A Review of A Tutorial on Adaptive MCMC with Applications

Darren Wang, Xi Zeng

May 8, 2020

Contents

Abstract

Markov chain Monte Carlo (MCMC) is a general strategy for generating samples from complex high-dimensional distributions. The choice of the proposal distribution is crucial to the performance of the Monte Carlo estimators as well as tuning of associated parameters such as proposal variances. Adaptive MCMC algorithms provide an alternative method which tune parameters with iterative computation thus automatically solve the problem with little user intervention. In this paper, we review the Adaptive MCMC methodology with inspection on the formulation of different Adaptive Metropolis (AM) algorithms, following our reference paper of Andrieu and Thoms (2008)[2]. Simulation study is given for different parameters with parallel model comparison. Applications of each algorithm on real data are discussed.

1 Review of Adaptive MCMC

Markov chain Monte Carlo, even though the most prevailing strategy to be used for generating random samples, can be inefficient due to inappropriate choice of proposal distribution. In more realistic scenarios, the efficiency of MCMC algorithms depends on not just one but a combination of Metropolis-Hastings updating procedures $\{P_{k,\theta},\ k=1,...,n,\ \theta\in\Theta\}$ which share π as common invariant distribution. A set of possible transitions for the Markov chain moving from X to Y is generated accordingly based on a family of proposal distributions $q_{k,\theta}$, each of which are accepted or rejected according to the probability $\alpha(X,Y)=min\{1,\frac{\pi(Y)q(Y,X)}{\pi(X)q(X,Y)}\}$. The combined MH strategy is a weighted mixture $P_\theta(x,dy)=\sum_{k=1}^n w_k(\theta)P_{k,\theta}(x,dy)$, $\sum_{k=1}^n w_k(\theta)=1, w_k(\theta)\geq 0$ which takes into account different possible parametrised proposal distributions. The main focus of such combination of strategies is optimization. Controlled MCMC encodes user-defined criteria for optimality or suboptimality which most time relies on the stochastic approximation framework.

However, controlled MCMC algorithms will not always preserve stationarility of π. Easy perturbation of π-ergodicity is one problem with controlled MCMC that shouldn't be overlooked as it violates the minimum requirement of generating samples using MCMC algorithms which could lead to inconsistent estimators. Let us assume P_θ as the transition matrix, and π an invariant distribution of P_θ following the toy example in Andrieu and Moulines (2006)[1]. For any $\theta\in\Theta$ the Markov chain is irreducible, aperiodic and hence ergodic. The time homogeneous Markov chain will fail to converge to π as soon as there is dependence on the current state. The use of vanishing adaptation in attempt to circumvent such loss of π-ergodicity of P_θ is intuitive but sometimes problematic. The general idea behind vanishing adaptation is to make θ_i depend less and less on recently visited states of the chain so that tuning of parameters eventually recedes from the algorithm. However, rate of convergence can be compromised and π-ergodicity can be lost if the trajectory of θ_i moves towards an unsatisfying direction. In addition, such condition is often difficult to verify which makes adaptation less favorable.

For better or worse, validity of controlled MCMC algorithms with vanishing adaptation is proven and can be used for deriving consistent estimation of expectations with respect to target distribution π. As a measure of optimality or suboptimality, one simple but efficient criterion is universal in practice which is formally defined as the jump rate of a MH update in the stationary regime

$$\begin{aligned}\bar{\alpha}_\theta &= \int_{X^2} min\{1, \frac{\pi(y)q_\theta(y,x)}{\pi(x)q_\theta(x,y)}\}\pi(x)q_\theta(x,y)dxdy \\ &= \mathbb{E}_{\pi\otimes q_\theta}(min\{1, \frac{\pi(y)q_\theta(y,x)}{\pi(x)q_\theta(x,y)}\})\end{aligned}$$

An optimum value θ^* corresponds to finding the root of the equation $h(\theta) = \bar{\alpha}_\theta - \alpha^*$ which ensures the minimization of the asymptotic variance of the random variable. Considering that the updating rule should apply to the situation where we in favor of the increments $\{\theta_{i+1} - \theta_i\}$ to be vanishing for the purpose of achieving convergence, Robbins-Monro stochastic optimization methodology is applied for solving the aforementioned root finding problem.

The first updating rule to be followed is motivated by the monotonicity assumption on $\bar{\alpha}_\theta$, which indicates that θ_i should be increased when $\hat{\alpha}_{\theta_i} > \bar{\alpha}^*$ and vice versa. It determines which way the data should flow to in order to achieve a balance (convergence). Assume $\{\gamma_i \subset (0, +\infty)^{\mathbb{N}}\}$ a sequence of possible step size, which based on different ways of approaching can be chosen at random or deterministicly. The sequential updating of $\{\theta_i\}$ is given by $\theta_{i+1} = \theta_i + \gamma_{i+1}(\hat{\alpha}_{\theta_i} - \bar{\alpha}^*)$. The recursion becomes deterministic as $\hat{\alpha}_\theta \to \bar{\alpha}_\theta$ is the baseline for the second updating rule which set the grounding for standard gradient algorithm to be applicable. The sequential updating of $\{\theta_i\}$ is given by $\theta_{i+1} = \theta_i + \gamma_{i+1}(\bar{\alpha}_{\theta_i} - \bar{\alpha}^*) + \gamma_{i+1}(\hat{\alpha}_{\theta_i} - \bar{\alpha}_{\theta_i})$. We can always write the recursion of interest in general form as

$$\theta_{i+1} = \theta_i + \gamma_{i+1}h(\theta_i) + \gamma_{i+1}\xi_{i+1}$$

Compared with standard MCMC algorithms without adaptation, controlled MCMC has many pros and cons worth mentioning about. First, recursive and stochastic updating processes means no extra computational and memory cost increased with iterations. In addition, controlled MCMC is flexible and requires limited user intervention, resulting in self-learning processes, while user-defined criterion can be adapted to match the desired statistical properties. However, even though that mathematically some convergence conditions are easily satisfied, in more realistic scenarios as the ones being discussed about in the following section, the convergence rate of such Markov chain can be rather slow. More often than desired, the dilemma of balancing between the stability of θ_i and the ergodicity of the the chain $\{X_i\}$ is encountered in practice. Restrictions can be imposed to ensure that θ_i drift away from bad values as much as possible, but convergence rate is compromised as a result.

