Adaptive Metropolis Hastings Algorithms

Tom Jin Xiao Yu Lu

October 23, 2014

Abstract

We present the R package 'AdaptiveMetropolis', which implements several adaptive Metropolis Hastings algorithms. These adaptive algorithms use various strategies to overcome a lack of knowledge about the target distribution by adapting the proposal distribution with the information inferred from sampling.

1 Introduction

```
require(AdaptiveMetropolis)
```

This package implements a sequence of adaptive Metropolis Hastings algorithms that build on the each other to adapt to more complex distributions and one algorithm to handle nested models.

1.1 Related Work

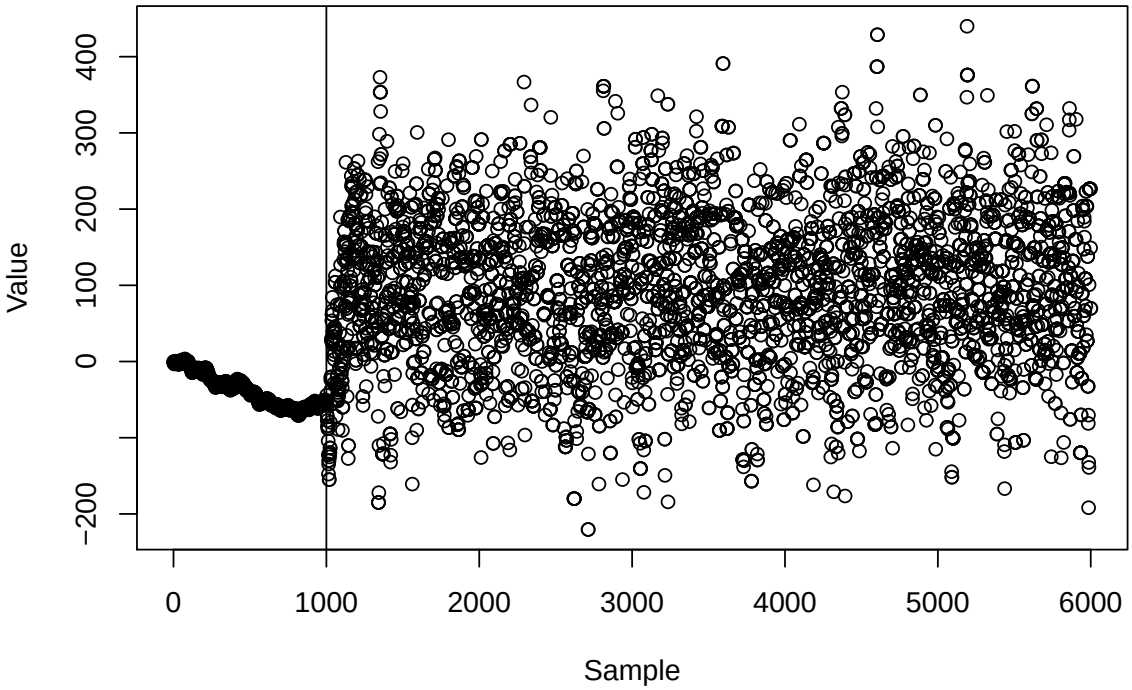
2 Adaptive MCMC Algorithms

2.1 Adaptive Metropolis

The Adaptive Metropolis algorithm as described by Haario et al. (2001) is the first in a family of Metropolis Hastings samplers that foregoes reversibility in the Markov chain in favour of continuous updates to the proposal distribution. Adaptation allows for distributions to be explored much quicker than standard Metropolis Hastings with a poorly tuned proposal distribution.

