ST407 Monte Carlo Methods Assignment 1

Student Number: 1983162

Hannah Craddock

Question 1

Part a Evaluate density

The density of interest $f(x)$ is as follows;

$$f(x) \propto s_7 \cdot \exp\left\{-sin\left(\frac{s_1 \cdot x^2}{15 - s_1}\right) - \frac{(x - 3 - s_2 - \pi)^2}{2 \cdot (5 + s_3)^2}\right\} + 2 \cdot (1 + s_7) \cdot \exp\left\{\frac{x^2}{32}\right\}$$
$$+ \quad (10 - s_7) \cdot \exp\left\{-\cos\left(\frac{s_4 \cdot x^2}{15 + s_4}\right) - \frac{(x + 3 + s_5 \cdot \pi)^2}{2(5 + s_6)^2}\right\}$$

Whereby;

s_i	s_1	s_2	s_3	s_4	s_5	s_6	s_7
	1	9	8	3	1	6	2

The function `fx.density` was used to evaluate $f(x)$ in R as follows;

```
fx.density = function(x, sn){

a=sn[7]*exp(-sin((sn[1]*x^2)/(15-sn[1]))-((x-3-
(sn[2]*pi))^2)/(2*(5+sn[3])^2))

b = 2*(1+sn[7])*exp(-((x^2)/32))

c= (10-sn[7])*exp( -cos((sn[4]*x^2)/(15+sn[4])) -     ((x+3+(sn[5]*pi))^2)/
(2*(5+sn[6])^2))

fx = a+b+c

fx

}
```

To implement the function `fx.density` and produce the required plot the following code was executed. The plot is shown in Figure 1

```
x = seq(-50,80,length=100000)
sn = c(1,9,8,3,1,6,2) #Student number
fx = fx.density(x,sn)
plot(x, fx)
```

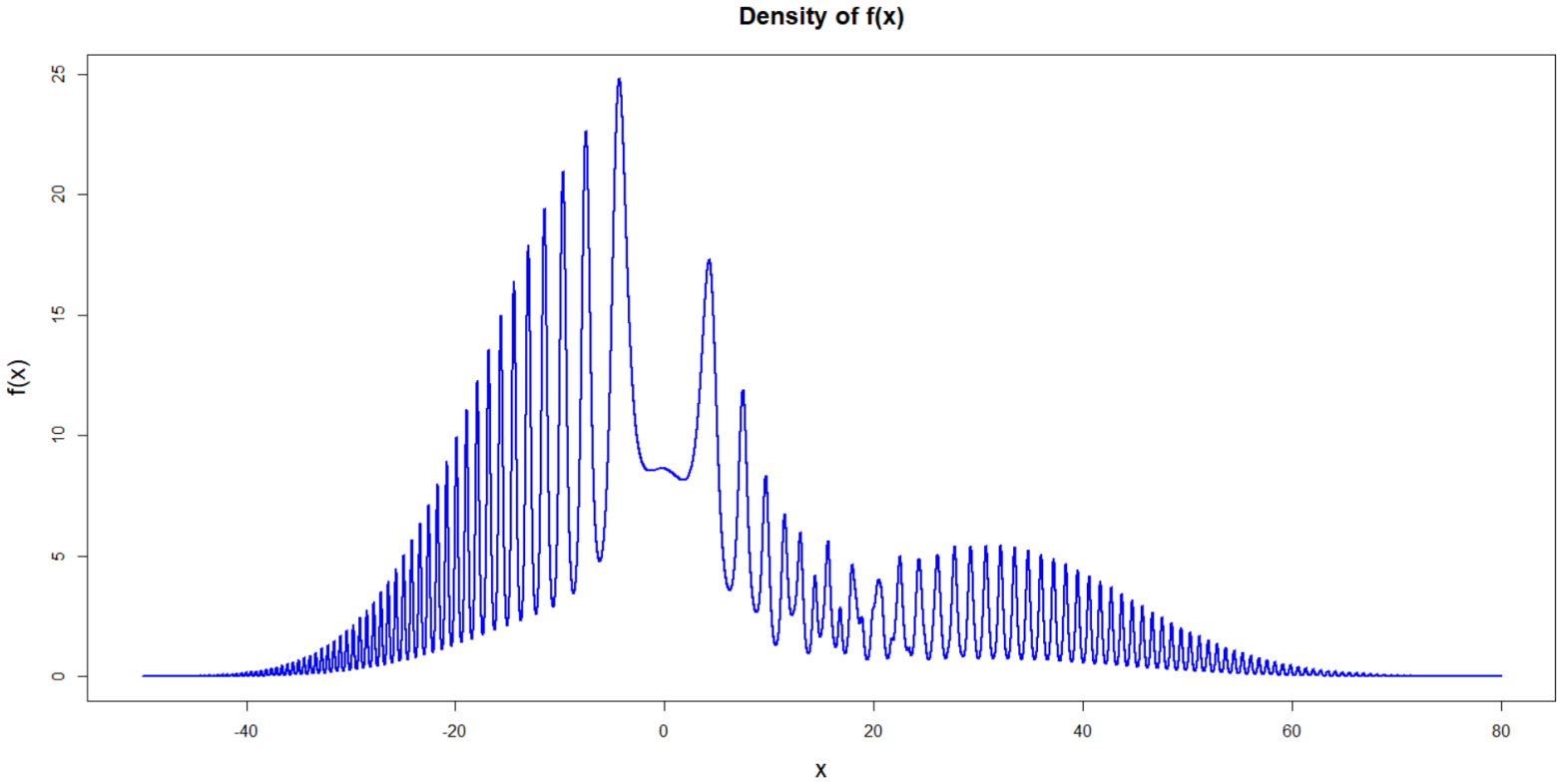


Figure 1.1 Plot of density f(x)

Part b: MCMC Algorithms

Part i. Simulated Annealing

The simulated annealing algorithm was used to provide an estimate of the global mode of f. The global mode for which the simulated annealing algorithm was targeting can be seen in *Figure 1bi* where it is marked by a red circle at the approximate co-ordinates $x = -4.328, y = 24.808$.

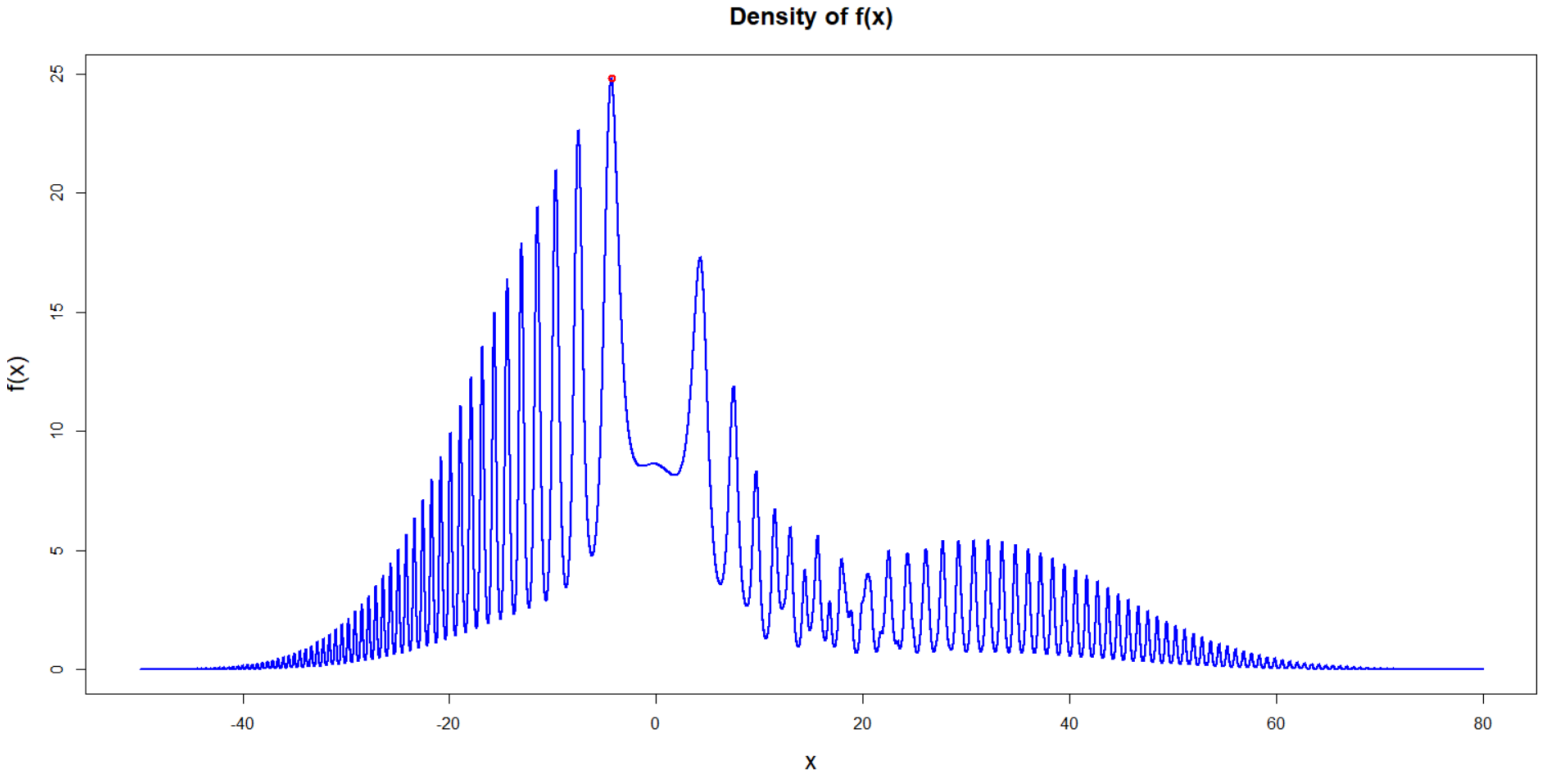


Figure 2.1. Plot of density f(x) with the global mode marked in red.

Simulated Annealing Algorithm in R

The function `simulated_annealing` was written in R to implement the simulated annealing algorithm. A normal random walk proposal was used. The 'temperature' (β) was increased by means of a geometric annealing schedule whereby $B_t = \alpha B_{t-1}$ for constant $\alpha > 1$. The function is as below. Note that commented code is in green.

Simulated Annealing in R

```
simulated_annealing = function(N, x0 = 0, beta1 = 1, alpha = 1.001, sigma =
1){

  #Vectors for storage
  x = vector('numeric', N)
  fn = numeric(N)
  count_na = 0

  #Current values of variables
  x_current = x0
  fx_current = get_fx(x_current)
  beta_current = beta1

  #Implement Markov Chain in loop
  for (i in 1: N){

    x_new = x_current + rnorm(1)*sigma #Normal random walk proposal
    fx_new = get_fx(x_new)
    alpha_accept = beta_current*(log(fx_new)-log(fx_current))#alpha
(acceptance probability)

   else if (alpha_accept > log(runif(1))){ #Criterion for acceptance
      #If holds set next sample in the chain as the proposed sample
      x_current <- x_new
      fx_current <- fx_new
    }
    #If not the next sample in the chain is again set to as the sample
value at previous step
    x[i] = x_current
    fn[i] = fx_current
    beta_current = beta_current*alpha #Update temperature/value of beta


  }

  list(x, fn) #Return x and f(x)

}
```

Tuning Parameters

The parameters considered for tuning in the case of simulated annealing were the proposal scale σ and the rate of annealing α. The set of proposal scales and annealing rates considered were respectfully;

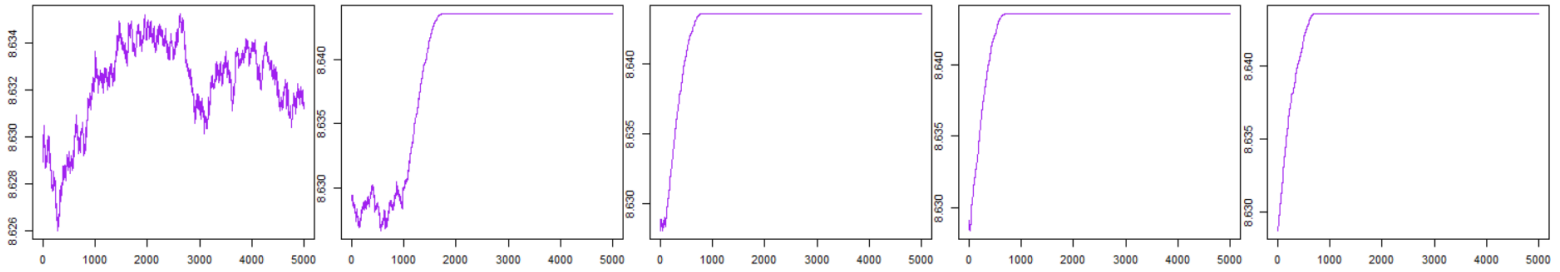
$\sigma \epsilon \{0.001, 0.01, 0.1, 1, 10\ 100\}$ and $\alpha = \{1.001, 1.01, 1.1, 2, 10\}$