2 Adaptive Metropolis (AM)

2.1 AM algorithm

The focus of our literature being reviewed is the adaptation of the multivariate symmetric increments random walk Metropolis-Hastings algorithm with Gaussian proposals (SRWM) which is motivated by the theoretical groundings set by Gelman et al. (1995)[3] suggesting a “optimal” covariance matrix for aforementioned target distribution. In practice, the tuning and determination of optimal covariance matrix requires expertise. But adaptive Metropolis algorithm first proposed by Haario et al. (2001)[4] solves the problem by updating the covariance matrix based on aggregated former knowledge of the past values of the simulations.

In general, the parameter θ_i is updated according to a single step of a stochastic approximation procedure,

$$\theta_{i+1} = \theta_i + \gamma_{i+1}H_{i+1}(\theta_i, X_i, X_{i+1})$$

where X_{i+1} is sampled form $P_{\theta_i}(X_i, \bullet)$. In our case of research, the proposal distribution H is chosen as Gaussian with mean at the current point X_{i-1} and covariance matrix Σ_i depends on the empirical covariance

matrix determined by all the previous points $X_0, ..., X_{i-1}$ and a λ parameter which only depends on dimension d. A possible choice of the increment distribution q in which $q(x, y) = q(y - x)$ is the multivariate normal with zero-mean and covariance matrix $\lambda\Sigma$.
The mean and covariance of the target distribution is given by

$$\mu_\pi \overset{def}{=} \int_X x\pi(dx) \ and \ \Sigma_\pi \overset{def}{=} \int_X (x - \mu_\pi)(x - \mu_\pi)^T \pi(dx)$$

Without loss of generality, the recursive updates of mean μ_i and covariance matrix Σ_i which intend to save computational power satisfy:

$$\begin{cases} \mu_{i+1} = \mu_i + \gamma_{i+1}(X_{i+1} - \mu_i) \\ \Sigma_{i+1} = \Sigma_i + \gamma_{i+1}((X_{i+1} - \mu_i)(X_{i+1} - \mu_i)^T - \Sigma_i) \end{cases} \tag{1}$$

The choice of dimension parameter λ is adopted from Gelman et al. (1995)[3] with $\lambda = 2.38^2/n_x$.

2.2 Rao-Blackwellised AM algorithm

The Rao-Blackwell theorem[5] is commonly utilized in statistics as the theoretical grounding for estimator optimization which often leads to variance reduction of an unbiased estimator. Given $g(X)$ any kind of an arbitrarily crude estimator of a parameter θ, the conditional expectation of $g(X)$ given $T(X)$, where T is a sufficient statistics, is an optimization of $g(X)$. In the Monte Carlo context, we call the method of replacing a naive estimator with its conditional expectation Rao-Blackwellization.

The Rao-Blackwellized procedure being proposed makes use of the conditional expectation $\alpha(X_i, Y_{i+1}) = E[f(X^{i+1})|X^i = X_i] = E[f(Y_{i+1})|X_i]$ which leads to the formulation of the expectation of $f(X_{i+1})$

$$\overline{\mathrm{f(X_{i+1})}} := \alpha(X_i, Y_{i+1})f(Y_{i+1}) + (1 - \alpha(X_i, Y_{i+1}))f(X_i)$$

Incorporate the Rao-Blackwellized procedure into standard AM recursions (1), which controls the flow of the data to move towards proposal distribution with an optimized estimator of our parameter of interest in intention to avoid too much sampling from undesired proposal distributions. However, it is not intuitive whether or not this optimization actually reduces asymptotic variance of the estimators as suggested by its standard counterpart.
The recursive updates of mean μ_i and covariance matrix Σ_i incorporating Rao-Blackwellization criterion satisfy:

$$\begin{cases} \mu_{i+1} = \mu_i + \gamma_{i+1}[\alpha(X_i, Y_{i+1})(Y_{i+1} - \mu_i) + (1 - \alpha(X_i, Y_{i+1}))(X_i - \mu_i)] \\ \Sigma_{i+1} = \Sigma_i + \gamma_{i+1}[\alpha(X_i, Y_{i+1})(Y_{i+1} - \mu_i)(Y_{i+1} - \mu_i)^T + (1 - \alpha(X_i, Y_{i+1}))(X_i - \mu_i)(X_i - \mu_i)^T - \Sigma_i] \end{cases} \tag{2}$$

2.3 Compound Criterion: Global Approach

Follow the discussion of standard AM algorithm in Sect. 2.1, we carry on to further optimization of the algorithm which this time focuses on optimizing the scaling parameter λ of the proposal distribution. In Sect. 2.1, we proposed an optimal λ which is adopted from Gelman et al. (1995)[3] with $\lambda = 2.38^2/n_x$. The suggested λ gives us a good choice of proposal distribution $N(0, \lambda\Sigma)$ to start from. However, in situations where we do not have enough information to make a proper guess for the initial value of the estimator of Σ_π, the learning curve can be comparatively flat when we are basically only exploring the nearby areas around the parameter. This can be particularly undesirable when the data follows multimodal distribution which gives us less chance of exploring broader area and finding the true distribution. In addition, too large or too small of $\lambda\Sigma_i$ could lead to bad acceptance probability accordingly. And fixing λ as the theoretical optimal value is not helping in this situation when we desire to learn Σ_i following different directions.

