alpha\sigma^2/2, \alpha\sigma^2\right)$ as the dimension (of the state space X) $T \to \infty$. Here σ^2 is the asymptotic variance of the estimator, and α is the asymptotic ratio of T to N as both $T, N \to \infty$, and is usually taken to be 1. Doucet et al. (2015) observe that this limiting distribution is a good fit for the noise, even for modest values of T and N. The log-normal CLT is very useful for theoretical analysis of PMCMC algorithms as shown, for example, by Doucet et al. (2015) and Medina-Aguayo et al. (2016). We assume log-normality of $\hat{p}(\boldsymbol{y}|\boldsymbol{\theta})$ in our GP model.

The first algorithm developed by Beaumont (2003), known as grouped independence Metropolis-Hastings (GIMH), is shown in Appendix A. This is essentially a standard Metropolis-Hastings algorithm in which the intractable likelihood at a new proposal $\boldsymbol{\theta}^*$ is replaced by an unbiased likelihood estimate. Note that the likelihood at the current value $\boldsymbol{\theta}$ is not re-estimated but simply recycled from the previous iteration.

Andrieu and Roberts (2009) show that the GIMH algorithm has the posterior $p(\boldsymbol{\theta}|\boldsymbol{y})$ as its limiting distribution. If we denote all the random numbers (assumed to be uniformly distributed without loss of generality) used to produce an unbiased likelihood estimate as $\boldsymbol{\eta} \in [0, 1]^s$ and considering a target posterior distribution on the extended space of $(\boldsymbol{\theta}, \boldsymbol{\eta})$, the $\boldsymbol{\theta}$ marginal target of interest is proportional to $p(\boldsymbol{\theta})\mathsf{E}[\hat{p}(\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{\eta})]$ where $\hat{p}(\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{\eta})$ is the likelihood estimate given the random numbers and the expectation is taken with respect to the distribution of $\boldsymbol{\eta}$ given $\boldsymbol{\theta}$. The unbiased nature of the likelihood estimator implies that the expectation is $p(\boldsymbol{y}|\boldsymbol{\theta})$ giving the desired posterior distribution as the target. This theoretically appealing property has led to the GIMH method becoming more prominent in the literature (e.g. Andrieu et al. (2010), Doucet et al. (2015) and Sherlock et al. (2017)) compared to the other approach of Beaumont (2003), the Monte Carlo within Metropolis (MCWM) algorithm. However, the GIMH method can get stuck when the likelihood is substantially overestimated at any given iteration. Doucet et al. (2015) suggest that for good performance in the GIMH algorithm, the log-likelihood should be estimated with a standard deviation between 1.0 and 1.7. However, in complex applications, it may be computationally difficult to achieve this goal.

The MCWM algorithm of Beaumont (2003) is the same as GIMH except for an extra step where the likelihood at the current $\boldsymbol{\theta}$ is re-estimated and not recycled from the previous iteration; see Appendix A. Although MCWM requires roughly double the amount of computational effort per iteration relative to GIMH, it does not suffer from stickiness in the Markov chain. However, the drawback is that the limiting distribution of MCWM is only the posterior $p(\boldsymbol{\theta}|\boldsymbol{y})$ in the limit as $N \to \infty$. Medina-Aguayo et al. (2016) develop sufficient conditions for the geometric ergodicity and hence the existence of an invariant distribution of MCWM for large enough N. The conditions are that the idealised chain (the chain that uses exact likelihood evaluations) is geometrically ergodic, the weights W are uniformly integrable and the weights satisfy uniform exponential bounds on their densities close to 0. These conditions are quite weak when the parameter space is compact. In our experience, MCWM generally produces an approximate posterior that is less precise than the actual posterior for finite N, and thus could be considered a conservative method in the sense that the posterior variances are overestimated.

Our aim is to accelerate the GIMH method by emulating the (unobserved) true log-likelihood surface as a function of $\boldsymbol{\theta}$ with a GP, where the GP is trained in relevant parts of the parameter space based on the output of a short run of MCWM.

2.2 Gaussian Processes

Gaussian processes can be used as a prior distribution to describe uncertainty about an unknown function $f(\cdot)$. They are characterised by a mean function $m_{\boldsymbol{\beta}}(\boldsymbol{\theta})$ and covariance function $C_{\boldsymbol{\gamma}}(\boldsymbol{\theta}, \boldsymbol{\theta}') = \text{cov}\{f(\boldsymbol{\theta}), f(\boldsymbol{\theta}')\}$, where $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ are so-called hyperparameters of the GP. A Gaussian process has the property that the joint distribution for the values of the function at a finite collection of points has a multivariate normal distribution; see, for example, Rasmussen and Williams (2006). In this paper, the function we wish to emulate is the log-likelihood function $f(\boldsymbol{\theta}) = \log p(\boldsymbol{y}|\boldsymbol{\theta})$. We assume a GP prior with mean function $m_{\boldsymbol{\beta}}(\boldsymbol{\theta}) = \beta_0 + \sum_{k=1}^{p} \beta_k \theta_k + \sum_{k=1}^{p} \beta_{k+p} \theta_k^2$, where θ_k is the kth component of the parameter vector $\boldsymbol{\theta}$ and $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_{2p})^\top$. We also assume that the log-likelihood surface is smooth and so take a squared exponential covariance function

$$C_{\boldsymbol{\gamma}}(\boldsymbol{\theta}, \boldsymbol{\theta}') = \delta_c \exp\left\{ -\frac{1}{2} \sum_{k=1}^{p} \frac{(\theta_k - \theta_k')^2}{r_k^2} \right\},$$

with hyper-parameters $\boldsymbol{\gamma} = (\delta_c, r_1, \ldots, r_p)^\top$.

Only noisy estimates of the likelihood are available and so we need to model their sampling distribution. We rely on the log-normal CLT (Bérard et al., 2014) and assume multiplicative noise with variance δ. This assumption is not quite correct as the estimates are not exactly log-normal for finite N, even when drawn from a particle filter. Additionally, there may be some dependence of their accuracy on the parameter value $\boldsymbol{\theta}$. Nevertheless we believe that this description captures the key aspects of the sampling distribution and has the great benefit of simplifying the form of GP prediction. Specifically, taking account of the variability in the function evaluations requires that we add a nugget term to the covariance function, so that $\text{cov}\{\hat{f}(\boldsymbol{\theta}), \hat{f}(\boldsymbol{\theta}')\} = C_{\boldsymbol{\gamma}}(\boldsymbol{\theta}, \boldsymbol{\theta}') + \delta \mathbb{1}(\boldsymbol{\theta} = \boldsymbol{\theta}')$, where $\mathbb{1}(\cdot)$ denotes the indicator function, which is 1 if its argument is true and 0 otherwise. We can estimate the GP hyperparameter $\boldsymbol{\xi} = (\boldsymbol{\beta}, \boldsymbol{\gamma}, \delta)$ using a training sample containing evaluations of the log-likelihood estimates at a set of J (input) values Θ. We denote this training sample by $\mathcal{D}_T = \{\boldsymbol{\theta}_j, \hat{f}(\boldsymbol{\theta}_j)\}_{j=1}^{J}$.

Denote the $J \times 1$ vectors of log-likelihood estimates and (prior) mean functions as $\boldsymbol{f}(\Theta)$ and $\boldsymbol{m}_{\boldsymbol{\beta}}(\Theta)$, where $\boldsymbol{f}(\Theta)_j = \hat{f}(\boldsymbol{\theta}_j)$ and $\boldsymbol{m}_{\boldsymbol{\beta}}(\Theta)_j = m_{\boldsymbol{\beta}}(\boldsymbol{\theta}_j)$, and the $J \times J$ covariance matrix derived from the covariance function as $C_{\boldsymbol{\gamma}}(\Theta, \Theta)$ where $C_{\boldsymbol{\gamma}}(\Theta, \Theta)_{ij} = C_{\boldsymbol{\gamma}}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j)$. Under the GP model assumption, we have that $\boldsymbol{f}(\Theta) \sim \mathcal{N}\{\boldsymbol{m}_{\boldsymbol{\beta}}(\Theta), C_{\boldsymbol{\gamma}}(\Theta, \Theta) + \delta I\}$, where I is the $J \times J$

identity matrix. This result is obtained by integrating over the random variables describing the actual function values at the training input values and is thus often referred to as the marginal likelihood. The hyperparameter can therefore be estimated via maximising the log marginal likelihood, that is, taking

$$\hat{\boldsymbol{\xi}} = \arg\min_{\boldsymbol{\xi}} \Big(\{\boldsymbol{f}(\Theta) - \boldsymbol{m}_{\boldsymbol{\beta}}(\Theta)\}^\top \{C_{\boldsymbol{\gamma}}(\Theta,\Theta) + \delta I\}^{-1} \{\boldsymbol{f}(\Theta) - \boldsymbol{m}_{\boldsymbol{\beta}}(\Theta)\} + \log |C_{\boldsymbol{\gamma}}(\Theta,\Theta) + \delta I| \Big). \tag{1}$$

In practice the estimate is obtained using a numerical scheme and so we use multiple starting values for the optimisation process in order to obtain a result that we believe to be close to the maximum marginal likelihood estimate.

The fitted GP is the posterior distribution of the log-likelihood function in light of the observed training data. It can be used to predict the value of the log-likelihood function $f(\boldsymbol{\theta})$ at any value $\boldsymbol{\theta}$ as its posterior distribution is $f(\boldsymbol{\theta})|\mathcal{D}_T \sim \mathcal{N}\{m^*(\boldsymbol{\theta}), s^*(\boldsymbol{\theta})^2\}$, where

$$m^*(\boldsymbol{\theta}) = m_{\hat{\boldsymbol{\beta}}}(\boldsymbol{\theta}) + C_{\hat{\boldsymbol{\gamma}}}(\Theta,\boldsymbol{\theta})^\top \{C_{\hat{\boldsymbol{\gamma}}}(\Theta,\Theta) + \hat{\delta} I\}^{-1} \{\boldsymbol{f}(\Theta) - \boldsymbol{m}_{\hat{\boldsymbol{\beta}}}(\Theta)\}, \tag{2}$$

$$s^*(\boldsymbol{\theta})^2 = C_{\hat{\boldsymbol{\gamma}}}(\boldsymbol{\theta},\boldsymbol{\theta}) - C_{\hat{\boldsymbol{\gamma}}}(\Theta,\boldsymbol{\theta})^\top \{C_{\hat{\boldsymbol{\gamma}}}(\Theta,\Theta) + \hat{\delta} I\}^{-1} C_{\hat{\boldsymbol{\gamma}}}(\Theta,\boldsymbol{\theta}), \tag{3}$$

and $C_{\hat{\boldsymbol{\gamma}}}(\Theta,\boldsymbol{\theta})$ is a $J \times 1$ vector, with $C_{\hat{\boldsymbol{\gamma}}}(\Theta,\boldsymbol{\theta})_j = C_{\hat{\boldsymbol{\gamma}}}(\boldsymbol{\theta}_j,\boldsymbol{\theta})$. Note that the computational complexity of the GP prediction is $\mathcal{O}(J^3)$. If the hyperparameter estimate $\hat{\boldsymbol{\xi}}$ and the training data $\mathcal{D}_T$ are static then the inverse computation $\{C_{\hat{\boldsymbol{\gamma}}}(\Theta,\Theta) + \hat{\delta} I\}^{-1}$ can be re-used for each new $\boldsymbol{\theta}$. However, if additional training data is added then the inverse must be re-computed. In such cases it is of interest to limit the number of training points J.

2.3 Pseudo-Marginal MCMC using Gaussian Processes

This section describes how to deploy a GP to accelerate the GIMH algorithm. The new algorithm, called GP-GIMH, is given in Algorithm 1. More detail about each of the steps is given below.

As described in the previous section, we use a pilot run of the MCWM algorithm to generate the design points Θ, during which we record the parameter value and both estimated log-likelihood values encountered during the MCWM algorithm (even those from rejected proposals) in the training sample. Our motivation for using MCWM to determine the training sample for the GP is three-fold: (i) it harnesses the observed data and thus most of the training points will be parameter values with non-negligible posterior density; (ii) it has good mixing properties ensuring that the GP is trained at a wide range of plausible $\boldsymbol{\theta}$ values; and (iii) the MCWM method is conservative and so it can provide good coverage of the tails of the posterior distribution. However, since MCWM is conservative, it may visit parameter regions where the log-posterior is very low (especially for proposals rejected by the algorithm) and so we suggest removing points with very low log-posterior (ignoring the normalising constant independent of $\boldsymbol{\theta}$) values from the training sample so that the training of the GP focuses on plausible regions of the parameter space.

Once the GP is fitted, giving the functions $m^*(\boldsymbol{\theta})$ and $s^*(\boldsymbol{\theta})$, one might consider an algorithm that samples from the following approximate posterior

$$p_{\mathrm{GP}}(\boldsymbol{\theta}|\boldsymbol{y}) \propto \exp\left\{m^*(\boldsymbol{\theta}) + \frac{s^*(\boldsymbol{\theta})^2}{2}\right\} p(\boldsymbol{\theta}),$$

where the exponential term is the mean of the log-normal distribution implied by the GP assumption. A standard MCMC algorithm could be applied to sample from $p_{GP}(\boldsymbol{\theta}|\boldsymbol{y})$. However, for $p_{\mathrm{GP}}(\boldsymbol{\theta}|\boldsymbol{y})$ to be a reasonable approximation of $p(\boldsymbol{\theta}|\boldsymbol{y})$ we require $s^*(\boldsymbol{\theta})$ to not be too large across important parts of the parameter space. When $s^*(\boldsymbol{\theta})$ is too large at a proposed value $\boldsymbol{\theta}^*$, i.e. $s^*(\boldsymbol{\theta}^*) > \epsilon$ for some chosen threshold ϵ, we apply an intervention. Also, to reduce computation, we would like to limit the number of times where interventions are necessary. To achieve this, we adopt a GIMH-style algorithm. Instead of evaluating the mean of the log-normal density directly, we simulate a log-likelihood estimate from the fitted GP, $\mathcal{N}\{m^*(\boldsymbol{\theta}^*), s^*(\boldsymbol{\theta}^*)^2\}$. When $s^*(\boldsymbol{\theta}^*) > \epsilon$ and $\boldsymbol{\theta}^*$ is accepted, there is an increased risk of obtaining a sticky period at $\boldsymbol{\theta}^*$. If $s^*(\boldsymbol{\theta}^*) > \epsilon$ and $\boldsymbol{\theta}^*$ is rejected we do not apply an intervention. The intervention involves obtaining a more accurate GP prediction to check if $\boldsymbol{\theta}^*$ was wrongly accepted or to help reduce sticky periods. We obtain K independent estimates of $f(\boldsymbol{\theta}^*) = \log p(\boldsymbol{y}|\boldsymbol{\theta}^*)$ using the same method (e.g. importance sampling or particle filter estimates) as in the training phase and determine their mean. Under the log-normal CLT, this mean also has a normal distribution: $\bar{f}_K(\boldsymbol{\theta}^*) \sim \mathcal{N}\{f(\boldsymbol{\theta}^*), \hat{\delta}/K\}$ where $\hat{\delta}$ is the maximum likelihood estimate of δ found in (1). We can incorporate this information into our beliefs about the log-likelihood at this point using a simple Bayes update, to give

$$f(\boldsymbol{\theta}^*)|\mathcal{D}_T, \bar{f}_K(\boldsymbol{\theta}^*) \sim \mathcal{N}\left(\frac{m^*(\boldsymbol{\theta}^*)/s^*(\boldsymbol{\theta}^*)^2 + K\bar{f}_K(\boldsymbol{\theta}^*)/\hat{\delta}}{1/s^*(\boldsymbol{\theta}^*)^2 + K/\hat{\delta}} . \frac{1}{1/s^*(\boldsymbol{\theta}^*)^2 + K/\hat{\delta}}\right), \tag{4}$$

Therefore the total number of independent log-likelihood estimates needed to secure a sufficiently accurate GP prediction is roughly $K = \lceil \hat{\delta}\{\epsilon^{-2} - s^*(\boldsymbol{\theta}^*)^{-2}\}\rceil$. If these multiple estimates can be farmed out across say A available processors then the number of estimates in each batch to achieve this goal is $\lceil K/A \rceil$.

We allow a burn-in phase consisting of B iterations where such additional likelihood estimates can be appended to the training sample $\mathcal{D}_T$. The motivation for this burn-in phase is to assist the GP in being trained in important regions not explored sufficiently in the MCWM phase. We find in Section 3 that the burn-in phase is useful in applications where it is very expensive to estimate the likelihood. The computational drawback of the burn-in phase is that the matrix inversion in (2) and (3) must be re-computed. However, in demanding applications these additional matrix inversions can be relatively cheap. It would be feasible to re-estimate the GP hyperparameter after the burn-in phase if desired. We note that the relative computing time for this should be short in complex applications as the current hyperparameter estimate can be used as a starting value.

It might be tempting to continually grow the training sample through the entire algorithm and thereby obtain a more accurate GP across more of the parameter space. However, not only would this require additional time-consuming calculations of matrix inverses (of increasing size) but we also find that, as this alters the GP fit in areas previously explored, it creates unusual trace plots; essentially the target distribution of the algorithm is changing. We discuss this in more detail in Section 4. Once the additional training data at $\boldsymbol{\theta}^*$ is obtained we perform another Metropolis-Hastings accept/reject step but where the log-likelihood estimate is simulated based on (4).

The value of ϵ controls the level of uncertainty allowed in the GP prediction when parameter values are accepted. If ϵ is set too large then a parameter value may be accepted with a grossly overestimated log-likelihood value and lead to stickiness in the Markov chain, similar behaviour that can be observed in the standard GIMH method. Smaller values of ϵ will lead to runs that are generally less sticky, but more computation is required to satisfy the constraint $s^*(\boldsymbol{\theta}^*) < \epsilon$. Recall that Doucet et al. (2015) suggest that the log-likelihood should be estimated with a standard deviation of roughly 1 for the GIMH method to have similar statistical efficiency to a standard Metropolis-Hastings method where the likelihood is available. For our examples we find that $\epsilon < 2$ is a suitable choice.

In practice we choose ϵ by performing some very short pilot runs and ensuring that for the majority of accepted $\boldsymbol{\theta}^*$ we have $s^*(\boldsymbol{\theta}^*) < \epsilon$. Some insight into a suitable value of ϵ may also be obtained by inspecting the empirical distribution of the GP prediction standard deviations at the training points $\{s^*(\boldsymbol{\theta}_j),\ \boldsymbol{\theta}_j \in \mathcal{D}_T\}$. If ϵ is not in the upper tail of this distribution then this indicates that the GP training set $\mathcal{D}_T$ requires additional training points as otherwise many additional log-likelihood estimates will be needed in GP-GIMH and little algorithmic speed-up will be obtained.

The speed-up of the GP-GIMH approach is roughly $F/(2\rho F + G)$ where F is the time taken to run GIMH for Q iterations, $2\rho F$ is the time for L iterations of the MCWM pre-computation step where $\rho = L/Q$ and the value of 2 denotes the fact that two likelihood estimations are required at each iteration of MCWM, and G is the remaining time of the GP-GIMH method. In this paper we assume that F is very large; it is computationally demanding to estimate the likelihood. In such cases G may be negligible in comparison, in which case the speed-up is roughly $Q/(2L)$. Thus it is of interest to set L small. However, if L is set too small then G may become non-negligible since the GP may be too uncertain across much of the parameter space. Furthermore we note that L will naturally need to increase as the parameter dimension grows (although it may be of interest to increase Q as well). The value of G will also likely increase with the parameter dimension since it becomes increasingly difficult to train the GP in areas of non-negligible posterior support. In summary, the GP-GIMH approach does suffer from the curse of dimensionality. Nonetheless, in Section 3, we demonstrate that it is possible to achieve significant speed-ups on non-trivial models of moderate dimension. It is important to note that the above discussion only considers the computational gains. We generally find also with GP-GIMH that sticky periods can be mitigated, adding to the overall efficiency gains of GP-GIMH.

Our approach uses the MCWM algorithm to train the GP whereas Wilkinson (2014) uses a history matching approach, which iteratively fits a GP to samples drawn from a hypercube, which shrinks upon each successive GP fit. This process may be expensive in moderate dimensions if uninformative priors are used and it does not exploit the correlation between parameters as our approach does. We compare the two training approaches on one of the examples later. Our remaining MCMC algorithm that uses GP predictions is similar to Wilkinson (2014). However, we include the novel aspects of obtaining additional likelihood estimates when required and a burn-in phase.

All computations involving GPs are facilitated by the *gpml* package (Rasmussen and Williams,

Algorithm 1 GP-GIMH algorithm.

```
Input: threshold $\epsilon$, burn-in $B$ and the number of MCMC iterations (iters)
Output: MCMC output $\boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^{\text{iters}}$
Perform an MCWM algorithm for $L$ iterations to help determine an initial training sample. See the text for more details. The training sample after these processes is denoted as $\mathcal{D}_T = \{\boldsymbol{\theta}_j, \hat{f}(\boldsymbol{\theta}_j)\}_{j=1}^J$
Estimate the hyperparameter $\boldsymbol{\xi} = (\boldsymbol{\beta}, \boldsymbol{\gamma}, \delta)$ of a GP using the training sample $\mathcal{D}_T$
Simulate $\phi^0 \sim \mathcal{LN}\{m^*(\boldsymbol{\theta}^0), s^*(\boldsymbol{\theta}^0)^2\}$ from the GP with hyperparameter $\hat{\boldsymbol{\xi}}$. $\boldsymbol{\theta}^0$ can be chosen based on the MCWM pilot run
for $i = 1$ to iters do
  Propose $\boldsymbol{\theta}^* \sim q(\cdot|\boldsymbol{\theta}^{i-1})$
  Simulate $\phi^* \sim \mathcal{LN}\{m^*(\boldsymbol{\theta}^*), s^*(\boldsymbol{\theta}^*)^2\}$ from the GP with hyperparameter $\hat{\boldsymbol{\xi}}$
  Compute $\alpha = \min\left\{1, \frac{\phi^* p(\boldsymbol{\theta}^*) q(\boldsymbol{\theta}^{i-1}|\boldsymbol{\theta}^*)}{\phi^{i-1} p(\boldsymbol{\theta}^{i-1}) q(\boldsymbol{\theta}^*|\boldsymbol{\theta}^{i-1})}\right\}$
  Draw $u \sim \mathcal{U}(0, 1)$
  if $u < \alpha$ then
    if $s^*(\boldsymbol{\theta}^*) \leq \epsilon$ then
      Set $\phi^i = \phi^*$ and $\boldsymbol{\theta}^i = \boldsymbol{\theta}^*$
    else
      Obtain $A$ batches of $\lceil K/A \rceil$ independent likelihood estimates and update $m^*(\boldsymbol{\theta}^*)$ and $s^*(\boldsymbol{\theta}^*)$ using (4)
      If $i \leq B$ then append the likelihood estimates to the training sample $\mathcal{D}_T$.
      Simulate $\phi^* \sim \mathcal{LN}\{m^*(\boldsymbol{\theta}^*), s^*(\boldsymbol{\theta}^*)^2\}$ from the GP with hyperparameter $\hat{\boldsymbol{\xi}}$
      Compute $\alpha = \min\left\{1, \frac{\phi^* p(\boldsymbol{\theta}^*) q(\boldsymbol{\theta}^{i-1}|\boldsymbol{\theta}^*)}{\phi^{i-1} p(\boldsymbol{\theta}^{i-1}) q(\boldsymbol{\theta}^*|\boldsymbol{\theta}^{i-1})}\right\}$
      if $u < \alpha$ then
        Set $\phi^i = \phi^*$ and $\boldsymbol{\theta}^i = \boldsymbol{\theta}^*$
      else
        Set $\phi^i = \phi^{i-1}$ and $\boldsymbol{\theta}^i = \boldsymbol{\theta}^{i-1}$
      end if
    end if
  else
    Set $\phi^i = \phi^{i-1}$ and $\boldsymbol{\theta}^i = \boldsymbol{\theta}^{i-1}$
  end if
end for
```

2006) in Matlab, particularly the functions *minimize* and *gp*. The function *minimize* runs the optimisation process to estimate the hyperparameter. The function *gp* returns the mean and variance of the GP prediction with and without the nugget. However, we note that in (2) and (3), information regarding the expensive matrix inversion can be pre-stored as the training sample is static for all iterations in GP-GIMH after the burn-in phase. Further, the *gp* function contains extensive error checking and additional unnecessary computations for our purposes. Thus we implement our own function, based on *gp*, to obtain the mean and variance of the GP prediction. We find that GP-GIMH based on our GP prediction code runs roughly two to three times faster compared to the version that uses *gp*.

2.4 Related Literature

GPs have been used for the emulation of complex deterministic models by Kennedy and O'Hagan (2000, 2001) and for complex stochastic models by Henderson et al. (2009, 2010) and Baggaley et al. (2012). Conrad et al. (2016) use local polynomials or GPs in a Metropolis-Hastings algorithm to reduce the number of model evaluations that are required. However, the focus of Conrad et al. (2016) is not on applications where a stochastic likelihood estimator is available.

Tran et al. (2017) develop a variational Bayes approach that can be used in any application where an unbiased likelihood estimator is available to approximate the posterior more efficiently compared to GIMH. However, the variational approximation is typically of a parametric form and assumptions are sometimes made that the parameters are independent *a posteriori*.

Rasmussen (2003) use GPs to accelerate the Hamiltonian Monte Carlo (HMC) method for Bayesian inference when posterior density evaluation is expensive. The proposal distribution in HMC involves (approximately) solving a Hamiltonian system, and requires several evaluations of the posterior distribution. The GP is trained from a pilot HMC run and used to approximate posterior evaluations required in the HMC proposal. This method remains exact as the GP is used only for the proposal and a Metropolis-Hastings (MH) correction is applied to account for the fact that the Hamiltonian is only solved approximately. The MH correction step requires an evaluation of the (expensive) exact posterior density.

A related literature is on the delayed-acceptance MCMC method (Christen and Fox, 2005). Here each proposed parameter goes through an initial Metropolis-Hastings step where the actual posterior density is replaced by a computationally cheap posterior approximation. The idea of the method is that 'poor' proposals can be rejected quickly and the majority of 'promising' proposals make it through to the next Metropolis-Hastings stage, which involves evaluations of the exact posterior density. The second Metropolis-Hastings step is constructed so that the Markov chain has the correct limiting distribution. As an example, Golightly et al. (2015) perform Bayesian inference for Markov jump processes using the corresponding linear noise approximation in the screening Metropolis-Hastings step. Sherlock et al. (2017) consider a more general approach and use a k nearest neighbour surrogate within a delayed-acceptance MCMC algorithm when the likelihood is expensive or when the likelihood is estimated unbiasedly. Although the delayed-acceptance MCMC approach is exact, Golightly et al. (2015) and Sherlock et al. (2017) report efficiency gains of generally less than an order of magnitude. Our motivation here is to obtain a significant speed-up in very complex applications whilst accepting some approximation to the posterior.

3 Examples

In the examples below we consider models that can be placed in the state space formulation. For likelihood estimation we use the bootstrap particle filter (Gordon et al., 1993). The particle filter is implemented in C whilst the rest of the code is written in Matlab.

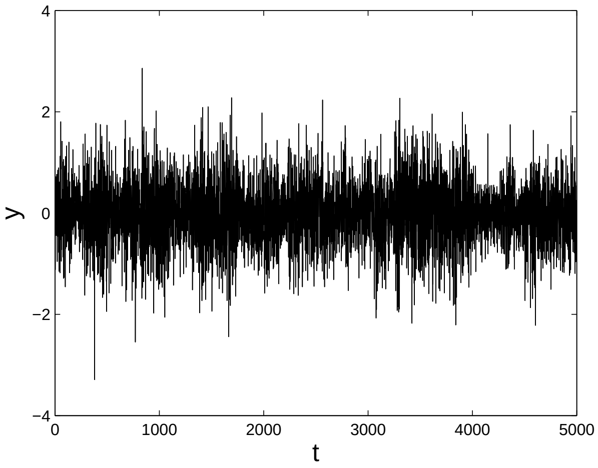


Figure 1: Data simulated from the stochastic volatility model of Section 3.1.1.

