Efficiency of the Pseudo-Marginal Random Walk Metropolis Algorithm

Valerie Bradley, Emmanuelle Dankwa, Hector McKimm

2019-09-24

Abstract

We motivate the need for pseudo-marginal Metropolis Hastings algorithms and present the theory behind such methods. We consider the assumptions that noise of the estimate of the target density is additive, Gaussian and independent of position, and implement the pseudo-marginal random walk Metropolis algorithm under these assumptions for a simple example. The optimal variance in the noise of the estimator of the target density and the acceptance rate are computed and compared to theoretical results.

1 Introduction

Markov Chain Monte Carlo (MCMC) methods are widely used to sample from distributions of interest. MCMC tends to be used in situations where independent samples cannot be drawn from the target distribution; the methods provide dependent samples, which for many purposes are just as useful as independent samples. Many MCMC methods are based on the Metropolis-Hastings algorithm (Gilks, Richardson, and Spiegelhalter (1995)). This sampler has the advantage of only requiring the target denstiy to be known up to a proportionality constant. Despite their flexibility, for traditional M-H sampling, it is still necessary to evaluate the target distribution, up to a proportionality constant. There are a number of settings in which exact evaluation is either too computationally expensive, or even impossible due to missing data. For example, a distribution for which the likelihood contains a complex integral or latent variable.

Pseudo-Marginal Metropolis Hastings (PsMMH) is an adaptation of traditional Metropolis-Hastings that makes it possible to sample from target distributions without exact evaluation (Knape and De Valpine (2012)). Using PsMMH, it is sufficient to unbiasedly *estimate* a distribution. Substituting the exact target density values for unbiased estimates of those values in the acceptance ratio caluclation still results in an algorithm that produces a Markov Chain whose stationary distribution is the target density.

In Section 2, we discuss the basic idea behind the Pseudo-Marginal Metropolis-Hastings algorithm. We then introduce the concept of the Pseudo-Marginal Random Walk Metropolis-Hastings algorithm which has the distinct characteristic of a Gaussian proposal function. Section 3 looks closely at the noise term introduced in the estimation of the target density and discusses the main assumptions used in the study. In Section 4, we examine the efficiency of the algorithm by considering two important factors : the jump distance and the computing time. In section 5, drawing upon the discussions in the previous sections, we implement an actual example with the multivariate normal distribution as the target density. Here, we also check optimization of the algorithm and check that the assumptions hold. We conclude by mentioning some application results from other literature and proposing directions for further research.

2 The pseudo-marginal random walk Metropolis algorithm

2.1 Metropolis-Hastings

Let $X \subseteq \mathbb{R}^d$ be a state space and $\pi(\boldsymbol{x})$ be the density of some distribution on X that we want to explore. The Metropolis-Hastings algorithm draws dependent samples from the *target density* $\pi(\boldsymbol{x})$ by constructing a

Markov Chain with limiting distribution $\pi(\boldsymbol{x})$. New samples $\boldsymbol{x}^*$ are proposed, given the current position $\boldsymbol{x}$, via some density $q(\boldsymbol{x}, \boldsymbol{x}^*)$. The new sample is accepted with probability:

$$\alpha(\boldsymbol{x}, \boldsymbol{x}^*) = \min\left\{1, \frac{\pi(\boldsymbol{x}^*)q(\boldsymbol{x}^*, \boldsymbol{x})}{\pi(\boldsymbol{x})q(\boldsymbol{x}, \boldsymbol{x}^*)}\right\}.$$

If the proposed point is rejected, then the next sample is taken to be the same as the previous.