We once again take a look at the monotonicity assumption on $\bar{\alpha}_\theta$ being introduced in the review of literature. With $\theta := (\lambda, \mu, \Sigma)$ our parameter and our focus of optimizing λ, it is intuitive to assume an updating strategy which combines the idea of a logarithm transformation of λ and the standard Robbins-Monro recursion. the direction for the estimated acceptance probability $\hat{\alpha}_{\theta_i} = \alpha(X_i, Y_{i+1})$ corresponding to each parameter θ_i to be updated is controlled by $\bar{\alpha}^*$. If the acceptance rate is too low, then λ_i should be increased accordingly, and vice versa. Therefore, the learning curve is expected to be steepen especially when in practice little knowledge is known about the distribution and we make poor choice on initial values.
The recursive updates of mean μ_i, covariance matrix Σ_i and scaling parameter λ satisfy:

$$\begin{cases} log(\lambda_{i+1}) = log(\lambda_i) + \gamma_{i+1}[\alpha(X_i, Y_{i+1}) - \bar{\alpha}^*] \\ \mu_{i+1} = \mu_i + \gamma_{i+1}(X_{i+1} - \mu_i) \\ \Sigma_{i+1} = \Sigma_i + \gamma_{i+1}((X_{i+1} - \mu_i)(X_{i+1} - \mu_i)^T - \Sigma_i) \end{cases} \tag{3}$$

3 Simulations and Algorithm Comparison

In this section, the algorithms aforementioned in section 2 were examined and compared under two scenarios. This section has two focus, first, to examine the validity of the algorithms, second, to show the power of these algorithms to adapt for a optimal step size when an inappropriate step size in a MCMC sampler was chosen initially.

Scenario 1 - Univariate Gaussian Distribution

In the first scenario, the goal is to draw samples from a univariate Gaussian distribution $X \sim N(10, 16)$. Using a most standard Metropolis-Hasting algorithm with symmetric Gaussian distribution $q(|X_{t-1}) \sim N(X_{t-1}, \sigma^2)$ as proposal distribution, the scaling parameter σ^2 was set as suggested by Gelman et al. (1995)[3] to be $2.38^2/n_x = 2.38^2$, and a logical initial point $X_0 = 12$ was chosen, lastly, 10,000 iterations were ran.

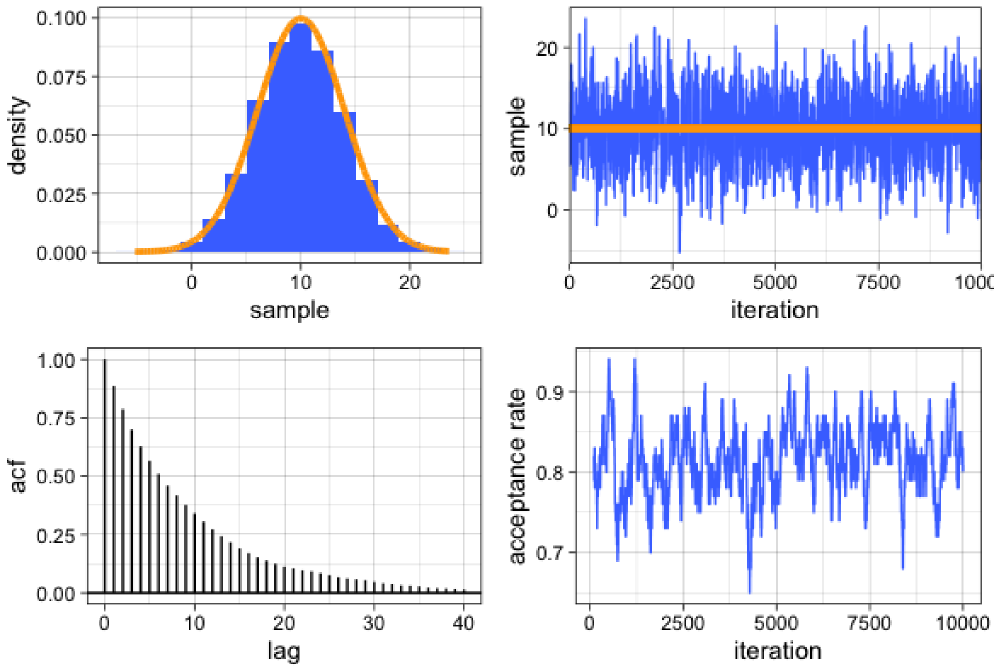

Figure 1: MH Sampler - Optimal Step Size

In Fig 1, the algorithm was evaluated in 4 different sub-plot, the panels are (from top left to bottom right): Histogram of samples, sample vs iteration (i.e. trace plot), acf plot, and proposal acceptance rate(window size

100 iterations) vs iteration. Please be noted that all the algorithms would be evaluated with these four plots.

From Fig 1, we see that samples were drawn successfully from the desired distribution, the sample chain mixed well and the autocorrelation was almost eliminated after lag = 30, the acceptance rate was rater high at around 0.8 compare to the rule of thumb for a 1-d distribution around 0.44, but overall the algorithm performed well.

However, often time we would want to draw sample from an unknown distribution, for example drawing from the posterior distribution in a Bayesian framework, or drawing from a distribution that we only known up to an normalizing constant. We use the following example to illustrate the importance of an appropriate step size in an MCMC algorithm. In the example, σ was chosen to be $100 * 2.38 = 238$.

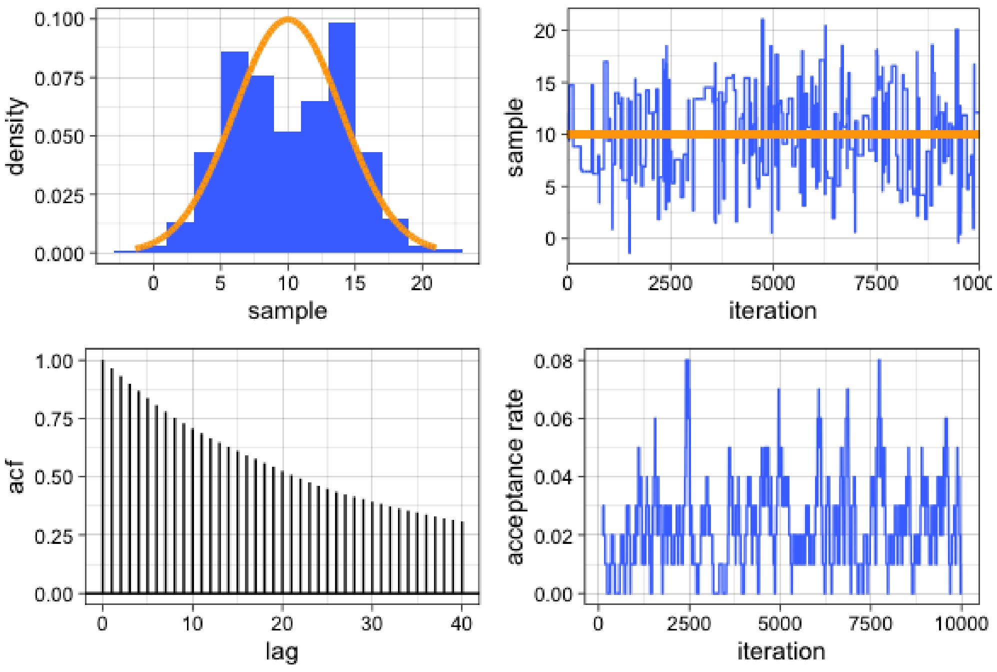

Figure 2: MH Sampler - Inappropriate Step Size

As shown in Fig 2, the algorithm performed poorly, as the samples were barely representing the true distribution, the acceptance rate was too low(about 0.02), the choppiness in the trace plots indicated that the samples often stuck in some values for a while and the acf is high even after lag = 40.