3.1 Stochastic Volatility Example

3.1.1 Model and Data

We illustrate our method by analysing data simulated from a stochastic volatility model in Chopin et al. (2013); this paper (and some of the references therein) give more details of the model construction and its interpretation. The observation at time t is scalar and has distribution $y_t \sim \mathcal{N}(\mu+\beta v_t, v_t)$, where v_t denotes the evolving and unobserved variance of the observation process. Here μ and β are static parameters. The evolution of the state process follows $\lambda v_t = z_{t-1} - z_t + \sum_{j=1}^{k} f_j$, where $k \sim \mathcal{P}o(\lambda\xi^2/w^2)$, $c_{1:k} \overset{\text{iid}}{\sim} \mathcal{U}(t-1, t)$, $f_{1:k} \overset{\text{iid}}{\sim} \mathcal{E}xp(\xi/w^2)$, and $z_t = e^{-\lambda} z_{t-1} + \sum_{j=1}^{k} e^{-\lambda(t-c_j)} f_j$, and $\mathcal{P}o$, $\mathcal{U}$ and $\mathcal{E}xp$ denote the Poisson, uniform and exponential distributions respectively. Here w^2, ξ and λ are additional static parameters so that the full parameter set is $\boldsymbol{\theta} = (w^2, \mu, \xi, \beta, \lambda)$. We assume our prior has independent components with $w^2, \xi \sim \mathcal{E}xp(5)$, $\mu, \beta \sim \mathcal{N}(0, 2)$ and $\lambda \sim \mathcal{E}xp(1)$. Also we initialise the z-process using $z_0 \sim \mathcal{G}a(\xi^2/w^2, \xi/w^2)$, that is, a gamma distribution with mean ξ and standard deviation w.

The observed data for this example has been generated from the model using the same parameter values as in Chopin et al. (2013), namely $w^2 = 0.0625$, $\mu = 0$, $\xi = 0.25$, $\beta = 0$ and $\lambda = 0.01$. However, our data are a longer series with 5000 observations (as opposed to their 1000 observations). The length of our time series creates a challenging problem for PMCMC algorithms. The data are shown in Figure 1.

3.1.2 Implementation Details

Unbiased likelihood estimates are determined by using the bootstrap particle filter of Gordon et al. (1993) with $N = 800$ particles. This value of N produces an estimated log-likelihood with a standard deviation of roughly 2.5 at the true parameter value. For the GIMH method, we also consider $N = 500$ and $N = 1000$ so that an essentially un-optimised GP-GIMH approach is compared to a close-to-optimal GIMH run. To further improve the performance of GIMH, we use the multiple-core PMCMC approach of Drovandi (2014) which uses multiple cores (here $A = 16$ cores) to obtain independent likelihood estimates for each proposed parameter value and takes the average in order to reduce the variance of the estimated likelihood. We run 100K iterations of GIMH. We also consider MCWM with $N = 800$ but only run it for 50K iterations as this algorithm requires two likelihood estimates at each iteration. We use the 16 cores in a similar way to improve the accuracy of MCWM. To use as a gold standard for comparison purposes, we run the GIMH algorithm for 400K iterations (with $N = 800$).

For the MCWM pre-computation step of GP-GIMH we use $L = 1500$, producing ≈ 3000 likelihood estimates. During this step, we adapt the covariance matrix of the multivariate normal random walk proposal every 10th iteration, following an initial 50 iterations. Recall that this MCWM output is only used to train the GP. We do not take advantage of any parallel computing in the MCWM pre-computation phase to illustrate that GP-GIMH can be effective even when the likelihood is not estimated as precisely as in our implementations of the standard GIMH and MCWM methods. The maximum log-posterior estimate encountered during the MCWM pre-computation phase is roughly -4610. We discard any log-posterior estimates below -4700. The main motivation for selecting this value is to eliminate two extreme log-likelihood estimates (-8926 and -6362) from the training. There are 18 other log-likelihood estimates that are also below this chosen threshold. To investigate the effect of the training sample size, J, we use all samples obtained from the MCWM pre-computing step and also thin the output by a factor of 2, producing (roughly) $J = 1500$ and $J = 3000$. For comparison purposes the remaining part of GP-GIMH (again 100K iterations) uses the same multivariate normal random walk as GIMH/MCWM. No burn-in phase is used for GP-GIMH (i.e. no additional log-likelihood estimates are added to the training sample). We consider ϵ values of 1, 1.5 and 2. When the tolerance condition is not met, we obtain $\lceil K/A \rceil$ batches of $A = 16$ independent likelihood estimates. We also make the 16 cores available in the remaining part of the GP-GIMH algorithm as Matlab automatically parallelises some of the computations. We run the full GP-GIMH procedure independently five times for each combination of J and ϵ.

To simplify comparisons between methods, we assume that a suitable starting value and multivariate normal random walk covariance matrix have already been determined, which are used for each algorithm (except the MCWM pre-computing phase, which adaptively determines a covariance matrix). Here we use the true parameter value to initialise the chain (unless otherwise stated) and an updating matrix obtained from some pilot runs. We note that, in reality, it is likely that the GP-GIMH algorithm would be faster at determining a suitable covariance matrix given that pilot runs can be performed quickly. This could also be a strong motivation for using a GP-GIMH algorithm if exact inferences are necessary (up to Monte Carlo error). We now explore the accuracy of the GP-GIMH algorithm in determining the posterior distribution.

3.1.3 Results

We find the marginal posterior estimates of GP-GIMH are insensitive to the choices of ϵ (see Appendix B). Very occasionally there is a distortion in the posterior tail when $\epsilon = 2.0$. Given the insensitivity to ϵ, in Figures 2 ($J \approx 1500$) and 3 ($J \approx 3000$) we only present

Table 1: Upper triangular part of the posterior correlation matrix for $\boldsymbol{\theta}$ obtained from the gold standard GIMH/GP-GIMH (first run with $J \approx 1500$ and $\epsilon = 1.0$) algorithms for the stochastic volatility model.

	μ	ξ	β	λ
w^2	-0.91/-0.90	-0.03/0.00	-0.05/-0.02	0.03/0.01
μ		0.02/0.02	0.05/0.01	-0.03/0.00
ξ			0.63/0.68	-0.07/-0.06
β				-0.40/-0.33

results for $\epsilon = 1$. Shown in the figures are results from the 5 independent runs of GP-GIMH, as well as the results produced by GIMH (gold standard run) and MCWM. One possible metric to assess the accuracy of the results from the different approximations is the total variation (TV) distance between the posterior estimates (e.g. kernel density estimates from the posterior samples, see Appendix K for more details) obtained from an approximation and the gold standard run. It is important to note that even the GIMH run suffers from Monte Carlo variability. Tables 1 and 2 in Appendix C show the TV distances between the univariate and bivariate posteriors, respectively, for the results obtained from GIMH ($N = 800$), MCWM and GP-GIMH (results averaged over 5 runs) with respect to the gold standard GIMH run. Due to the computational expense of computing the TV over two dimensions we only consider GP-GIMH results for $\epsilon = 1.0$ and $J \approx 3000$. From the tables it is clear that, unsurprisingly, GIMH is the most accurate. However, GP-GIMH appears to be more accurate than MCWM, except for any marginal or bivariate posterior that involves the parameter λ. Figures 2 and 3 show a clear bias in the estimated posterior from GP-GIMH for λ. Further, it appears that the GP-GIMH algorithm has more difficulty approximating the posterior distributions that show some skewness.

One drawback of GP-GIMH is there is some between-run variability, which is more apparent for the posterior distributions that deviate away from symmetry. It is likely that this variability comes from different GP fits resulting from different MCWM pre-computing runs. From Figure 3 it appears that the between-run variability is reduced slightly when using the larger training sample size, $J \approx 3000$. We attempt to validate this numerically. Here we consider the TV distances between the marginal densities produced from an individual run of GP-GIMH and the average marginal densities over the five runs of GP-GIMH. We then average these five distances to produce a single measure of the between-run variability. The table for different combinations of ϵ and J are shown in Appendix C. It is indeed evident that the between-run variability is generally reduced for larger J. Further, when $J \approx 1500$, we see an increase in the between-run TV distances for the parameters ξ, β and λ when ϵ is increased to 2.0. This is due to the distortion in the tails of these posteriors that is occasionally present when $\epsilon = 2.0$. When $J \approx 3000$, we see an increase in the between-run TV for λ when ϵ is increased for the same reason.

Table 1 shows the upper triangular part of the estimated posterior correlation matrix from a run of GIMH and one of the runs of GP-GIMH. It is evident that GP-GIMH is able to produce reasonable estimates of the posterior correlation matrix in this example, confirming that the algorithm could be useful for determining a suitable proposal distribution for GIMH.

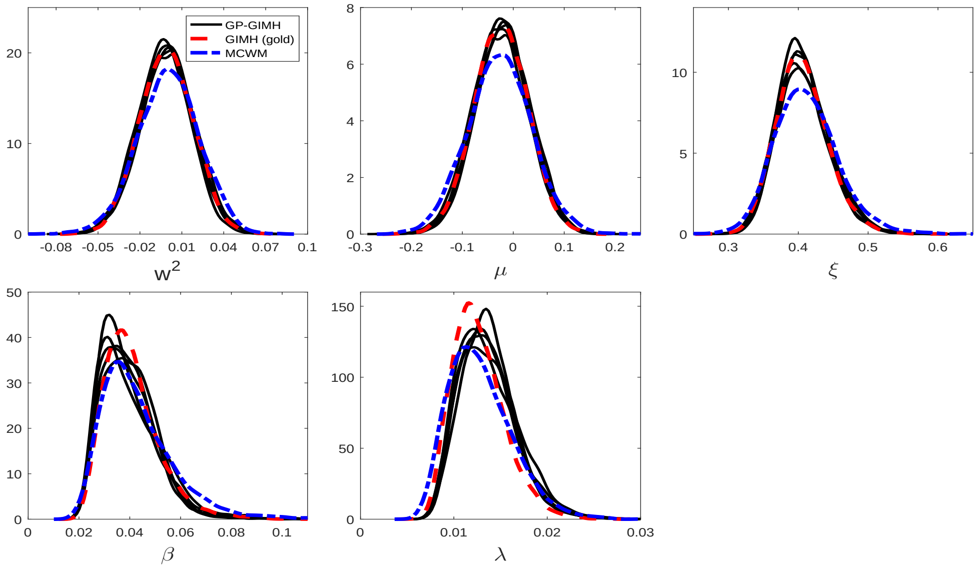


Figure 2: Estimated marginal posterior densities for the stochastic volatility example from runs of the gold standard GIMH (red dash) and MCWM (blue dash-dot) algorithms and five independent runs of the GP-GIMH (black solid) algorithm with $J \approx 1500$ and $\epsilon = 1$.

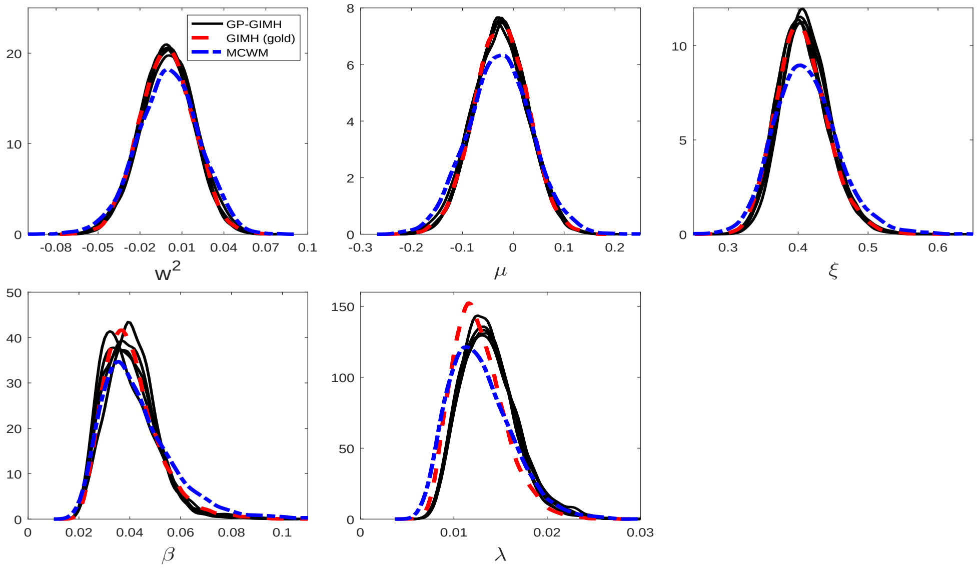


Figure 3: Estimated marginal posterior densities for the stochastic volatility example from runs of the gold standard GIMH (red dash) and MCWM (blue dash-dot) algorithms and five independent runs of the GP-GIMH (black solid) algorithm with $J \approx 3000$ and $\epsilon = 1$.

Table 2: Average run times for GP-GIMH (excluding the MCWM pre-computing phase and GP fitting) for different combinations of J and ϵ for the stochastic volatility example. The average time for GP training and fitting is 2.1 hours.

Configuration	avg run time (h)
$\epsilon = 1.0$, $J \approx 1500$	0.6
$\epsilon = 1.5$, $J \approx 1500$	0.6
$\epsilon = 2.0$, $J \approx 1500$	0.7
$\epsilon = 1.0$, $J \approx 3000$	1.1
$\epsilon = 1.5$, $J \approx 3000$	1.1
$\epsilon = 2.0$, $J \approx 3000$	1.2

Table 2 shows the run times (averaged over the 5 independent runs) for GP-GIMH (excluding the MCWM pre-computing phase and GP fitting) for different combinations of J and ϵ. There is little sensitivity to the run times with respect to ϵ. Intuitively, the run time should decrease with an increase in ϵ, however only a very small proportion of iterations had $s^*(\boldsymbol{\theta}^*) > \epsilon$ even when $\epsilon = 1$. The increase in computing time for larger J can be explained by the additional computation involved in generating the GP prediction. There is actually a complex interaction between J and ϵ with respect to the computing time, as we highlight in the next example.

Table 3 compares the statistical and computational efficiency of the three algorithms. The results for GP-GIMH are based on average results over the five runs. The effective sample size (ESS) for each parameter in each parameter set is calculated from the output of each algorithm using the *coda* package in R (Plummer et al., 2006). An overall summary of the algorithm output is taken as the minimum ESS value or the average ESS value over all parameters in the output. Shown also is the clock time in hours. For GP-GIMH, the time represents the cumulative time for the pre-computation step, GP fitting (allocated 10 minutes for $J \approx 1500$ and 20 minutes for $J \approx 3000$) and the remaining MCMC algorithm. For a final measure of performance we look at the ESS measures divided by the computing time. Here the GIMH method is slightly more efficient than MCWM. The GIMH method is able to better take advantage of the 16 cores, which results in an increase in acceptance rate from 6% (single core) to 19% (16 cores), whereas the use of additional cores does not greatly affect the acceptance rate for MCWM (although it does improve the posterior accuracy). The GP-GIMH algorithm eclipses both of the other two in terms of statistical efficiency. This is due to the increased acceptance rate of GP-GIMH relative to GIMH and the fact that GP-GIMH is run for double the number of iterations compared with MCWM. The GP-GIMH method has a much higher acceptance rate than GIMH as the log-likelihood estimates generated from the fitted GP generally have much less noise relative to the log-likelihood estimates obtained from the bootstrap filter. The total computing time of the GP-GIMH algorithm is considerably lower than the other two. Improved performance on both the ESS and computing time leads to an overall performance gain of one to two orders of magnitude for GP-GIMH over GIMH and MCWM. We also run the GP-GIMH method with 5 different Markov chain starting values (obtained from the MCWM pre-computation step) and obtain similar efficiency results (based on run 1 with $J \approx 3000$ and $\epsilon \in \{1.0, 2.0\}$).

Finally we consider whether or not the quality of the GP fit has an impact on the approxima-

Table 3: Comparison of the computational and statistical efficiency of the GIMH, GP-GIMH and MCWM algorithms for the stochastic volatility model. For GP-GIMH the results are averaged over the five independent runs for each combination of J and ϵ.

method	acc rate (%)	min ESS	avg ESS	time (hrs)	min ESS/time	avg ESS/time
GIMH ($N = 500$)	11	541	955	32	17	30
GIMH ($N = 800$)	19	815	1899	77	11	25
GIMH ($N = 1000$)	22	696	1737	81	9	21
MCWM	34	653	1385	77	8	18
$J \approx 1500, \epsilon = 1.0$	30	1457	3399	2.7	559	1305
$J \approx 1500, \epsilon = 1.5$	29	1448	3074	2.6	565	1207
$J \approx 1500, \epsilon = 2.0$	29	1324	2645	2.8	501	1012
$J \approx 3000, \epsilon = 1.0$	30	1501	3651	3.2	471	1146
$J \approx 3000, \epsilon = 1.5$	30	1574	3564	3.2	500	1135
$J \approx 3000, \epsilon = 2.0$	30	1434	3308	3.3	441	1023

tion error we observe for GP-GIMH. We do this by comparing the GP prediction with other log-likelihood estimates at training points not used to determine the GP fit. Specifically we use the fitted GP from the first run of the MCWM pre-computation step and the training points (and log-likelihood estimates) from the other four MCWM pre-computation runs. The fit is assessed using a standardised residual

$$r_i = \frac{\hat{f}(\boldsymbol{\theta}_i) - m^*(\boldsymbol{\theta}_i)}{\sqrt{s^*(\boldsymbol{\theta}_i)^2 + \hat{\delta}}}.$$

Note that the nugget is included in the GP variance term as the comparisons are made with noisy log-likelihood estimates. In Appendix D, we show normal quantile-quantile plots of the standardised residuals and plots of these residuals against parameter value components. Note that we only include residuals at training points with an accurate GP prediction (standard deviation below ϵ) as it is only at these points that the GP prediction alone is used. Appendix D shows the residual plots for different combinations of ϵ and J. It is evident that the residuals depart further from normality with an increase in ϵ. However, from the sensitivity results in Appendix B, it appears that the GP-GIMH method is somewhat robust to the lack of normality in this example. In some cases there is a small amount of curvature in the residuals when plotted against the training samples of each parameter component.

3.2 Gene Network Example

3.2.1 Model and Data

Golightly and Wilkinson (2005) consider a Markov jump process for an autoregulatory gene network consisting of four species DNA, RNA, P and P_2 and explain its relevance and appli-

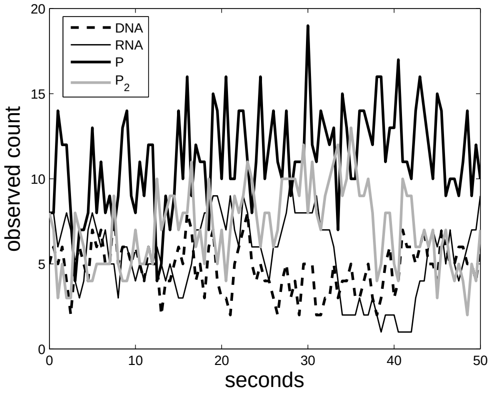


Figure 4: Data simulated from the gene network model of Section 3.2.1.

cation. The system is described by eight reactions

$$\begin{array}{rlrl} \mathrm{DNA} + \mathrm{P}_2 & \xrightarrow{c_1 \mathrm{DNA} \times \mathrm{P}_2} \mathrm{DNA} \cdot \mathrm{P}_2, & 2\mathrm{P} & \xrightarrow{c_5 \mathrm{P}(\mathrm{P}-1)/2} \mathrm{P}_2, \\ \mathrm{DNA} \cdot \mathrm{P}_2 & \xrightarrow{c_2 (k - \mathrm{DNA})} \mathrm{DNA} + \mathrm{P}_2, & \mathrm{P}_2 & \xrightarrow{c_6 \mathrm{P}_2} 2\mathrm{P}, \\ \mathrm{DNA} & \xrightarrow{c_3 \mathrm{DNA}} \mathrm{DNA} + \mathrm{RNA}, & \mathrm{RNA} & \xrightarrow{c_7 \mathrm{RNA}} \emptyset, \\ \mathrm{RNA} & \xrightarrow{c_4 \mathrm{RNA}} \mathrm{RNA} + \mathrm{P}, & \mathrm{P} & \xrightarrow{c_8 \mathrm{P}} \emptyset, \end{array}$$

where k is a conservation constant (number of copies of the gene) and $\boldsymbol{c} = (c_1, \ldots, c_8)$ are the stochastic rate constants governing the speed at which the system evolves. We study the scenario in Golightly and Wilkinson (2005) where data are simulated using rates $\boldsymbol{c} = (0.1, 0.7, 0.35, 0.2, 0.1, 0.9, 0.3, 0.1)$, with $k = 10$, and initial species levels $(\mathrm{DNA}, \mathrm{RNA}, \mathrm{P}, \mathrm{P}_2) = (5, 8, 8, 8)$. We simulate equi-spaced data as the next 100 observations (on all species) recorded at 0.5 unit time intervals (see Figure 4). We now investigate how our method performs in making inferences from these simulated data for the stochastic rate constants. Note that we assume that the conservation constant k and initial species levels are known as this is typical in designed experiments (and is as in Golightly and Wilkinson (2005)). Note that we assume these data are observed without error. Even though the observed counts are small, due to the large number of species, this model does not allow a computationally tractable likelihood function. As in Fearnhead et al. (2014), we take independent half-Cauchy priors for the parameters, with density $p(c_i) \propto 1/(1 + 4c_i^2)$, $c_i > 0$ for $i = 1, \ldots, 8$. Also, in our analysis, we remove the positivity constraint on the rate parameters by working on the log scale, that is, with $\theta_i = \log c_i$ for $i = 1, \ldots, 8$.

One approach to perform inference for such models is to assume that each species is observed with Gaussian error with a standard deviation of σ (Holenstein, 2009). This facilitates an analysis using a standard particle MCMC approach with a bootstrap filter. More accurate inferences are obtained with a low value of σ, with the correct posterior obtained in the limit as $\sigma \to 0$. However as σ decreases, more particles (N) are required to obtain an accurate likelihood estimate and this makes the procedure increasingly computationally demanding.

3.2.2 Implementation Details

Here we use $\sigma = 0.6$ and $N = 6000$ for the bootstrap particle filter. As mentioned earlier, the smaller the value of σ the closer the approximate posterior is to the true posterior with the drawback that a larger N is required to estimate the likelihood to a reasonable level of precision. At the true parameter value, the log-likelihood is estimated with a standard deviation of 1.9 when $N = 6000$. Similar to the previous example, pilot runs of GIMH are used to determine a suitable covariance matrix for a multivariate random walk proposal, which all methods use. For simplicity we start all chains at the true parameter value (unless otherwise stated). We run GIMH for 100K iterations and MCWM for 50K iterations and make available the $A = 16$ cores. At the true parameter value, the standard deviation of the log-likelihood estimate is roughly 0.75 when using the 16 cores. We also consider $N = 4000$, 5000 and 8000 for GIMH (70K iterations for $N = 8000$). For comparison purposes we consider a gold standard GIMH run of 350K iterations (with $N = 6000$).

For the GP-GIMH algorithm, we use a similar approach to the previous example. We first run the MCWM pre-computation step for $L = 2000$ iterations. During this step, after an initial 50 iterations, we adapt the covariance matrix of the multivariate normal random walk proposal every 10th iteration. Again we do not use the $A = 16$ cores in the MCWM pre-computing phase. The maximum log-posterior encountered during the MCWM pre-computation phase is roughly -708 and we do not discard any proposals (the minimum is -789). To investigate the effect of the training sample size, J, we compare the results of using a GP trained on all likelihood estimates from the MCWM pre-computing step with that of a GP trained on the same estimates but after thinning the parameter output by a factor of two, producing training data with (roughly) $J = 4000$ and $J = 2000$. The remaining part of GP-GIMH (again 100K iterations) uses the same multivariate normal random walk as GIMH/MCWM. No burn-in phase is used for GP-GIMH (in Section 3.2.4 we consider the impact of using a burn-in phase). We consider ϵ values of 1.2, 1.5 and 2 and make the 16 cores available. When the tolerance condition is not met, we obtain $\lceil K/A \rceil$ batches of $A = 16$ independent likelihood estimates. We run the full GP-GIMH procedure independently five times for each combination of J and ϵ.

Code to implement our method for this example can be found at `http://www.runmycode.org/companion/view/2663`.

3.2.3 Results

Before discussing posterior approximations, we compare our MCWM training procedure with the history matching approach of Wilkinson (2014). The full details are provided in Appendix E. We find that with less training time our trained GP is able to predict much more accurately than the history matching trained GP for samples from the posterior distribution generated from the exact GIMH method.

As before we find that the GP-GIMH results are generally insensitive to ϵ, though we occasionally observe incorrect tail behaviour when $\epsilon = 2$; see the plots in Appendix F. The marginal posterior density estimates for the different approaches are shown in Figures 5 (with $J \approx 2000$ for GP-GIMH) and 6 (with $J \approx 4000$ for GP-GIMH). Both GP-GIMH implemen-

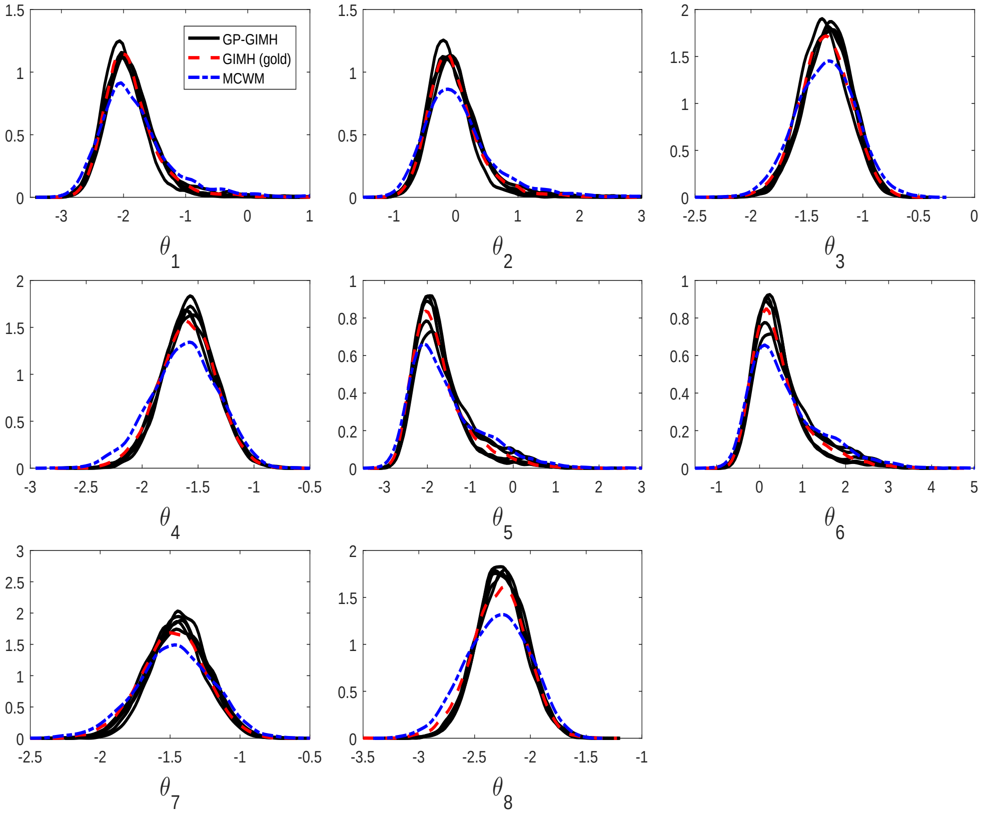


Figure 5: Estimated marginal posterior densities for the gene network example from runs of the gold standard GIMH (red dash) and MCWM (blue dash-dot) algorithms and five independent runs of the GP-GIMH (black solid) algorithm with $J \approx 2000$ and $\epsilon = 1.2$. The results for GIMH and MCWM are based on $N = 6000$.

tations use a tolerance of $\epsilon = 1.2$. We also show the univariate and bivariate TV distances between the different approximations and the gold standard run in Appendix G. In general it is evident that the GP-GIMH method is producing results not quite as accurate as GIMH but more accurate than MCWM for this example. From the between-run TV distances shown in Appendix G, it is evident that the between-run variability is reduced when increasing J. Again, for some of the parameters, we see an increase in the between-run TV distances when $\epsilon = 2.0$.