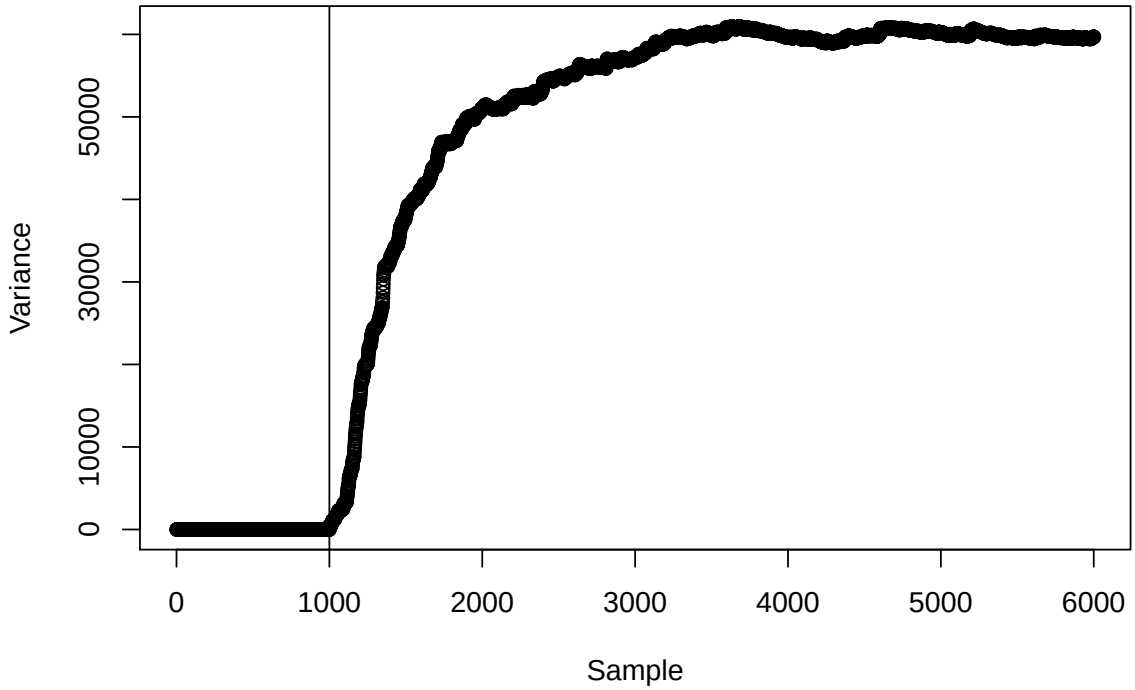
For this algorithm it is essential that a burn in of some length is run to prevent trapping the chain in a lower dimensional subspace. This happens when the initial jumps are very close together and very correlated causing subsequent samples to be highly correlated until the chain breaks free after several thousand iterations. By running a burn in phase of random walk Metropolis Hastings the chain has had the opportunity to explore a little and adapt much faster.

```
data <- AM(function(x) {dnorm(x, 100,100)}, 5000, 0, 1)
plot(data)
```



In this example run no knowledge of the target distribution is assumed and the proposal distribution is set up as a univariate Gaussian with mean 0 and variance 1 but unknown to the sampler target distributions is a univariate Gaussian with mean 100 and variance 100. The sampler will have to find the mode and adapt its variance in order to explore it effectively.