The first two examples showed the importance of an appropriate step size is well illustrated. We now refer to the adaptive MCMC algorithms, examining their validity and showcase their abilities to deal with an badly chosen step size. The algorithms were implemented as suggested by Haario et al. (2001)[4], with initial period $t_0 = 500$, $s_d = 0.001$. The result is summarized in Fig 3.

With the same starting point $X_0 = 12$, and step size $\sigma = 238$, the initial period ran just as bad as the previous MH sampler, however, once the adaptation starts, the acceptance rate quickly converged to around the optimal value 0.44, the distribution plot, trace plot and acf plot all voted for the fact that the adaptive Metropolis algorithm is valid. (Note that the spike in distribution plot was caused by the first 500 iterations when the adaptation haven't started yet, low accpetance rate made the samples stuck in the initial value.)

We further implement the Rao-Blackwellized AM algorithm with same specifications, and the result shown in Fig 4 indicated that the algorithm is valid, and the resulting acceptance rate was slightly better(closer to 0.44).

Lastly, the AM algorithm was implemented again with global adaptive scaling, in the implementation, $\bar{\alpha}_*$ is set as the optimal acceptance rate 0.44. It was shown in Fig 5 that, although the algorithm performed

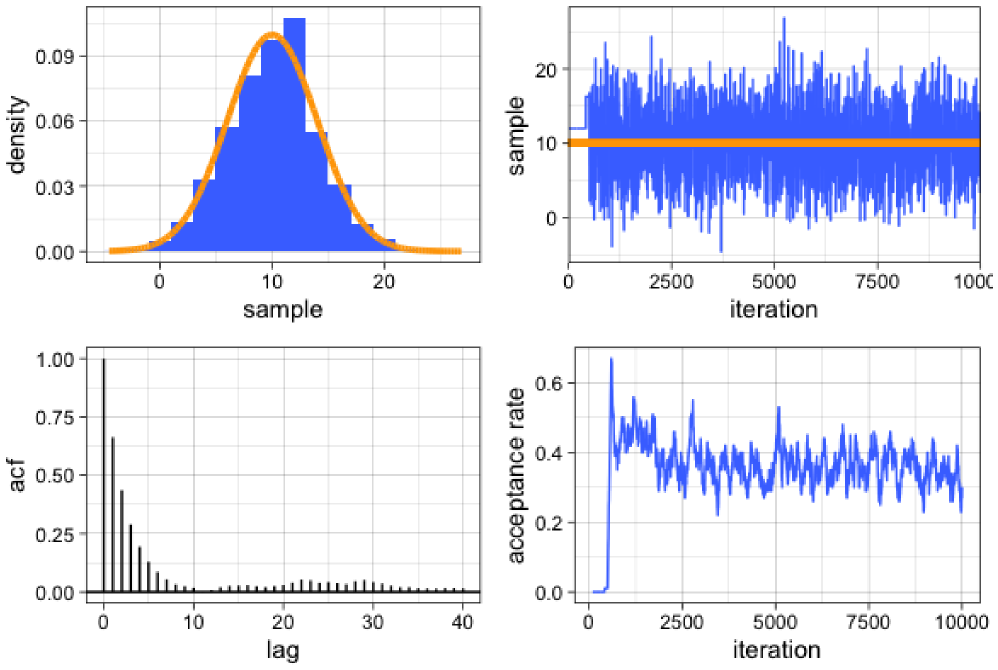

Figure 3: AM Sampler - Inappropriate Step Size

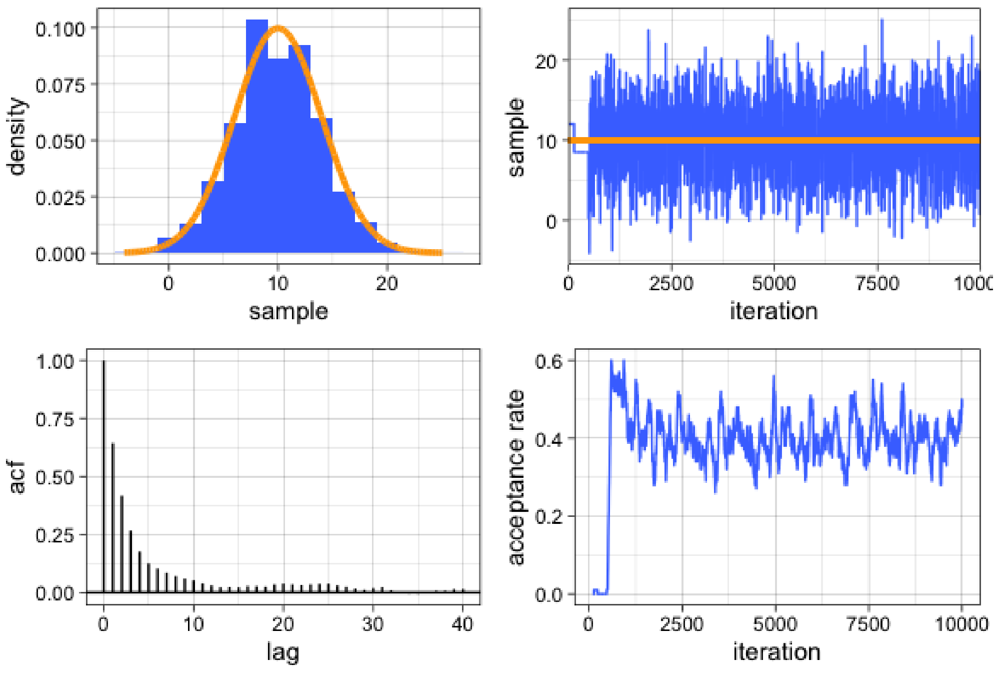

Figure 4: Rao-Blackwellized AM Sampler

rather well on drawing samples from the desired distribution, the global adaptive scaling of λ also raised the acceptance rate in the long run, making it not ideal as the original AM algorithm.