From Table 4 it is evident that GP-GIMH is capturing the true posterior correlation matrix quite accurately. In the GIMH run, there is a period where the Markov chain does not move for roughly 1700 iterations, which highlights the difficulty encountered with the GIMH approach. With the GP-GIMH method, as long as ϵ is set low enough, we do not observe such sticking behaviour. Trace plots from the different approaches are shown in Appendix H.

Table 5 shows the run times (averaged over the five independent runs) for GP-GIMH for different combinations of J and ϵ. Note that these times exclude the MCWM pre-computing phase and GP fitting. Again these times appear to be fairly insensitive to the tolerance level ϵ. It is interesting to note that there is only a small increase in run times for $J \approx 4000$. The

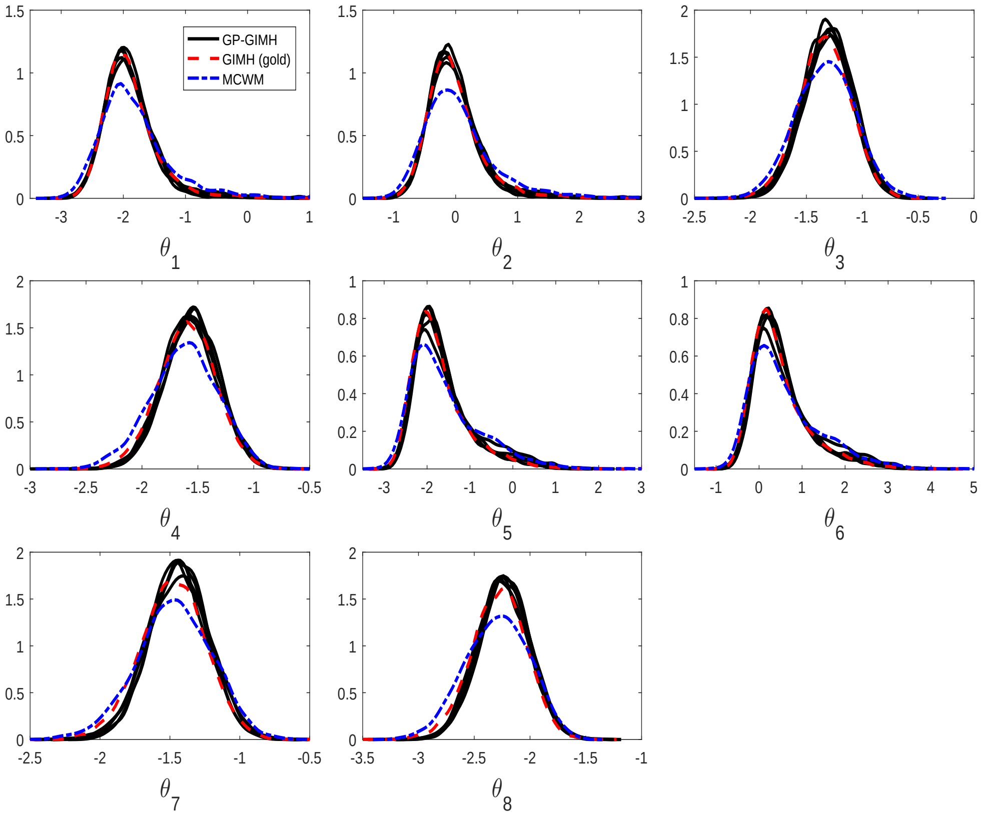


Figure 6: Estimated marginal posterior densities for the gene network example from runs of the gold standard GIMH (red dash) and MCWM (blue dash-dot) algorithms and five independent runs of the GP-GIMH (black solid) algorithm with $J \approx 4000$ and $\epsilon = 1.2$.

Table 4: Upper triangular part of the posterior correlation matrix for $\boldsymbol{\theta}$ obtained from the gold standard GIMH/GP-GIMH (first run with $J \approx 2000$ and $\epsilon = 1.2$) algorithms for the gene network model.

	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_8
θ_1	0.98/0.97	0.03/0.06	0.00/-0.02	-0.01/-0.04	0.00/-0.04	0.03/0.06	-0.01/0.01
θ_2		0.03/0.05	0.00/-0.01	-0.01/-0.04	-0.01/-0.04	0.04/0.05	-0.01/0.02
θ_3			-0.01/0.02	-0.01/-0.07	-0.01/-0.07	0.70/0.71	-0.02/-0.01
θ_4				0.00/0.06	0.01/0.07	-0.01/-0.02	0.70/0.73
θ_5					0.99/0.99	0.00/-0.04	0.01/0.09
θ_6						0.00/-0.04	0.01/0.09
θ_7							-0.01/-0.04

Table 5: Average run times for GP-GIMH (excluding the MCWM pre-computing phase and GP fitting) for different combinations of J and ϵ for the gene network example. The average time for GP training and fitting is around 4 hours.

Configuration	avg run time (h)
$\epsilon = 1.2$, $J \approx 2000$	1.9
$\epsilon = 1.5$, $J \approx 2000$	1.9
$\epsilon = 2.0$, $J \approx 2000$	2.3
$\epsilon = 1.2$, $J \approx 4000$	2.9
$\epsilon = 1.5$, $J \approx 4000$	2.5
$\epsilon = 2.0$, $J \approx 4000$	2.6

Table 6: Comparison of the computational and statistical efficiency of the GIMH, GP-GIMH and MCWM algorithms for the gene network model. For GP-GIMH the results are averaged over the five independent runs for each combination of J and ϵ. † 70K iterations are used for GIMH when $N = 8000$.

method	acc rate (%)	min ESS	avg ESS	time (hrs)	min ESS/time	avg ESS/time
GIMH ($N = 4000$)	7	517	700	58	9	12
GIMH ($N = 5000$)	9	689	902	78	9	12
GIMH ($N = 6000$)	10	537	984	108	5	9
GIMH ($N = 8000$)†	12	684	944	123	6	8
MCWM	19	445	769	121	4	6
$J \approx 2000, \epsilon = 1.2$	13	956	1452	5.9	176	260
$J \approx 2000, \epsilon = 1.5$	13	868	1389	5.9	149	240
$J \approx 2000, \epsilon = 2.0$	13	495	1190	6.3	81	194
$J \approx 4000, \epsilon = 1.2$	13	892	1419	7.1	132	206
$J \approx 4000, \epsilon = 1.5$	13	971	1476	6.7	151	226
$J \approx 4000, \epsilon = 2.0$	13	716	1354	6.8	110	204

increase in time for GP prediction is offset by the fact that the GP is trained at more points so that unreliable GP predictions (with $s^*(\boldsymbol{\theta}^*) > \epsilon$) occur less often. Furthermore, there is lower variability in the run times with $J \approx 4000$: in the 15 runs of GP-GIMH with $J \approx 4000$ (five independent runs for each of three different ϵ values) the standard deviation of the run time is 0.9 hours whereas the corresponding number for $J \approx 2000$ is 1.3 hours.

The computational, statistical and overall efficiency of the different approaches is shown in Table 6. Again it is clear that the GP-GIMH algorithm offers generally at least an order of magnitude improvement in overall efficiency, resulting from the large reduction in computing time and a slightly higher acceptance rate than GIMH. The overall efficiency for GP-GIMH depends little on J in this example. There is perhaps a slight decrease in overall efficiency as ϵ increases, especially for $\epsilon = 2$. We also run the GP-GIMH method with 5 different Markov chain starting values (obtained from the MCWM pre-computation step) and obtain similar efficiency results (based on run 1 with $J \approx 4000$ and $\epsilon \in \{1.2, 2.0\}$).

The GP residual plots are shown in Appendix I. As with the previous example, the normality assumption of the residuals is increasingly violated with an increase in ϵ. Again there is a small amount of curvature in some of the residual plots, however generally the GP appears to fit reasonably well.

3.2.4 Increasing Accuracy with GP-GIMH

Finally, we attempt to use the increased efficiency of the GP-GIMH approach to target a smaller value of σ. The idea is to obtain an approximate solution to a more accurate posterior (in the sense that it is closer to the true posterior) rather than an exact solution to a less accurate posterior. Here we set $\sigma = 0.4$ and use $N = 10000$ for the MCWM pre-computing step. Given the challenging nature of this problem, we use the 16 cores in the MCWM pre-computing step to obtain a more accurate log-likelihood estimate at each iteration of the MCWM phase by taking the average of 16 independent likelihood estimates. Using the 16 cores, the log-likelihood estimate has a standard deviation of roughly 2.2 at the true parameter value. The MCWM pre-computing phase is run for 2000 iterations with an adaptive MCMC strategy as detailed earlier. None of the training samples are discarded. We use $\epsilon = 1.2$ for the remaining GP-GIMH algorithm with a multivariate normal proposal estimated from a few pilot runs of the method. Once the proposal distribution is determined, GP-GIMH is run for 100K iterations.

Given that all 16 cores are already used to obtain a single log-likelihood estimate, additional likelihood estimates required when $s^*(\boldsymbol{\theta}^*) > \epsilon$ can only be obtained in batches of size 1, increasing the time needed to generate these additional likelihood estimates (compounded by the fact that we use the larger $N = 10000$ to accommodate the smaller σ). Given the complexity of this application, we examine the impact of including for the first time a burn-in phase. Here we use $B = 20000$.

Since the likelihood cannot be estimated accurately even with the 16 cores here, GIMH does not perform well with $N = 10000$ (acceptance rate of 6% over 10K iterations). Instead we trial GIMH using $N = 14000$ (with this choice the standard deviation of the log-likelihood estimate is roughly 1.6 at the true value).

For GP-GIMH, the MCWM phase takes 10 hours. The remaining part of the GP-GIMH algorithm takes 7.5 hours without the burn-in phase and 2.3 hours with the burn-in phase. The improvement in computing time afforded with the burn-in is however reduced in that 20K less iterations can be used for inference. Using the burn-in period would be even more beneficial if more iterations were performed. Appendix J shows that the posterior estimates obtained by GP-GIMH depend very little on whether a burn-in is used. The GIMH method takes roughly 470 hours to run only 100K iterations. Further, there is a dramatic reduction in acceptance rate, down from 35% for GP-GIMH to 12% for GIMH (note that this is even with smaller random walk standard deviations to improve the acceptance rate). This amounts to an efficiency improvement of roughly 100 times for GP-GIMH (with and without the burn-in).

Posterior distribution estimates are shown in Figure 7 for GP-GIMH ($\sigma = 0.4$ with burn-in phase) and GIMH (for both $\sigma = 0.6$ and $\sigma = 0.4$). It is important to note that the GIMH posterior estimates with $\sigma = 0.4$ are rough as they are based on a low ESS (average ESS over parameters of about 80). There is a marked shift in the posteriors towards the true values for θ_3 and θ_7 when σ is reduced to 0.4, which the GP-GIMH approach is able to capture

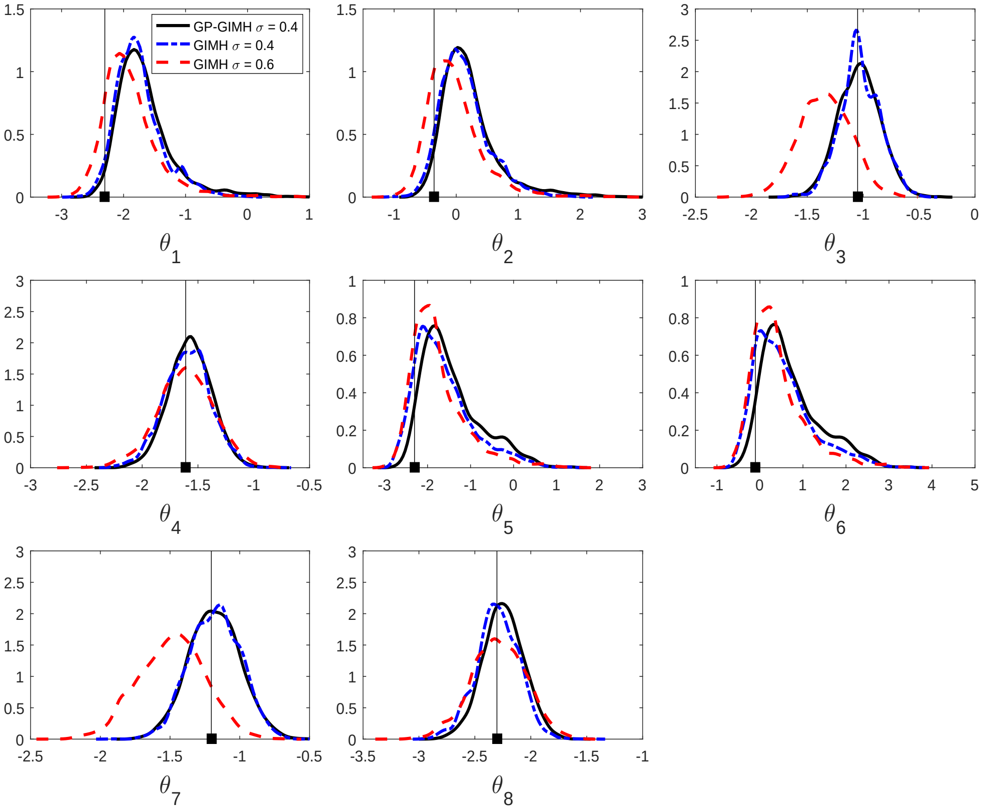


Figure 7: Estimated marginal posterior densities for the gene network example from GIMH with $\sigma = 0.6$ (red dash) and GP-GIMH with $\sigma = 0.4$ and a burn-in phase of $B = 20000$ iterations (black solid).

accurately. There is a gain in precision for θ_4 and θ_8, which GP-GIMH is also able to capture. The GP-GIMH approach is recovering the skewed posteriors (θ_1, θ_2, θ_7 and θ_8) with less accuracy, with the method appearing to spend too much time in the right tails. Overall, the GP-GIMH method is performing well here given the massive improvement in efficiency. The results of GIMH suggest that the posteriors are moving away from the true parameter values for θ_1 and θ_2 when σ is reduced. This might be explained by the fact that only a relatively small (partially observed) dataset is used here, and so the parameter values most favourable for the dataset generated may be away from the true parameter values.

4 Discussion

In this paper we have presented an approach to accelerate pseudo-marginal methods using GPs. We expect our method to be useful in applications with a relatively low number of parameters and where precise likelihood estimation is very expensive. For accurate posterior approximations, we also require that the underlying true log-likelihood surface can be well approximated with a GP. Thus we require a smooth log-likelihood function.

Other choices for the mean function, covariance function and observation model may be selected. Such selections are likely to be problem dependent, for example some noisy log-likelihood functions might be better modelled with an error variance that depends on $\boldsymbol{\theta}$, such as heteroscedastic GP models (Goldberg et al., 1998; Kersting et al., 2007). In practice one may use a model selection procedure (Rasmussen and Williams, 2006, chap. 5) to determine the most appropriate GP for a given training sample. Järvenpää et al. (2016) use a GP with input-dependent noise for ABC and discuss model selection in this context. However, a more sophisticated GP will result in more complex computations involving the GP. This requires further investigation.

In this article we have not investigated the optimal standard deviation of the log-likelihood estimator to use in the MCWM pre-computing phase. Such an investigation would require an extensive and significant simulation study. Here we found success when the standard deviation of the log-likelihood estimator is roughly 2 in the MCWM pre-computing phase. This is of interest as it is larger than that recommended for GIMH (Doucet et al., 2015), which suggests that our approach should be useful in complex scenarios.

Here we used a multivariate normal random walk proposal in the MCMC. Alternatively, the pre-computed approximation to the log-likelihood could also be used to design improved proposals for GP-GIMH. The covariance function of the GP could be viewed as a smoothed representation of the geometry of the parameter space. Where there is strong dependence between parameters, information such as gradient and curvature can be utilised to propose large moves with high acceptance probability. For example, Zhang et al. (2017) incorporate a pre-computation step in their approximate Hamiltonian Monte Carlo method (see also Rasmussen (2003)).

Our approach could also be used to accelerate approximate Bayesian inferences when an expensive biased likelihood estimator is used. Alquier et al. (2016) present a noisy MCMC framework that provides bounds on the error when a biased likelihood estimator is used in an MCMC method. Another example is the synthetic likelihood of Wood (2010) (see also Price et al. (2017)), which assumes a multivariate normal approximation for the likelihood of a summary statistic, with a mean and covariance matrix estimated by repeated model simulation.

A GP surrogate may be incorporated into delayed-acceptance MCMC to facilitate exact Bayesian inferences in the presence of models where only an unbiased likelihood estimator is available or those with expensive likelihood functions. There is scope here to adapt the GP during the MCMC algorithm by adding training samples when necessary and/or re-estimating hyperparameters.

If exact results are necessary, the output of our GP-GIMH approach may be used to form an importance distribution for importance sampling (IS) or sequential Monte Carlo (SMC). The IS and SMC approaches have been extended to allow for unbiased likelihood estimators to be used (see Chopin et al. (2013), Tran et al. (2014), Duan and Fulop (2015) and Drovandi and McCutchan (2016)). The IS and SMC methods are of additional interest as they produce also an estimate of the evidence, which can be used for fully Bayesian model comparisons; see, for example, Drovandi and McCutchan (2016) and Carson et al. (2017).

Acknowledgements

CCD was supported by an Australian Research Council's Discovery Early Career Researcher Award funding scheme (DE160100741). MTM was supported by the UK Engineering and Physical Sciences Research Council as part of a programme grant (EP/K014463/1). The authors are grateful to Andy Golightly for useful comments on an earlier draft. The authors are also grateful to an Associate Editor and two referees for their helpful suggestions that led to improvements in the presentation and content of this paper.

References

Alquier, P., Friel, N., Everitt, R., and Boland, A. (2016). Noisy Monte Carlo: Convergence of Markov chains with approximate transition kernels. Statistics and Computing, 26(1):29–47.

Andrieu, C., Doucet, A., and Holenstein, R. (2010). Particle Markov chain Monte Carlo methods. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 72(3):269–342.

Andrieu, C. and Roberts, G. O. (2009). The pseudo-marginal approach for efficient Monte Carlo computations. The Annals of Statistics, 37(2):697–725.

Baggaley, A. W., Boys, R. J., Golightly, A., Sarson, G. R., and Shukurov, A. (2012). Inference for population dynamics in the neolithic period. The Annals of Applied Statistics, 6(4):1352–1376.

Beaumont, M. A. (2003). Estimation of population growth or decline in genetically monitored populations. Genetics, 164(3):1139–1160.

Bérard, J., Del Moral, P., and Doucet, A. (2014). A lognormal central limit theorem for particle approximations of normalizing constants. Electronic Journal of Probability, 19(94):1–28.

Carson, J., Crucifix, M., Preston, S., and Wilkinson, R. D. (2017). Bayesian model selection for the glacial-interglacial cycle. To appear in Journal of the Royal Statistical Society: Series C (Applied Statistics).

Chopin, N., Jacob, P. E., and Papaspiliopoulos, O. (2013). SMC^2: an efficient algorithm for sequential analysis of state space models. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 75(3):397–426.

Christen, J. A. and Fox, C. (2005). Markov chain Monte Carlo using an approximation. Journal of Computational and Graphical Statistics, 14(4):795–810.

Conrad, P. R., Marzouk, Y. M., Pillai, N. S., and Smith, A. (2016). Accelerating asymptotically exact MCMC for computationally intensive models via local approximations. Journal of the American Statistical Association, 111(516):1591–1607.

Doucet, A., Pitt, M. K., Deligiannidis, G., and Kohn, R. (2015). Efficient implementation of Markov chain Monte Carlo when using an unbiased likelihood estimator. Biometrika, 102(2):295–313.

Drovandi, C. C. (2014). Pseudo-marginal algorithms with multiple CPUs. http://eprints.qut.edu.au/61505/.

Drovandi, C. C. and McCutchan, R. A. (2016). Alive SMC^2: Bayesian model selection for low-count time series models with intractable likelihoods. Biometrics, 72(2):344–353.

Duan, J.-C. and Fulop, A. (2015). Density-tempered marginalized sequential Monte Carlo samplers. Journal of Business & Economic Statistics, 33(2):192–202.

Fearnhead, P., Giagos, V., and Sherlock, C. (2014). Inference for reaction networks using the linear noise approximation. Biometrics, 70(2):457–466.

Goldberg, P. W., Williams, C. K. I., and Bishop, C. M. (1998). Regression with input-dependent noise: a Gaussian process treatment. In Advances in Neural Information Processing Systems (NIPS), volume 10, pages 493–499. The MIT Press.

Golightly, A., Henderson, D. A., and Sherlock, C. (2015). Delayed acceptance particle MCMC for exact inference in stochastic kinetic models. Statistics and Computing, 25(5):1039–1055.

Golightly, A. and Wilkinson, D. J. (2005). Bayesian inference for stochastic kinetic models using a diffusion approximation. Biometrics, 61(3):781–788.

Gordon, N. J., Salmond, D. J., and Smith, A. F. M. (1993). Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In Radar and Signal Processing, IEE Proceedings F, volume 140, pages 107–113.

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):4256–4302.

Henderson, D. A., Boys, R. J., Krishnan, K. J., Lawless, C., and Wilkinson, D. J. (2009). Bayesian emulation and calibration of a stochastic computer model of mitochondrial dna deletions in substantia nigra neurons. Journal of the American Statistical Association, 104(485):76–87.

Henderson, D. A., Boys, R. J., and Wilkinson, D. J. (2010). Bayesian calibration of a stochastic kinetic computer model using multiple data sources. Biometrics, 66(1):249–256.

Holenstein, R. (2009). Particle Markov Chain Monte Carlo. PhD thesis, The University Of British Columbia.

Järvenpää, M., Gutmann, M., Vehtari, A., and Marttinen, P. (2016). Gaussian process modeling in approximate Bayesian computation to estimate horizontal gene transfer in bacteria. arXiv:1610.06462 [stat.ML]. arXiv preprint.

Kennedy, M. C. and O'Hagan, A. (2000). Predicting the output from a complex computer code when fast approximations are available. Biometrika, 87(1):1–13.

Kennedy, M. C. and O'Hagan, A. (2001). Bayesian calibration of computer models. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 63(3):425–464.

Kersting, K., Plagemann, C., Pfaff, P., and Burgard, W. (2007). Most likely heteroscedastic Gaussian process regression. In Proceedings of the 24th International Conference on Machine Learning (ICML), volume 227 of ACM International Conference Proceeding Series, pages 393–400.

Medina-Aguayo, F. J., Lee, A., and Roberts, G. O. (2016). Stability of noisy Metropolis–Hastings. Statistics and Computing, 26(6):1187–1211.

Meeds, E. and Welling, M. (2014). GPS-ABC: Gaussian process surrogate approximate Bayesian computation. In Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence (UAI), pages 593–602, Quebec City, Canada.

Plummer, M., Best, N., Cowles, K., and Vines, K. (2006). CODA: Convergence diagnosis and output analysis for MCMC. R News, 6(1):7–11.

Price, L. F., Drovandi, C. C., Lee, A., and Nott, D. J. (2017). Bayesian synthetic likelihood. Journal of Computational and Graphical Statistics, doi: 10.1080/10618600.2017.1302882.

Rasmussen, C. E. (2003). Gaussian processes to speed up hybrid Monte Carlo for expensive Bayesian integrals. In Bernardo, J., Bayarri, M., Berger, J., Dawid, A., Heckerman, D., Smith, A., and West, M., editors, Bayesian Statistics, volume 7, pages 651–659.

Rasmussen, C. E. and Williams, C. K. I. (2006). Gaussian processes for machine learning. The MIT Press, Cambridge, Massachusetts.

Sherlock, C., Golightly, A., and Henderson, D. A. (2017). Adaptive, delayed-acceptance MCMC for targets with expensive likelihoods. Journal of Computational and Graphical Statistics, 26(2):434–444.

Tran, M.-N., Nott, D. J., and Kohn, R. (2017). Variational Bayes with intractable likelihood. To appear in Journal of Computational and Graphical Statistics.

Tran, M.-N., Scharth, M., Pitt, M. K., and Kohn, R. (2014). Importance sampling squared for Bayesian inference in latent variable models. Available at SSRN 2386371.

Wilkinson, R. (2014). Accelerating ABC methods using Gaussian processes. Journal of Machine Learning Research, 33:1015–1023.

Wood, S. N. (2010). Statistical inference for noisy nonlinear ecological dynamic systems. Nature, 466:1102–1107.

Zhang, C., Shahbaba, B., and Zhao, H. (2017). Precomputing strategy for Hamiltonian Monte Carlo method based on regularity in parameter space. Computational Statistics, 32(1):253–279.

Appendices for Accelerating Pseudo-Marginal MCMC using Gaussian Processes

C. C. Drovandi*, M. T. Moores† and R. J. Boys‡

*Australian Centre of Excellence for Mathematical and Statistical Frontiers and Queensland University of Technology, Brisbane, Australia 4001

† University of Warwick, Coventry CV4 7AL, United Kingdom

‡ Newcastle University, Newcastle upon Tyne NE1 7RU, United Kingdom

email: c.drovandi@qut.edu.au

August 28, 2017

A GIMH and MCWM algorithms

The pseudo-marginal algorithms discussed in the main text are shown in Algorithm 1 (GIMH) and Algorithm 2 (MCWM).