2.2 Pseudo-Marginal Metropolis Hastings

Sometimes it is compuatationally infeasible to evaluate the target density $\pi(\boldsymbol{x})$. In these situations, PsMMH can be used instead. The method uses an approximation $\hat{\pi}_{\boldsymbol{v}}(\boldsymbol{x})$, which depends on a variable $\boldsymbol{v}$ whose distribution is $q_{aux}(\boldsymbol{v} \mid \boldsymbol{x})$. The approximation of the target must satisfy:

$$\mathbb{E}_{q_{aux}}[\hat{\pi}_{\boldsymbol{v}}(\boldsymbol{x})] = \pi(\boldsymbol{x})$$

The proposal density $q(\boldsymbol{x}, \boldsymbol{x}^*)$ is used to sample a new point $\boldsymbol{x}^*$. This point is in turn used to attain a new value for the auxilliary variable: $\boldsymbol{v}^*$. This pair is then either jointly accepted or rejected, according to the acceptance probability:

$$\alpha = \min\{1, \frac{\hat{\pi}_{\boldsymbol{v}^*}(\boldsymbol{x}^*)q(\boldsymbol{x}^*, \boldsymbol{x})}{\hat{\pi}_{\boldsymbol{v}}(\boldsymbol{x})q(\boldsymbol{x}, \boldsymbol{x}^*)}\}$$

2.3 Pseudo-Marginal Random Walk Metropolis Hastings

In the case of the Pseudo-Marginal Random Walk Metropolis Hastings (PsMRWM) algorithm, the proposal function, $q(\boldsymbol{x}, \boldsymbol{x}^*)$, is Gaussian. That is:

$$\boldsymbol{X}^* = \boldsymbol{x} + \lambda \boldsymbol{Z}$$

with $\boldsymbol{Z} \sim \boldsymbol{N}(\boldsymbol{0}, \boldsymbol{I})$. In other words, we simulate a proposal point from a normal distribution centred at the current position of the Markov chain:

$$\boldsymbol{X}^* \sim \boldsymbol{N}(\boldsymbol{x}, \lambda \boldsymbol{I}).$$

Hence the term *Random Walk* in the algorithm's name. The proposal function will generate points which randomly explore the sample space and the accepted points (obtained using an acceptance probability) are considered as being sampled from the target density.

3 Assumptions on the Noise

In this paper, the log-density of the target will be used. We define the *noise* term at the current point in the Markov chain to be:

$$W := \log \hat{\pi}_{\boldsymbol{v}}(\boldsymbol{x}) - \log \pi(\boldsymbol{x}).$$

Consider in addition the noise in the estimated log-target at the proposal, defined as:

$$W^* := \log \hat{\pi}_{\boldsymbol{v}^*}(\boldsymbol{x}^*) - \log \pi(\boldsymbol{x}^*).$$

For the sake of deriving results on the efficiency of the algorithm, this paper will make two main assumptions. The PMRWM algorithms outlined in Sherlock et al. (2015) make use of these assumptions which when adhered to, result in the *optimal* results. To be clear, these assumptions are not necessary conditions for the functioning of the PsMRWM algorithm, but are made solely in order to make progress on the theory of the algorithm's efficiency.

First, we replicate the algorithms holdng fast to these assumptions. We then modify the assumptions and evaluate the impact on the algorithm outcome. We also evaluate the impact with respect to computational time and speed of convergence to the target density. The assumptions as proposed are discussed below:

Assumption 1: The Markov chain $\{(\boldsymbol{X}_i, W_i)\}_{i>0}$ is stationary and the distribution of W^* is independent of the the proposal $\boldsymbol{x}^*$.

In other words, when considering the Markov chain, we assume that the chain is long enough for its limiting distribution to have been reached and in addition that the initial "burn-in" period has been discarded. Thus all that is left are values which can be thought of as samples from the target distribution. Note that in many scenarios it may in fact be unrealistic to assume that the noise is independent of the propsal point: $q_{aux}(\boldsymbol{v} \mid \boldsymbol{x}) = q_{aux}(\boldsymbol{v})$. However these assumptions are made in this instance in order to derive theoretical results on the algorithm's efficiency.

Assumption 2: For every $\boldsymbol{x}$ in X, we have an unbiased estimator $\hat{\pi}_{\boldsymbol{v}}(\boldsymbol{x})$ of $\pi(\boldsymbol{x})$, such that $\log\hat{\pi}(\boldsymbol{x})$ follows a Gaussian distribution with variance $\sigma^2 > 0$. In addition, the expected one-step computing time of the estimator is inversely proportional to σ^2.