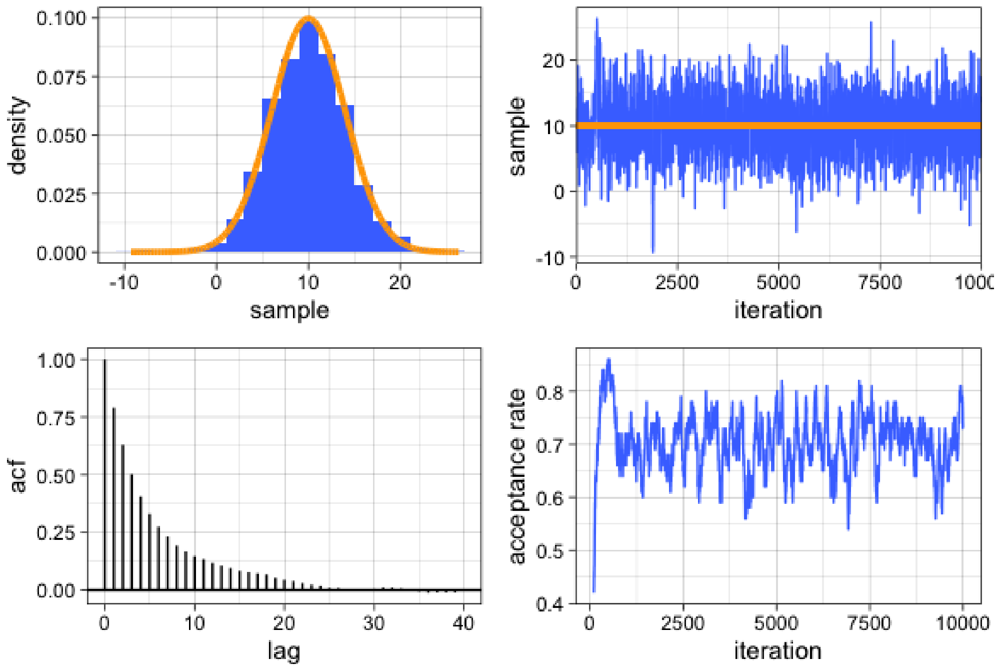

Figure 5: AM with global adaptive scaling

As pointed out in Andrieu and Thoms (2008)[2], the global adaptive scaling on λ is likely not to be useful in the long-run. However, the adaptation indeed made the acceptance rate better in the initial period. Therefore, we can choose to adapt λ only in the initial period and stop adapting when the adaptation of Σ_i starts. This global adaptation technique is proved to be very useful in the early stage of the algorithm and might be useful in the case when there is a need for a longer initial period, and the computational cost is high.

Scenario 2 - Bivariate Gaussian Distribution

In this scenario, our target distribution is a bivariate normal distribution as follows:

$$N(\begin{bmatrix} 4 \\ 30 \end{bmatrix}, \begin{bmatrix} 4 & 3 \\ 3 & 9 \end{bmatrix})$$

Starting at initial values $[10, 10]^T$, same with the former scenario, when an inappropriate step size was chosen for Metropolis-Hasting algorithm(100 times the suggested step), low acceptance rate made the algorithm stuck in a few local values. Therefore, the samples failed to represent the desired distribution with 10,000 iterations. Details of the results can be found in Fig 6

Using the same initial step size with an initial period of 500 iterations, adaptive metropolis algorithm escaped from the obvious not ideal proposal distribution immediately after the adaptation starts. The algorithm worked very well in creating samples that represent the target distribution. It is also worth notice that the algorithm explored some low density area, but once the acceptance rate drop lower, it was able to adapt the step size back using only few hundreds of iterations. Overall, although the acceptance rate is lower than the optimal(0.234), the algorithm is still valid and performed rather well. This simulation is summarized in Fig 7.

We further incorporated Rao-Blackwellization to adaptive metropolis algorithm, the results is summarized in Fig 8. The algorithm was again valid as it produced samples that converged to the target distribution. However Rao-Blackwellization in the simulation caused the acceptance rate to rise over 0.4 at around 2500th iterations and continued to rise after.