Algorithm 1 GIMH algorithm of Beaumont (2003).

```
Input: $\boldsymbol{\theta}^0$ and iters
Output: MCMC output $\boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^{\text{iters}}$
Compute $\phi^0 = \hat{p}(\boldsymbol{y}|\boldsymbol{\theta}^0)$
for $i = 1$ to iters do
    Propose $\boldsymbol{\theta}^* \sim q(\cdot|\boldsymbol{\theta}^{i-1})$
    Compute $\phi^* = \hat{p}(\boldsymbol{y}|\boldsymbol{\theta}^*)$
    Compute $\alpha = \min\left\{1, \frac{\phi^* p(\boldsymbol{\theta}^*) q(\boldsymbol{\theta}^{i-1}|\boldsymbol{\theta}^*)}{\phi^{i-1} p(\boldsymbol{\theta}^{i-1}) q(\boldsymbol{\theta}^*|\boldsymbol{\theta}^{i-1})}\right\}$
    Draw $u \sim \mathcal{U}(0,1)$
    if $u < \alpha$ then
        Set $\phi^i = \phi^*$ and $\boldsymbol{\theta}^i = \boldsymbol{\theta}^*$
    else
        Set $\phi^i = \phi^{i-1}$ and $\boldsymbol{\theta}^i = \boldsymbol{\theta}^{i-1}$
    end if
end for
```

Algorithm 2 MCWM algorithm of Beaumont (2003).

```
Input: $\boldsymbol{\theta}^0$ and iters
Output: MCMC output $\boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^{\text{iters}}$
Compute $\phi^0 = \hat{p}(\boldsymbol{y}|\boldsymbol{\theta}^0)$
for $i = 1$ to iters do
    Compute $\phi^{i-1} = \hat{p}(\boldsymbol{y}|\boldsymbol{\theta}^{i-1})$
    Propose $\boldsymbol{\theta}^* \sim q(\cdot|\boldsymbol{\theta}^{i-1})$
    Compute $\phi^* = \hat{p}(\boldsymbol{y}|\boldsymbol{\theta}^*)$
    Compute $\alpha = \min\left\{1, \frac{\phi^* p(\boldsymbol{\theta}^*) q(\boldsymbol{\theta}^{i-1}|\boldsymbol{\theta}^*)}{\phi^{i-1} p(\boldsymbol{\theta}^{i-1}) q(\boldsymbol{\theta}^*|\boldsymbol{\theta}^{i-1})}\right\}$
    Draw $u \sim \mathcal{U}(0,1)$
    if $u < \alpha$ then
        Set $\boldsymbol{\theta}^i = \boldsymbol{\theta}^*$
    else
        Set $\boldsymbol{\theta}^i = \boldsymbol{\theta}^{i-1}$
    end if
end for
```

B Sensitivity of GP-GIMH to ϵ for the stochastic volatility example

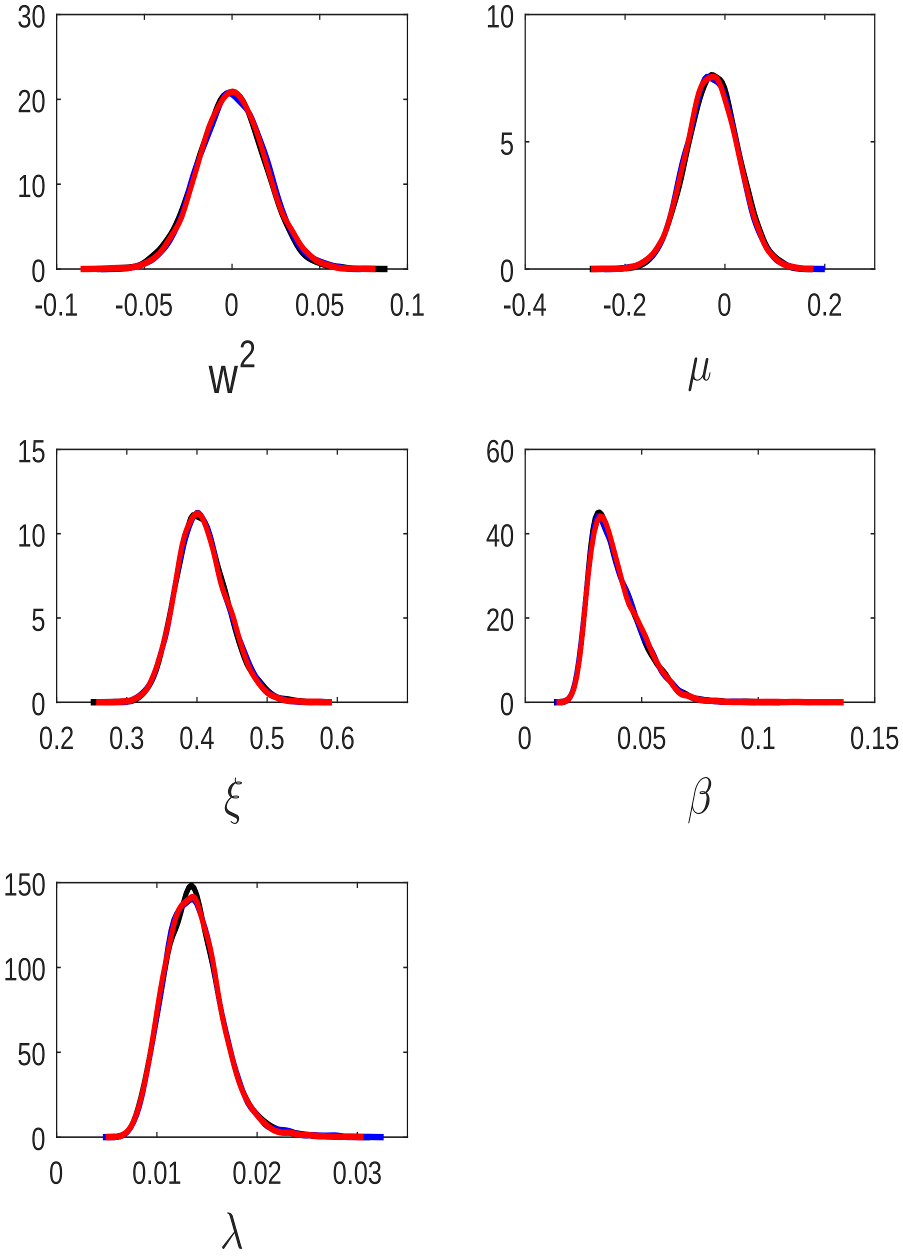


Figure 1: Sensitivity to ϵ for the stochastic volatility example for the first independent run when $J \approx 1500$. Posterior estimates are shown for $\epsilon = 1.0$ (black), $\epsilon = 1.5$ (blue) and $\epsilon = 2.0$ (red).

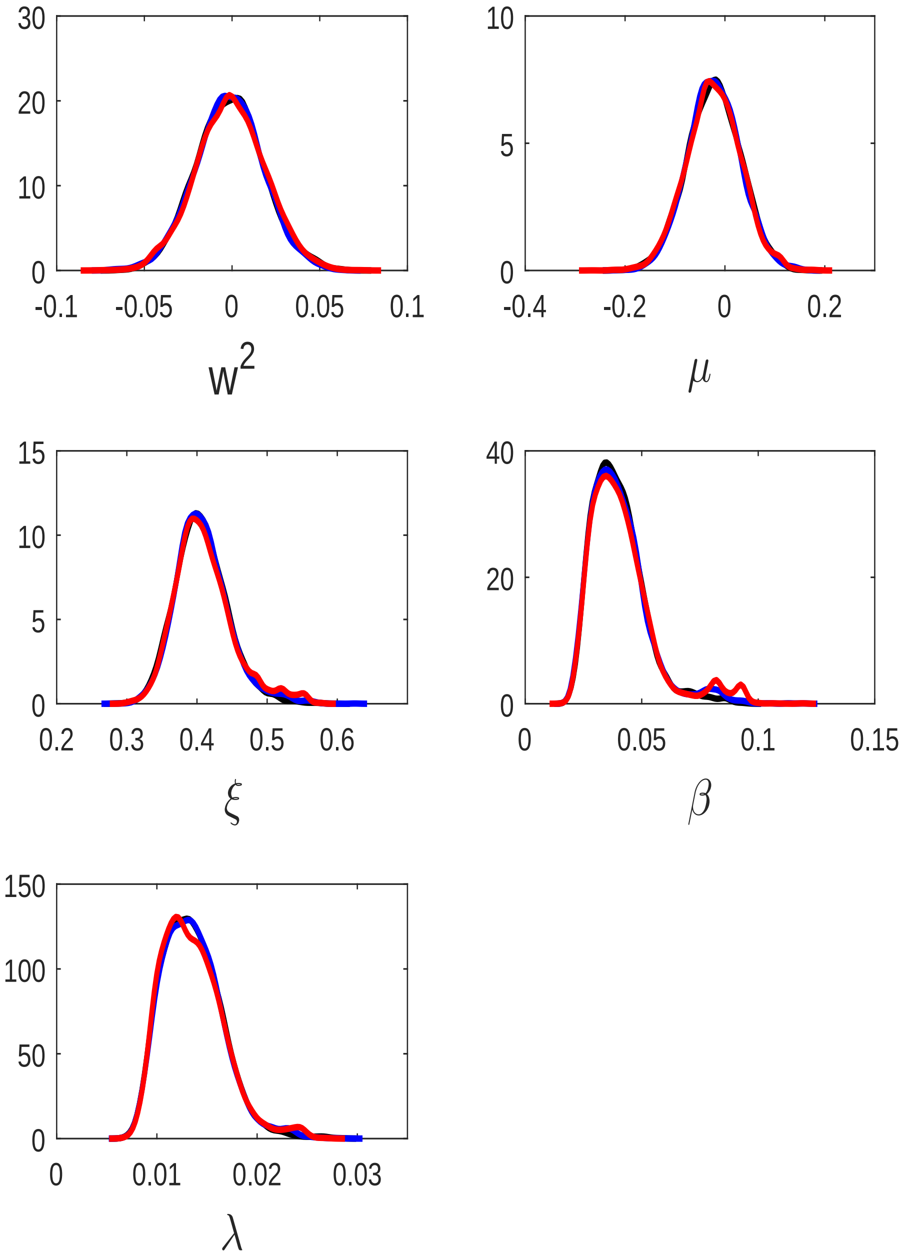


Figure 2: Sensitivity to ϵ for the stochastic volatility example for the second independent run when $J \approx 1500$. Posterior estimates are shown for $\epsilon = 1.0$ (black), $\epsilon = 1.5$ (blue) and $\epsilon = 2.0$ (red).

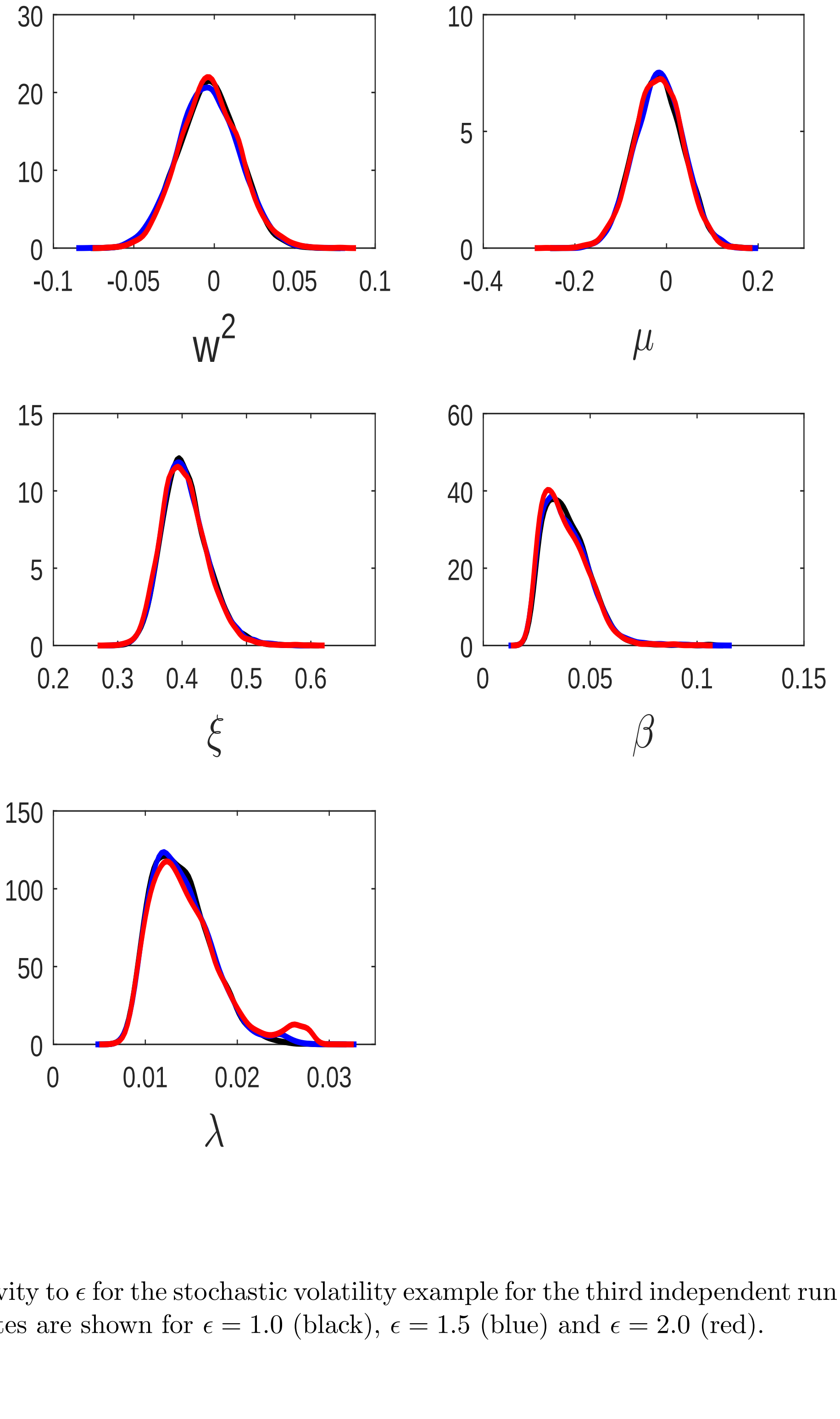


Figure 3: Sensitivity to ϵ for the stochastic volatility example for the third independent run when $J \approx 1500$. Posterior estimates are shown for $\epsilon = 1.0$ (black), $\epsilon = 1.5$ (blue) and $\epsilon = 2.0$ (red).

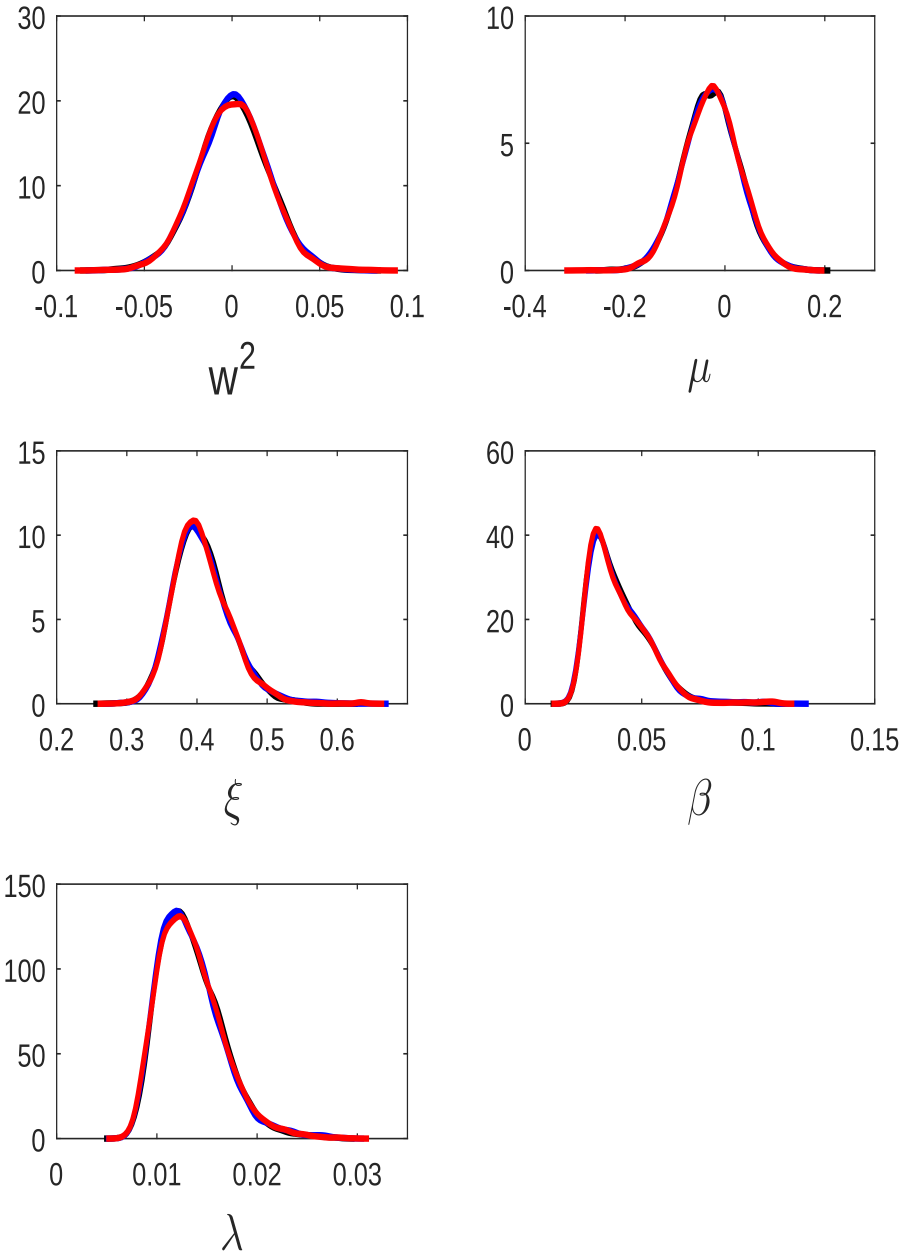


Figure 4: Sensitivity to ϵ for the stochastic volatility example for the fourth independent run when $J \approx 1500$. Posterior estimates are shown for $\epsilon = 1.0$ (black), $\epsilon = 1.5$ (blue) and $\epsilon = 2.0$ (red).

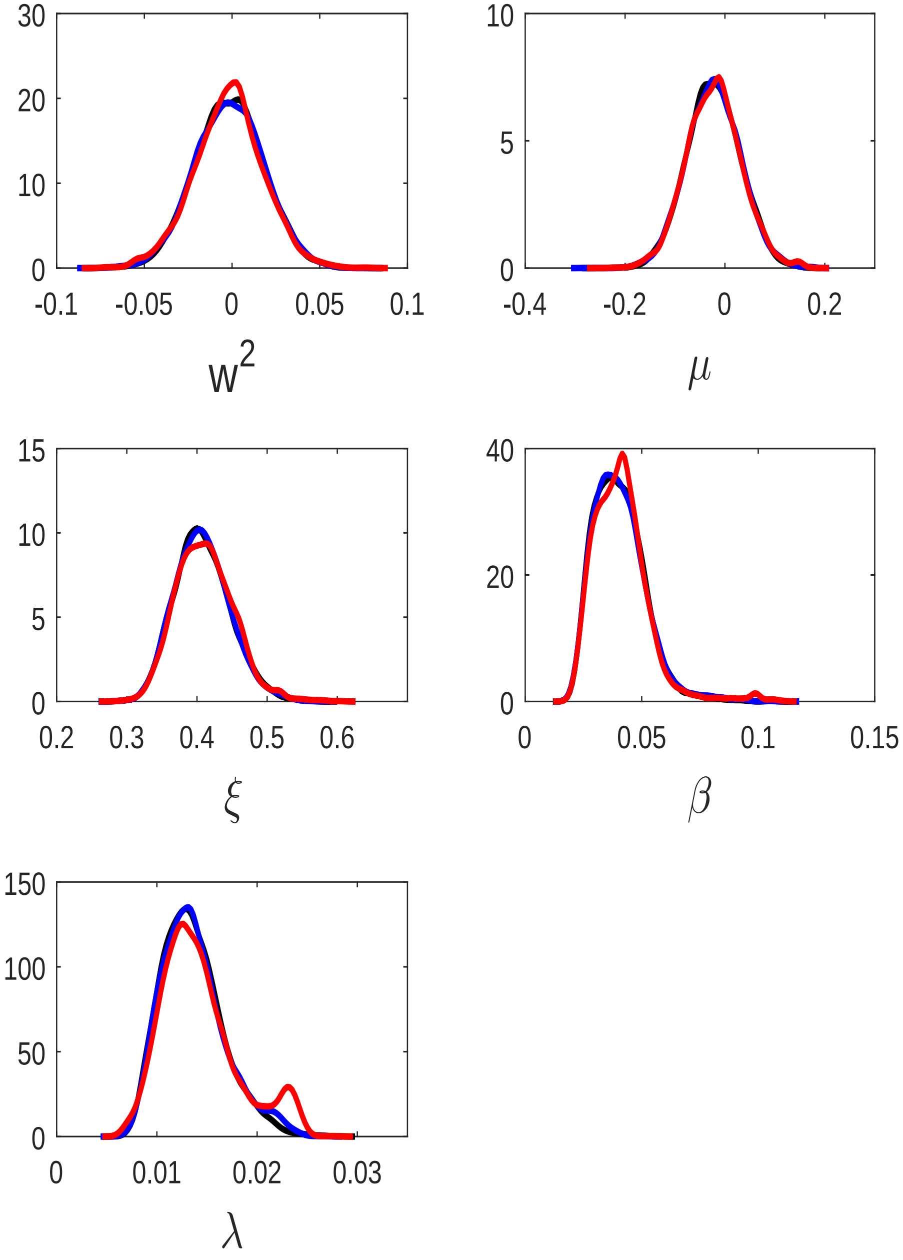


Figure 5: Sensitivity to ϵ for the stochastic volatility example for the fifth independent run when $J \approx 1500$. Posterior estimates are shown for $\epsilon = 1.0$ (black), $\epsilon = 1.5$ (blue) and $\epsilon = 2.0$ (red).

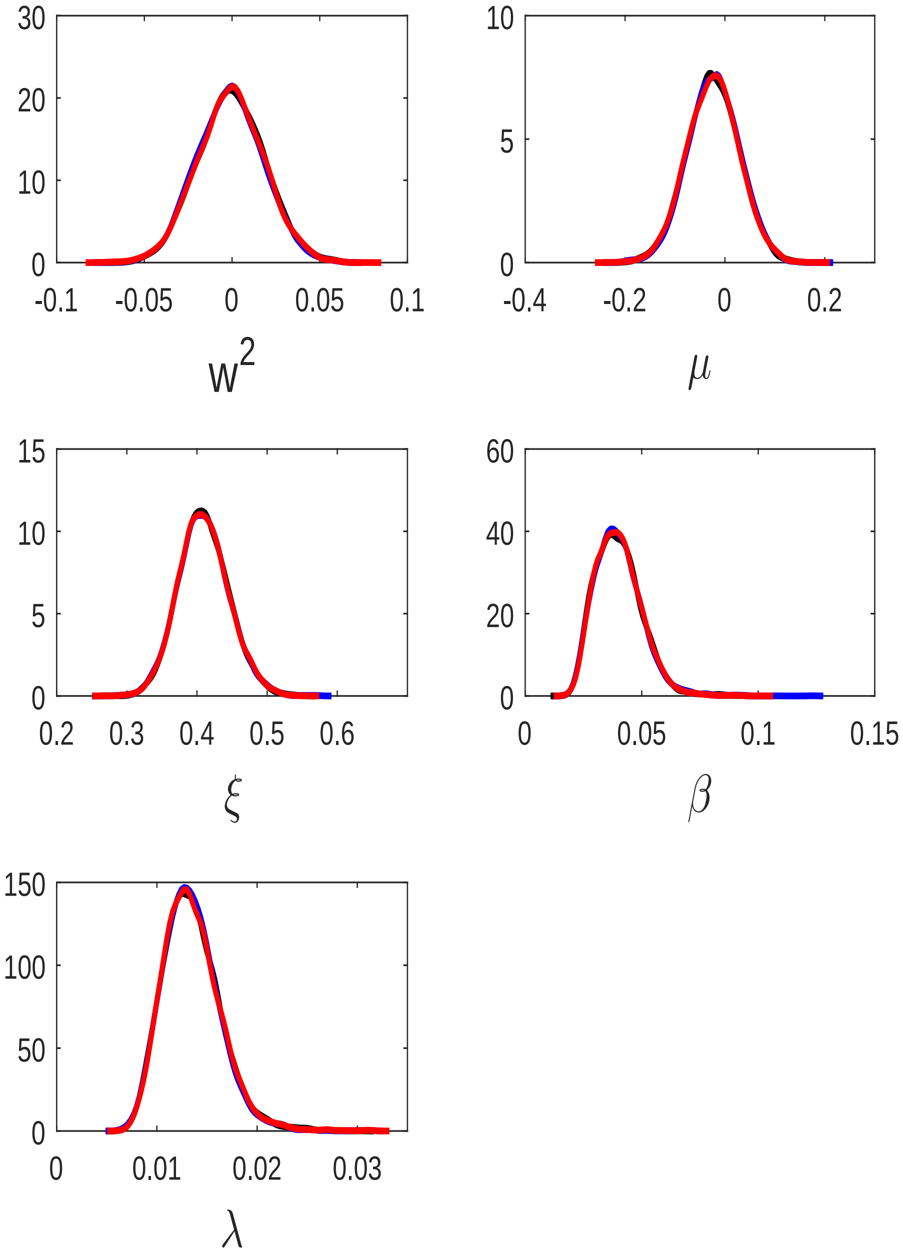


Figure 6: Sensitivity to ϵ for the stochastic volatility example for the first independent run when $J \approx 3000$. Posterior estimates are shown for $\epsilon = 1.0$ (black), $\epsilon = 1.5$ (blue) and $\epsilon = 2.0$ (red).

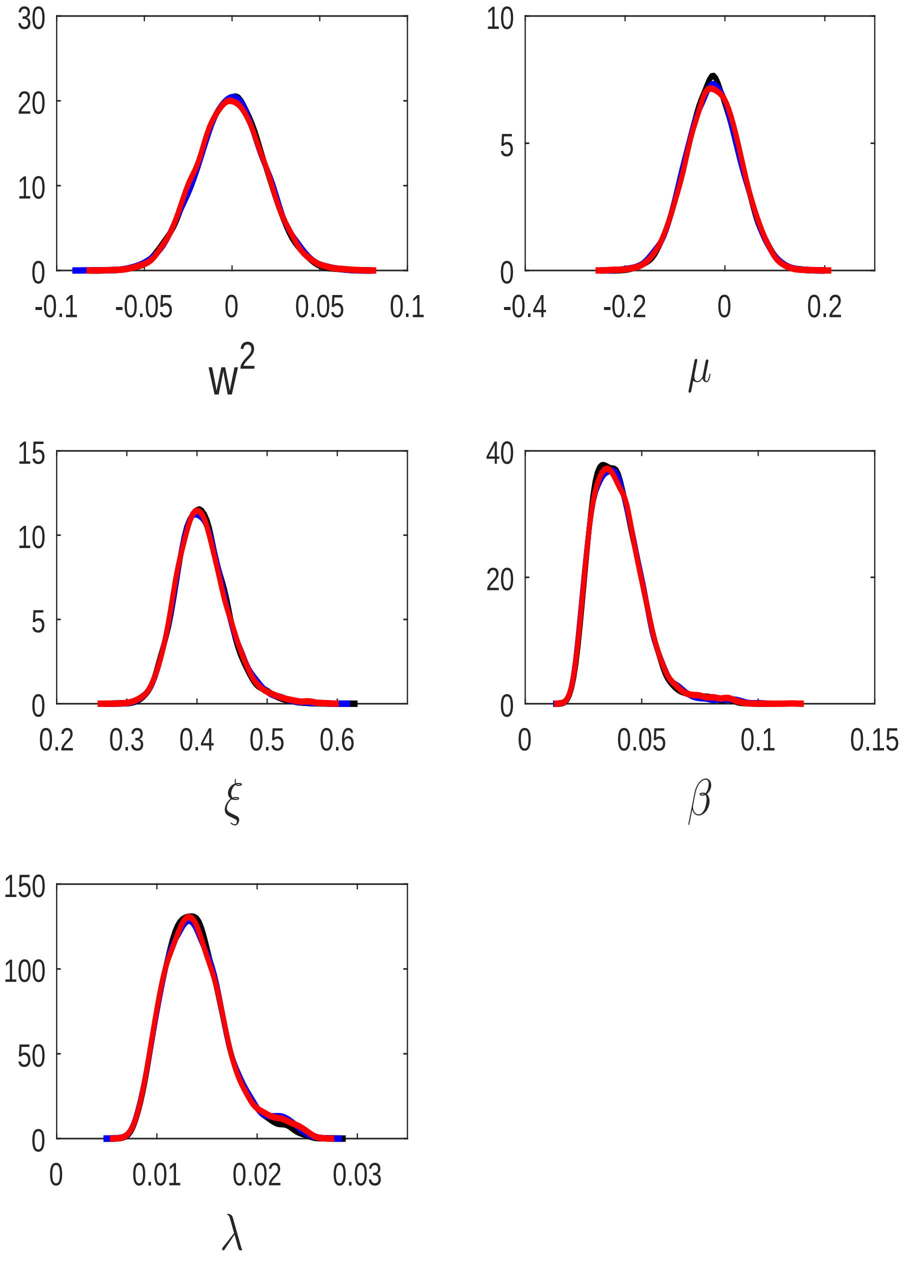


Figure 7: Sensitivity to ϵ for the stochastic volatility example for the second independent run when $J \approx 3000$. Posterior estimates are shown for $\epsilon = 1.0$ (black), $\epsilon = 1.5$ (blue) and $\epsilon = 2.0$ (red).