To get an idea of the best configurations, the algorithm was run for all combinations of σ and α and their respective plots were then examined and compared. The chain was firstly run for 5000 iterations to get an idea of how fast the various configurations were and subsequently it was run for 100,000 to determine which configurations did in fact converge to the global optimum as time approached infinity. The case of 5000 iterations will firstly be considered, as seen in Figure 1.1.2

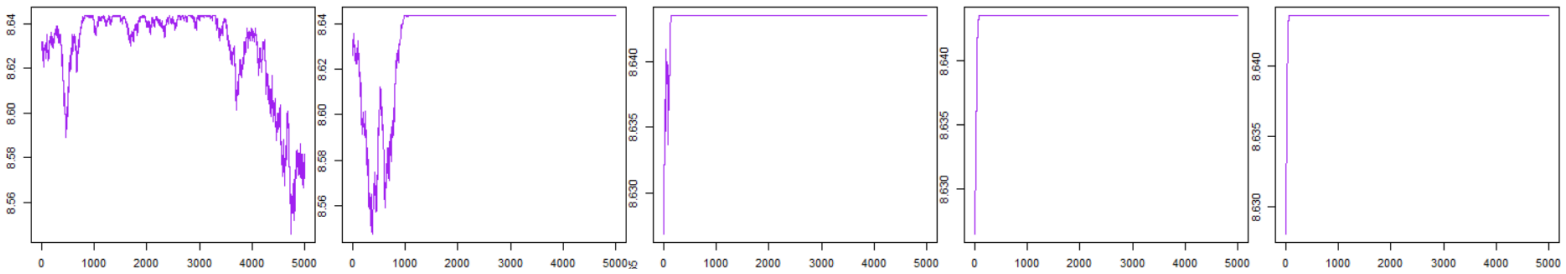
Simulated Annealing, N = 5000

$\alpha = 1.001$ $\alpha = 1.01$ $\alpha = 1.1$ $\alpha = 2$ $\alpha = 10$

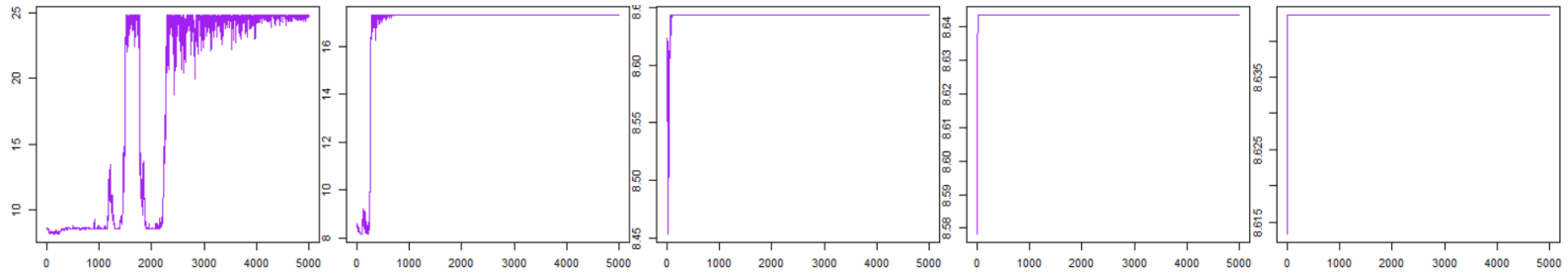
$\boldsymbol{\sigma = 0.001}$



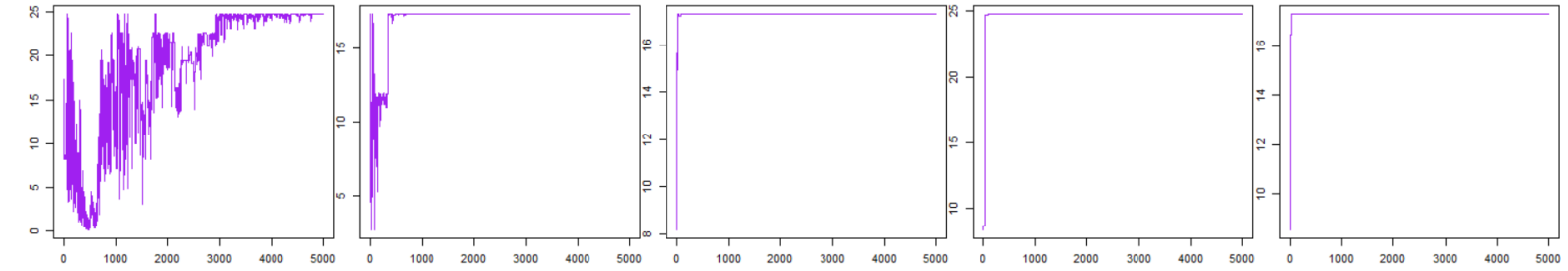
$\boldsymbol{\sigma = 0.01}$



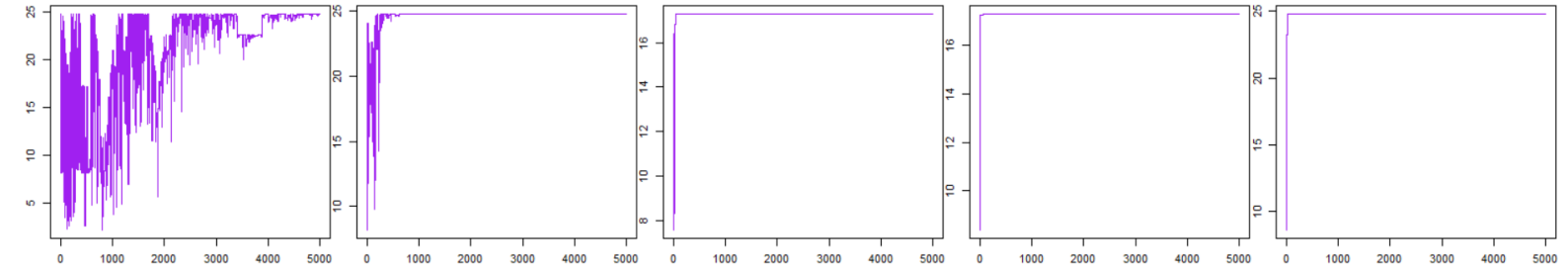
$\sigma = 0.1$



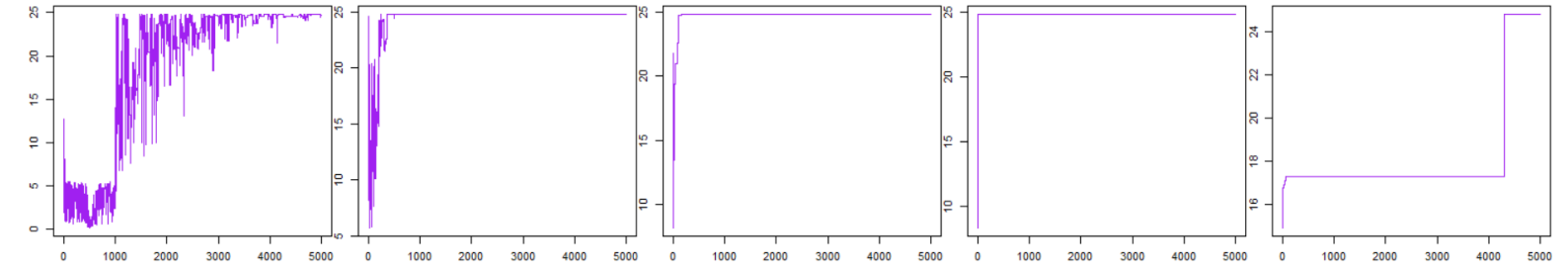
$\sigma = 1$



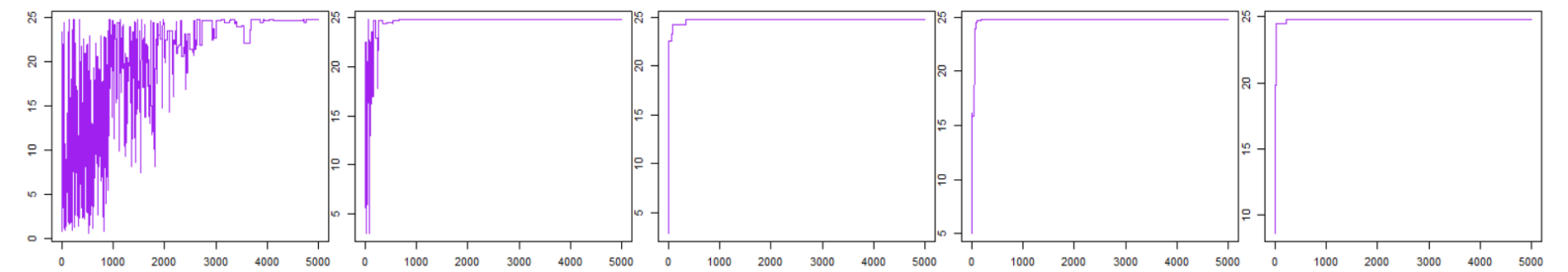
$\sigma = 2$



$\sigma = 5$



$\sigma = 10$



$\boldsymbol{\sigma = 100}$

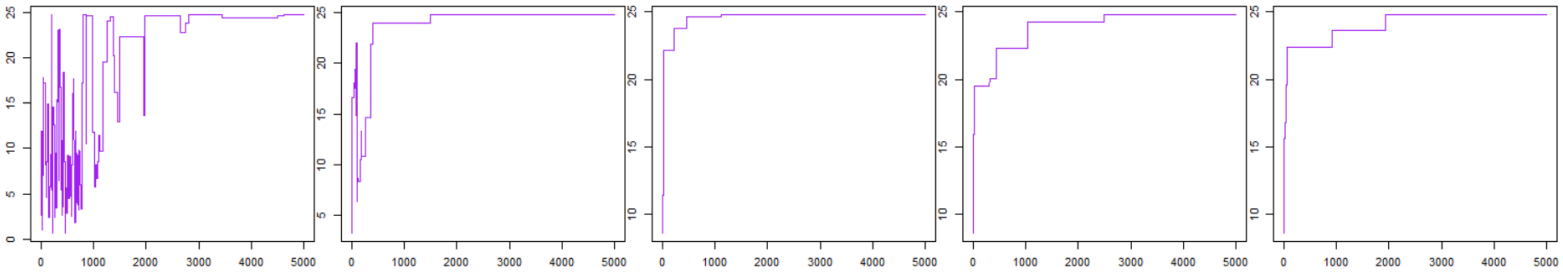


Figure 2.2 Simulated Annealing Algorithm run for 5000 iterations for $\sigma = \{0.001, 0.01, 0.1,1, 10,100\}, \alpha = \{1.001, 1.01, 1.1, 2, 10\}$.

Assessment

Several interesting features regarding the convergence of the algorithm can be observed from Figure 2.1 The best configurations of the chain converge very fast, that is for $\sigma \sim 1$ and $\alpha = \{1.01,\ 1.1\}$. In both cases the algorithm converges to the correct global mode of approximated 25 within 1000 iterations.

Considering the proposal scale σ across all values tested ($\sigma \epsilon \{0.001, 0.01, 0.1, 1,10, 100\}$), the outcome is as expected. In cases were sigma is large the chain doesn't move around the parameter space in an optimal way ($\sigma \epsilon \{5,10,100\}$). Due to the large variance of the random walk proposal, many extreme proposals are put forward. The algorithm rejects many such proposals and ends up getting trapped at the same location for a number of iterations as seen in the last three rows of Figure 2.1 This is undesirable as the chain is therefore slower to converge to the targeted extremum, the global mode in this case. On the other hand, in cases were sigma is small, the algorithm is much slower at moving around the parameter space and finding the global mode ($\sigma \epsilon \{0.001,0.01,0.1\}$). The algorithm can often get trapped at a local mode as seen in the first three rows of Figure 2.1 In all such cases the algorithm has failed to reach the global mode within 5000 iterations, getting trapped at local maxima.

Considering the annealing rate α across all values implemented, the value doesn't have as a dramatic impact on the nature of convergence of the chain as does the proposal scale σ. On the whole, there is little difference in the chain convergence between iterations of say α equal to 1.1, 2 or 10. However for smaller values of α, namely 1.001 and to some extent 1.01, the convergence of the chain is much slower. This is as expected, as in this case the annealing temperature β is relatively small in comparison to iterations with greater annealing rates. As a result the global mode isn't as prominent in comparison to other local maxima and so it takes longer for the chain to converge.

Diagnostics

Diagnostics will only be considered across the α values for one value of σ chosen to be one, i.e;

$\sigma = 1$ and $\alpha = \{1.001, 1.01, 1.1, 2, 10\}$

Note that this set contains the best configurations in terms of mixing, namely; $\sigma \sim 1$ and $\alpha = \{1.01,\ 1.1\}$.