To understand the first part of this assumption, note that it applies for every $\boldsymbol{x}$ in X. Each $\hat{\pi}_{\boldsymbol{v}}(\boldsymbol{x})$, where $\boldsymbol{x}$ is fixed, has an associated distribution. The value of $\hat{\pi}_{\boldsymbol{v}}(\boldsymbol{x})$ changes due to changes in the auxilliary variable $\boldsymbol{v}$. We are assuming that the distribution of $\hat{\pi}_{\boldsymbol{v}}(\boldsymbol{x})$ is such that $\log \hat{\pi}_{\boldsymbol{v}}(\boldsymbol{x})$ follows a Gaussian distribution.

In the next section, it will be shown in a simulation example how it was ensured that the distribution of the noise was Gaussian and independent of the proposal point.

4 Definition of Efficiency

The Euclidean Expected Square Jump Distance (ESJD) is often used to measure the efficiency of MCMC algorithms. This quantity, defined as $\mathbb{E} \mid\mid \boldsymbol{X}_{i+1} - \boldsymbol{X}_i \mid\mid^2$, describes how large the distances between consecutive points in the Markov chain can be expected to be. If the ESJD is large, this can be intrepted as the sampler exploring the sample space well. Alternatively if the ESJD is small, the sampler is struggling to move around the sample space. Reasons for the latter may be that the acceptance probability is very small, leading to the the chain making few moves; or perhaps the proposal function generates proposals that are very close to the original point. Sherlock et al. (2015) shows that the ESJD iof a chain decomposes into $l^2 \times \alpha(l) = J(l)$, where l is the proposed jumping distance and $\alpha(l)$ is the acceptance probability of that jump.

Normally when considering the efficiency of a standard Random Walk Metropolis algorithm, it would be enough to only consider the ESJD, because expected computation time doesn't depend on the choice of tuning parameters, such as the scaling parameter for the proposal function.

When it comes to the PsMRWM algorithm however, computation time is no longer independent, but rather inversely proportional to the variance in the estimator of the target density. In evaluating the efficiency of the PMRWM algorithm, we must therefore account for this additional computation time. Therefore, Sherlock et al. (2015) proposes considering the joint optimization of the ESJD, and the distribution of the noise in the posterior estimate.

Efficiency is now defined as:

$$\text{Efficiency} = \frac{\text{Expected Square Jump Distance}}{\text{Expected one-step computing time}}$$

Where the expexcted one-step computing time can be estimated by:

$$\text{Expected one-step computing time} = \frac{\text{Total computing time}}{\text{Number of iterations (n)}}$$

5 Implementation with an example

Consider the task of simulating from a multivariate normal distribution with mean zero and a random covariate matrix. To allow for better visualisation of the algorithm's performance, we will consider the two dimensional case. Let the target density be:

$$\pi(\boldsymbol{x}) = \boldsymbol{N}_2(\mathbf{0}, \boldsymbol{\Sigma}),$$

with

$$\Sigma = \begin{pmatrix} 1.0 & 2.0 \\ 2.0 & 5.0 \end{pmatrix}$$

In this case, one could easily exactly the target distribution, therefore standard Metropolis-Hastings could be used. However, this example however is meant to be illustrative. It will demonstrate how even when the target distribution can only be estimated, it is still possible to sample from the target distribution using PsMRWM.