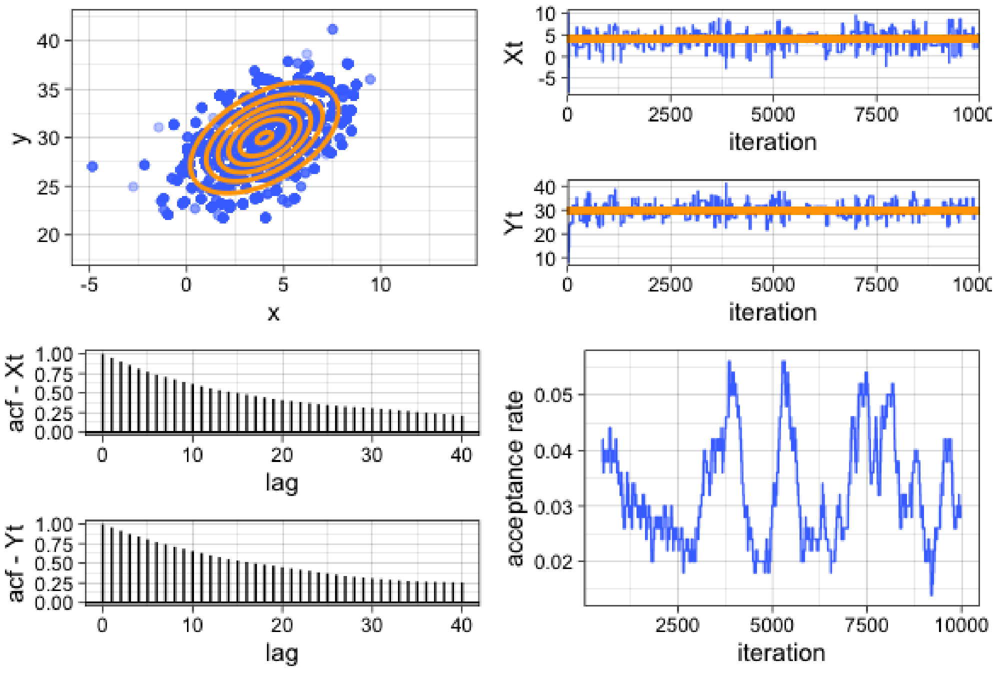

Figure 6: 2-d MH - Inappropriate Step Size

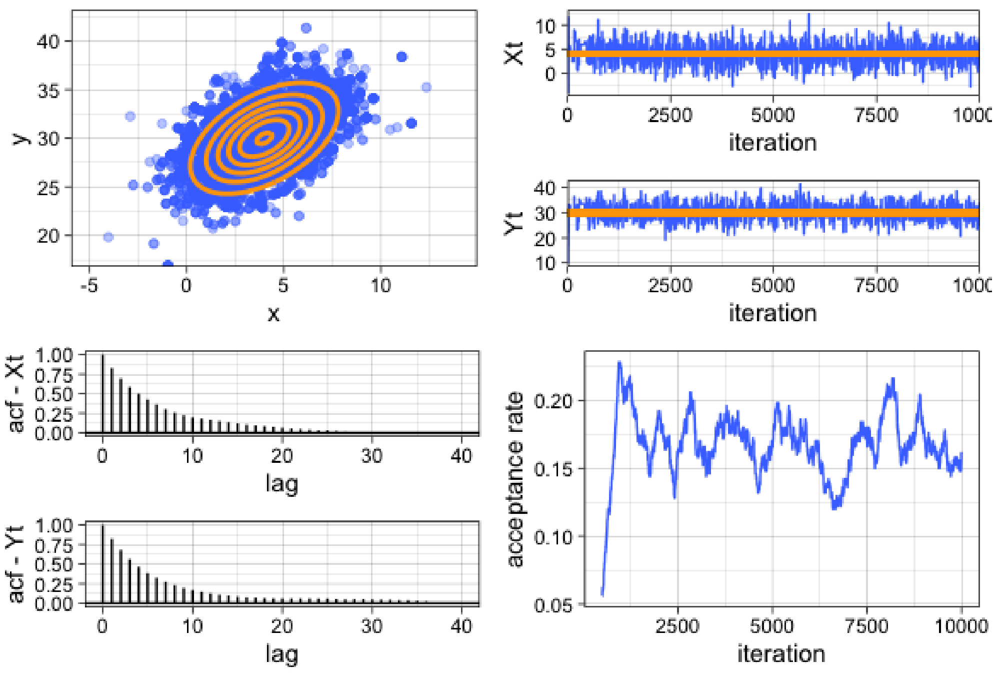

Figure 7: 2-d AM - Inappropriate Step Size

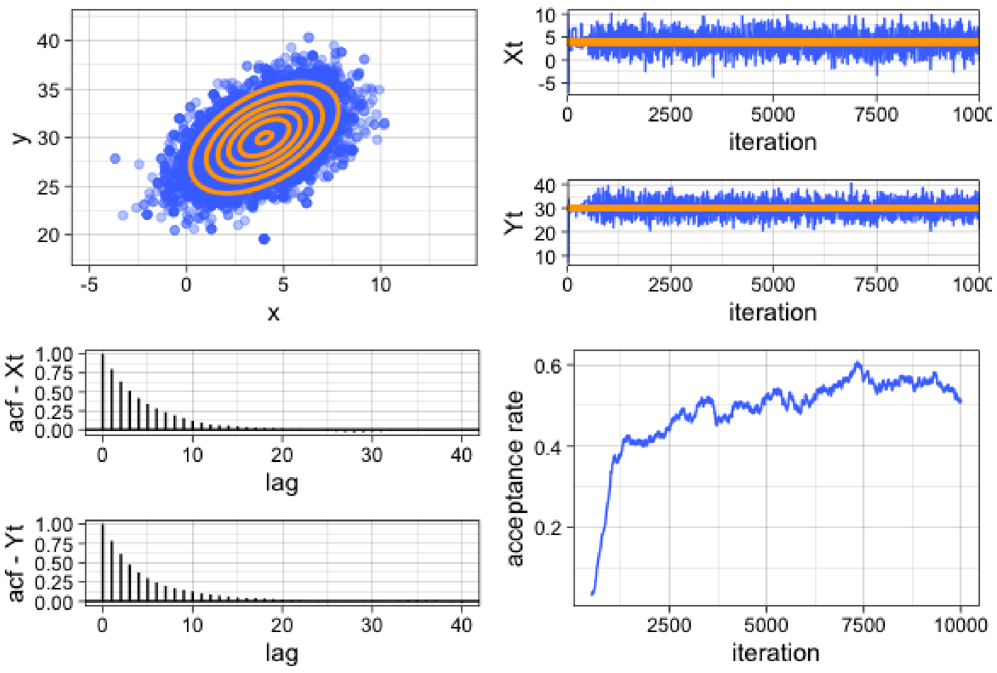

Figure 8: 2-d Rao-Blackwellized AM

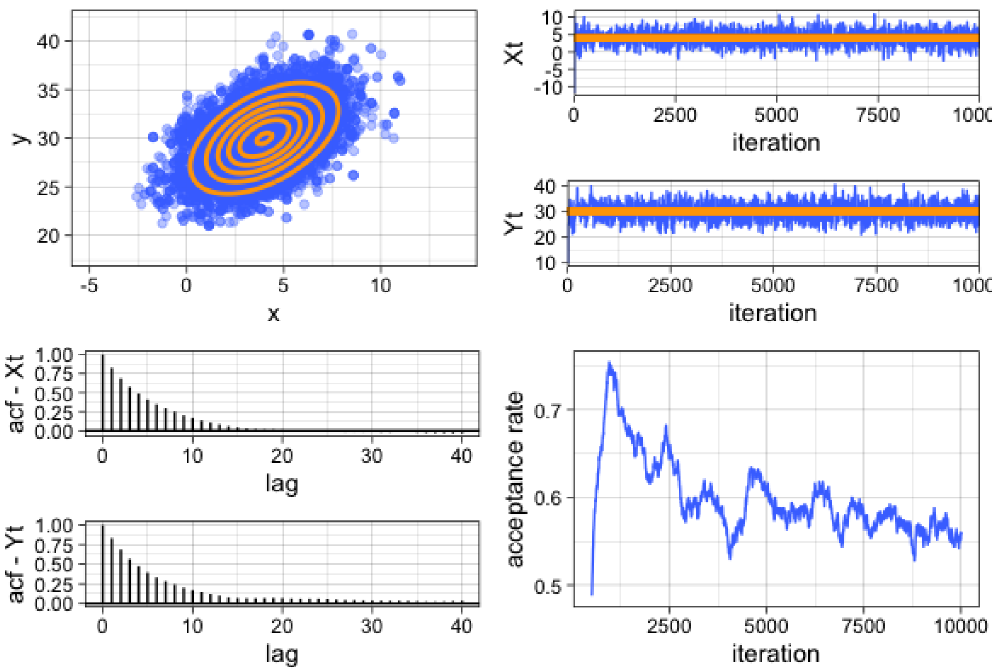

Figure 9: 2-d AM with Global Adaptive Scaling

Lastly, the same technique of global adaptive scaling on parameter λ_i was implemented, in this implementation $\bar{\alpha}^*$ is set as the optimal acceptance rate 0.234. As shown in Fig 9 the algorithm still provided valid samples from the target distribution. Although the global adaptive scaling of λ also raised the acceptance rate in the long run, this simulation perfectly corresponded with Andrieu and Thoms (2008)[2] that we see a more rapid exploration of the target distribution as the algorithm started with high acceptance rate and gradually moved towards a bigger step size.

4 Discussion

In this project, we studied the standard Adaptive Metropolis algorithm along with its various counterparts which adopt different criterion in aims of optimization one step further. Briefly reviewed the framework and all the algorithms introduced, we focused on the first three algorithm, i.e., the standard AM algorithm, the Rao-Blackwellised AM algorithm and AM algorithm with global adaptive scaling. We applied the aforementioned three algorithms in simulation on both 1-dimension and 2-dimension cases. All three algorithms gave us promising results on and therefore possible directions on how we can improve MCMC.

There can be stopping rules for simulations and algorithms, but in reality world there may never be such rules on when and where to stop the research. For further comparison, readers can contrast these algorithms with the rest algorithms being introduced in our reference paper. In addition, since that AM algorithms are supposed to have more advantages in high-dimension setting, readers can delve deeper into this direction of research and look up on how AM algorithm can improve simulation both computationally and efficiently. The algorithms can also be applied on real data sets, especially in today's information rich new world where large-scale of data is prevalent and what we studied in this paper is inspiring.