Autocorrelation

Although the values are quite high, the parameterisation of alpha that displayed the best mixing properties i.e α = 1.1, also has the lowest autocorrelation

	σ = 1				
α	1.001	1.010	1.10	2.00	10.00
Autocorrelation (first)	0.994	0.969	0.923	0.950	0.951

Table 2.1

Acceptance Rate

The acceptance rate is defined as;

$$\text{Acceptance Rate} = \frac{\text{Number of Accepted Proposals}}{\text{Number of Proposals made}}$$

The acceptance rate is quite low at these configurations. It highlights the apparent inefficiency of the this form of the algorithm when these configurations are applied

	σ = 1				
α	1.001	1.010	1.10	2.00	10.00
Acceptance Rate	0.152	0.173	0.174	0.165	0.179

Table 2.2

Average Squared Jumping Distance (ASJD)

Although there is not a major difference across values of α, the α value with the best 'Average Squared Jumping Distance' (ASJD) is that of 1.010 which also lines up with the results shown in Figure 2.2. That is given this value of the ASJD, it appears to explore the state space quite well and so is able to identify and converge to the global mode quickly

	σ = 1				
α	1.001	1.010	1.10	2.00	10.00
ASJD	0.121	0.228	0.194	0.226	0.221

Table 2.3

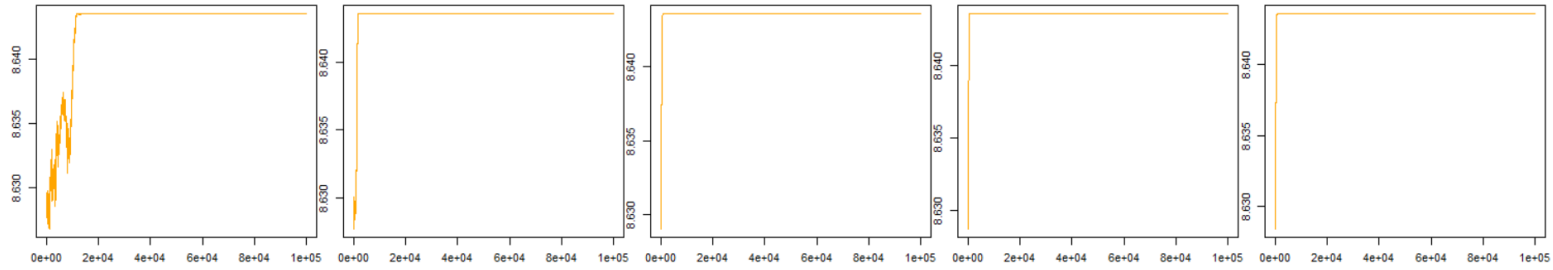
Comparison to Objective

The chain was again run for 100,000 iterations to determine if, in all cases, the algorithm did in fact converge to the global mode in the limit. The traces are shown in Figure 2.3, however the results are somewhat comparable to those in Figure 2.2 when the chain was run for 5,000 iterations. Most configurations do converge to the global mode well within 100,000 iterations. That is they converge to a value of approximately 24.808, which is the targeted global mode as shown in Figure 2.1 It is still the case for small proposal scales, that the algorithm gets trapped in local maxima, never reaching the global mode. The best configurations of the chain are still at values for $\sigma \sim 1$ and $\alpha = \{1.01,\ 1.1\}$.

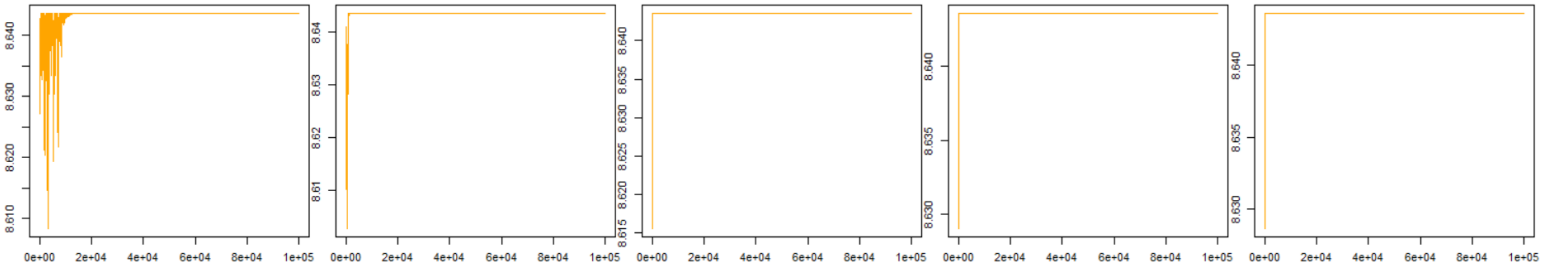
Simulated Annealing, N = 100,000

$\alpha = 1.001$ $\alpha = 1.01$ $\alpha = 1.1$ $\alpha = 2$ $\alpha = 10$

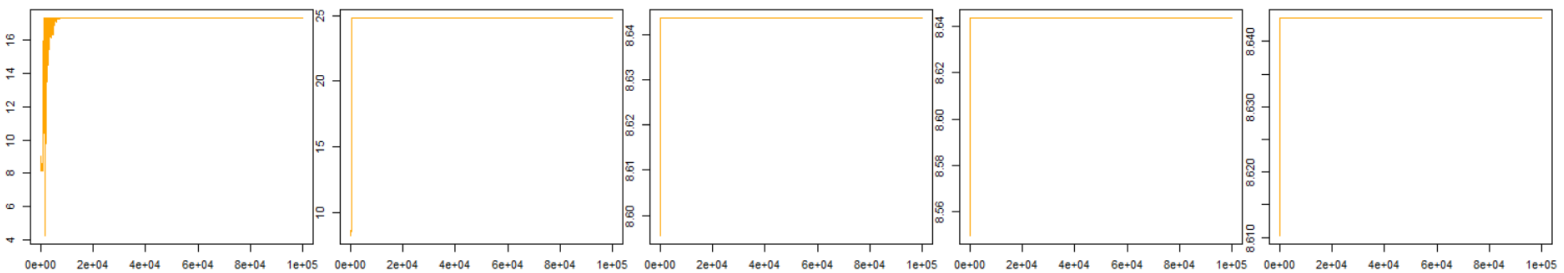
$\boldsymbol{\sigma = 0.001}$



$\boldsymbol{\sigma = 0.01}$

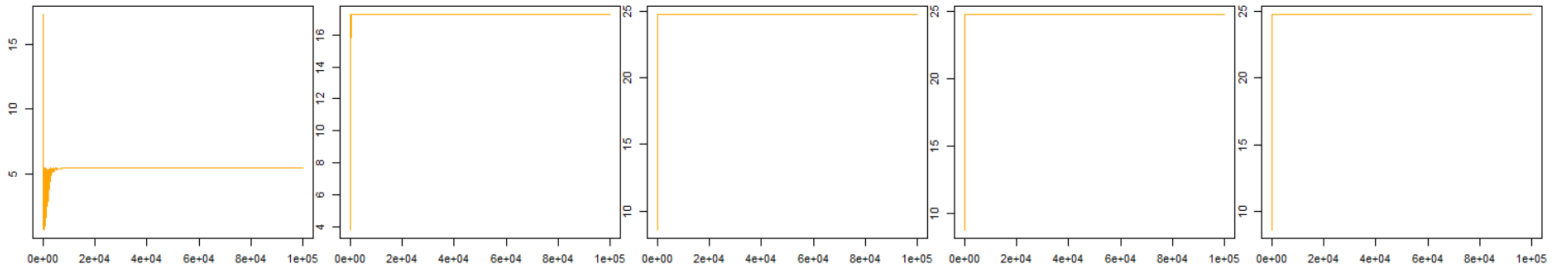


$\boldsymbol{\sigma = 0.1}$

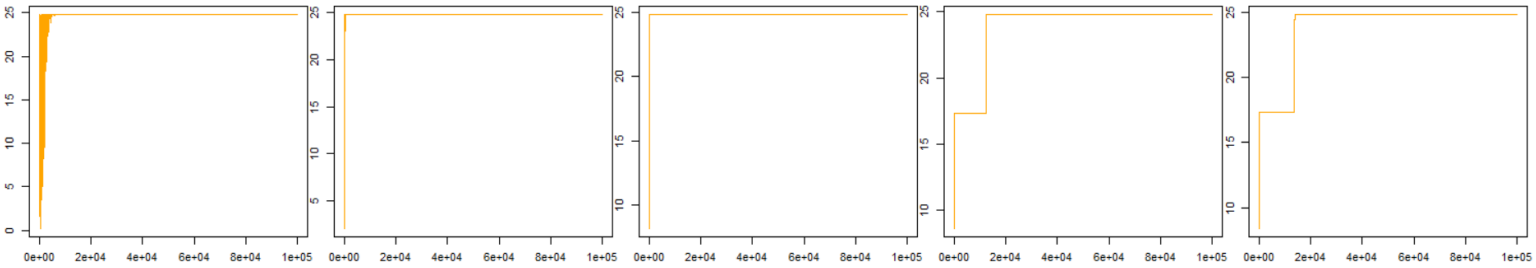


$\alpha = 1.001$ $\alpha = 1.01$ $\alpha = 1.1$ $\alpha = 2$ $\alpha = 10$

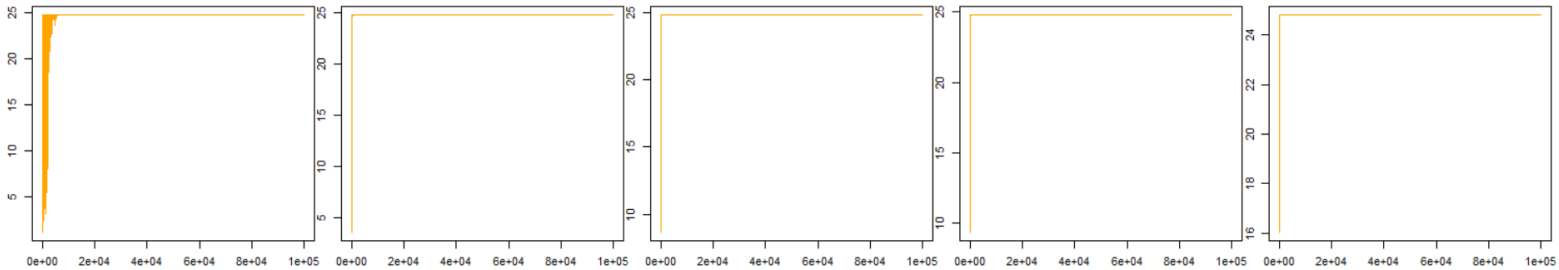
$\sigma = 1$



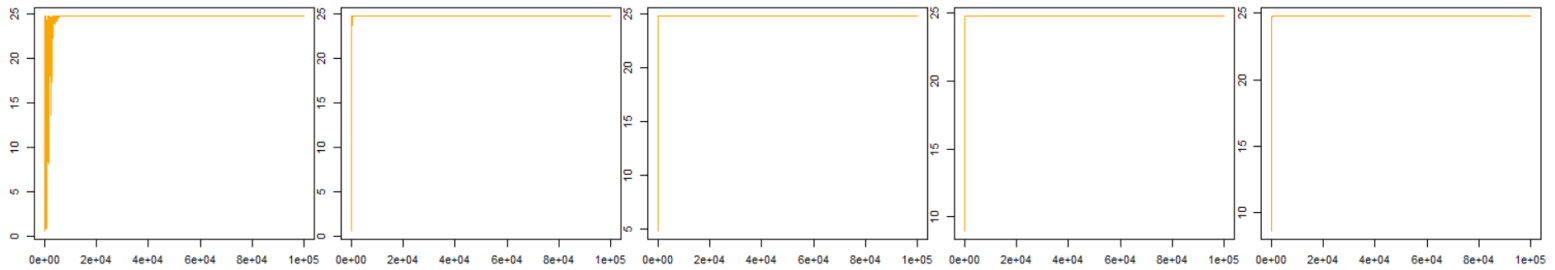
$\sigma = 2$



$\sigma = 5$



$\sigma = 10$



$\boldsymbol{\sigma = 100}$

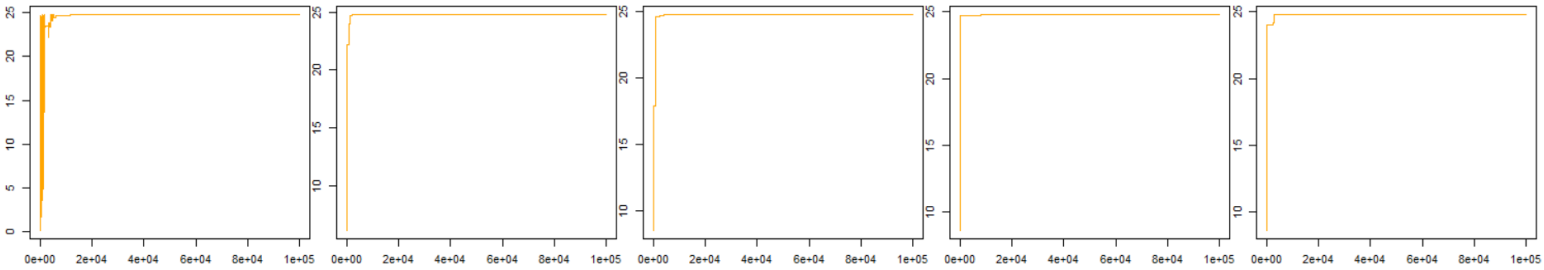


Figure 2.3 Simulated Annealing Algorithm run for 100,000 iterations for $\sigma = \{0.001, 0.01, 0.1, 1, 10, 100\}, \alpha = \{1.001, 1.01, 1.1, 2, 10\}$.