The code below initializes the chain at a value $\boldsymbol{x} = (x_1, x_2)$ randomly sampled from a 2-dimensional Normal distribution with mean (3,1) and covariance matrix $I^{(d)}$. This distribution was selected such that the initial proposal was likely to be far away from the mean of the true distribution, so that we could verify that the chain was successfully making a random walk toward the correct target distribution. We then implemented PsMRWM to sample from the specified target distribution.

```r
set.seed(1234)

#specify target covariance matrix
covmat = matrix(c(1, 2, 2, 5), byrow = T, nrow = 2)

# define initial value of x
x = rmvnorm(n = 1, mean = c(3,-1), sigma = diag(2))

# Run Pseudo-Marginal Random Walk Metropolis-Hastings algorithm to sample from target
example = runMHmvnorm(theta = x                # initial value of x
                            , covmat = covmat # target covariance matrix
                            , N = 10000       # numer of iterations
                            , sigma = 2       # variance of proposal distribution
                            , sigma_aux = 1   # variance of noise
                      )
save(example, file = "example_chain.RData")
```

5.1 Checking assumptions

Figure 1 below allows us to visually verify the assumptions we made about the noise in the estimates of the density of the target distribution. The first plot shows the distribution of the additive noise, $W*$, used to simulate Monte Carlo the target density at each proposal point $\boldsymbol{x}*$ relative to the actual log density of the target distribution $\hat{\pi}(\boldsymbol{x})$, calculated analytically. The plot allows us to visually confirm that the assumptions we made about the noise in the target densities estiamtes have held. More specifically, the noise appears to be distributed relatively normally with a mean of 0, satisfying Assumption 2. The next two plots show the additive noise relative to each dimension of $(x*)$. The noise appears to be distributed independently of $\boldsymbol{X}*$, thus also satisfying Assumption 2.

5.2 Results

Now that we have verified that our assumptions have held, we can begin to examine the actual sample distribution. Figure 2 shows a heatmap of the sample distribution, overlaid with contours of the actual target density. The plot clearly shows that the sample distribution has correctly captured the high correlation between the two elements of the target distribution, higher variance of the second component and mean of the target distribution, $(0, 0)$.