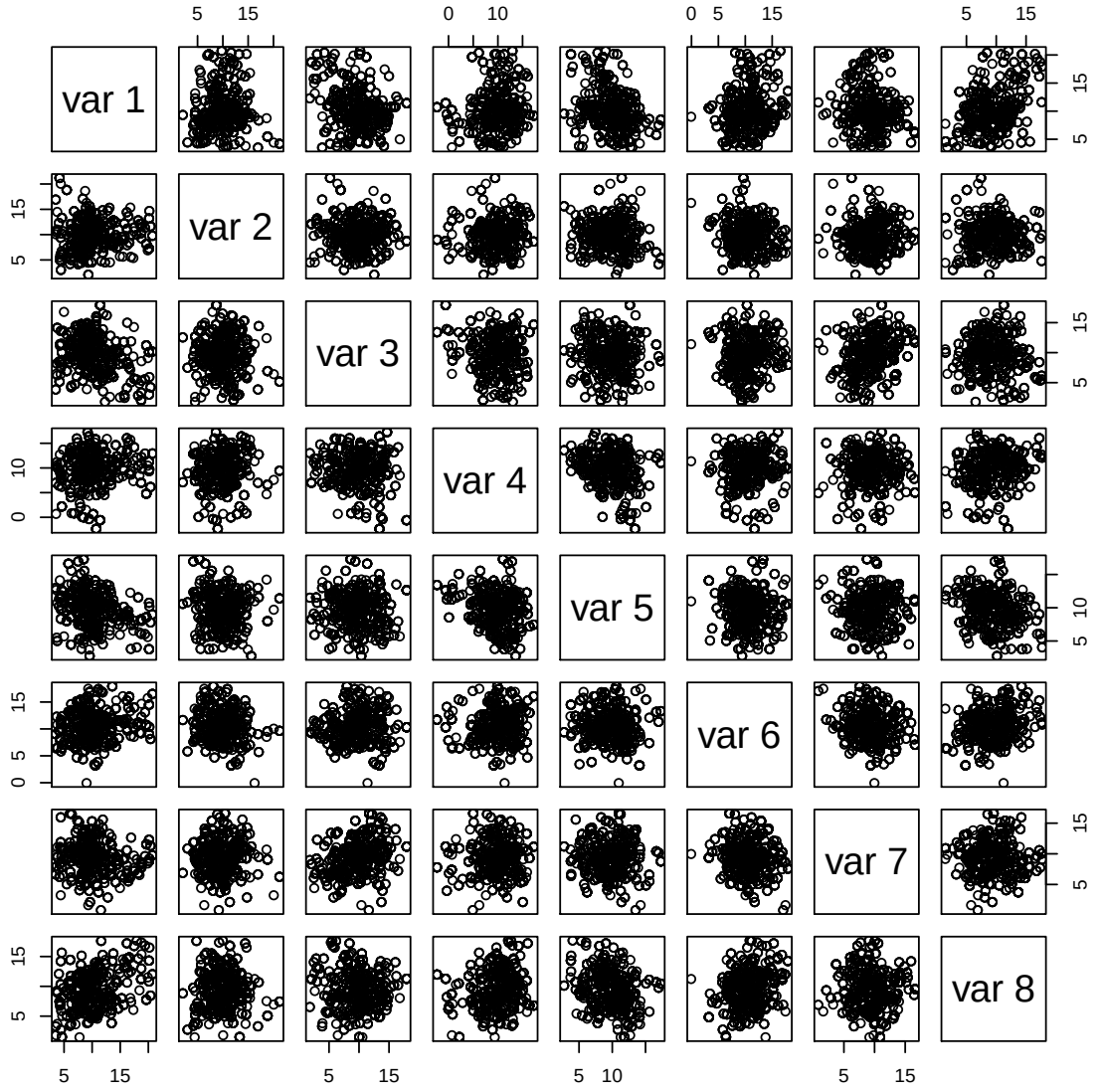
Normally the burn in samples are discarded due to poor mixing. They can be observed on the left of the vertical line at sample 1000. As soon as the adaptation begins the varaince exploads and slowly tends towards variance 50000. Within a small number of iterations much of the mode has been explored.



2.2 Adaptive Scaling Within Adaptive Metropolis

An adaptive scaling with adaptive Metropolis implementation based on Andrieu and Thoms (2008) scales its variance matrix in order to achieve a 0.23 acceptance ratio.

```
library(mvtnorm)
data <- ASWAM(function(x) {dmvnorm(x, rep(10, 8), 10*diag(8))},
              1000, rep(0, 8), diag(8))
pairs(data)
```



2.3 Robust Adaptive Metropolis

2.3.1 Introduction

The Robust Adaptive Metropolis Algorithm (RAM) is an extension of the normal Metropoliis-Hasting Algorithm, where it estimates the shape of the target distribution and coerces the acceptance rate simultaneously, by adapting the proposal covariance structuring according to the data one has simulated. The adaptation rule is computationally simple adding no extra cost compared with the AM algorithm.