Part ii. Random Walk Metropolis Algorithm

The random walk Metropolis algorithm was implemented with f being its invariant distribution. The Metropolis algorithm is a special case of the Metropolis-Hastings algorithm whereby the proposal is a symmetric random walk. The function *rwm* was written in R to implement the algorithm. The function requires that an intial value of the chain. 'Rwm' returns samples from the invariant distribution f aswell as the count of accepted values.

Random Walk Metropolis Hastings function in R

```
rwm = function(N, sigma, x0 = 2){

  #Vectors to store samples
  X = vector('numeric', N)
  X[1] = x0 #Initialise 1st sample in chain in order to start the chain
running
  count_accept = N #Count of number of accepted values

  #Loop - Markov chain
  for (t in 2:N){
    X[t] = X[t-1] + rnorm(1, mean = 0, sd= sigma) #Symmetric random walk
proposal
    u = runif(1)

    if(u > (get_fx(X[t])/get_fx(X[t-1]))) { #Criterion for rejection
      X[t] = X[t-1] #If criterion for acceptance is not met the next sample
in chain is set to current sample
      count_accept = count_accept - 1
    }
  }
  list(X, count_accept) #Return samples and count of accepted values
}
```

Tuning Parameters

The proposal scale σ was considered for tuning in the case of the random walk metropolis. The set of proposal scales considered was;

$$\sigma \in \{0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8, 25.6, 51.2, 102.4, 153.6, 204.8\}$$

The chain was firstly run for 5000 iterations to get an idea of how well the chain was mixing. The traces are shown in Figure 3.1. The first two rows exhibit very poor mixing. The chain moves very slowly around the space of interest as the proposals are so conservative given the small proposal scale. The mixing improves greatly as σ increases from 6.4 to 12.8, with the best mixing occurring at σ values of 25.6 and 51.2. The mixing then begins to worsen again as σ is increased further to 102.4, 153.6 and 204.8 as shown in the final row. Here the large proposal scale results in extreme proposals, many

of which are rejected and as a result the chain remains at a given state for a large number of iterations, i.e the mixing is suboptimal.

Random Walk Metropolis N = 5000

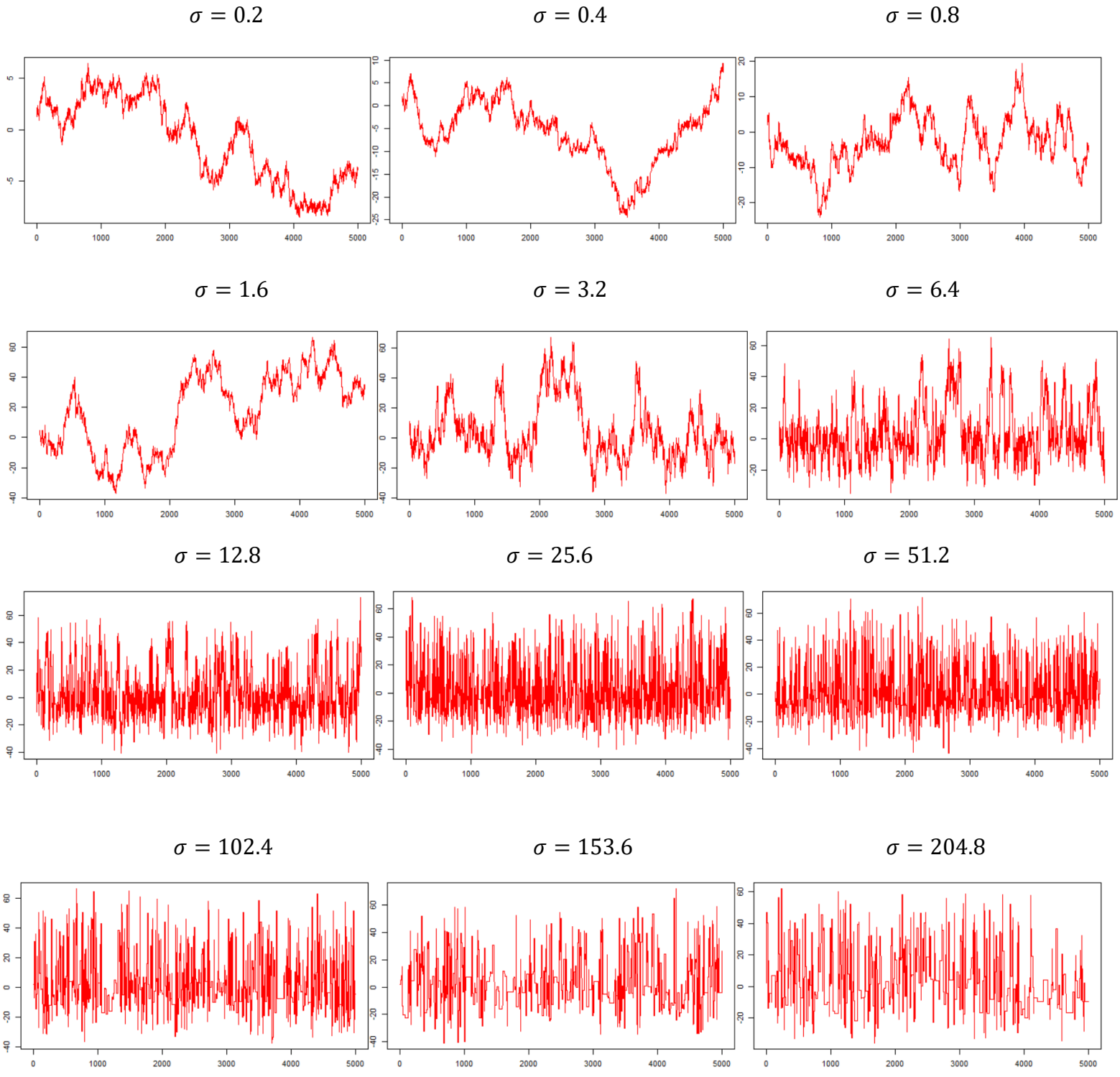


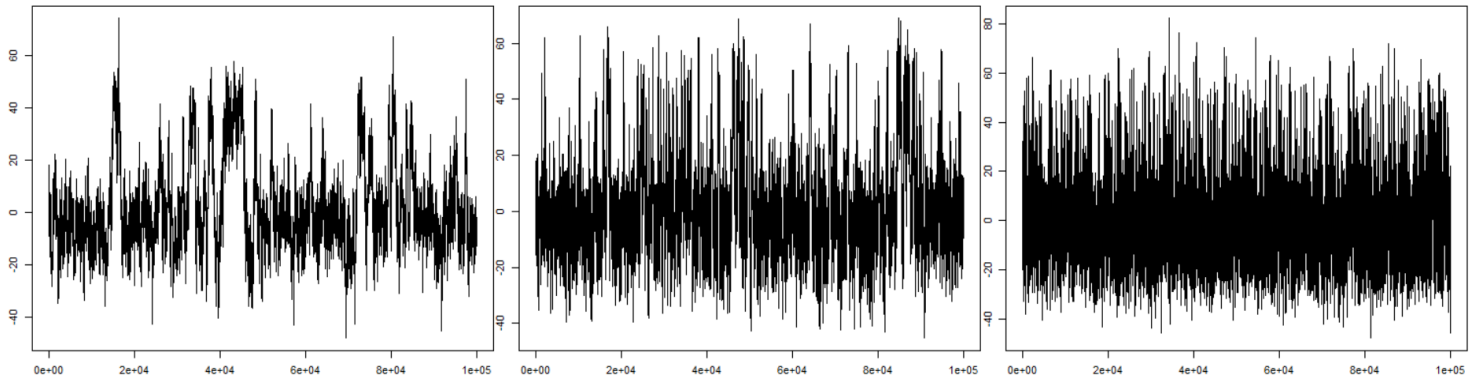
Figure 3.1 Sample paths of X_t resulting from the Random Walk Metropolis algorithm run for 5,000 iterations for $\sigma \in \{0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8\ 25.6, 51.2, 102.4, 153.6, 204.8\}$

Given the poor mixing displayed by the chain for small values of σ (Figure 3.1 row 1; $\{\sigma = 0.2, 0.4, 0.8\}$) and the largest value of σ $\{\sigma = 204.8\}$, these values were discarded going forward and the tuned set of proposal scales considered were;

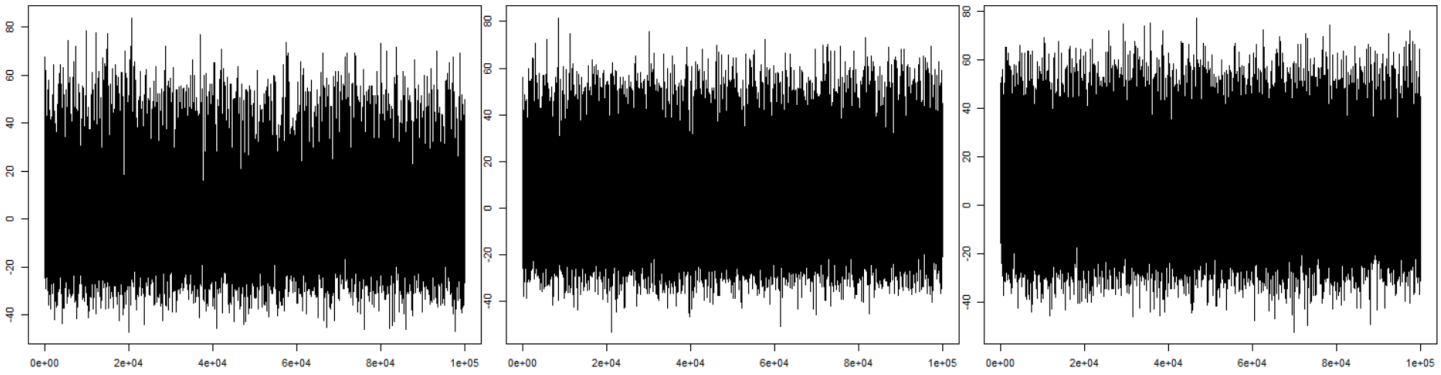
$\sigma \epsilon \{1.6, 3.2, 6.4, 12.8, 25.6, 51.2\ 102.4, 153.6\}$

To continue with the analysis the chain was again run for 100,000 iterations with the traces shown in Figure 3.2. As expected the chain displays very good mixing and exploration of the state space, particularly for σ increases from 6.4 to 12.8, 25.6 and 51.2. Comparatively, the mixing displayed by the chain at the smallest scales considered $\{\sigma = 1{,}6, 3.2\}$ isn't as good or 'explorative'. In both cases and particularly in the case of the former, the chain spends significant periods of time within the modes, namely 'sticking' within the mode, which prevents it from fully exploring the state space.