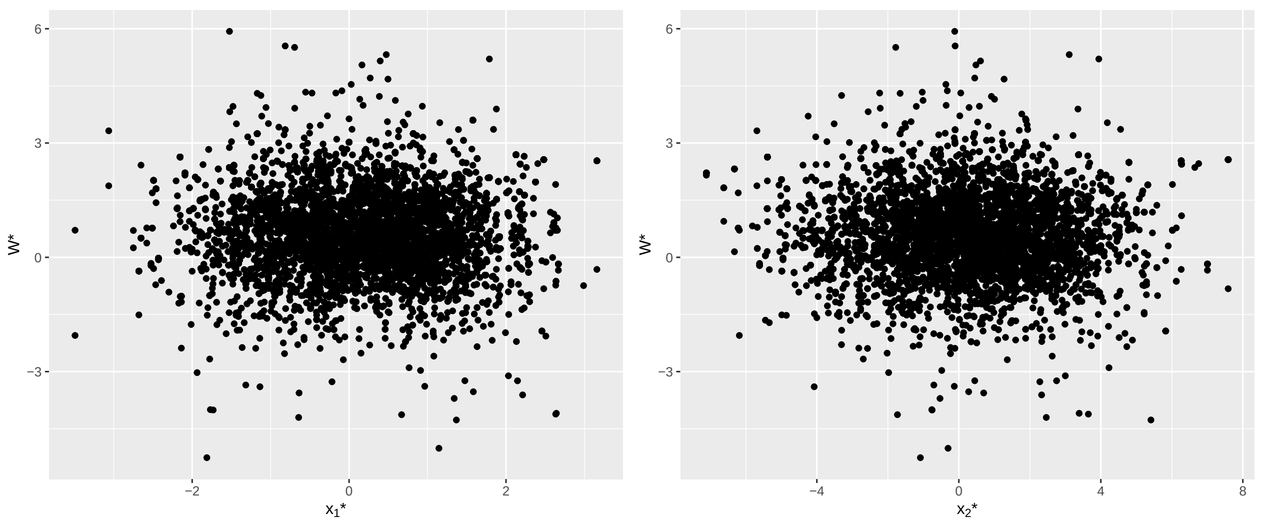

Figure 1: Noise in estimates of target distribution

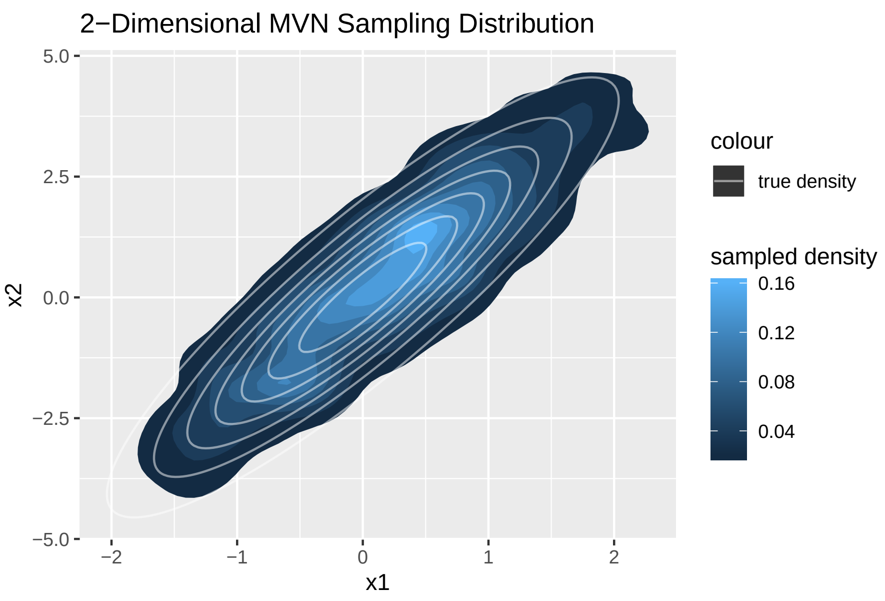

Figure 2: Sampled and target densities

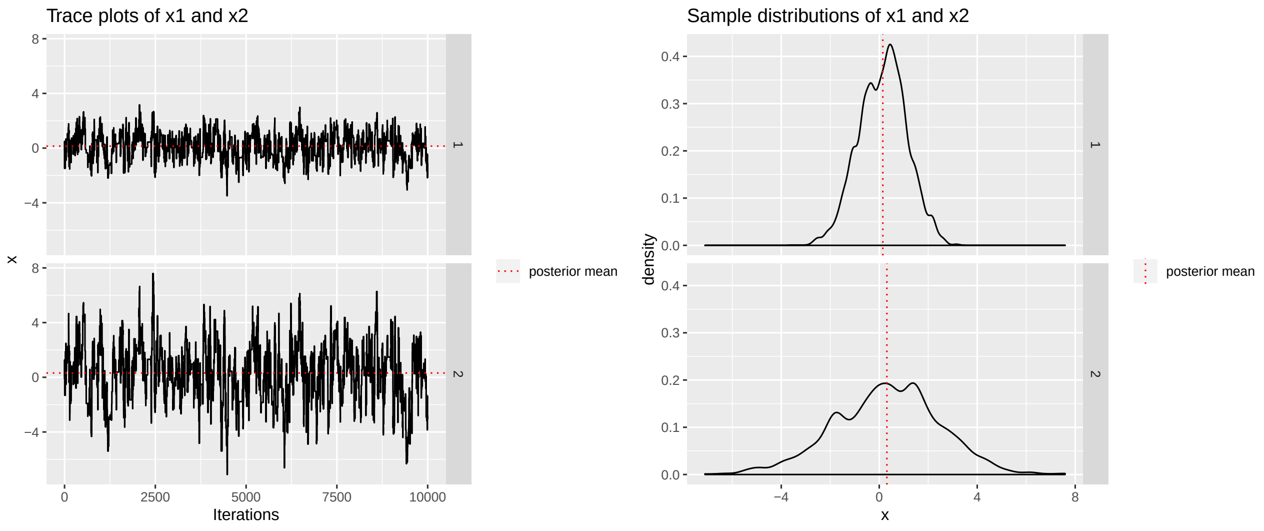

Figure 3: Trace plots and posterior marginal densities of the target multivariate normal distribution