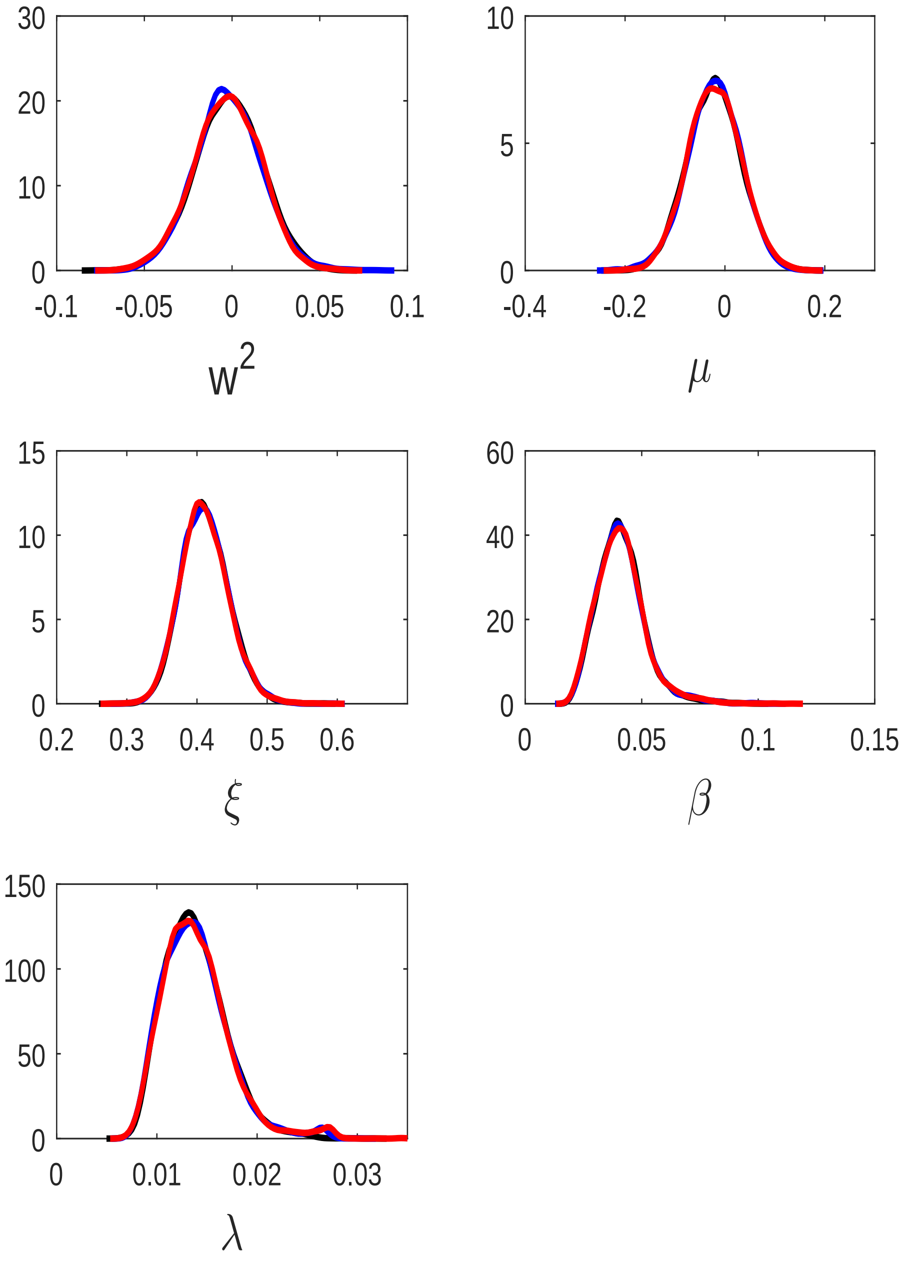


Figure 8: Sensitivity to ϵ for the stochastic volatility example for the third independent run when $J \approx 3000$. Posterior estimates are shown for $\epsilon = 1.0$ (black), $\epsilon = 1.5$ (blue) and $\epsilon = 2.0$ (red).

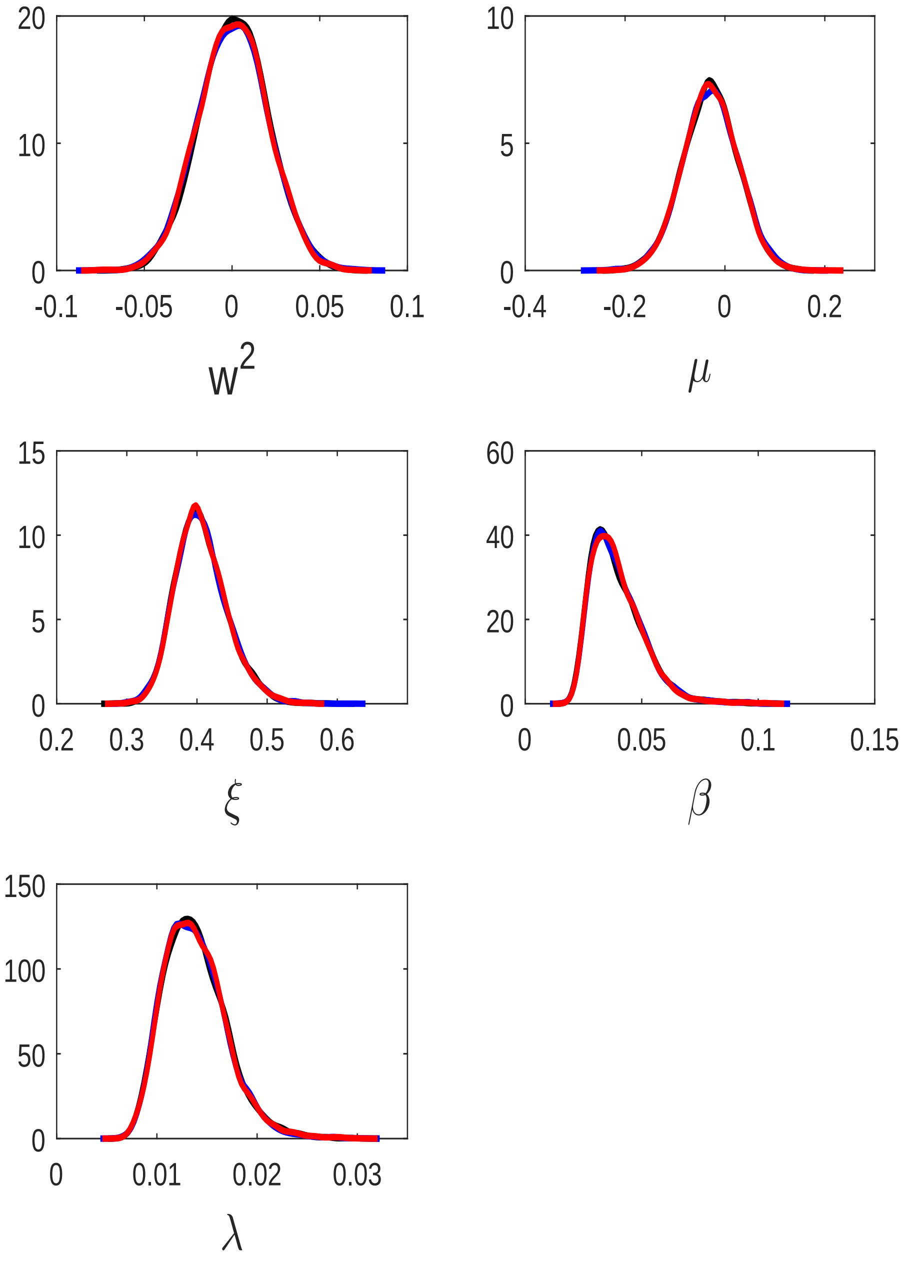


Figure 9: Sensitivity to ϵ for the stochastic volatility example for the fourth independent run when $J \approx 3000$. Posterior estimates are shown for $\epsilon = 1.0$ (black), $\epsilon = 1.5$ (blue) and $\epsilon = 2.0$ (red).

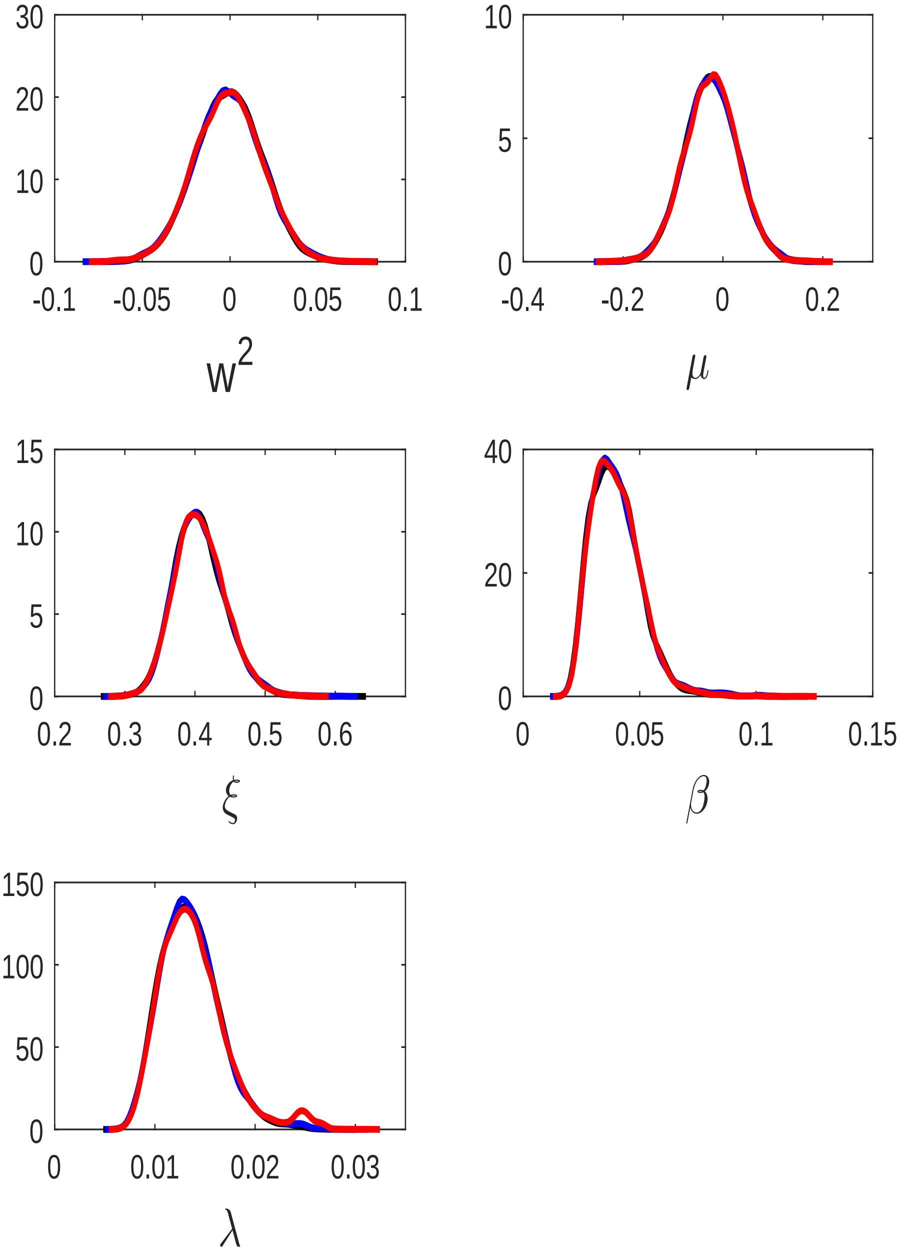


Figure 10: Sensitivity to ϵ for the stochastic volatility example for the fifth independent run when $J \approx 3000$. Posterior estimates are shown for $\epsilon = 1.0$ (black), $\epsilon = 1.5$ (blue) and $\epsilon = 2.0$ (red).

C Total variation distances for the stochastic volatility example

Table 1: TV distances between the marginals of the gold standard GIMH run and the GIMH, MCWM and GP-GIMH (for different values of ϵ and J where the results are averaged over the five runs) methods for the stochastic volatility example.

method	w^2	μ	ξ	β	λ
GIMH ($N = 800$)	0.032	0.018	0.027	0.023	0.016
MCWM	0.068	0.079	0.095	0.099	0.052
GP-GIMH ($\epsilon = 1.0$, $J \approx 1500$)	0.033	0.032	0.032	0.086	0.158
GP-GIMH ($\epsilon = 1.5$, $J \approx 1500$)	0.033	0.034	0.038	0.088	0.151
GP-GIMH ($\epsilon = 2.0$, $J \approx 1500$)	0.031	0.031	0.042	0.097	0.157
GP-GIMH ($\epsilon = 1.0$, $J \approx 3000$)	0.026	0.024	0.044	0.061	0.190
GP-GIMH ($\epsilon = 1.5$, $J \approx 3000$)	0.025	0.025	0.044	0.064	0.192
GP-GIMH ($\epsilon = 2.0$, $J \approx 3000$)	0.023	0.024	0.042	0.060	0.192

Table 2: TV distances between the bivariate posterior distributions of the gold standard GIMH run and the GIMH/MCWM/GP-GIMH (based on the average over five runs with $\epsilon = 1.0$ and $J \approx 3000$) methods for the stochastic volatility example.

	μ	ξ	β	λ
w^2	0.24/0.50/0.22	0.25/0.50/0.28	0.21/0.54/0.30	0.33/0.58/0.93
μ		0.24/0.54/0.28	0.22/0.56/0.30	0.14/0.64/0.93
ξ			0.24/0.56/0.42	0.25/0.68/0.94
β				0.14/0.69/0.98

Table 3: The mean TV distances over the five independent runs of GP-GIMH between each run and the mean of the five runs for the stochastic volatility example. Results are shown for different combinations of ϵ and J.

GP-GIMH parameters	w^2	μ	ξ	β	λ
($\epsilon = 1.0$, $J \approx 1500$)	0.031	0.028	0.027	0.048	0.044
($\epsilon = 1.5$, $J \approx 1500$)	0.032	0.029	0.032	0.052	0.051
($\epsilon = 2.0$, $J \approx 1500$)	0.030	0.028	0.039	0.063	0.062
($\epsilon = 1.0$, $J \approx 3000$)	0.024	0.021	0.032	0.041	0.014
($\epsilon = 1.5$, $J \approx 3000$)	0.021	0.021	0.030	0.043	0.021
($\epsilon = 2.0$, $J \approx 3000$)	0.020	0.021	0.029	0.039	0.024

D GP residual plots for the stochastic volatility example for different values of J and ϵ

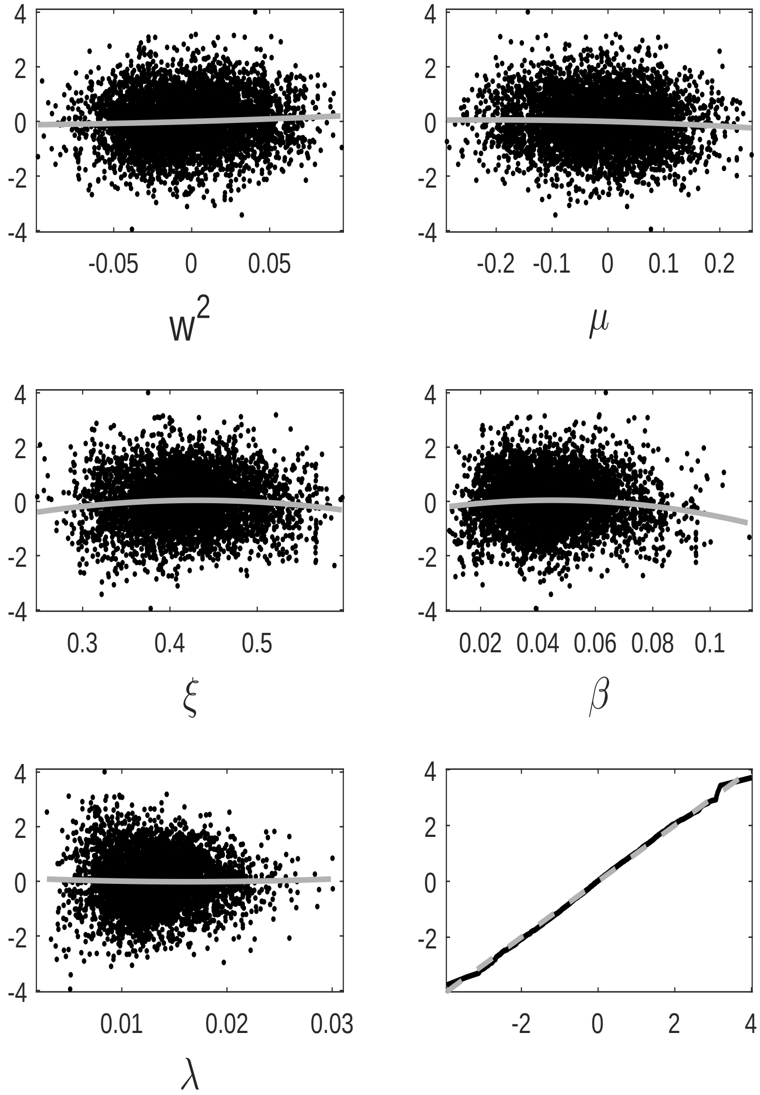


Figure 11: Residual plots of the GP fit for the stochastic volatility example when $J \approx 1500$ and $\epsilon = 1.0$. The grey curves show a quadratic fit to the residuals.

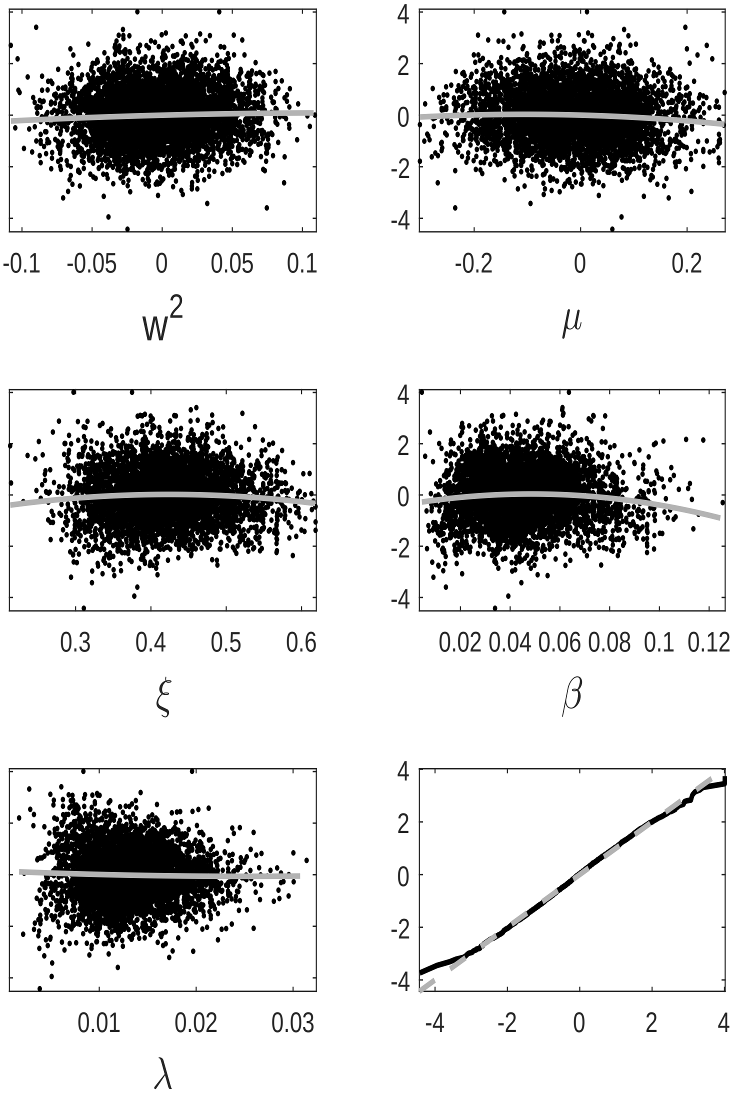


Figure 12: Residual plots of the GP fit for the stochastic volatility example when $J \approx 1500$ and $\epsilon = 1.5$. The grey curves show a quadratic fit to the residuals.

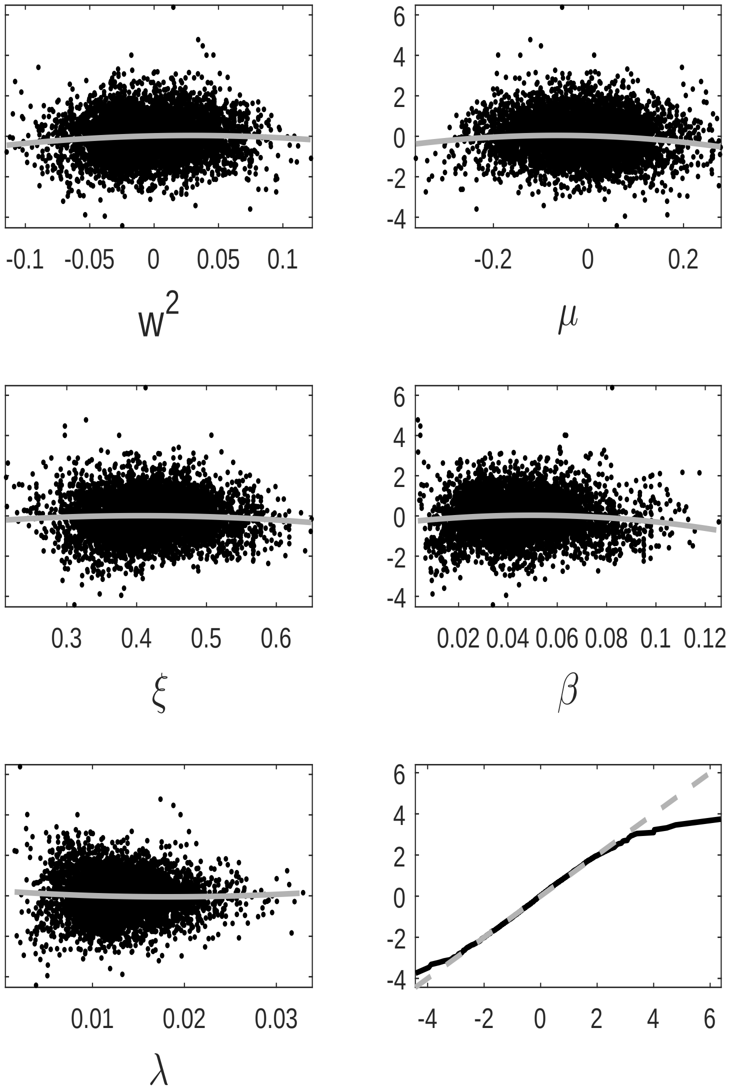


Figure 13: Residual plots of the GP fit for the stochastic volatility example when $J \approx 1500$ and $\epsilon = 2.0$. The grey curves show a quadratic fit to the residuals.

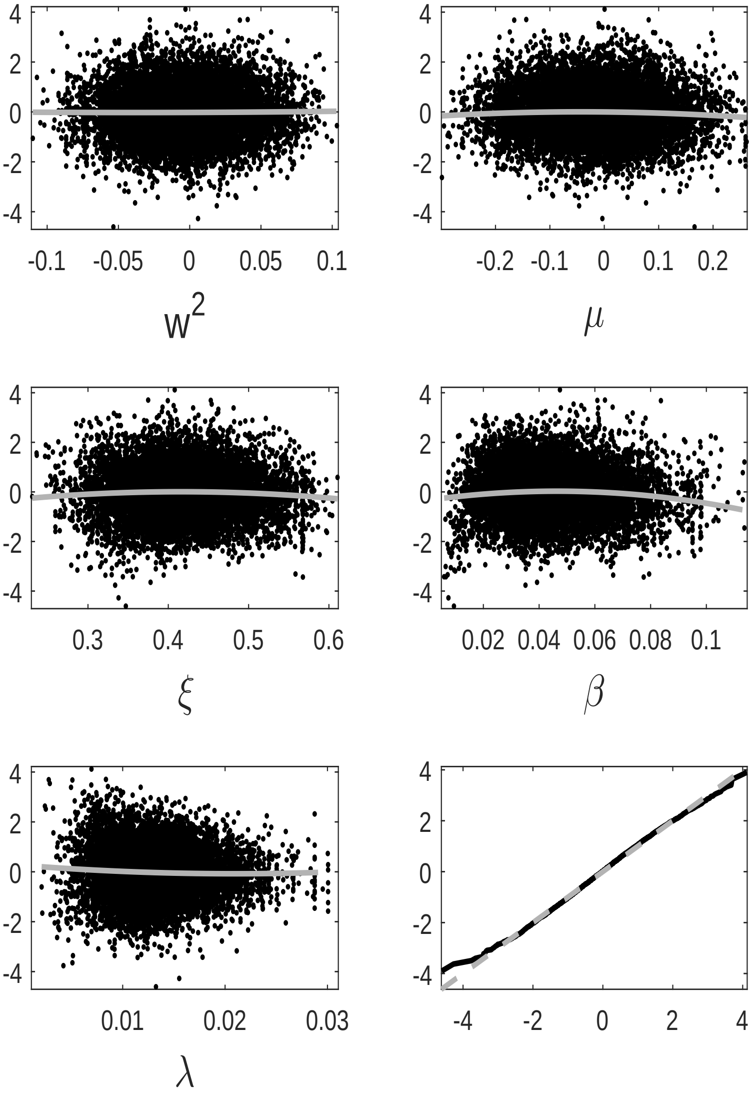


Figure 14: Residual plots of the GP fit for the stochastic volatility example when $J \approx 3000$ and $\epsilon = 1.0$. The grey curves show a quadratic fit to the residuals.

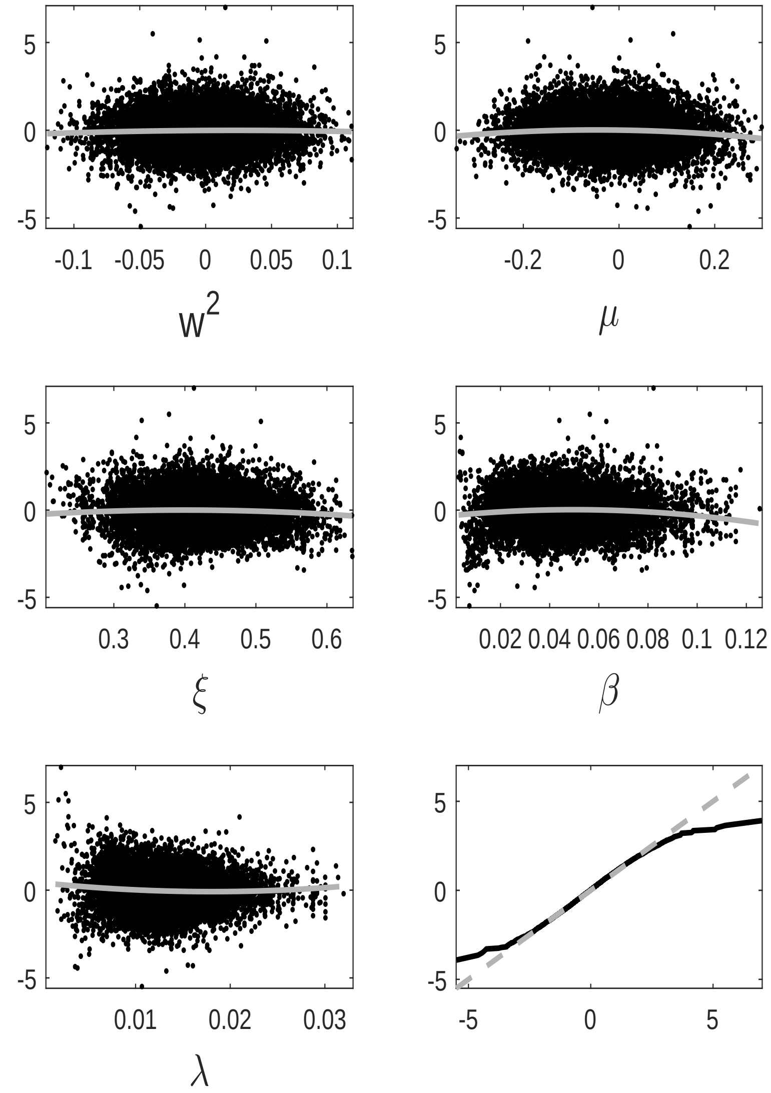


Figure 15: Residual plots of the GP fit for the stochastic volatility example when $J \approx 3000$ and $\epsilon = 1.5$. The grey curves show a quadratic fit to the residuals.