$\sigma = 1.6$ $\sigma = 3.2$ $\sigma = 6.4$



$\sigma = 12.8$ $\sigma = 25.6$ $\sigma = 51.2$



$\sigma = 102.4$ $\sigma = 153.6$

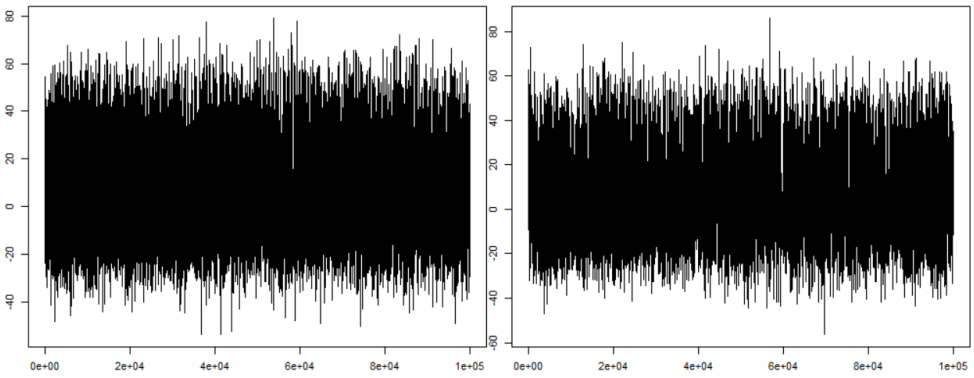


Figure 3.2. Sample paths of X_t resulting from the Random Walk Metropolis algorithm run for 100,000 iterations for $\sigma \in \{1.6, 3.2, 6.4, 12.8, 25.6, 51.2, 102.4, 153.6\}$

Diagnostics

Autocorrelation

The first autocorrelation for each of the proposal scales considered is given in Table 3.1. The figures line up with what was observed in terms of the chain's mixing ability in Figure 3.2. That is, the chains which displayed the best mixing also have the best autocorrelation $\{\sigma = 25.6, 51.2, 102.4\}$ In these cases, the chain makes reasonable moves from one step to the next and therefore successive samples are less correlated than they are in cases where the proposal scale is smaller and in which the proposals are much more conservative. The best autocorrelation of 0.713 is at a proposal scale of 51.2. The chain appears to mix in an optimal manner at this value of σ, it has found a good balance between exploring the space sufficiently without proposing candidates that are too extreme. As in the case of the later, when proposals are too extreme, the chain gets stuck in the same state for a number of iterations, thus increasing the overall autocorrelation value. This can be seen as sigma increases from 102.4 to 153.6. The worst autocorrelation of 0.997 is at the smallest proposal scale of 1.6. This is unsurprising given that successive samples are highly correlated due to the small variance of the random walk proposal.

Table 3.1 First autocorrelation of the chain for a given proposal scale σ

σ	**1.6**	**3.2**	**6.4**	**12.8**	**25.6**	**51.2**	**102.4**	**153.6**
Autocorrelation (first)	0.997	0.991	0.966	0.901	0.776	0.713	0.797	0.853

Acceptance Rate

The acceptance rate for each of the proposal scales considered is shown in Table 3.2. The results align with that which was seen in the trace plots. The chains with smaller proposal scales accept a lot more proposals than those with greater proposal scales. This follows from the form of the probability of acceptance α in the Metropolis step, namely the ratio of the proposal density to that of the current density. In cases where σ is small, a proposal will be very similar to that of the current state and therefore this ratio will be close to 1. As a result the acceptance criterion of the Metropolis step will be met, more often than not as seen for σ values of 1.6, 3.2 and 6.4. The opposite is true in cases where σ is large, the acceptance rate is significantly smaller, it is merely 0.165 and 0.11 for sigma values of 102.4 and 153.6 respectfully. This reflects what was seen in Figure 3.1 where the chain remains at a given state for a number of iterations, rejecting many proposals.

Table 3.2. Acceptance Rate of the chain for a given proposal scale σ

σ	**1.6**	**3.2**	**6.4**	**12.8**	**25.6**	**51.2**	**102.4**	**153.6**
Acceptance Rate	0.713	0.686	0.653	0.578	0.447	0.296	0.165	0.110

Average Squared Jumping Distance

The 'Average Squared Jumping Distance ' (ASJD) for each value of σ are shown in Table 3.3. The values are unexpectedly high. It highlights the fact that for large proposal scales, extreme candidates are proposed and the overall jumping distance is large. Remark that it is a squared value which will naturally lead to larger values.

Table 3.3 The Average Squared Jumping Distance for a given proposal scale σ

σ	**1.6**	**3.2**	**6.4**	**12.8**	**25.6**	**51.2**	**102.4**	**153.6**	**204.8**
ASJD	1.748	6.693	23.491	69.106	155.624	193.302	139.087	102.68	77.237

Assesment

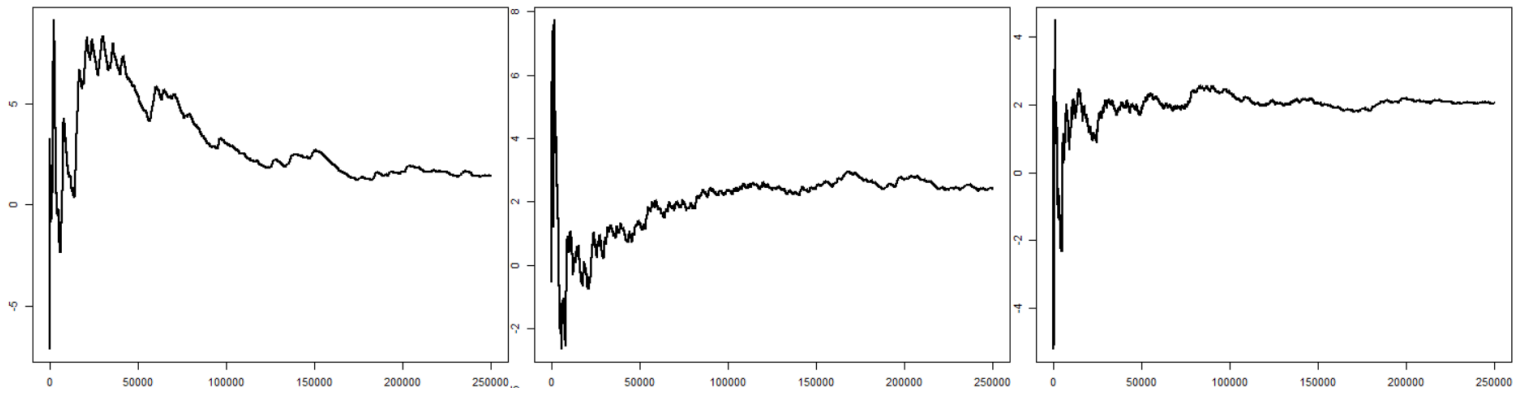
It now follows to ascertain whether the chain did in fact converge to the invariant distribution f. The ergodic average of the chains will be considered as well as the overall distribution of the samples

Ergodic Average

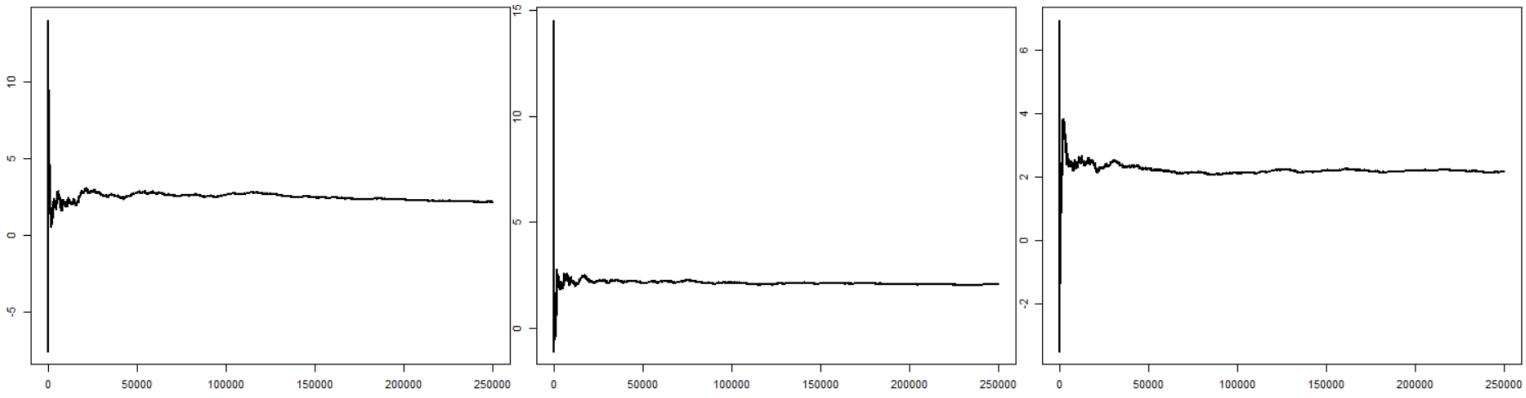
The ergodic mean of the chain was firstly examined for 100,000 iterations of the markov chain. However, it appeared that, for several values of sigma, the chain hadn't quite settled down to its equilibrium distribution and so the process was repeated again for 250,000 iterations as shown in Figure 3.3. For $\sigma \geq 12.8$, the chain has reached a steady state or invariant distribution, with an approximate value ~ 2.15. In contrast to this the plots for the smaller values of sigma of 1.6 and 3.2 contain some fluctuations, although they are broadly close enough to the correct value.

The final value of the chain across the last 1000 and 10 samples are shown respectfully in table 3.4. For the optimal values of σ of 25.6, 51.2 and 102.4 there is only a difference of 0.001 between the average of the last 1000 samples and that of the last 10 samples, which suggests that the chain has reached it's invariant distribution f.

$\sigma = 1.6$ $\sigma = 3.2$ $\sigma = 6.4$



$\sigma = 12.8$ $\sigma = 25.6$ $\sigma = 51.2$



$\sigma = 102.4$ $\sigma = 153.6$

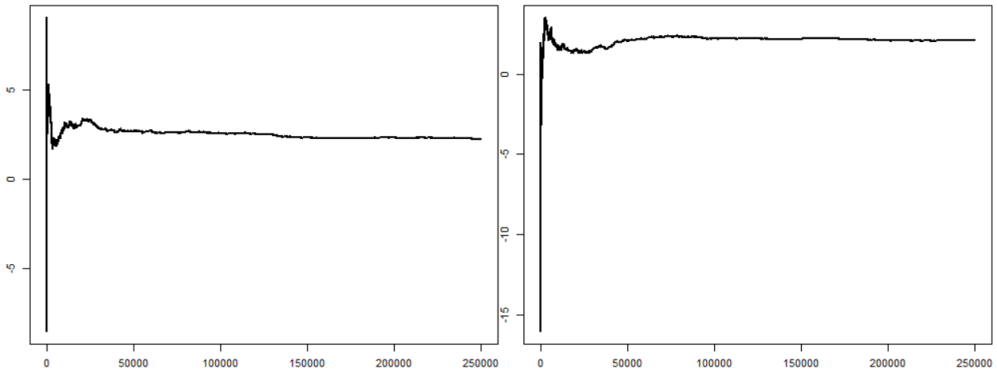


Figure 3.3. Ergodic mean of the chains resulting from the random walk Metropolis algorithm run for 250,000 iterations for $\sigma \in \{1.6, 3.2, 6.4, 12.8, 25.6, 51.2, 102.4, 153.6\}$

Table 3.4. The average of the chain across the last 1000 & last 10 samples for the RWM run for 250,000 iterations

σ	1.6	3.2	6.4	12.8	25.6	51.2	102.4	153.6
Ergodic Mean (final 1000 samples)	1.453	2.430	2.044	2.157	2.073	2.154	2.203	2.112
Ergodic Mean (final 10 samples)	1.451	2.413	2.049	2.149	2.072	2.153	2.204	2.117

Empirical Density - Histogram

To compare with the objective, that is the function $f(x)$, the empirical histogram of the samples resulting from the RWM for each value of σ were rendered (Figure 3.4). Qualitatively assessing the output, the empirical histograms appear to be samples from the target $f(x)$, that is the RWM algorithm was able to produce the correct samples, with $f(x)$ being it's invariant distribution.