2.3.2 Algorithm

Suppose the proposal density q is spherically symmetric. Let $S \in \mathbf{R}^{d\times d}$ be a lower-diagonal matrix with positive diagonal elements, $\{\eta_n\}$ be a step size sequence decaying to zero, α_* be the target mean acceptance probability. The RAM algorithm is defined as:

for n = 1,2,..., iterate:

- compute $X := X_{n-1} + S_{n-1}U_n$, where $U_n \sim q$ is an independent random vector.
- with probability $\alpha_n := \min(1, \pi(X)/\pi(X_{n-1}))$, set $X_n = X$ and $X_n = X_{n-1}$ otherwise.

- update S_n through the equation:

$$S_n S_n^T = S_{n-1}(I + \eta_n(\alpha_n - \alpha_*)\frac{U_n U_n^T}{\|U_n\|^2})S_{n-1}^T,$$

where $I \in \mathbf{R}^{d\times d}$ stands for the identity matrix.

Note there is a unique S_n satisfying 3 above, since the right hand side is symmetric and positive definite.

2.3.3 Inputs and outputs

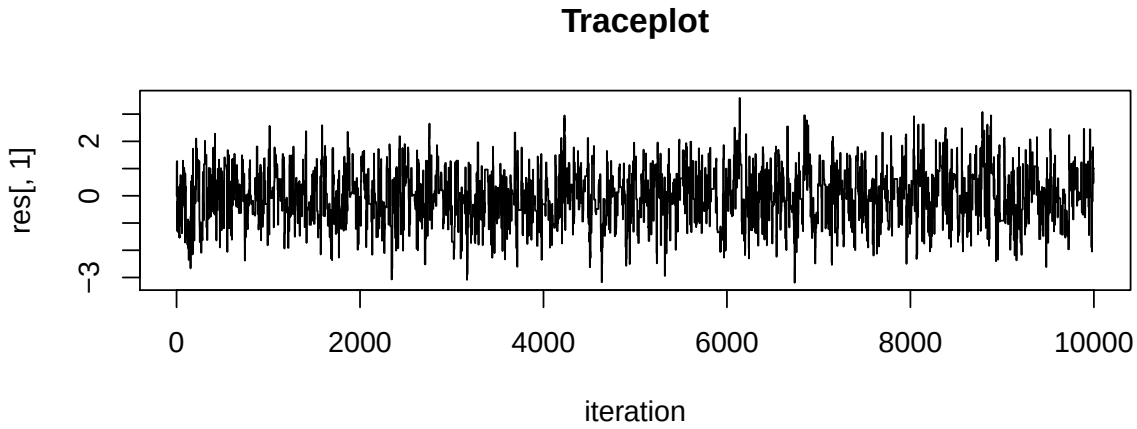
the RAM algorithm takes 4 inputs including the target distribution π (target), the number of iterations (N), the initial state (init.state) and the initial covariance structure for the MVN proposal (init.cov). The output is of a matrix form, recording the simulated path.