All code are attached in appendix.

References

[1] Andrieu, C., & Moulines, É. (2006). *On the ergodicity properties of some adaptive MCMC algorithms.* The Annals of Applied Probability, 16(3), 1462-1505. doi:10.1214/105051606000000286
[2] Andrieu, C., & Thoms, J. (2008). *A tutorial on adaptive MCMC.* Statistics and Computing, 18(4), 343-373. doi:10.1007/s11222-008-9110-y
[3] Gelman, A., Roberts, G., Gilks, W. (1995). Efficient Metropolis jumping rules. In: *Bayesian Statistics*, vol. 5. Oxford University Press, New York
[4] Haario, H., Saksman, E., & Tamminen, J. (2001). *An Adaptive Metropolis Algorithm.* Bernoulli, 7(2), 223. doi:10.2307/3318737
[5] Rao–Blackwell theorem. (2020, April 21). Retrieved from https://en.wikipedia.org/wiki/Rao%E2%80%93Blackwell_theorem

Appendix - Code for Simulations

Metropolis-Hasting

```r
MH = function(inits, m) {
  # dimension
  d = length(inits)
  # initialize vector for the first 100 iterations
  Xt0 = matrix(0, ncol = d, nrow = m + 1)
  # initial vector
  Xt0[1, ] = inits
  # acceptance vector
  accept = rep(0, m)
  # original MH
  for (i in 1 : m) {
    # Change these 2 lines for different target distribution
    proposed = Xt0[i, ] + rnorm(1, 0, 238)
    rho = dnorm(proposed, 10, 4) / dnorm(Xt0[i, ], 10, 4)
    ########################################################

    alpha = min(1, rho)

    if(runif(1) < alpha) {
      Xt0[i + 1, ] = proposed
      accept[i] = 1
    } else {
      Xt0[i + 1, ] = Xt0[i, ]
    }
  }

  return(list(samples = Xt0, accept = accept))
}
```

Helper Functions for Adaptive Algorithm

```
# Function for calculating Ct
ct = function(sd, xt_prev_mat, epsl) {
  return (sd * cov(xt_prev_mat) + sd * epsl * diag(dim(xt_prev_mat)[2]))
}

# Function for updating Ct recursively to save computational time
ct_update = function(prev_ct, t, xt, xt_prev_bar, sd, epsl) {
  comp_1 = (t - 1) / t * prev_ct
  comp_2 = t * xt_prev_bar %*% t(xt_prev_bar)
  xt_bar = t / (t + 1) * (xt_prev_bar) + xt / (t + 1)
  comp_3 = (t + 1) * xt_bar %*% t(xt_bar)
  comp_4 = xt %*% t(xt)
  comp_5 = epsl * diag(length(xt))

  return (comp_1 + sd / t * (comp_2 - comp_3 + comp_4 + comp_5))
}
```