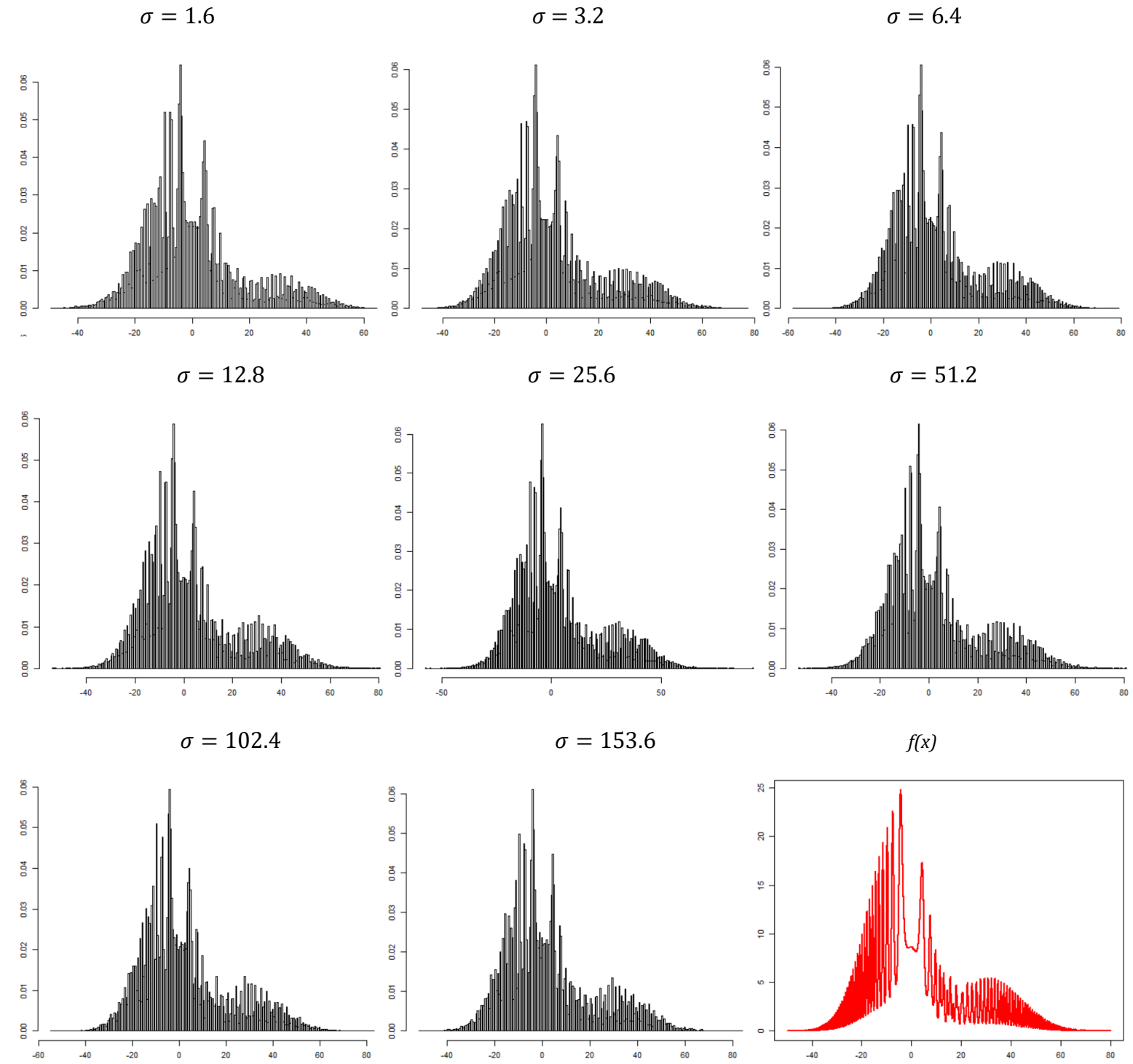


Figure 3.4. Empirical histogram of the chains resulting from the random walk Metropolis algorithm run for 250,000 iterations for $\sigma \in \{1.6, 3.2, 6.4, 12.8, 25.6, 51.2, 102.4, 153.6\}$

Part iii. The Slice Sampler

The Slice sampler was implemented with f being its invariant distribution.

Algorithm

The Slice sampler is an algorithm which allows expectations to be approximated with respect to $f(x)$ by Gibbs Sampling from $\bar{f}(x,u) \propto \mathbb{I}_{[0,f(x)]}u$. Slice Sampling works on the basic principle that a target density can be sampled from by sampling uniformly from the region under the graph of its density function. Remark the algorithm works by adding the auxiliary variable u, temporarily augmenting the sample space.

There are various forms of the Slice Sampling algorithm and in this instance the Metropolised Slice Sampler was implemented, given that it is much more generally applicable than the purer form (Johansen A.M, 2018) However the 'pure' Slice Sampling method (Neal, 2003) was also investigated as an additional method for (b.iv) and the function created in R is included in the Appendix.

The Metropolised Slice Sampler Algorithm works as follows (Johansen A.M, 2018);

- Initialise the chain , i.e set $(X^{(0)}, U^{(0)}$ and iterate for t = 1,2,...;
  - Draw $X \sim \bar{q}(.|X^{(t-1)}, U^{(t-1)})$
  - Accept and set $X^t = X$ with probability given by;
    - Otherwise Reject and set $X^t = X^{t-1}$
  - Draw $U^t \sim \bar{f}_{U|X}(.|X^{(t)})$

The function `slice_sampler_metropolis` was written in R to implement the algorithm ***as below.***

Slice Sampler function in R

```
slice_sampler_metropolis = function(N, sigma, x0 = 2){

  #x- markov chain
  x = vector('numeric', N)
  u = vector('numeric', N)
  x[1] = x0        #Initialise
  count_accept = N

  #Markov Chain starts
  for (t in 2:N){

    #Auxilliary Uniform variable u
    u[t] = runif(1, 0, get_fx(x[t-1]))

    #X sampled from markov chain using a symmetric random
    x[t] = x[t-1] + rnorm(1, mean = 0, sd= sigma)

    #Using augmented distribution f(x,u)
```

```
    if(u[t] > get_fx(x[t])) {          #Rejection criterion
      x[t] = x[t-1]
      count_accept = count_accept - 1
    }
  }
  print(count_accept/N) #Acceptance rate

  x          #Return samples only (auxiliary variable discarded)
}
```

Tuning Parameters

The proposal scale σ was considered for tuning in the case of the slice sampler algorithm. Given that, like the RWM algorithm, the Metropolised Slice Sampler contains a metropolis step for drawing $X^{(t)}$, and that a significant amount of investigation was carried out into the proposal scale in the case of the RWM algorithm, the same set of attuned proposal scale will be considered here, i.e;

$\sigma \epsilon \{1.6, 3.2, 6.4, 12.8, 25.6, 51.2, 102.4, 153.6, 204.8\}$

The chain was run for 100,000 iterations with the traces shown in Figure 4.1. The results are very similar to that of the RWM algorithm. The chain displays very good mixing and exploration of the state space, particularly for σ increases from 6.4 to 12.8, 25.6 and 51.2. Comparatively, the mixing displayed by the chain at the smallest scales considered $\{\sigma = 1{,}6, 3.2\}$ isn't as good or 'explorative'. In both cases and particularly in the case of the former, the chain spends significant periods of time within the modes, namely 'sticking' within the mode, which prevents it from fully exploring the state space.

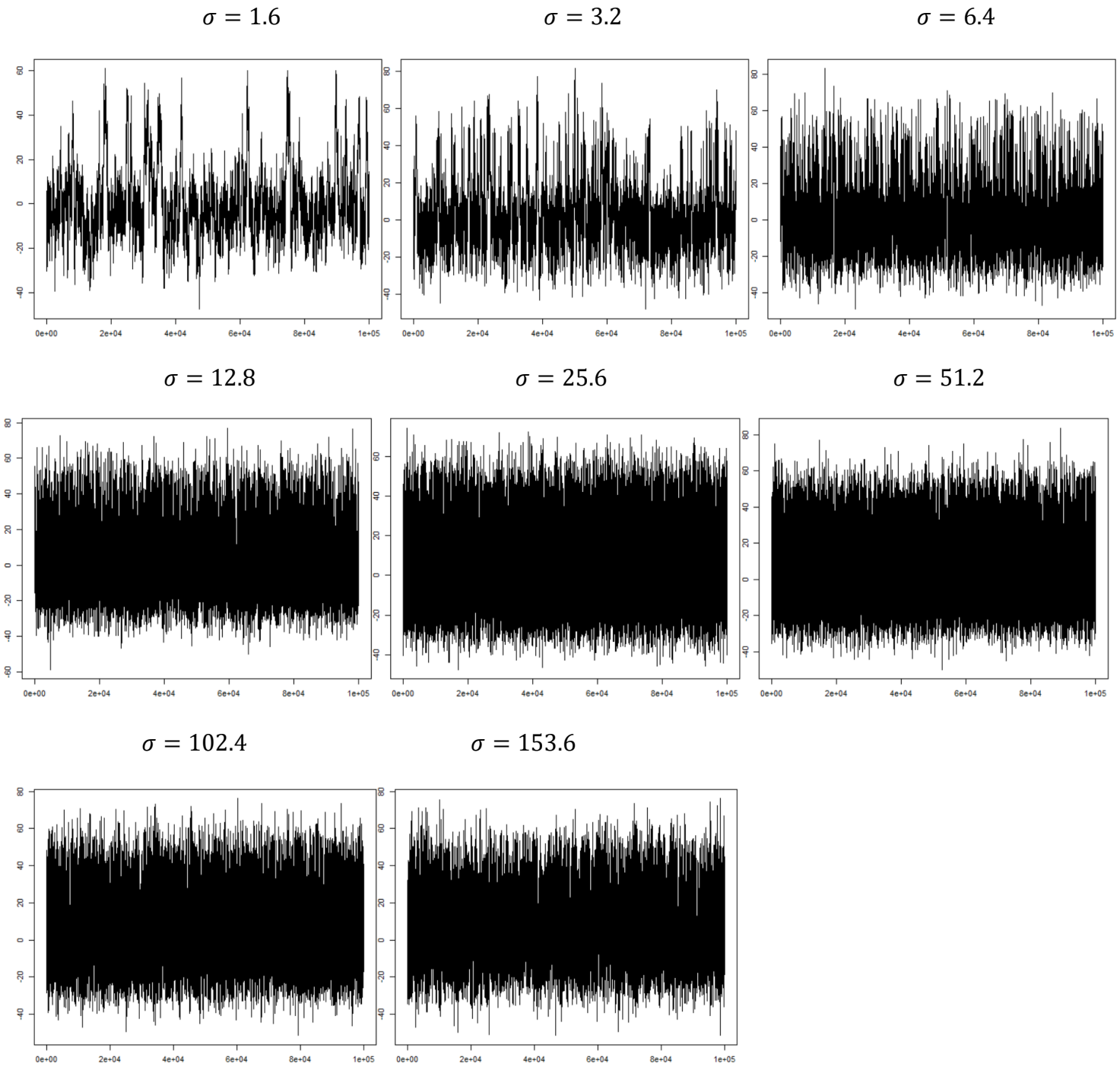


Figure 4.1. *The Slice Sampler Algorithm (Metropolised) run for 100,000 iterations for* $\sigma \in \{1.6, 3.2, 6.4, 12.8, 25.6, 51.2, 102.4, 153.6\}$

Diagnostics

It transpired that the diagnostic results of the Slice Sampler were extremely close to those of the RWM Algorithm. To save from repetition, only the three best sigma values, in terms of their apparent mixing ability and autocorrelation, were considered for Diagnostics for the Slice sampler namely;

$$\sigma \epsilon \{ 25.6, 51.2, 102.4\}$$

Autocorrelation

The first autocorrelation for each of the proposal scales considered is given in Table 4.1. Similar to the case of the RWM algorithm, the autocorrelation values are quite good at these 'well-balanced' proposal scales.

Table 4.1 First autocorrelation of the chain for a given proposal scale σ

σ	**25.6**	**51.2**	**102.4**
Autocorrelation (first)	0.837	0.791	0.847

Acceptance Rate

The acceptance rates are reasonable. Again we can see for greater proposal scales, more extreme candidates are proposed and so the acceptance rate is lower.

Table 4.2 Acceptance rate of the chain for a given proposal scale σ

σ	**25.6**	**51.2**	**102.4**
Acceptance Rate	0.451	0.289	0.160

Average Squared Jumping Distance

The 'Average Squared Jumping Distance' (ASJD) for each value of σ are shown in Table 3.3. Again the values are very high, highlighting the extent to which the chain moves around the state space at larger values of the proposal scale.

Table 4.3 The Average Squared Jumping Distance for a given proposal scale σ

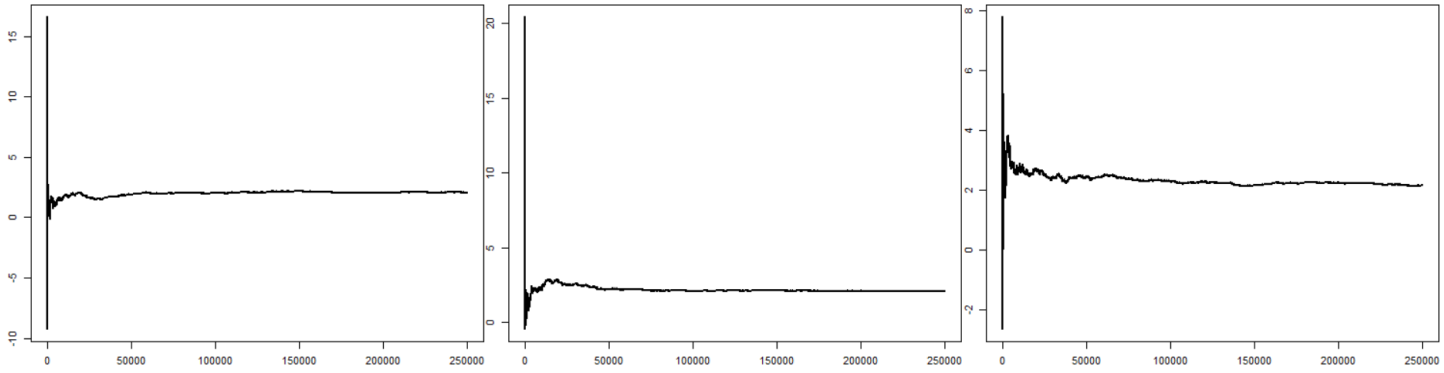
σ	**25.6**	**51.2**	**102.4**
ASJD	157.320	195.681	143.169

Assessment

The ergodic mean of the chains will be considered as well as their overall distribution.