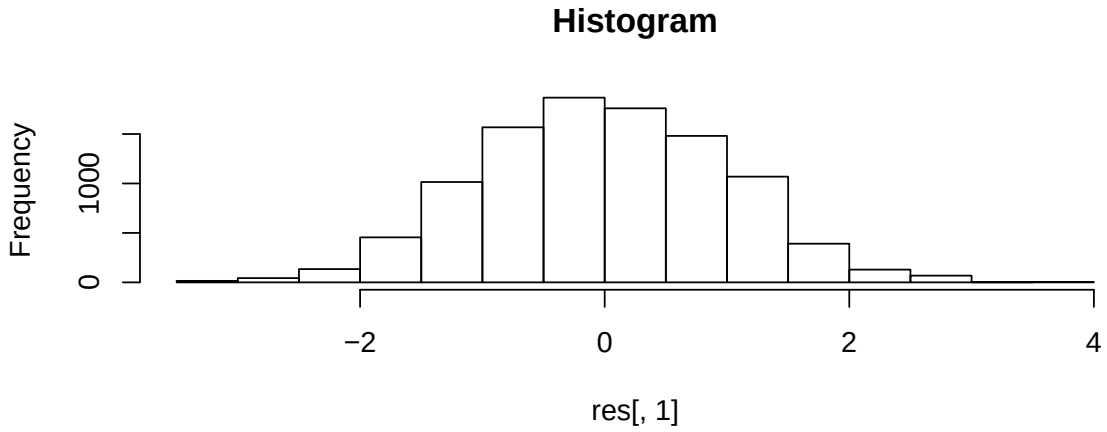
2.3.4 Example

We implement a toy example here. We set the initial state to be the origin in a 3-dimensional space, and the target distribution is a standard MVN with identity covariance structure, and we iterate for 10,000 times:

```
library(mvtnorm)
res <- RAM(function(x) {dmvnorm(x, rep(0, 3), diag(3))}, 10000,
           c(0,0,0), diag(3))
```

The traceplot of the 1st component and the histogram can be found below:





2.4 Adaptive Metropolis-Within-Gibbs

2.4.1 introduction

We implement one of the examples in Roberts and Rosenthal (2009), namely the Metropolis Within Gibbs Algorithm (MWGA), which applies the algorithm to a hierarchical model:

- Each observation $Y_{ij} \sim N(\theta_i, V)$ for $[1 \leq j \leq r_i]$.
- Each parameter $\theta_i \sim Cauchy(\mu, A)$ for $[1 \leq i \leq K]$.
- μ, A, V follow $N(0,1), IG(1,1), IG(1,1)$ respectively.

2.4.2 Algorithm

The AMWG algorithm extends the usual Gibbs sampler, so that we update each parameter using a random walk proposal, and accept or reject it with certain acceptance probability, which is adapted in each iteration according to the simulated path one has already obtained at that time.

In each n^{th} "batch", we append A, V, μ to θ and update each of the (K+3) parameters in turn by proposing a $N(0, \sigma_{i,n}^2)$ increment, where *sigma* is adjusted according to the following rule:

- Set $\sigma_{i,n} = 1$ when $n < 50$ for $1 \leq i \leq K+3$.
- Update $\log(\sigma_{i,n}) = \begin{cases} \log(\sigma_{i,n}) + \delta(n) & \text{if proportion(accepted } \theta_i\text{s)} \geq 0.44; \\ \log(\sigma_{i,n}) - \delta(n) & \text{if proportion(accepted } \theta_i\text{s)} \leq 0.44. \end{cases}$

  where $\delta(n)$ decays to 0 and is chosen to be $\min(0.005, n^{-1/2})$.