Adaptive Metropolis

```
AM = function(inits, m, t0, epsl = 0.001) {
  # dimension
  d = length(inits)
  # initialize vector for the first 100 iterations
  Xt0 = matrix(0, ncol = d, nrow = m + 1)
  # initial vector
  Xt0[1, ] = inits
  # acceptance vector
  accept = rep(0, m)
  # original MH
  for (i in 1 : t0) {

    # Change these 2 lines for different target distribution
    proposed = Xt0[i, ] + rnorm(1, 0, 238)
    rho = dnorm(proposed, 10, 4) / dnorm(Xt0[i, ], 10, 4)
    ########################################################

    alpha = min(1, rho)

    if(runif(1) < alpha) {
      Xt0[i + 1, ] = proposed
      accept[i] = 1
    } else {
      Xt0[i + 1, ] = Xt0[i, ]
    }
  }

  #Ct for next iteration
  sd_ = 2.38 ^ 2 / d
  Ct = ct(sd_, as.matrix(Xt0[1 : (t0 + 1), ]), epsl)
  Xt_p_bar = colMeans(as.matrix(Xt0[1 : (t0 + 1), ]))
  # complete the remaining iterations with adaptation
  for (i in (t0 + 1) : m) {

    # Change these 2 lines for different target distribution
    proposed = Xt0[i, ] + rnorm(1, 0, sqrt(Ct))
    rho = dnorm(proposed, 10, 4) / dnorm(Xt0[i, ], 10, 4)
    ########################################################
    alpha = min(1, rho)

    if(runif(1) < alpha) {
      Xt0[i + 1, ] = proposed
      accept[i] = 1
    } else {
      Xt0[i + 1, ] = Xt0[i, ]
    }

    Ct = ct_update(Ct, i, Xt0[i + 1, ], Xt_p_bar, sd, epsl)
    Xt_p_bar = i / (i + 1) * (Xt_p_bar) + Xt0[i + 1, ] / (i + 1)
  }

  return(list(samples = Xt0, accept = accept))
}
```

Rao-Blackwellizaed Adaptive Metropolis

```
AM_R = function(inits, m, t0, epsl = 0.001) {
  # dimension
  d = length(inits)
  # initialize vector for the first 100 iterations
  Xt0 = matrix(0, ncol = d, nrow = m + 1)
  # initial vector
  Xt0[1, ] = inits
  # acceptance vector
  accept = rep(0, m)
  # original MH
  for (i in 1 : t0) {
```

```r
    # Change these 2 lines for different target distribution
    proposed = Xt0[i, ] + rnorm(1, 0, 238)
    rho = dnorm(proposed, 10, 4) / dnorm(Xt0[i, ], 10, 4)
    ########################################################

    alpha = min(1, rho)

    if(runif(1) < alpha) {
      Xt0[i + 1, ] = proposed
      accept[i] = 1
    } else {
      Xt0[i + 1, ] = Xt0[i, ]
    }
  }

  #Ct for next iteration
  sd_ = 2.38 ^ 2 / d
  Ct = ct(sd_, as.matrix(Xt0[1 : (t0 + 1), ]), epsl)
  Xt_p_bar = as.matrix(colMeans(as.matrix(Xt0[1 : (t0 + 1), ])))
  # complete the remaining iterations with adaptation
  for (i in (t0 + 1) : m) {

    # Change these 2 lines for different target distribution
    proposed = Xt0[i, ] + rnorm(1, 0, sqrt(Ct))
    rho = dnorm(proposed, 10, 4) / dnorm(Xt0[i, ], 10, 4)
    ########################################################
    alpha = min(1, rho)

    if(runif(1) < alpha) {
      Xt0[i + 1, ] = proposed
      accept[i] = 1
    } else {
      Xt0[i + 1, ] = Xt0[i, ]
    }

    # Rao-Blackwellization
    Xt_bar = as.matrix(alpha * (proposed) + (1 - alpha) * Xt0[i, ])

    Ct = ct_update(Ct, i, Xt_bar, Xt_p_bar, sd, epsl)
    Xt_p_bar = i / (i + 1) * (Xt_p_bar) + Xt_bar / (i + 1)
  }

  return(list(samples = Xt0, accept = accept))
}
```

Adaptive Metropolis with Global Scaling

```r
AM_4 = function(inits, m, t0, epsl = 0.001) {
  # dimension
  d = length(inits)
  sd_ = 2.38 ^ 2 / d
  # alpha_star
  alpha_star = 0.44
  # initialize vector for the first 100 iterations
  Xt0 = matrix(0, ncol = d, nrow = m + 1)
  # initial vector
  Xt0[1, ] = inits
  # acceptance vector
  accept = rep(0, m)
  # original MH
  for (i in 1 : t0) {
    # Change these 2 lines for different target distribution
    proposed = Xt0[i, ] + rnorm(1, 0, sd_ * 238)
    rho = dnorm(proposed, 10, 4) / dnorm(Xt0[i, ], 10, 4)
    ########################################################
    alpha = min(1, rho)

    sd_ = i / (i + 1) * exp(log(sd_) + 1 / (i + 1) * (alpha - alpha_star))

    if(runif(1) < alpha) {
      Xt0[i + 1, ] = proposed
      accept[i] = 1
    } else {
      Xt0[i + 1, ] = Xt0[i, ]
    }
  }

  #Ct for next iteration
  Ct = ct(sd_, as.matrix(Xt0[1 : (t0 + 1), ]), epsl)
  Xt_p_bar = colMeans(as.matrix(Xt0[1 : (t0 + 1), ]))
  # complete the remaining iterations with adaptation
  for (i in (t0 + 1) : m) {

    # Change these 2 lines for different target distribution
    proposed = Xt0[i, ] + rnorm(1, 0, sqrt(Ct))
    rho = dnorm(proposed, 10, 4) / dnorm(Xt0[i, ], 10, 4)
    ########################################################

    alpha = min(1, rho)

    if(runif(1) < alpha) {
      Xt0[i + 1, ] = proposed
      accept[i] = 1
    } else {
      Xt0[i + 1, ] = Xt0[i, ]
    }

    # update sd
    sd_ = i / (i + 1) * exp(log(sd_) + 1 / (i + 1) * (alpha - alpha_star))

    Ct = ct_update(Ct, i, Xt0[i + 1, ], Xt_p_bar, sd, epsl)
    Xt_p_bar = i / (i + 1) * (Xt_p_bar) + Xt0[i + 1, ] / (i + 1)
  }

  return(list(samples = Xt0, accept = accept))
}
```