Ergodic Mean

The ergodic mean of the chain was examined for 250,000 iterations of the markov chain. as shown in Figure 4.2 In all three case the chain has reached a steady state or invariant distribution, with an approximate value ~ 2.15. The final value of the chain across the last 1000 and 10 samples are shown respectfully in table 4.4. There is only a difference of 0.002 between the average of the last 1000 samples and that of the last 10 samples, which suggests that the chain has reached it's invariant distribution f.



$\sigma = 25.6$ $\sigma = 51.2$ $\sigma = 102.4$

Figure 4.2 Ergodic mean of the chains resulting from the random walk Metropolis algorithm run for 250,000 iterations for $\sigma \in \{1.6, 3.2, 6.4, 12.8, 25.6, 51.2, 102.4, 153.6\}$

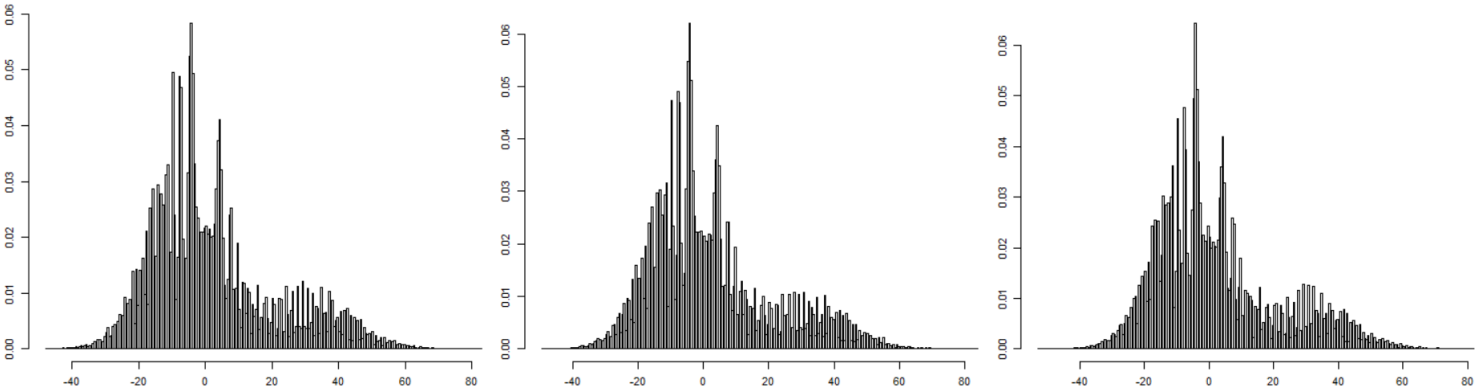
Table 4.4. The average of the chain across the last 1000 & last 10 samples for the RWM run for 250,000 iterations

σ	25.6	51.2	102.4
Ergodic Mean (final 1000 samples)	2.071	2.111	2.161
Ergodic Mean (final 10 samples)	2.073	2.113	2.160

Empirical Density - Histogram

Again to compare with the objective f the empirical histogram of the samples resulting from the Simulated Annealing algorithm were rendered (Figure 3.4). Qualitatively assessing the output, the empirical histograms appear to be samples from the target f (red) that is the RWM algorithm was able to produce the correct samples, with f being it's invariant distribution.

$\sigma = 25.6$ $\sigma = 51.2$ $\sigma = 102.4$



$f(x)$

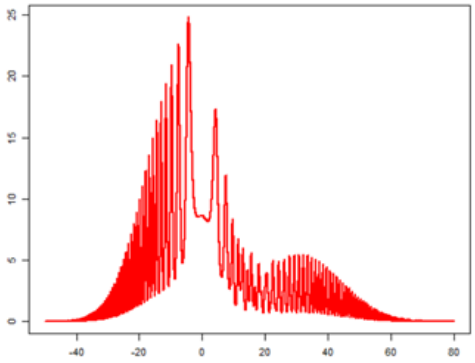


Figure 4.3 Ergodic average of the chains resulting from the Slice Sampling algorithm run for 100,000 iterations

Part iv: Own Proposed Method

Out of interest, two additional Monte Carlo methods were implemented with f as their invariant distribution; one outside the scope of the course, namely Hamiltonian MC and one covered in the course; the general form of the Slice Sampler (See Appendix for the later).

Hamiltonian MCMC

Choice of Algorithm - Interest

After a brief literature review, the area of Hamiltonian MCMC emerged as one method that is of current interest in the area of MCMC research. Given its recent popularity it was deemed an interesting method to explore and ascertain whether it would work sufficiently to sample from f.

Algorithm

Hamiltonian Monte Carlo (HMC) is a Markov chain Monte Carlo (MCMC) method that makes use of Hamiltonian dynamics to produce distant proposals for the Metropolis algorithm. (See Livingstone (2015), Betancourt, Michael (2013) and Neal (2011)). It uses an approximate Hamiltonian dynamics simulation based on numerical integration; this is then corrected for via the Metropolis acceptance step.

The HMC algorithm introduces auxiliary momentum variables, p, to the parameters of the target distribution, q. HMC makes use of the Hamiltonian functions from physics that can be written as;

$$H(q,p) = U(q) + K(p) \qquad [5.1]$$

Whereby;

- $K(p)$ – Kinetic Energy of the system
- $U(q)$ – Is the Potential energy of the system and for HMC defined as the negative of the log probability of the density of the distribution of q that we wish to sample (f in this case), i.e;

$$U(q) = -\log[f(x)] \qquad [5.2]$$

This Hamiltonian function generates a move by firstly sampling the ancillary momentum variables p and then evolving the total system via the following Hamiltonian equations;

Hamiltonian Differential Equations

$$\frac{dq}{dt} = +\frac{\delta H}{\delta p} = +\frac{\delta K}{\delta p}$$

$$\frac{dp}{dt} = -\frac{\delta H}{\delta p} = -\frac{\delta K}{\delta q} - \frac{\delta U}{\delta q} \qquad [5.3]$$

Algorithm implementation

In order to implement the algorithm on a computer, the Hamiltion equations need to be approximated by discretizing time using a small stepsize ε. That is, it involves an iterative process starting at time zero and computing the state at each time step; ε, $2\varepsilon, 3\varepsilon$ etc.

Approximating the solution to a system of Differential Equations

The leap-frog method is one such method approximating the solution to a system of Differential Equations. In the case of Hamilton's equations, for each component of the position and momentum the method performs the following steps, with the components indexed $by\ i = 1, \dots, d$ (d – number of dimensions);

$$p_i\left(t + \frac{\varepsilon}{2}\right) = p_i(t) - \frac{\varepsilon}{2}\frac{\delta U}{\delta q_i}(q(t))$$

$$q_i(t + \varepsilon) = q_i(t) + \varepsilon\frac{p_i(t + \frac{\varepsilon}{2})}{m_i}$$

$$p_i(t + \varepsilon) = p_i\left(t + \frac{\varepsilon}{2}\right) - \frac{\varepsilon}{2}\frac{\delta U}{\delta q_i}(q(t + \varepsilon))$$

Hamiltonian function in R

The function $HMC\_samples$ was implemented in R in order to carry out HMC. It requires as input arguments; ε; the leapfrog stepsize, L; the number of leapfrog steps in the trajectory, q_0 the starting position of the trajectory and N, the number of iterations for which the chain will run. In addition it requires calling the function get_fx_log which returns the negative of the log probability of the density and the function $gradient$ from the package $rootSolve$ which returns the vector of partial derivatives of U given q. The function is as below.

```
HMC_samples = function(N, epsilon, L, q0){

  #Initialise Parameters
  q_samp = vector('numeric', N)
  current_q = q0
  count_accept = 0

  #Start HMC (Markov Chain)
  for (t in 1:N){

    #Current position
    q = current_q
    #Momentum
    p = rnorm(length(q),0,1) # independent standard normal variates
    #Current momentum
    current_p = p

    #Leap frog method
    #U_gradient -> Gradient of Potential energy - Negative log probability
of the function
    grad_U_q = gradient(f = get_fx_log, q)

    #A half step for momentum is made at the beginning
    p = p - epsilon * grad_U_q / 2

    #Alternate full steps for the position and momentum
    for (i in 1:L){

      #A full step for the position is made
      q = q + epsilon * p
      #U_gradient
      grad_U_q = gradient(f = get_fx_log, q)

      #A full step for the momentum is made, except at end of trajectory
      if (i!=L){
        p = p - epsilon*grad_U_q
      }
    }

    #U_gradient
    grad_U_q = gradient(f = get_fx_log, q)
    #A half step for momentum at the end.
    p = p - epsilon*grad_U_q / 2

    #Momentum is negated at end of trajectory in order to make the proposal
symmetric
    p = -p

    # Evaluate potential and kinetic energies at start and end of
trajectory
    current_U = get_fx_log(current_q)# I.e U(current_q)
    current_K = sum(current_p^2) / 2
    proposed_U = get_fx_log(q)
```

```
    proposed_K = sum(p^2) / 2

    #Accept/Rejection step
    # Accept or reject the state at end of trajectory.
    #This returns either the position at the end of the trajectory or the
initial position

    prob_accept = current_U-proposed_U+current_K-proposed_K #Acceptance
probability
    if (runif(1) < exp(prob_accept)){
      current_q = q
      count_accept = count_accept + 1
    }
    q_samp[t] = current_q #Store current_q as sample

  }
  print('Acceptance rate:')
  print(count_accept/N)

  q_samp #Return samples
}
```

To apply this function requires simply executing the following;

```
HMC_samples(N, epsilon, L, q0)
```

In addition the function get_fx_log is written in R as;

```
get_fx_log = function(x){

  #Student number
  sn = c(1,9,8,3,1,6,2)
  #Density
  a = sn[7]*exp(-sin((sn[1]*x^2)/(15-sn[1])) - ((x-3-
(sn[2]*pi))^2)/(2*(5+sn[3])^2) )
  b = 2*(1+sn[7])*exp(-((x^2)/32))
  c = (10-sn[7])*exp( -cos((sn[4]*x^2)/(15+sn[4])) -
((x+3+(sn[5]*pi))^2)/(2*(5+sn[6])^2))
  fx = a+b+c

  -log(fx)
}
```

Tuning parameters

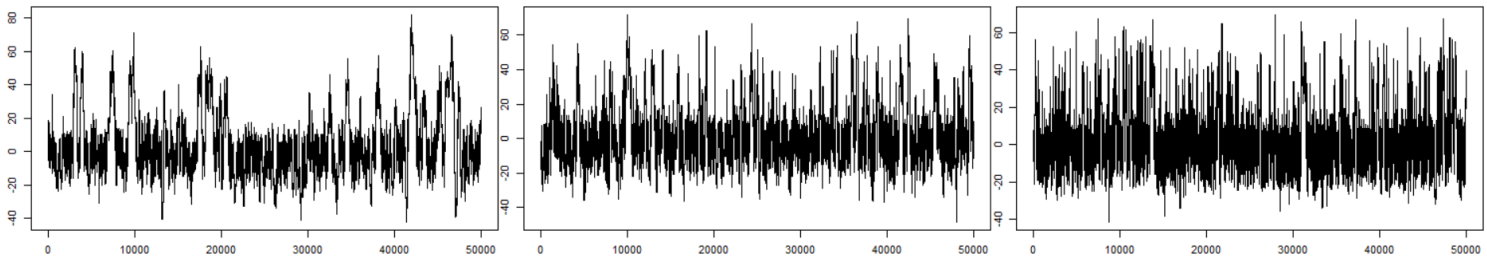
HMC requires tuning of the stepsize ε and the number of leapfrog steps, L, which together determine the length of the trajectory. Neal (2011) remarks that this is often an impediment to implementing HMC, and that it is generally harder than tuning a simple Metropolis algorithm. It requires preliminary runs with trial values for both ε and L. Neal (2011) illustrates examples with various values used, and based on this and some preliminary trials, the set of ε and L considered were respectfully;

$\varepsilon \in \{0.2, 0.3, 0.5\}$ and $L = \{15, 25, 40\}$

To get an idea of the best configurations, the HMC algorithm was run for all combinations of ε and L for 50,000 iterations of the chain. Their respective trace plots, diagnostics and ergodic averages were examined and compared. The results will first be presented and subsequently discussed.