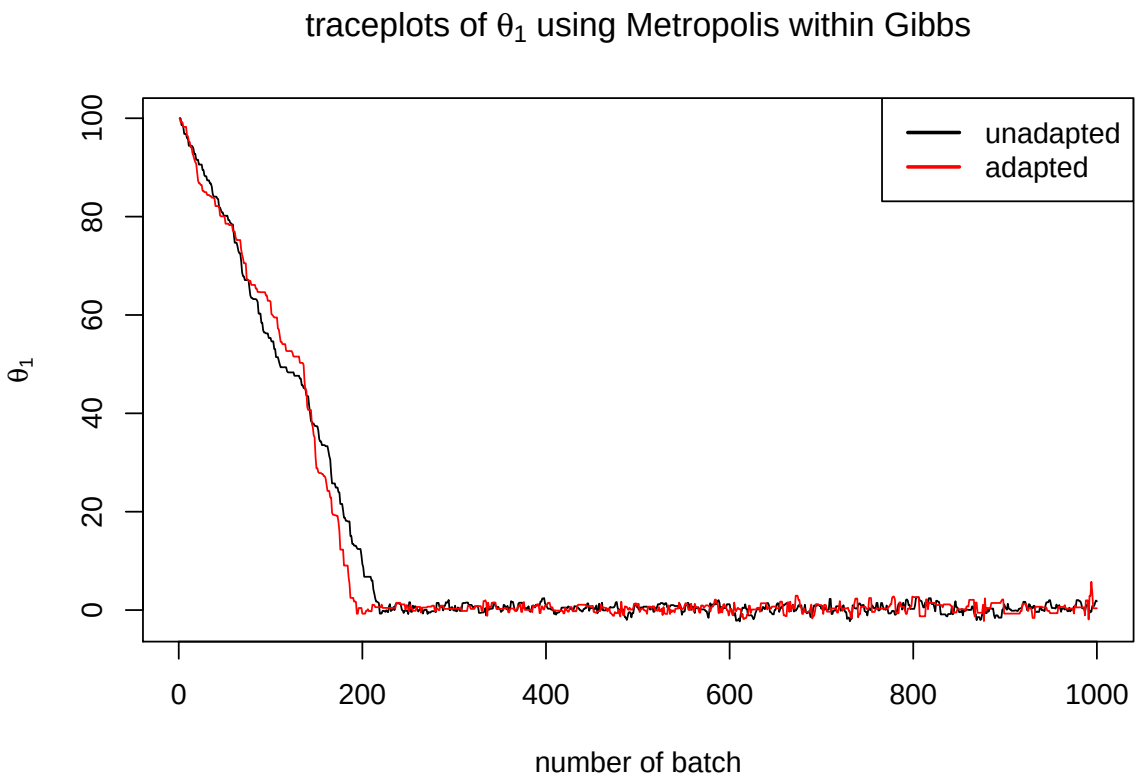
2.4.3 Inputs and Outputs

AMWG takes three inputs including whether the proposal variance is adapted (adapt), the number of iterations(N) and the number of parameters θ(K). The output contains a list of the parameter (θ, A, V, μ) paths and the acceptance/rejection paths.

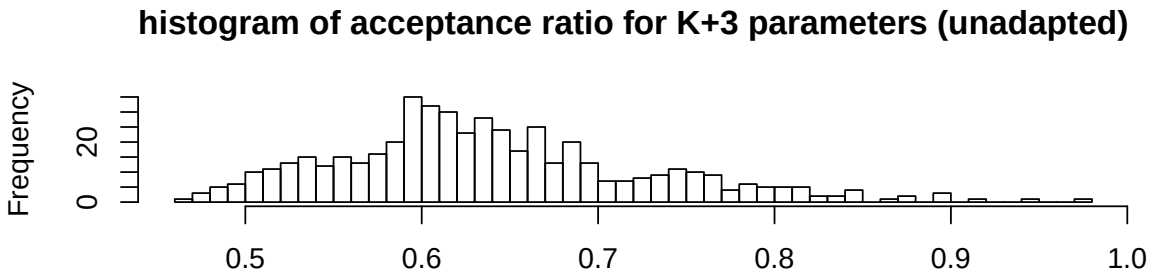
2.4.4 Example

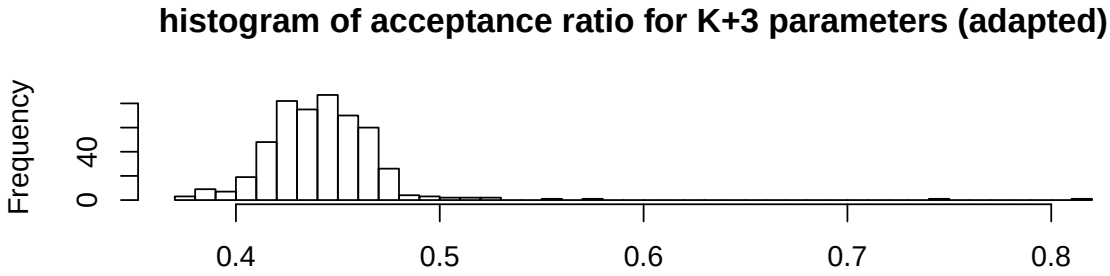
We can compare the adapted and unadapted version of the algorithm with $N = 1000$ and $K = 500$:

```
out_unadapt <- AMWG(adapt = 0, 1000, 500)
theta_unadapt <- out_unadapt[[1]]
acceptance_unadapt <- out_unadapt[[2]]
out_adapt <- AMWG(adapt = 1, 1000, 500)
theta_adapt <- out_adapt[[1]]
acceptance_adapt <- out_adapt[[2]]
```