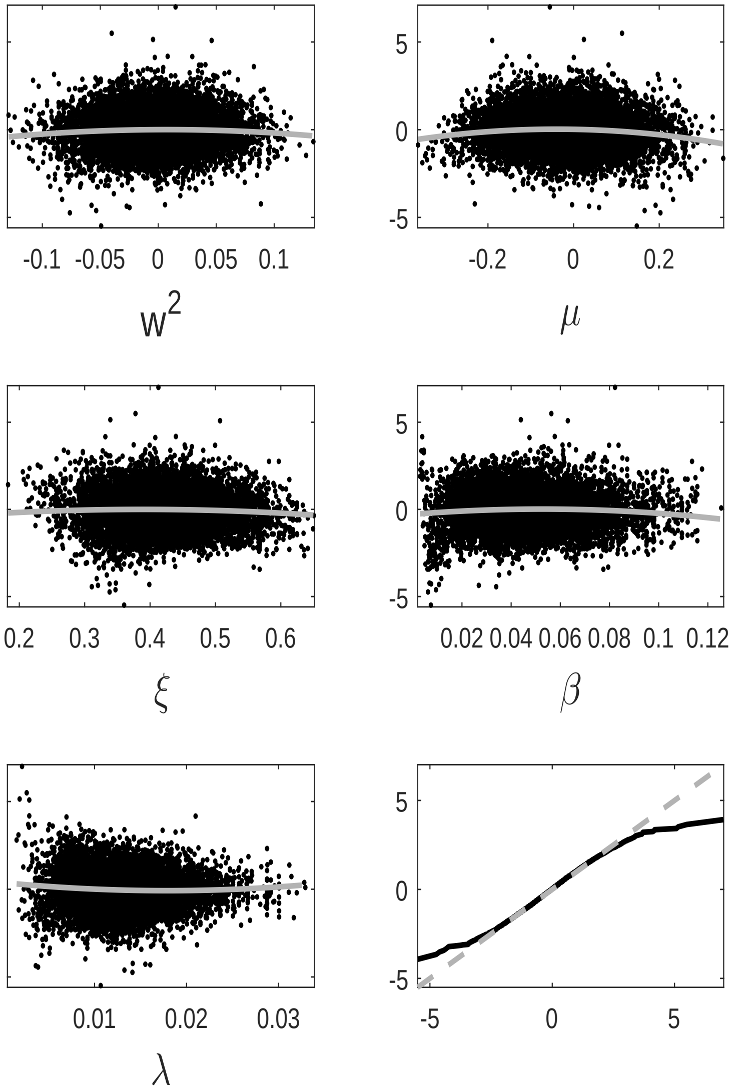


Figure 16: Residual plots of the GP fit for the stochastic volatility example when $J \approx 3000$ and $\epsilon = 2.0$. The grey curves show a quadratic fit to the residuals.

E History matching

The history matching approach of Wilkinson (2014) for GP training is shown in Algorithm 3. The method begins by sampling J trial values from the prior (Wilkinson (2014) suggest to use quasi-Monte Carlo numbers for this step since it has better space filling properties than pseudo-random numbers), estimating the log-likelihood for each trial value and fitting a GP where the training data is denoted as $\mathcal{D}_T = \{\boldsymbol{\theta}_j, \hat{f}(\boldsymbol{\theta}_j)\}_{j=1}^J$. Then, the fitted GP is used to define a region of the $\boldsymbol{\theta}$-space of implausibility. A proposed value $\boldsymbol{\theta}^*$ is determined as not implausible if $m^*(\boldsymbol{\theta}^*) + 3\sigma^*(\boldsymbol{\theta}^*) < \max\{\hat{f}(\boldsymbol{\theta}_j)\}_{j=1}^J - T$. A suitable value of T must be selected by the user. The portion of the training data that is deemed as not implausible by the fitted GP is retained and is replenished by continually simulating from the prior until J not implausible values are again generated.

From the new training set, another GP is fitted and the process continues in waves, where each proposed $\boldsymbol{\theta}^*$ is required to pass the non-implausibility test of the fitted GPs thus far.

Algorithm 3 History matching algorithm of Wilkinson (2014) for GP training.

```
Input: Number of trial points $J$, threshold $T > 0$ for determining implausibility, number of waves $W$.
Output: A set of training samples and fitted GPs for $W$ waves.
 1: Simulate $\{\boldsymbol{\theta}_j\}_{j=1}^J \overset{\text{iid}}{\sim} p(\boldsymbol{\theta})$.
 2: Estimate log-likelihood $f(\boldsymbol{\theta}_j)$ for $j = 1, \ldots, J$. This produces training data $\mathcal{D}_T = \{\boldsymbol{\theta}_j, \hat{f}(\boldsymbol{\theta}_j)\}_{j=1}^J$.
 3: for $w = 1$ to $W$ do
 4:   Fit GP model to $\mathcal{D}_T$. Denote the fitted GP as $\mathrm{GP}_w$.
 5:   Compute $m^*(\boldsymbol{\theta}_j)$ and $\sigma^*(\boldsymbol{\theta}_j)^2$ from the fitted $\mathrm{GP}_w$ for $j = 1, \ldots, J$.
 6:   Determine which of the current trial values $\{\boldsymbol{\theta}_j\}_{j=1}^J$ are not implausible based on $\mathrm{GP}_w$ where
      implausibility occurs when $m^*(\boldsymbol{\theta}_j) + 3\sigma^*(\boldsymbol{\theta}_j) < \max\{\hat{f}(\boldsymbol{\theta}_j)\}_{j=1}^J - T$. Remove implausible values
      from the training data.
 7:   while number of not implausbile trial values $< J$ do
 8:     while true do
 9:       Propose $\boldsymbol{\theta}^* \sim p(\boldsymbol{\theta})$.
10:       Check through all GP fits $\{\mathrm{GP}_k\}_{k=1}^w$ that $\boldsymbol{\theta}^*$ is not implausible. If it is implausible, go back
          to line 9, otherwise go to the next step.
11:       Estimate log-likelihood $f(\boldsymbol{\theta}^*)$ and add $(\boldsymbol{\theta}^*, f(\boldsymbol{\theta}^*))$ to the training data. Break out of the inner
          while loop.
12:     end while
13:   end while
14: end for
```

We apply the history matching method to the gene network example of the main paper. We encountered several issues. For the initial training data from the prior, there are several extremely low log-likelihood estimates. As suggested by Wilkinson (2014), the GP is fitted to the values $\log\{-\hat{f}(\boldsymbol{\theta}_j)\}$ for $j = 1, \ldots, J$. Wilkinson (2014) suggest that this might only be required for the first few waves. However, we find that even after 10 waves there are still proposed parameter values that are deemed not implausible even though they produce very low log-likelihood estimates. Thus our fitted GPs always use $\log\{-\hat{f}(\boldsymbol{\theta})\}$ as the response variable. Further, the samples from the prior can be particularly poor so that due to numerical problems the log-likelihood is estimated to be $-\infty$. We set such estimates to the minimum log-likelihood of the other training samples so that such parameter regions are discouraged in future waves. Unlike Wilkinson (2014), we also find that using a constant value of T through the waves is not suitable. Here we choose a value of T at each wave such that a certain percentage of the training data are deemed to be non-implausible by the fitted GP.

For the gene network example we use $J = 1000$ and select T at each wave using the bisection method so that roughly 250 of the training data are deemed non-implausible. We use $W = 10$ waves so that $750 \times 10 + 1000 = 8500$ log-likelihood estimates are required. This is more than our MCWM approach in the main paper, but the history matching approach is more suitable for parallel computing. However, the 9th and 10th waves are computationally intensive as many prior simulations are required to find parameter values that are deemed not implausible by all the fitted GPs. Thus the history matching approach is comfortably allocated more training time compared to our MCWM approach in the main paper.

Boxplots of the values of $\log\{-\hat{f}(\boldsymbol{\theta})\}$ in the training set over the 10 waves is shown in Figure 17. Even though there is general improvement in the log-likelihood estimates as the wave number increases, in the last wave there are still many parameter values that are not rejected by the previously fitted GPs that produce very low log-likelihood estimates.

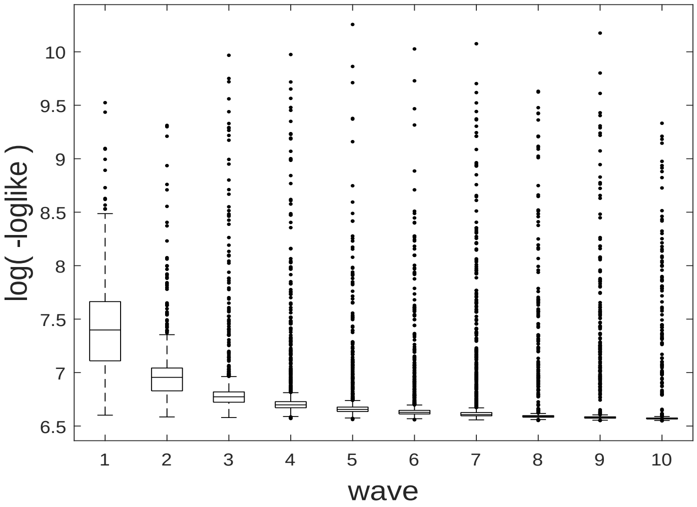
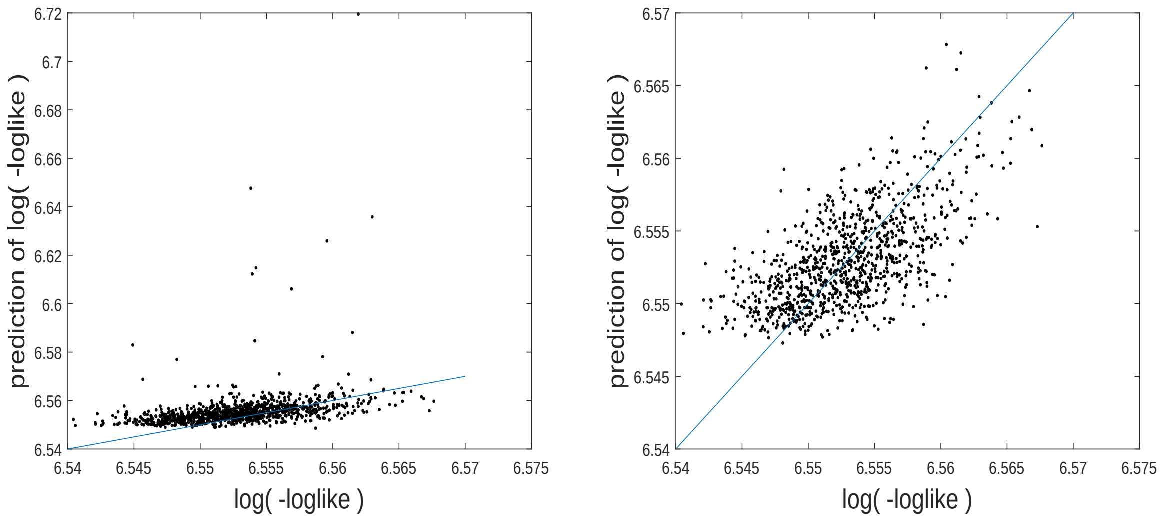


Figure 17: Boxplot of $\log\{-\hat{f}(\boldsymbol{\theta})\}$ values obtained during the history matching procedure.

To compare the performance of the history matching and MCWM to training in the context of our pseudo-marginal application, we took 1000 samples from the actual posterior of the gene network example by thinning the output of the exact GIMH method. We then predict from the different GP fits at these test samples and compare the predicted log-likelihood to the log-likelihood estimates from the particle filter. For the history matching method, we consider the fitted GP from the final wave. The results are shown in Figure 18. It is clear that the MCWM training approach leads to a GP that predicts more successfully in posterior support regions.



(a) history matching

(b) MCWM

Figure 18: Comparison of GP predictions of the log-likelihood estimates from a posterior sample for the gene network example. (a) GP fitted from the last wave of history matching and (b) GP fitted based on the MCWM approach of the main paper.

*

F Sensitivity of GP-GIMH to ϵ for the gene network example

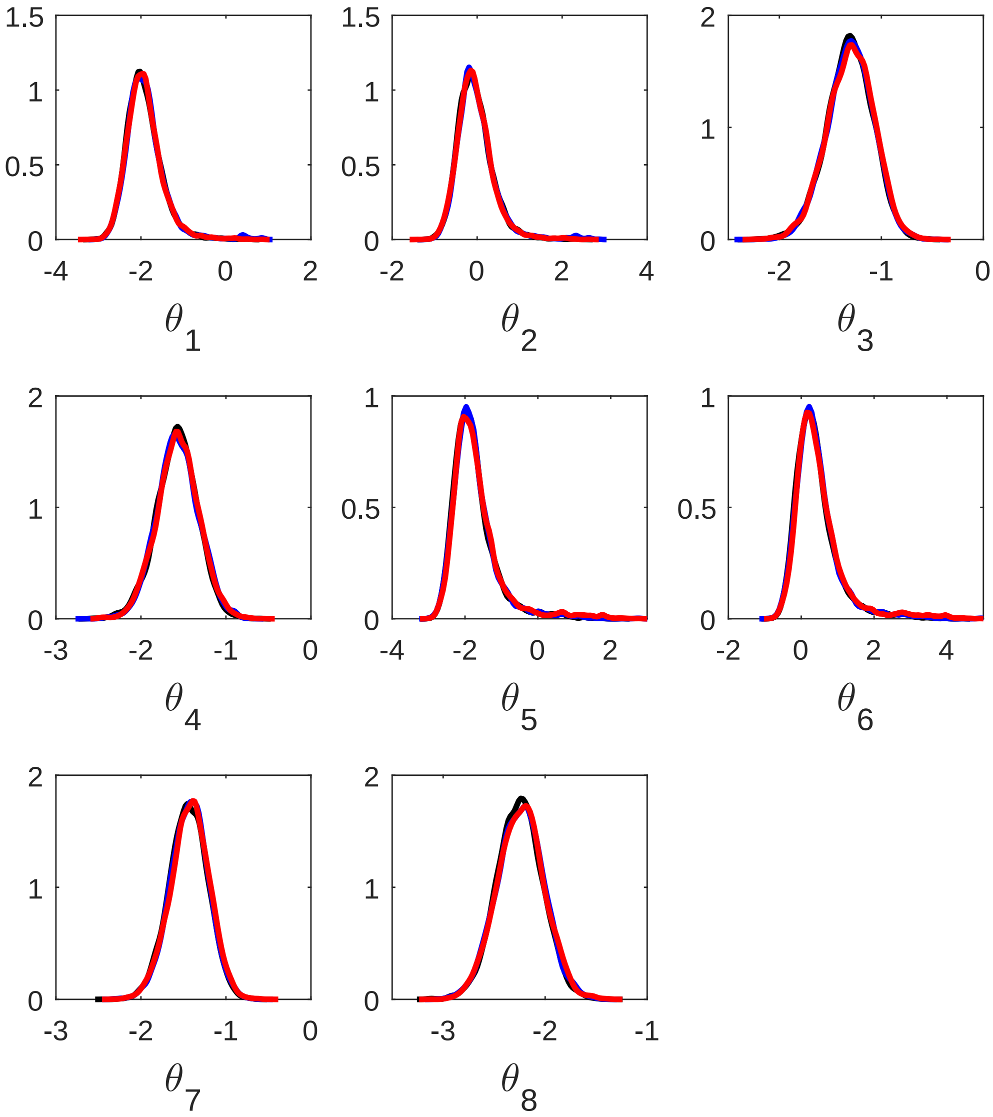


Figure 19: Sensitivity to ϵ for the gene network example for the first independent run when $J \approx 2000$. Posterior estimates are shown for $\epsilon = 1.2$ (black), $\epsilon = 1.5$ (blue) and $\epsilon = 2.0$ (red).

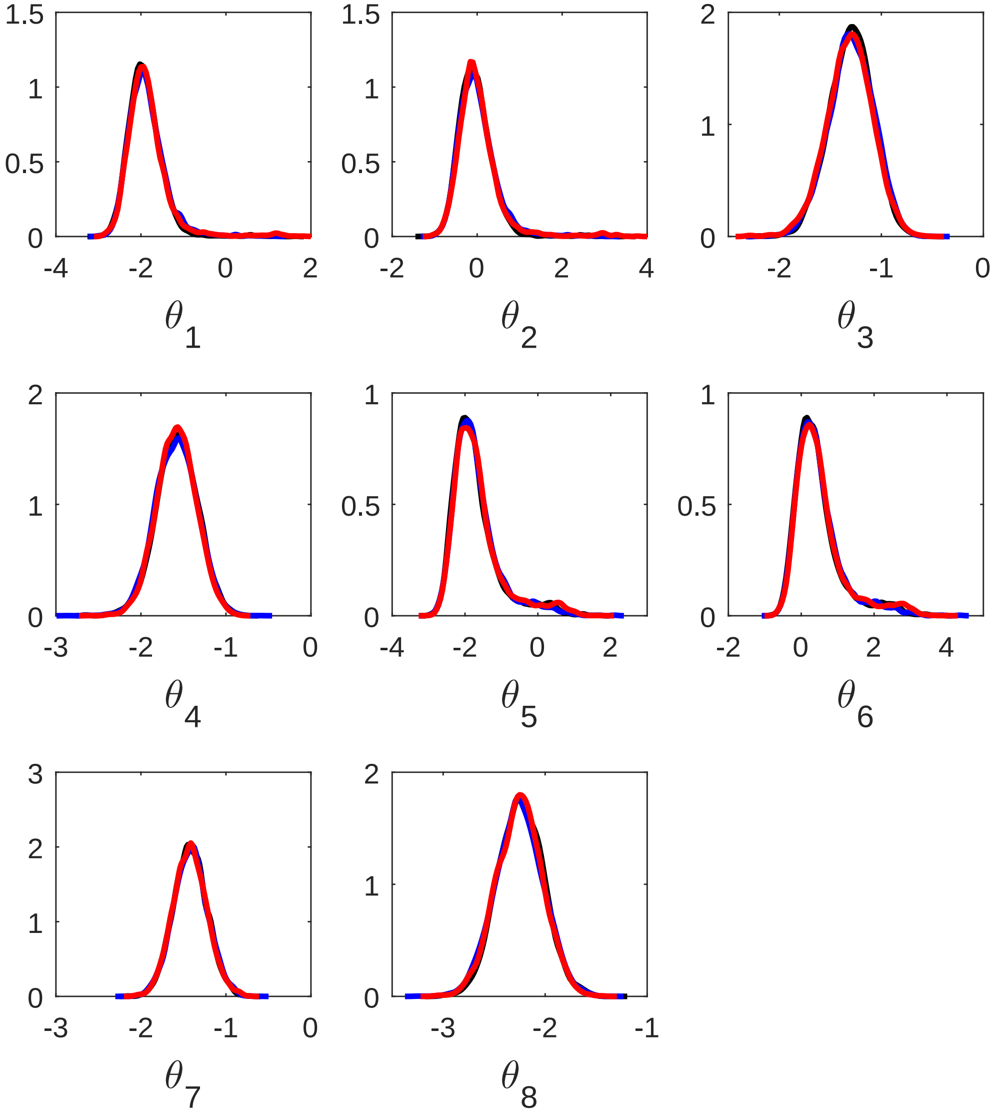


Figure 20: Sensitivity to ϵ for the gene network example for the second independent run when $J \approx 2000$. Posterior estimates are shown for $\epsilon = 1.2$ (black), $\epsilon = 1.5$ (blue) and $\epsilon = 2.0$ (red).

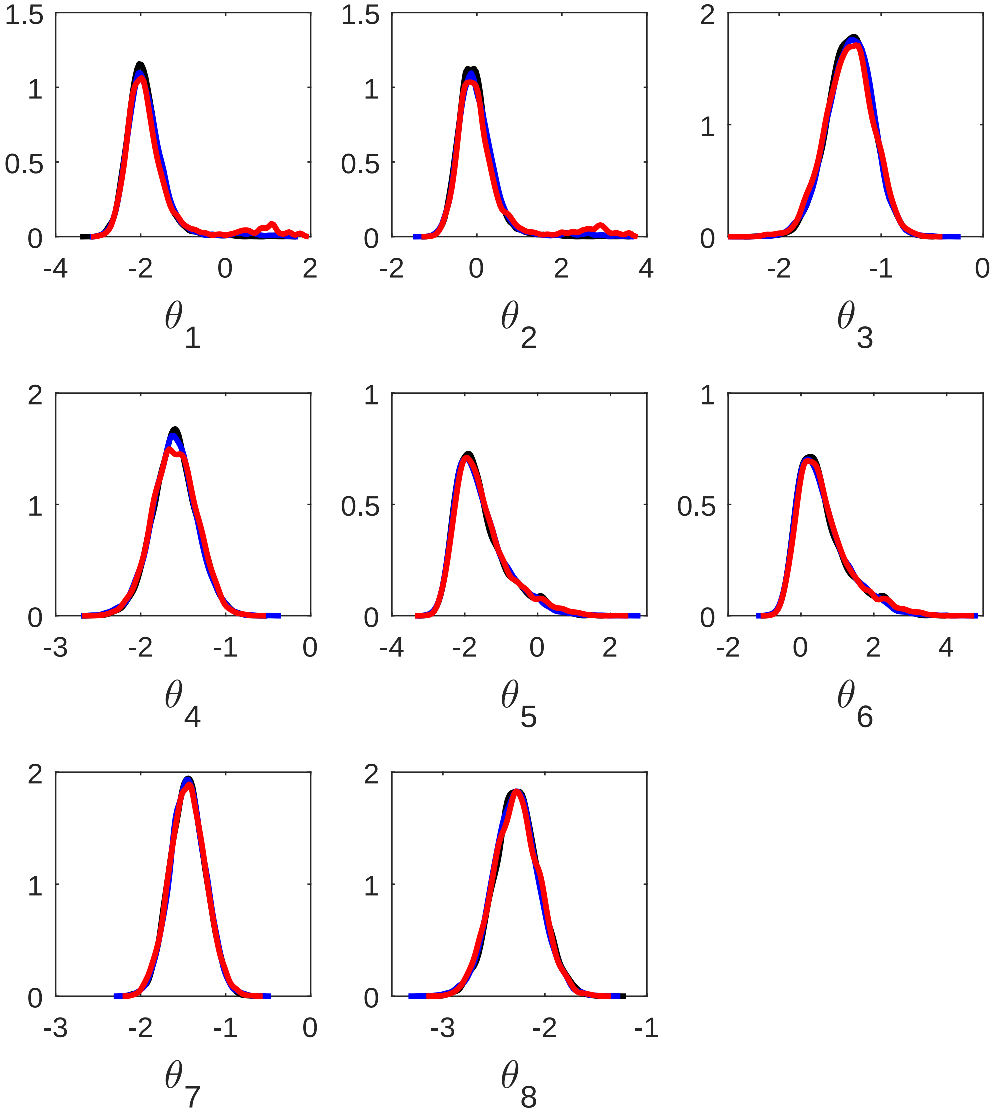


Figure 21: Sensitivity to ϵ for the gene network example for the third independent run when $J \approx 2000$. Posterior estimates are shown for $\epsilon = 1.2$ (black), $\epsilon = 1.5$ (blue) and $\epsilon = 2.0$ (red).

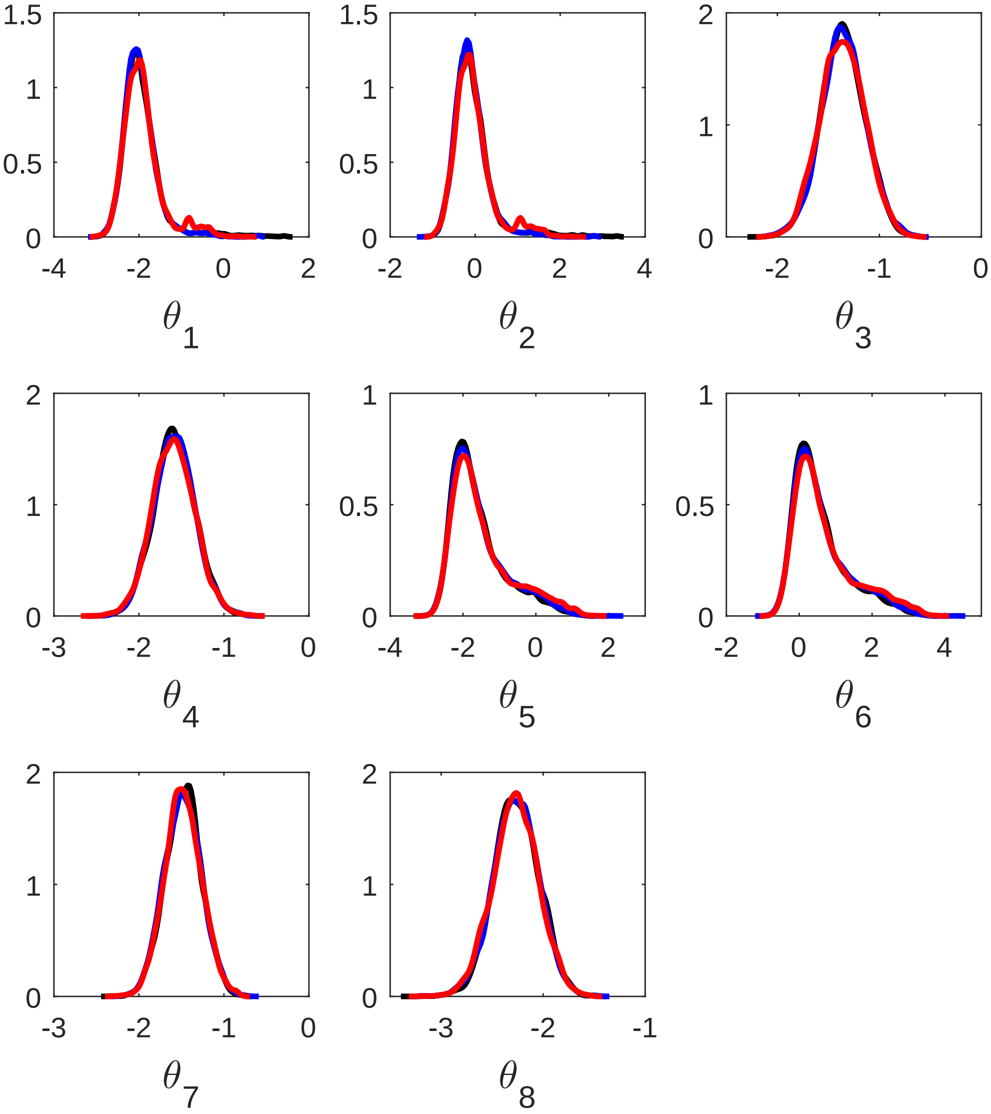


Figure 22: Sensitivity to ϵ for the gene network example for the fourth independent run when $J \approx 2000$. Posterior estimates are shown for $\epsilon = 1.2$ (black), $\epsilon = 1.5$ (blue) and $\epsilon = 2.0$ (red).

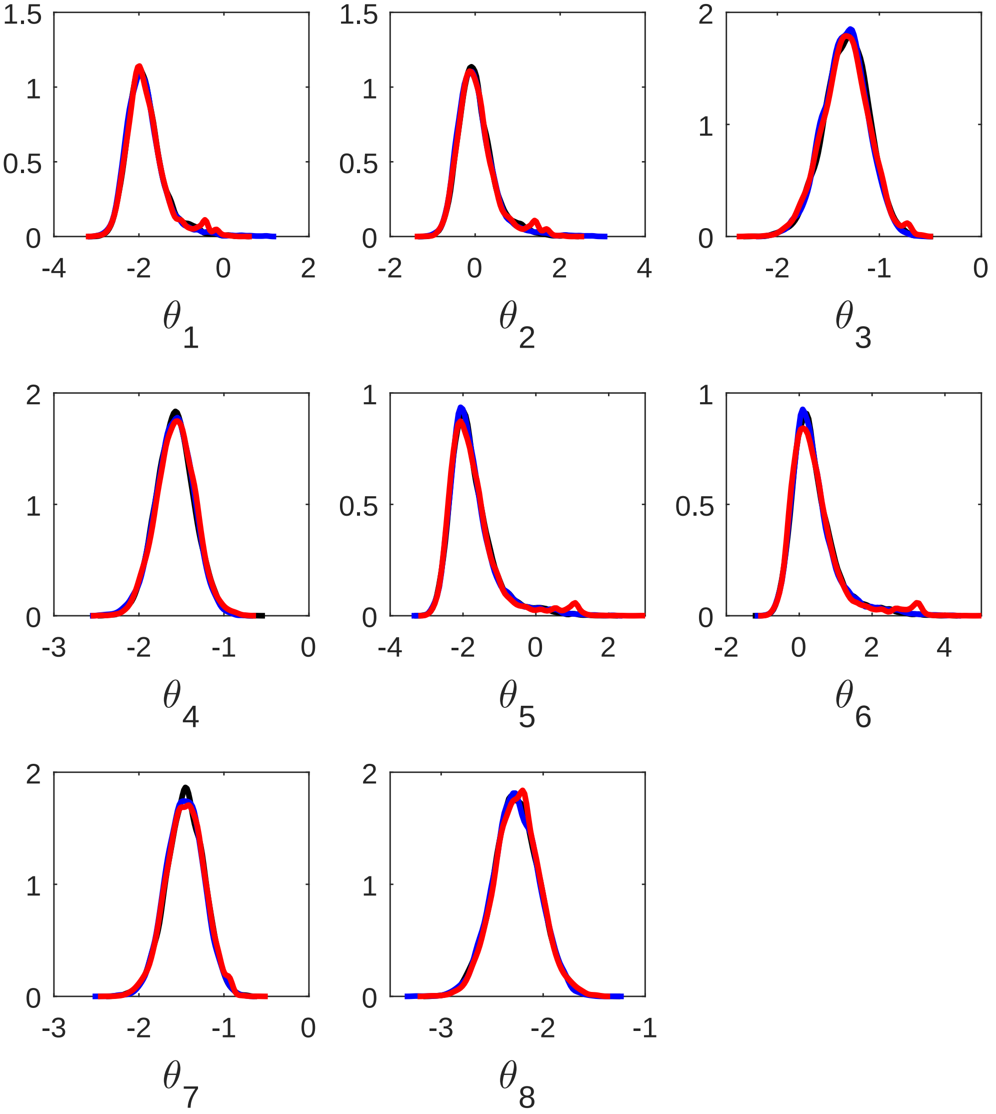


Figure 23: Sensitivity to ϵ for the gene network example for the fifth independent run when $J \approx 2000$. Posterior estimates are shown for $\epsilon = 1.2$ (black), $\epsilon = 1.5$ (blue) and $\epsilon = 2.0$ (red).