We can verify that the chain sampled accurately from the target density by examining the empirical covariance matrix. The empirical covariance matrix, shown below, nicely approximates the target covariance matrix, despite significant noise in target density estimation.

```
cov(example$theta)
#>           [,1]      [,2]
#> [1,] 0.9485873 1.865161
#> [2,] 1.8651605 4.652677
```

Similarly, the the empirical mean of the resulting chain is approximately that of the target distribution, $(0, 0)$.

```
apply(example$theta, mean, MARGIN = 2)
#> [1] 0.1461078 0.3122610
```

We can also examine the chains' trace plots. Trace plots are helpful tools for evaluting how well MCMC chains are mixing, or exploring the parameter space, and for identifying periods of 'burn in', or when the chain has not yet reached its stationary distribution. The plots in Figure 3 shows the chain mixing reasonably well, though later will will discuss how the chains' exploration may be optimised by examining a scaled version of ESJD.

Lastly, we look at the marginal distributions of each component of $\boldsymbol{x}$. In the second set of plots of Figure 3, we can see that the marginal densities of each component of $\boldsymbol{x}$ are roughly normal with mean 0 and different variances, as expected.

5.3 Efficiency

Recall that efficiency is defined as:

$$\text{Efficiency} = \frac{\text{Expected Square Jump Distance}}{\text{expected one-step computing time}}.$$

Let l^2 be the *expected squared proposed jumping distance.* That is, how far we can expect the new proposal point to be from the current point in the chain.

Define $J(l)$ to be the expected squared jump distance. This takes into account the fact that not all jumps are accepted and is found with the equation:

$$J(l) := \alpha(l)l^2,$$

where $\alpha(l)$ is the expected acceptance probability.

We assume that the Expected one-step computing time is inversely proportional to σ^2, therefore the equation for Efficiency is:

$$\text{Eff}_{\sigma^2}(\lambda) = \sigma^2 \times J(l)$$

Where σ^2 is the variance of the noise.

In order to calculate the efficiency in practice, since we do not know the exact expected squared jump distance, we estimate the quantity. Let $l^{(i)2}$ be the proposed squared jumping distance for iteration i. Let $\alpha^{(i)}(l)$ be the acceptance probability for that iteration. Then $J^{(i)}(l)$, the squared jump distance in that iteration, can be calculated. Averging over $J^{(i)}(l)$ for all iterations, we find an estimate of $J(l)$.

For this example chain, $\boldsymbol{Eff}_{\sigma^2}(l)$ is equal to 0.4164. However, this number alone is relatively meaningless. Instead l, and σ^2 should be chosen to maximize $\boldsymbol{Eff}_{\sigma^2}(\lambda)$.

5.4 Optimization

The code below implements a simulation to recover the optimal value of l. We run the example above for 30 values of l, equally distributed between 0.05 and 6, and 20 values of σ, equally distributed between 0.05 and 1. The parameter space of l and σ we defined is based on that explored in Sherlock et al. (2015), which was shown to contain the optimal values.

```r
l = seq(0.05, 6, length.out = 30)
noise = seq(0.05, 1, length.out = 20)
params = expand.grid(l = l, noise = noise)
# #all_chains = list()
# for(p in 1:nrow(params)){
#
#   temp = runMHmvnorm(theta = x
#               , covmat = covmat
#               , N = 4000
#               , sigma = (params[p,]$l)^2       # variance of proposal distribution
#               , sigma_aux = params[p,]$noise   # variance of noise
#               )
#
#   all_chains[[length(all_chains) + p]] = temp
# }

#save(all_chains, file = 'simulation_chains.RData')
```