The histograms of the proportion of accepted parameters until the N^{th} batch are displayed here:

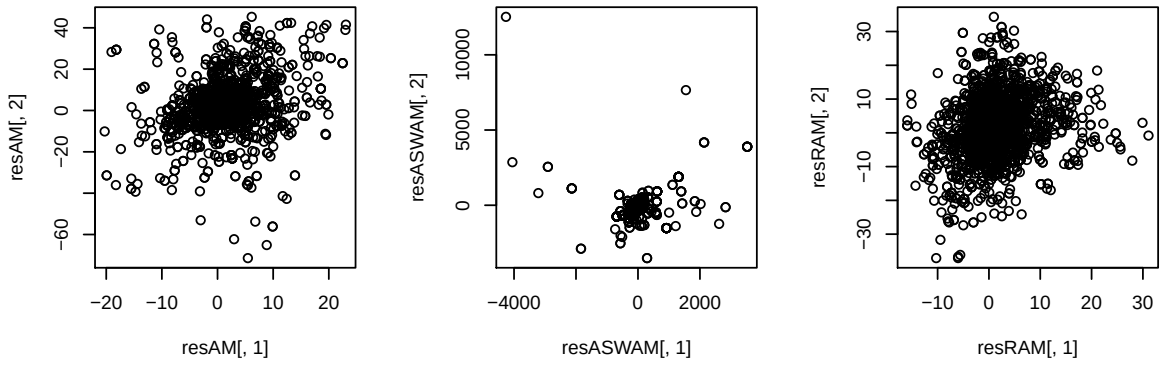




It can be seen that AMWG pulls the acceptance ratio towards 0.44, which is optimal in many settings.

2.5 Comparison

We compare the three generic algorithms(AM, ASWAM, RAM) here using a 2-dimensional student distribution with degrees of freedom equal to 1. The simulated patterns are displayed below:



It seems that in this case, AM and RAM have better performance compared with ASWAM.

3 Testing

This package includes a suite of unit tests written in testthat to validate the functions AM, ASWAM and RAM. The tests verify that all of the samplers produce sane results when run to sample from some common distributions in one and multiple dimensions. The tests also verify the the internal parameter validation function is correctly rejecting invalid parameters.

The testing of this package is integrated with Travis CI. The continuous integration platform builds and tests the package as code is committed to the package's repository.

4 Future Work

The looped nature of these algorithms would potentially benefit from an implementation using Rcpp to achieve near native performance with for loops. However as long as these samplers accept arbitrary density functions within R there will be a performance penalty when evaluating these densities from within C++. This penalty will remain even if the density itself is a native C/C++/Fortran function as a call will have to be made from C++ to R and then to another compiled language.

Adaptive Metropolis Hastings algorithms do not solve the issues caused by multimodal distributions. If the modes of a target distribution are far enough apart it is possible that a chain never finds the other modes and adapts itself only to the mode it has found.

References

Christophe Andrieu and Johannes Thoms. A tutorial on adaptive MCMC. *Statistics and Computing*, 18 (4):343–373, 2008. ISSN 0960-3174. doi: 10.1007/s11222-008-9110-y. URL `http://dx.doi.org/10.1007/s11222-008-9110-y`.

Heikki Haario, Eero Saksman, and Johanna Tamminen. An adaptive metropolis algorithm. *Bernoulli*, 7 (2):pp. 223–242, 2001. ISSN 13507265. URL `http://www.jstor.org/stable/3318737`.

Gareth O. Roberts and Jeffrey S. Rosenthal. Examples of adaptive mcmc. *Journal of Computational and Graphical Statistics*, 18(2):349–367, 2009. doi: 10.1198/jcgs.2009.06134. URL `http://dx.doi.org/10.1198/jcgs.2009.06134`.