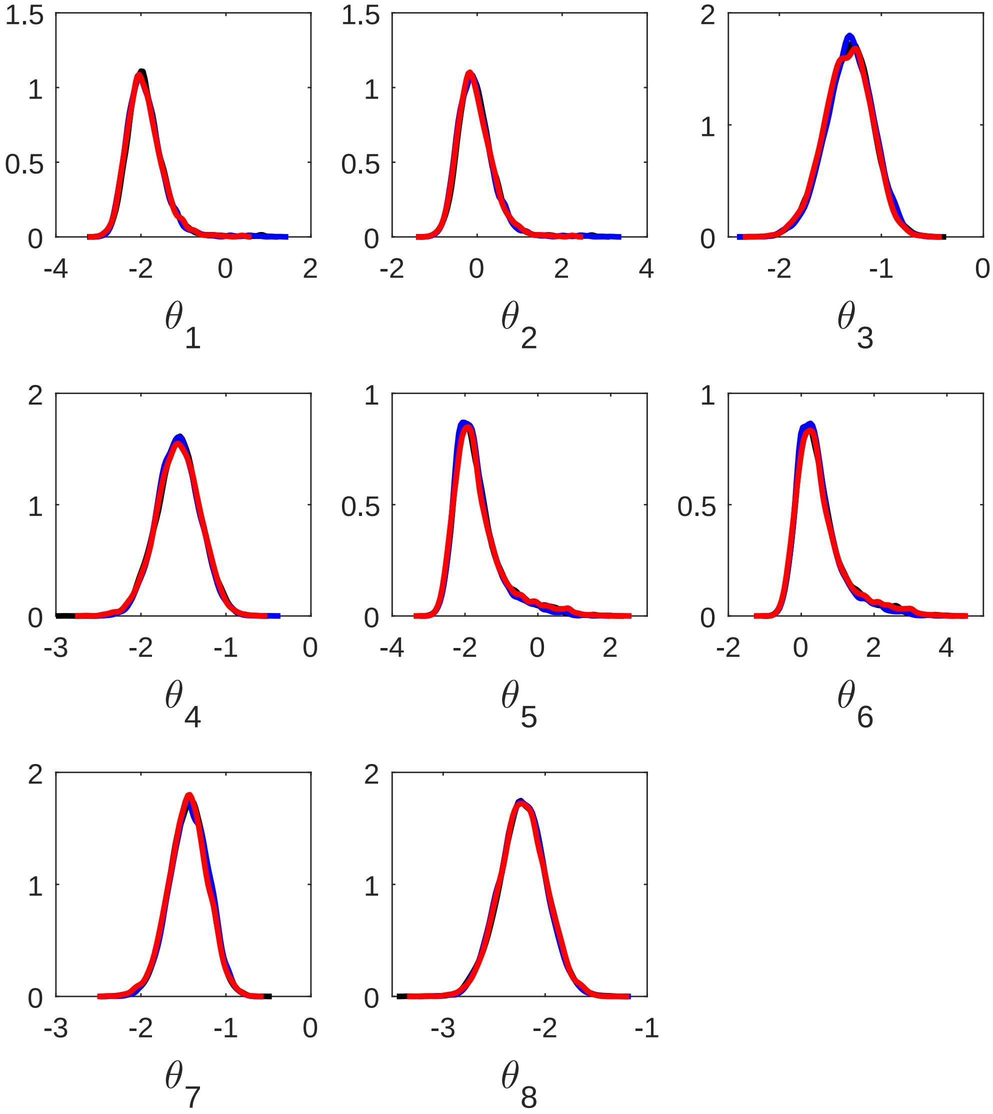


Figure 24: Sensitivity to ϵ for the gene network example for the first independent run when $J \approx 4000$. Posterior estimates are shown for $\epsilon = 1.2$ (black), $\epsilon = 1.5$ (blue) and $\epsilon = 2.0$ (red).

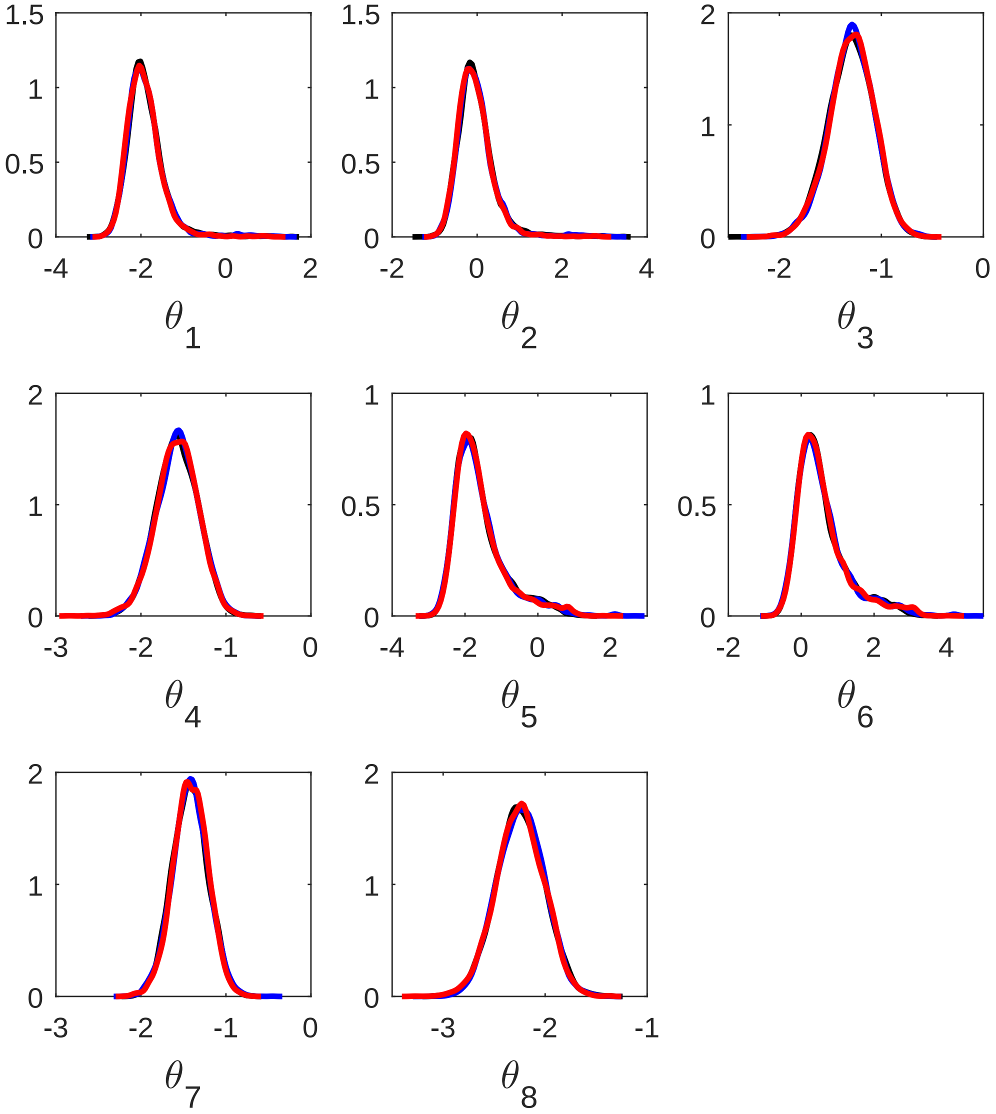


Figure 25: Sensitivity to ϵ for the gene network example for the second independent run when $J \approx 4000$. Posterior estimates are shown for $\epsilon = 1.2$ (black), $\epsilon = 1.5$ (blue) and $\epsilon = 2.0$ (red).

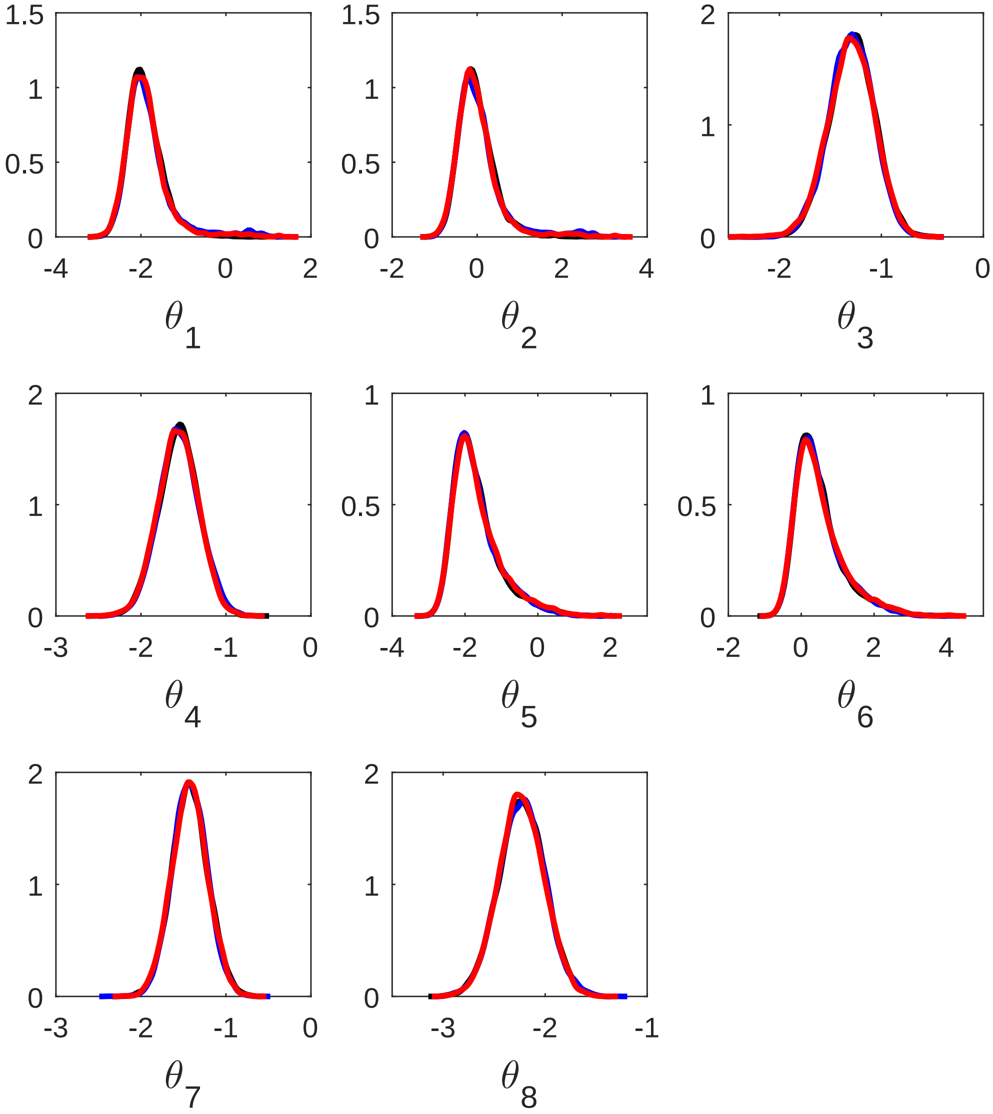


Figure 26: Sensitivity to ϵ for the gene network example for the third independent run when $J \approx 4000$. Posterior estimates are shown for $\epsilon = 1.2$ (black), $\epsilon = 1.5$ (blue) and $\epsilon = 2.0$ (red).

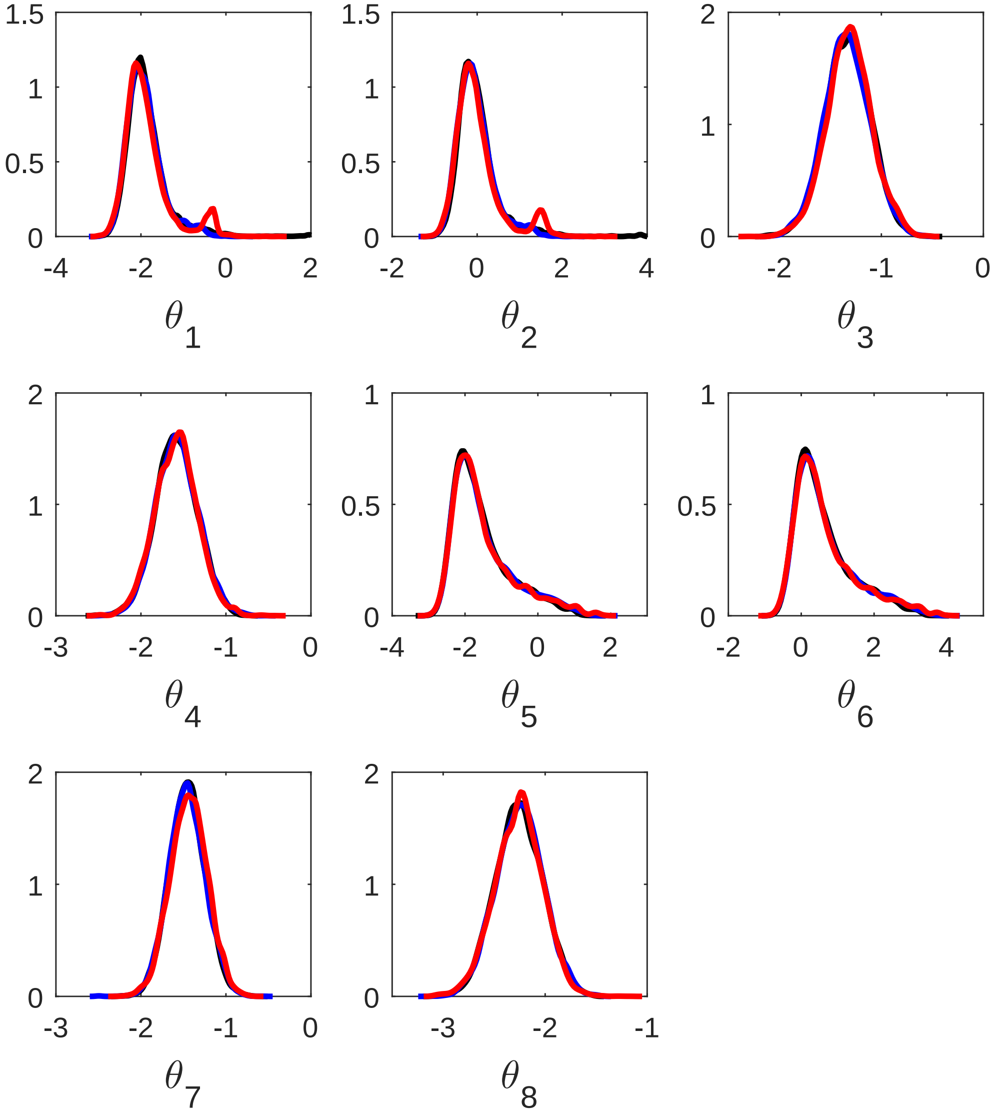


Figure 27: Sensitivity to ϵ for the gene network example for the fourth independent run when $J \approx 4000$. Posterior estimates are shown for $\epsilon = 1.2$ (black), $\epsilon = 1.5$ (blue) and $\epsilon = 2.0$ (red).

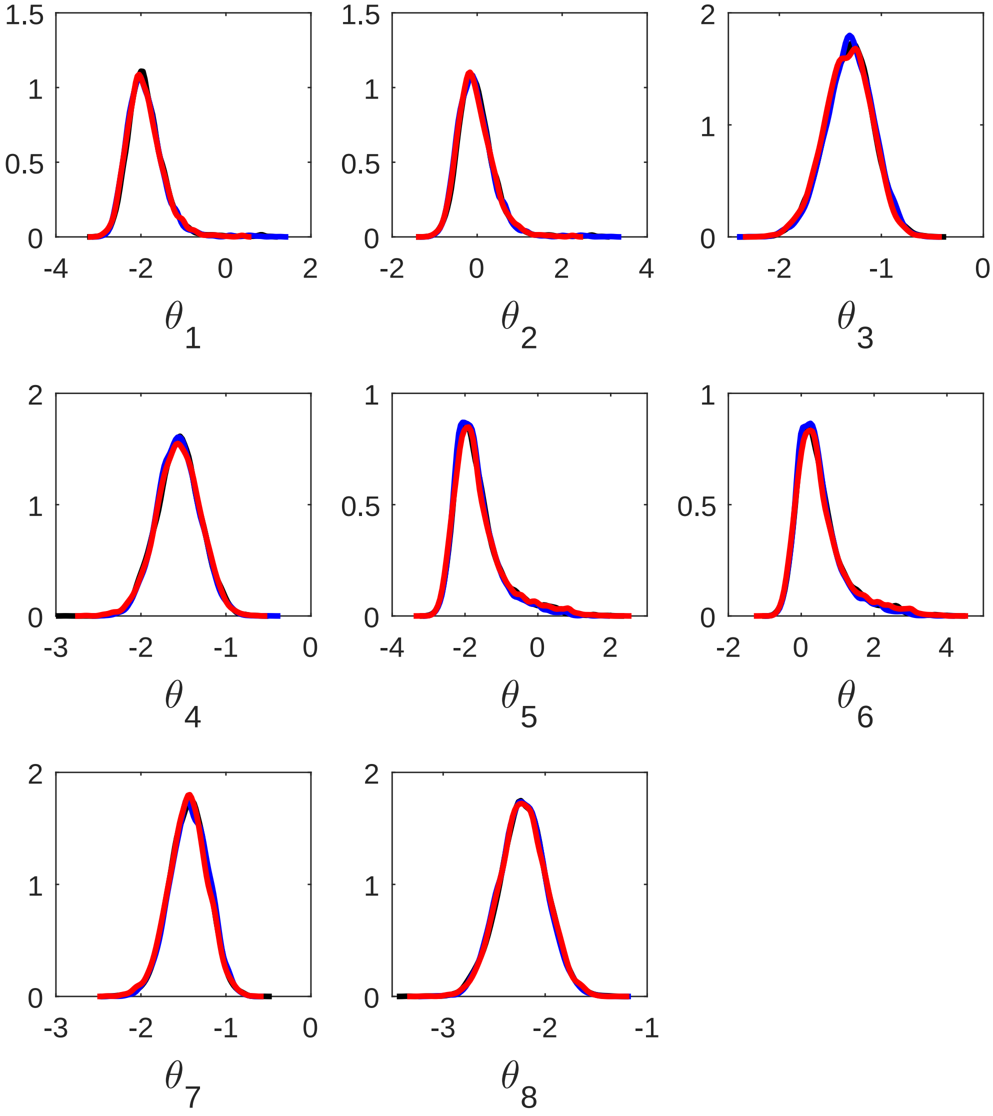


Figure 28: Sensitivity to ϵ for the gene network example for the fifth independent run when $J \approx 4000$. Posterior estimates are shown for $\epsilon = 1.2$ (black), $\epsilon = 1.5$ (blue) and $\epsilon = 2.0$ (red).

G Total variation distances for the gene network example

Table 4: TV distances between the marginal posterior distributions of the gold standard GIMH run and the GIMH, MCWM and GP-GIMH (for different values of ϵ and J where results are averaged over the five runs) methods for the gene network example.

method	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_8
GIMH ($N = 6000$)	0.026	0.031	0.016	0.026	0.047	0.044	0.028	0.033
MCWM	0.117	0.115	0.067	0.092	0.106	0.110	0.076	0.101
GP-GIMH ($\epsilon = 1.2$, $J \approx 2000$)	0.053	0.053	0.066	0.046	0.074	0.069	0.070	0.059
GP-GIMH ($\epsilon = 1.5$, $J \approx 2000$)	0.055	0.054	0.066	0.042	0.081	0.075	0.071	0.055
GP-GIMH ($\epsilon = 2.0$, $J \approx 2000$)	0.058	0.058	0.056	0.047	0.091	0.088	0.072	0.056
GP-GIMH ($\epsilon = 1.2$, $J \approx 4000$)	0.037	0.040	0.064	0.063	0.064	0.058	0.078	0.076
GP-GIMH ($\epsilon = 1.5$, $J \approx 4000$)	0.041	0.039	0.065	0.066	0.065	0.062	0.082	0.080
GP-GIMH ($\epsilon = 2.0$, $J \approx 4000$)	0.050	0.042	0.064	0.060	0.068	0.064	0.082	0.076

Table 5: TV distances between the bivariate posterior distributions of the gold standard GIMH run and GIMH/MCWM/GP-GIMH (based on the average over five runs with $\epsilon = 1.0$ and $J \approx 4000$) methods for the gene network example.

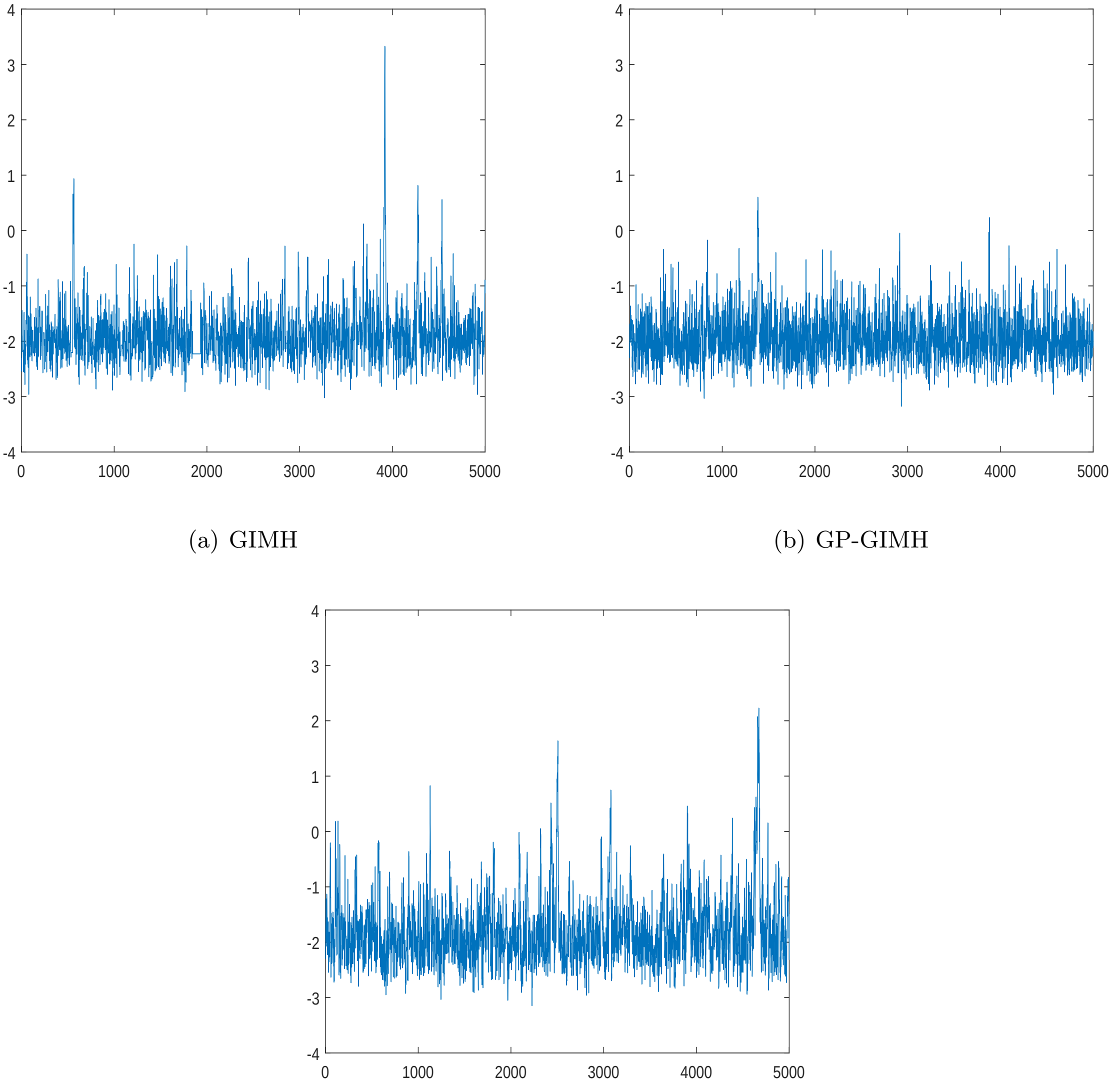
	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_8
θ_1	0.07/0.14/0.08	0.07/0.14/0.10	0.07/0.15/0.09	0.09/0.17/0.10	0.09/0.17/0.09	0.08/0.15/0.10	0.08/0.15/0.10
θ_2		0.07/0.14/0.10	0.07/0.15/0.09	0.09/0.17/0.10	0.09/0.17/0.09	0.08/0.15/0.11	0.08/0.15/0.10
θ_3			0.06/0.12/0.10	0.08/0.14/0.10	0.08/0.14/0.10	0.06/0.11/0.10	0.07/0.12/0.11
θ_4				0.08/0.15/0.10	0.08/0.15/0.10	0.07/0.13/0.11	0.06/0.13/0.10
θ_5					0.08/0.15/0.09	0.08/0.14/0.12	0.09/0.15/0.11
θ_6						0.09/0.14/0.11	0.08/0.15/0.11
θ_7							0.06/0.13/0.12

Table 6: The mean TV distances over the five independent runs of GP-GIMH between each run and the mean of the five runs for the gene network example. Results are shown for different combinations of ϵ and J.

GP-GIMH parameters	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_8
($\epsilon = 1.2$, $J \approx 2000$)	0.044	0.043	0.037	0.026	0.057	0.057	0.038	0.025
($\epsilon = 1.5$, $J \approx 2000$)	0.043	0.043	0.042	0.028	0.064	0.065	0.045	0.029
($\epsilon = 2.0$, $J \approx 2000$)	0.054	0.055	0.040	0.034	0.070	0.072	0.044	0.030
($\epsilon = 1.2$, $J \approx 4000$)	0.027	0.028	0.029	0.027	0.036	0.036	0.029	0.025
($\epsilon = 1.5$, $J \approx 4000$)	0.030	0.028	0.033	0.025	0.046	0.047	0.031	0.020
($\epsilon = 2.0$, $J \approx 4000$)	0.041	0.036	0.033	0.027	0.037	0.040	0.029	0.026

H Trace plots for the gene network example

Trace plots for θ_1 of the gene network example are shown in Figure 29 for the different approaches. The GIMH (Figure 29(a)) and GP-GIMH (Figure 29(b)) results are thinned by a factor of 20 whilst the MCWM results (Figure 29(c)) are thinned by a factor of 10. In Figure 29(a) it is evident that the GIMH method becomes stuck for a large number of iterations around iteration 2000 (after thinning) whilst the GP-GIMH run Figure 29(b) displays generally better mixing. The MCWM method (Figure 29(c)) has a tendency to mix well at it is less affected by over estimated likelihoods unlike GIMH.