By running PsMRWM on the above example for a range of values of proposed jump distance, l, and variance of the noise, σ^2, we can attempt to identify values of each parameter that maximize the efficiency of the algorithm. Figure 4 below replicates Figure 1 from Sherlock et al. (2015), showing that efficiency of the PsMRWM is highly dependent on proposed jump size l, and much less dependent on the standard deviation of the noise, σ.

```
#> `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

Furthermore, the second figure shows the profile efficiency of the PsMRWM relative the the proposal size, which we have determined to have a larger impact on overall efficiency of the algorithm. By fitting a smooth function to the observed values, we can empirically estimate the optimal value of l, l_{opt}, which we find to be `l_opt`. While this is not exactly identical to the value of l_{opt} determined by Sherlock et al. (2015), it is relatively close, and we are confident that given more simulation runs we would continute to converge to their findings.

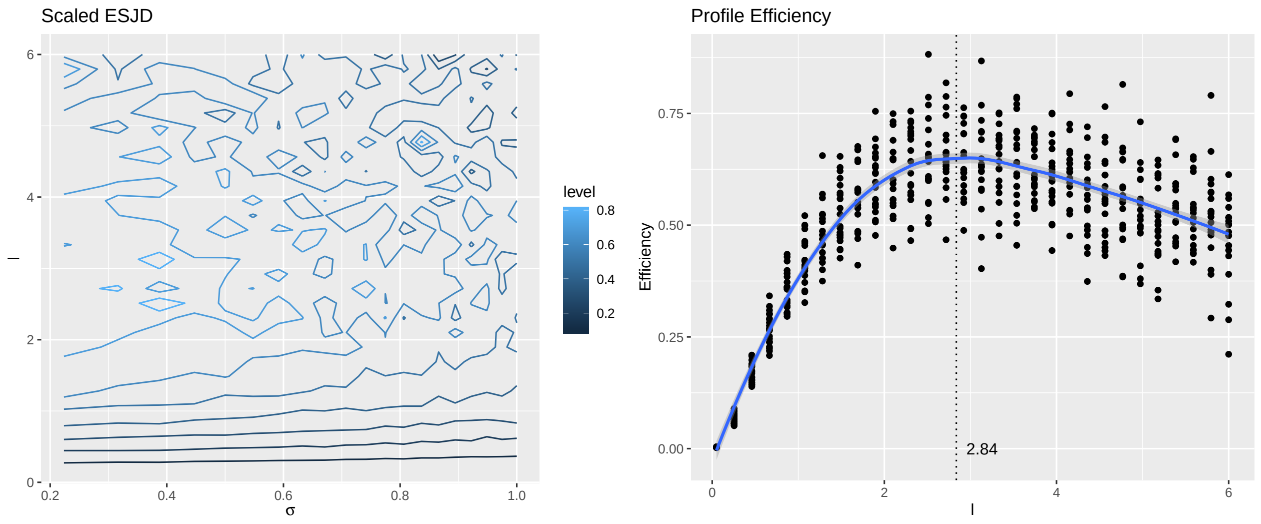

Figure 4: Left: Contour plot of efficiency relative to proposal size and standard deviation of the noise, Right: the profile efficiency relative to the proposal size

6 Conclusion

In conclusion, we have examined the PsMRWM for the multivariate normal distribution, considering the bivariate case. We implemented the algorithm with an example and obtained results which showedapproximate convergence to the target distribution, under some assumptions. We also analysed the efficiency of the algorithm, considering both jump size and computation time. Finally, we were able to approximately replicate the the optimal proposal jump size, l established in Sherlock et al. (2015).

In further research, we could consider evaluating the robustness of the assumptions used in the algorithm and examine whether modifying them could have any impact on convergence to the target distribution, or the optimal value of l that we recovered here.

References

Gilks, Walter R, Sylvia Richardson, and David Spiegelhalter. 1995. *Markov Chain Monte Carlo in Practice*. CRC press.

Knape, Jonas, and Perry De Valpine. 2012. "Fitting Complex Population Models by Combining Particle Filters with Markov Chain Monte Carlo." *Ecology* 93 (2): 256–63.

Sherlock, Chris, Alexandre H Thiery, Gareth O Roberts, Jeffrey S Rosenthal, and others. 2015. "On the Efficiency of Pseudo-Marginal Random Walk Metropolis Algorithms." *The Annals of Statistics* 43 (1): 238–75.