(a) GIMH

(b) GP-GIMH

(c) MCWM

Figure 29: Trace plots of θ_1 for the gene network example. (a) GIMH, (b) GP-GIMH (first of the 5 runs with $\epsilon = 1.2$) and (c) MCWM.

I GP residual plots for the gene network example for different values of J and ϵ

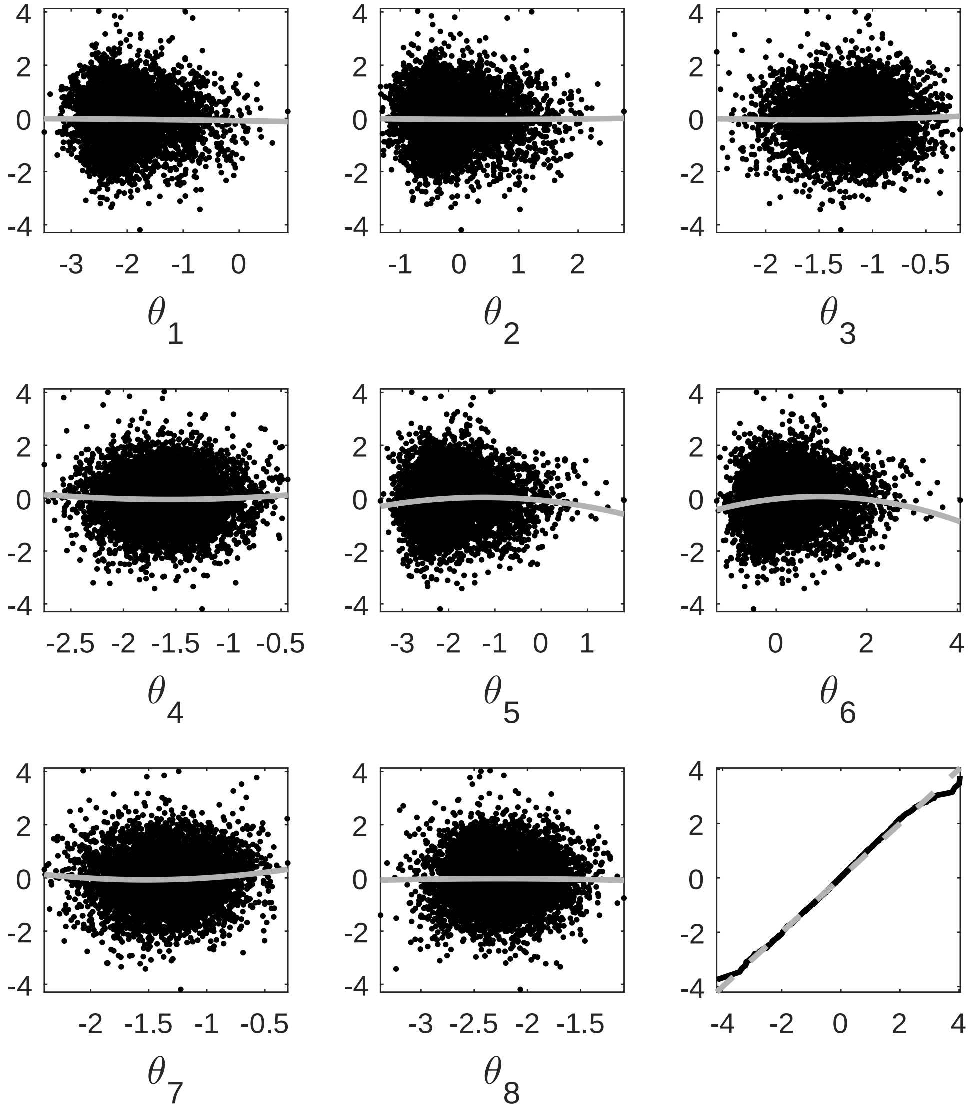


Figure 30: Residual plots of the GP fit for the gene network example when $J \approx 2000$ and $\epsilon = 1.2$. The grey curves show a quadratic fit to the residuals.

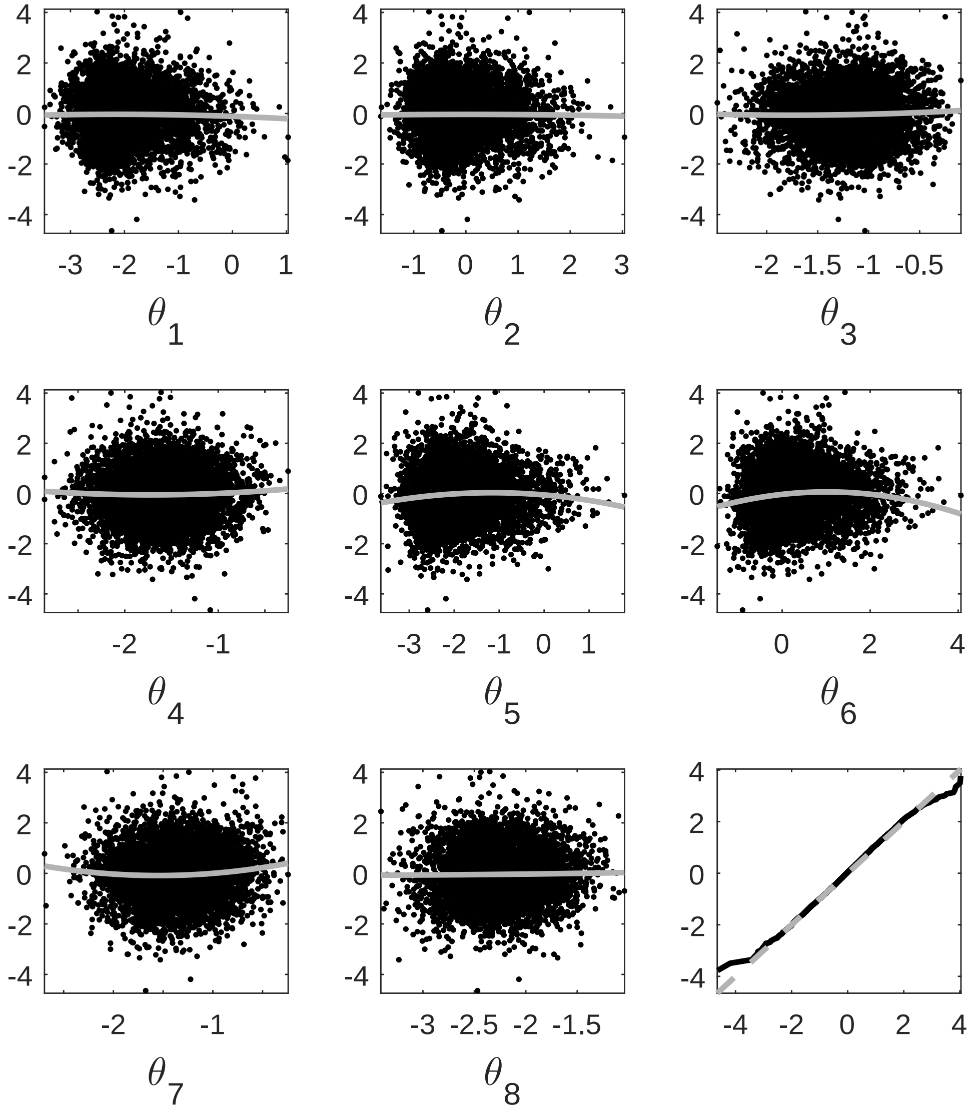


Figure 31: Residual plots of the GP fit for the gene network example when $J \approx 2000$ and $\epsilon = 1.5$. The grey curves show a quadratic fit to the residuals.

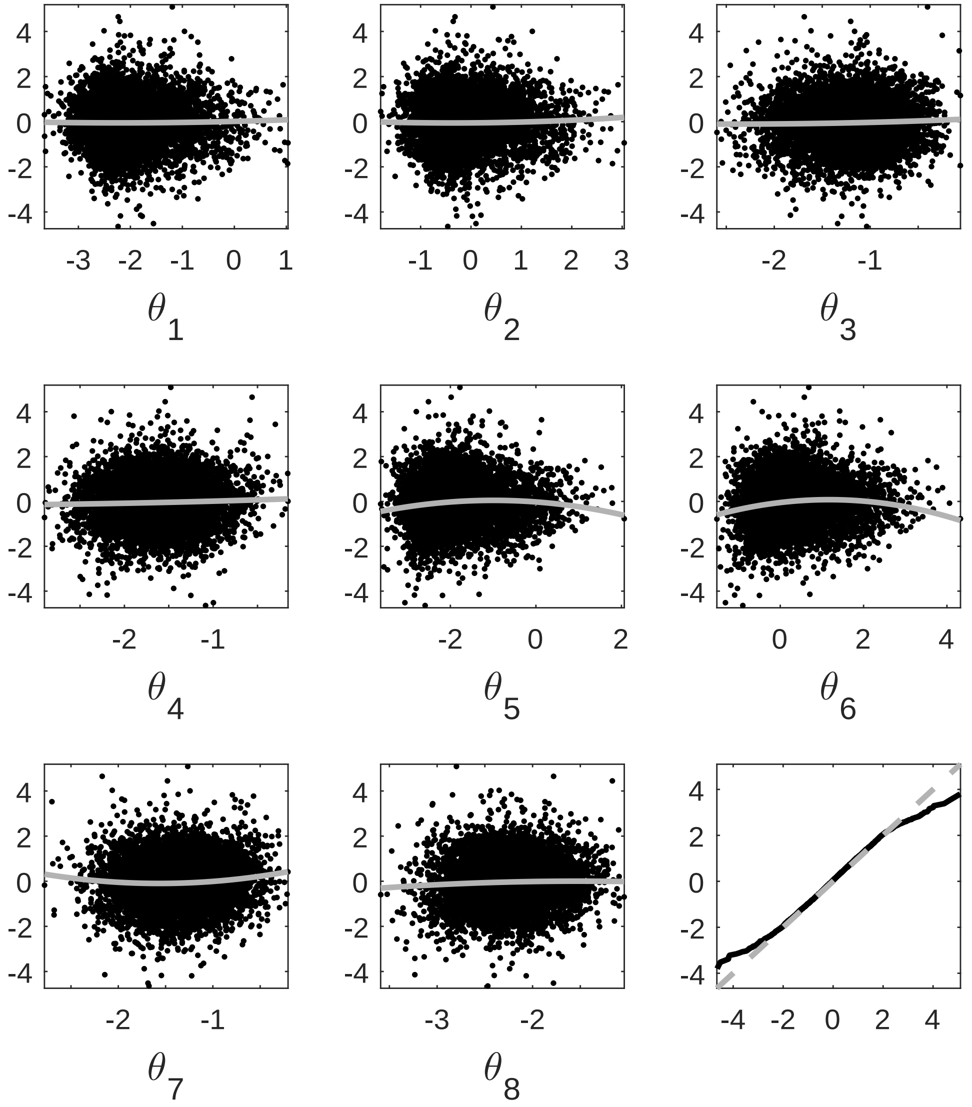


Figure 32: Residual plots of the GP fit for the gene network example when $J \approx 2000$ and $\epsilon = 2.0$. The grey curves show a quadratic fit to the residuals.

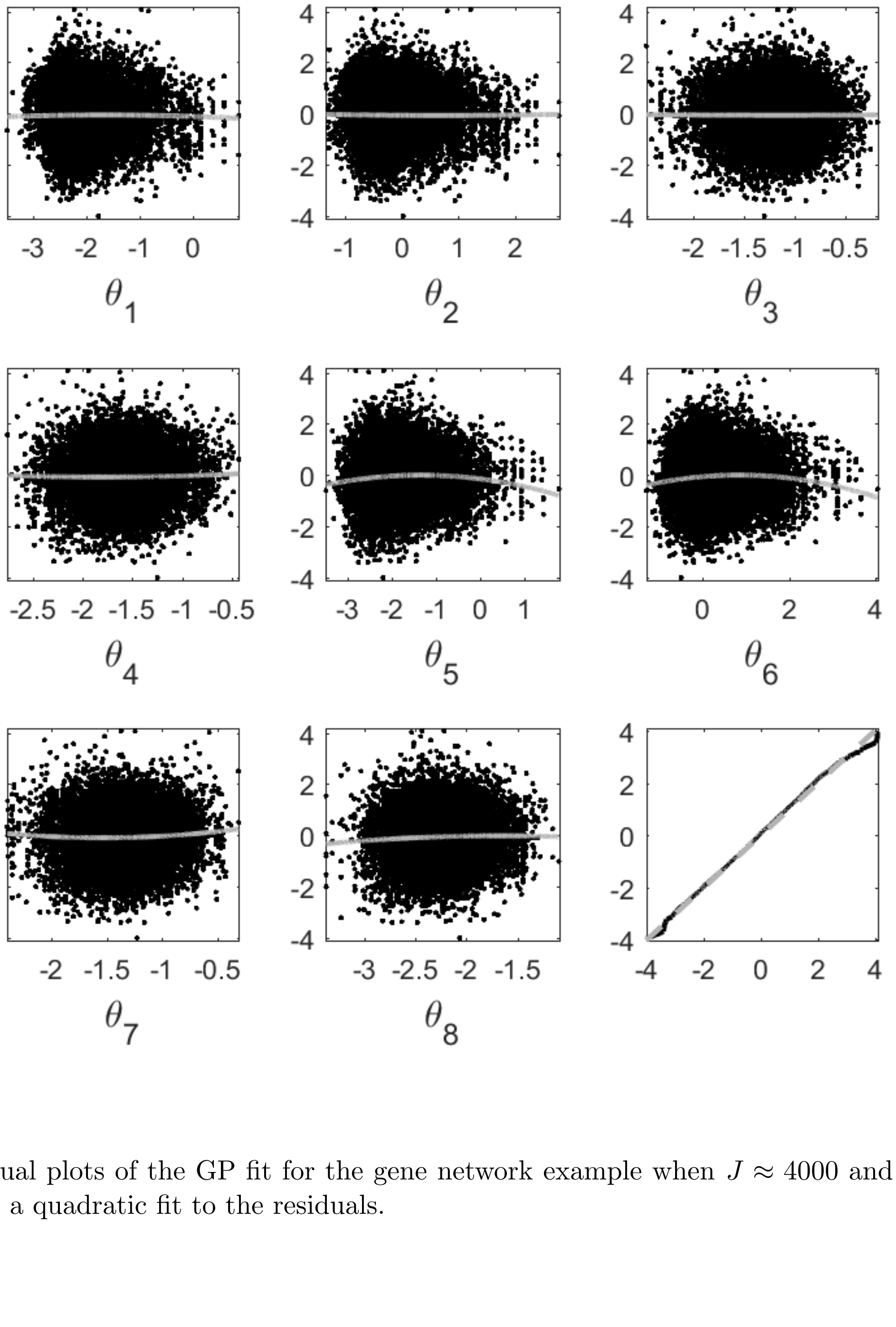


Figure 33: Residual plots of the GP fit for the gene network example when $J \approx 4000$ and $\epsilon = 1.2$. The grey curves show a quadratic fit to the residuals.

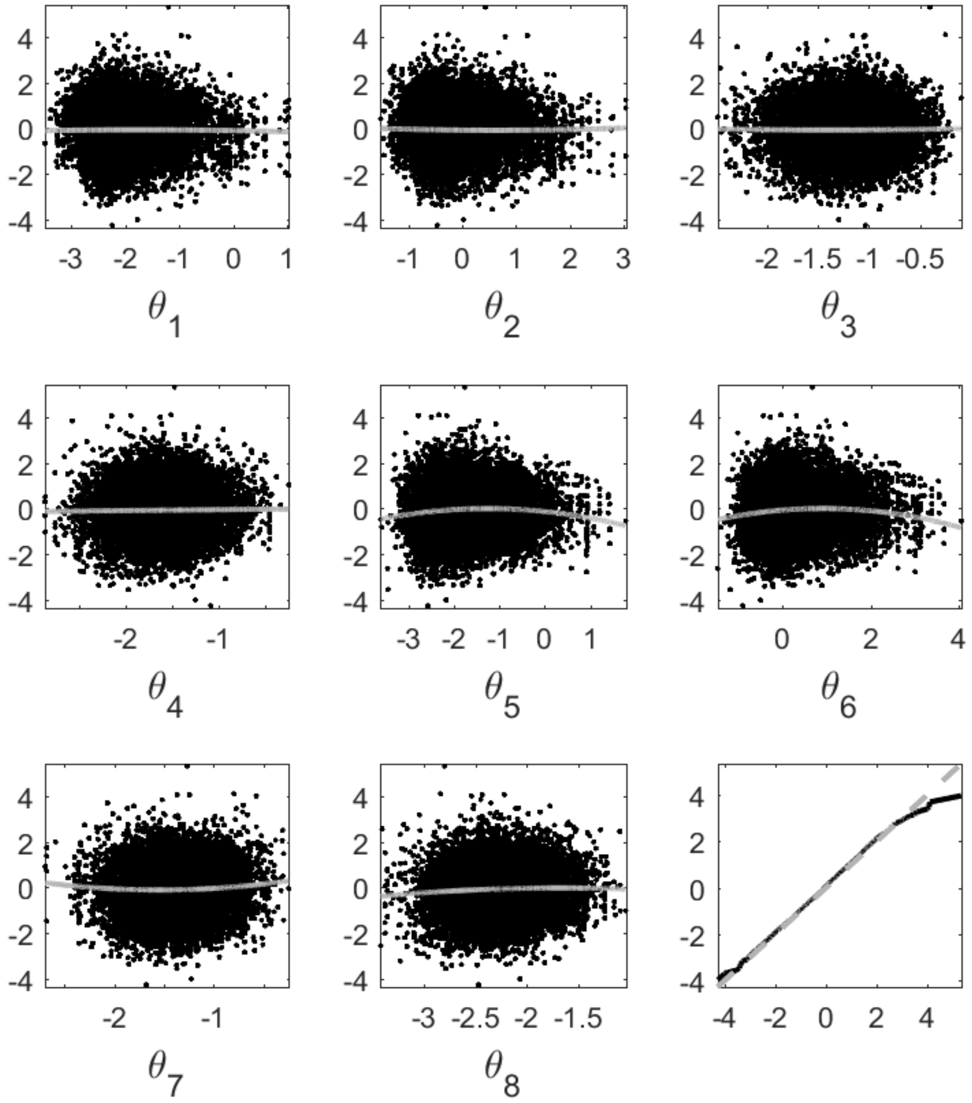


Figure 34: Residual plots of the GP fit for the gene network example when $J \approx 4000$ and $\epsilon = 1.5$. The grey curves show a quadratic fit to the residuals.

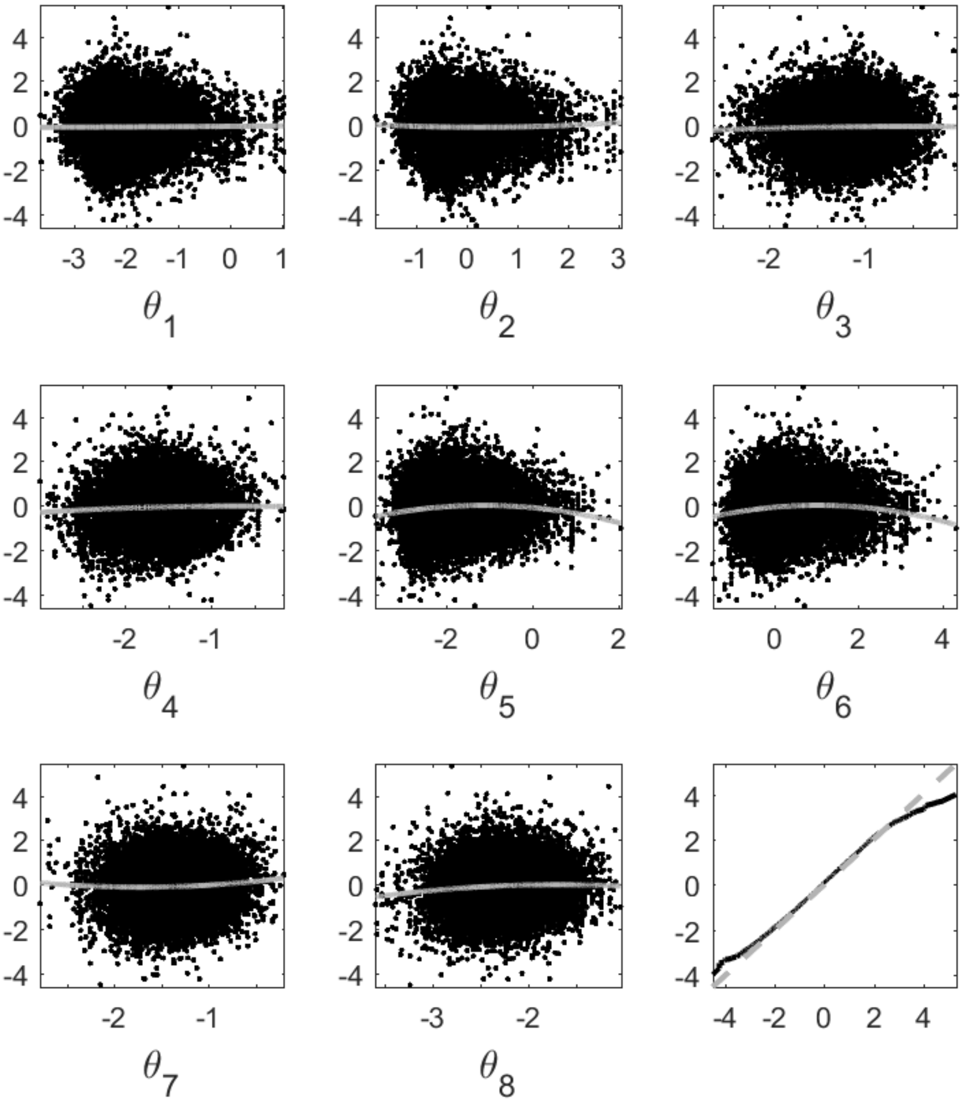


Figure 35: Residual plots of the GP fit for the gene network example when $J \approx 4000$ and $\epsilon = 2.0$. The grey curves show a quadratic fit to the residuals.

J Comparison of GP-GIMH results for the gene network model (with $\sigma = 0.4$) with and without the burn-in

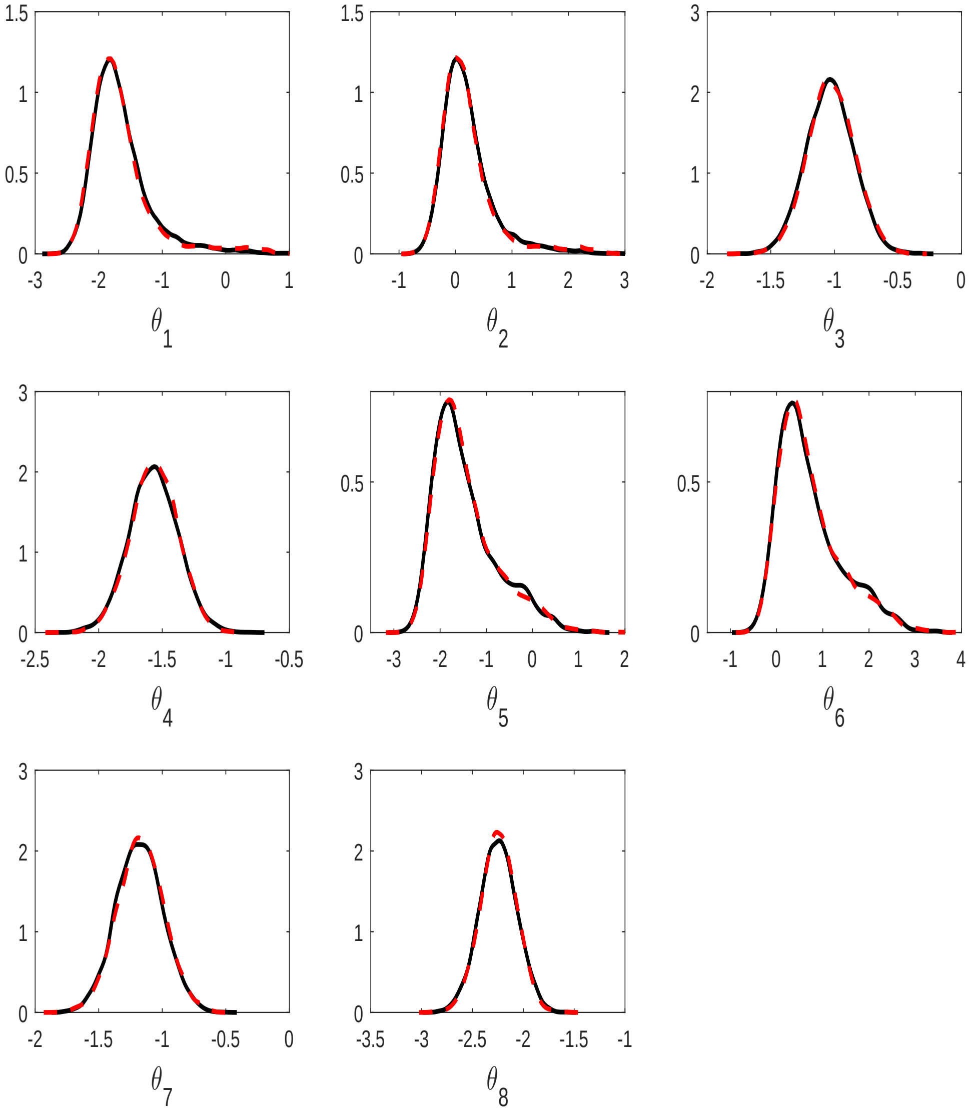


Figure 36: GP-GIMH results for the gene network model with $\sigma = 0.4$ with (black solid) and without (red dash) the burn-in phase.

K Total variation distance calculation

One measure of the difference between two probability distributions with densities f and g is the total variation (TV) distance, which is given by

$$\delta(f,g) = \frac{1}{2}\int_{\boldsymbol{\theta}} |f(\boldsymbol{\theta}) - g(\boldsymbol{\theta})| d\boldsymbol{\theta}.$$

In our context there is not an analytical expression for either f or g, but we are able to generate samples (albeit correlated ones from the output of MCMC methods) from these distributions. We can use these samples and non-parametric techniques to produce density estimates $\hat{f}$ and $\hat{g}$. Then, we consider a regular grid of $\boldsymbol{\theta}$ values where both f and g have non-negligible density in order to approximate the integral needed to compute the TV distance.

Due to the difficulty of estimating densities non-parametrically in high-dimension and also in approximating high-dimensional integrals we consider computing the TV distance over all the one and two dimensional marginals of f and g. Since MCMC samples from joint distributions it also automatically samples from the marginals. Consider the ith and jth component of the parameter vector $\boldsymbol{\theta}$. Consider a regular grid over this two dimensional space with spacings Δ_i and Δ_j in the ith and jth directions, respectively. Then the estimate of the TV distance is given by

$$\hat{\delta}(f,g) = \frac{1}{2}\Delta_i\Delta_j \sum_i \sum_j |\hat{f}(\theta_i,\theta_j) - \hat{g}(\theta_i,\theta_j)|.$$

For one dimensional density estimation we use the *ksdensity* function in MATLAB, which automatically selects a bandwidth. For two dimensional density estimation we use the *ks* package (Duong, 2015) in R and the automatic Matrix bandwidth selection approach of Wand and Jones (1994). We use thinning to reduce the dependence between the MCMC samples when estimating the densities and also choose the grid upon inspection of the approximated densities. We use a larger thinning interval and a coarser grid when computing the two dimensional TV distances to reduce the required computation.

References

Beaumont, M. A. (2003). Estimation of population growth or decline in genetically monitored populations. *Genetics*, 164(3):1139–1160.

Duong, T. (2015). *ks: Kernel Smoothing*. R package version 1.10.0.

Wand, M. P. and Jones, M. C. (1994). Multivariate plug-in bandwidth selection. *Computational Statistics*, 9(2):97–116.

Wilkinson, R. (2014). Accelerating ABC methods using Gaussian processes. *Journal of Machine Learning Research*, 33:1015–1023.