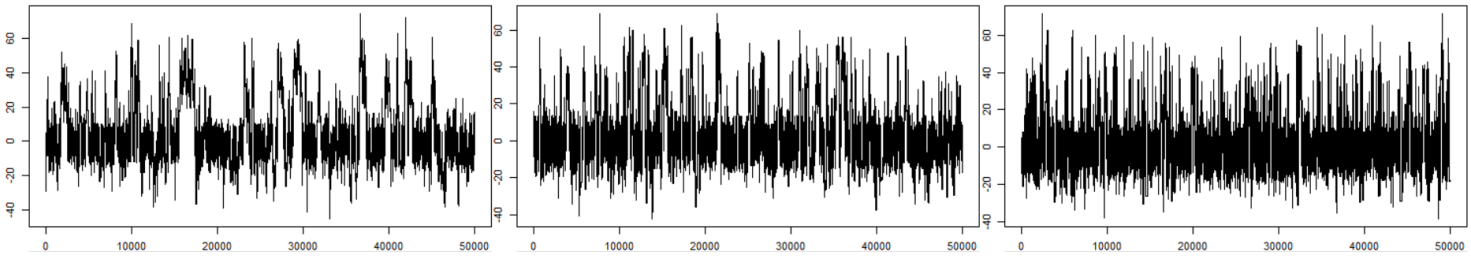
Sample Paths of the HMC Algorithm (N = 50,000)

ε = 0.2



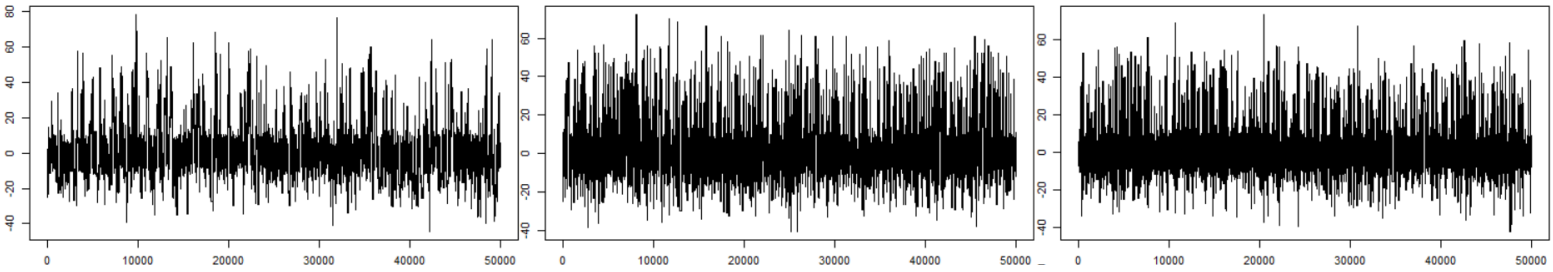
L = 15 *L = 25* *L = 40*

ε = 0.3



L = 15 *L = 25* *L = 40*

ε = 0.5



L = 15 *L = 25* *L = 40*

Figure 5.1. Sample paths of X_t (black) and Ergodic averages (blue) resulting from the Hamiltonian MC algorithm run for 50,000 iterations for $\epsilon \in \{0.2, 0.3, 0.5\}, L = \{15, 25, 40\}$

Diagnostics

Autocorrelation

L	**ε**		
	0.2	**0.3**	**0.5**
15	0.994	0.988	0.971
25	0.983	0.975	0.928
40	0.964	0.944	0.922

Table 5.1 First autocorrelation of the chain for a given ε and *L*

Acceptance Rate

L	**ε**		
	0.2	**0.3**	**0.5**
15	0.942	0.791	0.579
25	0.922	0.765	0.521
40	0.916	0.767	0.488

Table 5.2 Acceptance Rate of the HMC algorithm for a given proposal scale ε and *L*

ASJD

L	**ε**		
	0.2	**0.3**	**0.5**
15	4.74	8.682	17.980
25	11.701	18.181	53.916
40	25.25	39.689	52.578

Table 5.3 ASJD of the chain for a given ε and *L*

Discussion – Tuning Parameters, Diagnostics, Assessment

Considering the sample paths of the HMC chains first (Figure 5.1), it would appear that the mixing is noticeably worse in comparison to that of the standard MC chains. However, this exemplifies a key property of HMC (Neal, 2011). That is, using Hamiltonian dynamics, distant proposals can be produced for the Metropolis algorithm eschewing the slow exploration of the state space by the chain that is inherent in the case of simple random-walk proposals given their diffusive behaviour. For all sample paths shown in Figure 5.1, the majority of the chains seem to spend their time around the locations of where the bulk of the function $f(x)$ lies, i.e in the range $\sim \{0, 25\}$. Considering all values of the tuning parameters ε and L, the configurations which seems to display the best mixing is those at a ε value of 0.5 and L value of $\{25, 40\}$. This makes intuitive sense, that the HMC algorithm would explore the sample space more at greater trajectory lengths. The enhanced exploration of the chain at greater values of ε and L also aligns with the autocorrelation results given in Table 5.1. Although the autocorrelation is high overall, the best autocorrelation scores of 0.928 and 0.922 occur at these configurations of ε equal to 0.5 and L value of $\{25, 40\}$. That is, for a chain that displays better mixing, the correlations between successive samples will have a lower.

Efficiency

The advantage gained in avoiding the diffusive behaviour of the random-walk proposals is further highlighted when the acceptance rates are considered (Table 5.2). They are extremely high, ranging from 0.942 ($\varepsilon = 0.15, \mathrm{L} = 15$) to 0.488 ($\varepsilon = 0.5, \mathrm{L} = 40$). In HMC, moves are proposed to distant states which have a high probability of acceptance, due to the energy conserving properties of Hamiltonian dynamics. Specifically, it is the gradients given by equations guide the transitions through regions of high probability and admit the efficient exploration of the entire target distribution.

Assessment of Implementation

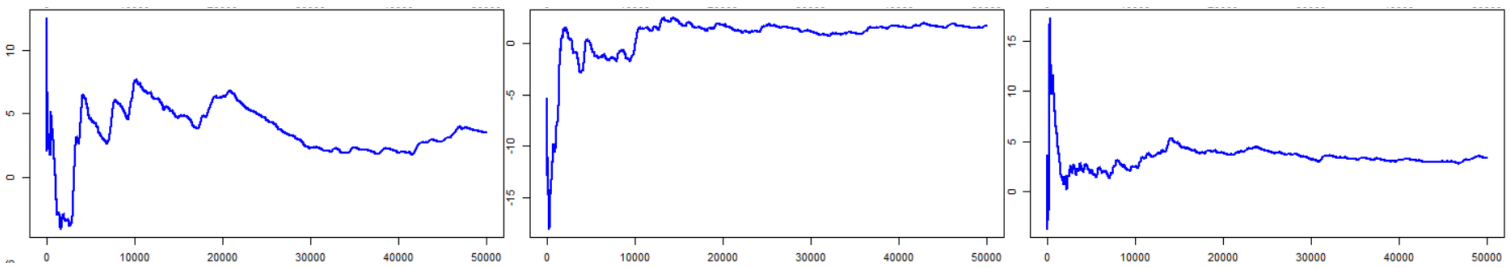
It now follows to determine whether the chain did infact converge to the invariant distribution f. The ergodic mean of the chains will be considered aswell as their overall distribution.

Ergodic Mean

The ergodic mean of the chain for 50,000 iterations is shown in Figure 5.2. In all cases, bar a few exceptions, the chain has broadly reached a steady state. For trajectory lengths L of 25, and 40 the chain has reached invariant distribution, with an approximate value ~ 2. The final value of the chain across the last 1000 and 10 samples are shown respectfully in table 4.4. In these cases, there is only a difference of 0.02 between the average of the last 1000 samples and that of the last 10 samples, which suggests that the chain has reached it's invariant distribution f. In contrast to this, there are still significant fluctuations in the chain at a trajectory length L equal to 15 (Figure

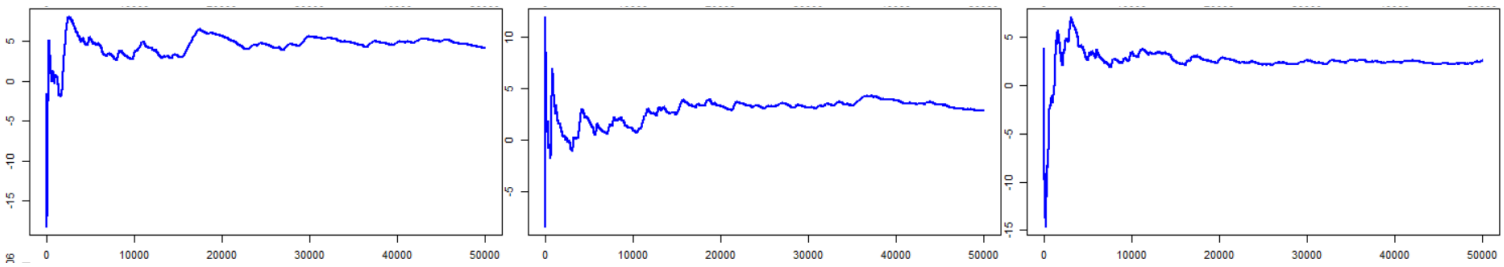
5.4, column 1). It appears that the chain needs to run longer at these parameterisations in order to settled down to equilibrium.

ε = 0.2



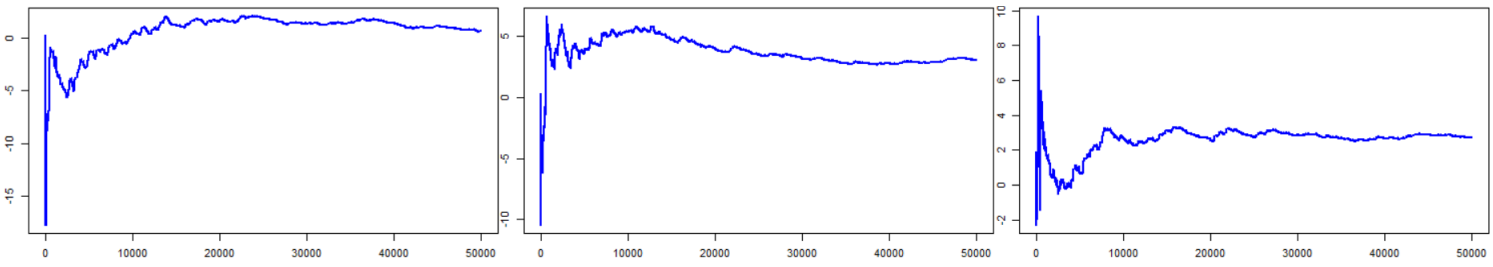
L = 15 *L = 25* *L = 40*

ε = 0.3



L = 15 *L = 25* *L = 40*

ε = 0.5



L = 15 *L = 25* *L = 40*

Figure 5.2. Ergodic averages of X_t resulting from the Hamiltonian MC algorithm run for 50,000 iterations for $\epsilon \in \{0.2, 0.3, 0.5\}, L = \{15, 25, 40\}$

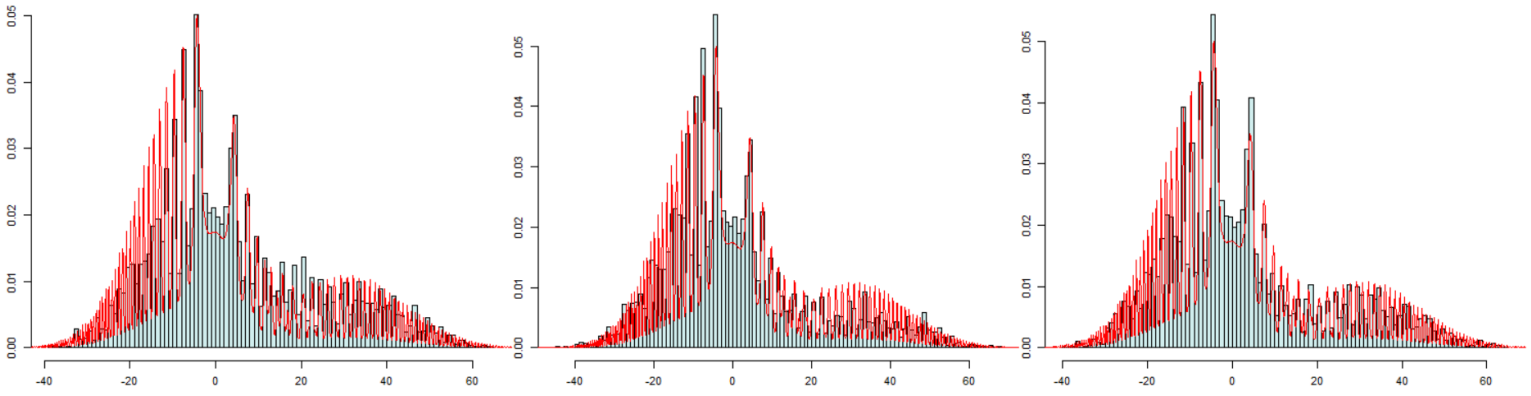
Table 5.4. The average of the chain across the last 1000 & last 10 samples for HMC run for 50,000 iterations

L		**ε**		
	Mean value	**0.2**	**0.3**	**0.5**
15	Final 1k	3.57	4.20	0.73
	Final 10	3.554	4.09	0.69
25	Final 1k	1.55	2.85	3.08
	Final 10	1.65	2.87	3.02
40	Final 1k	3.42	2.48	2.73
	Final 10	3.36	2.57	2.71

Comparison with the Objective - Empirical Density

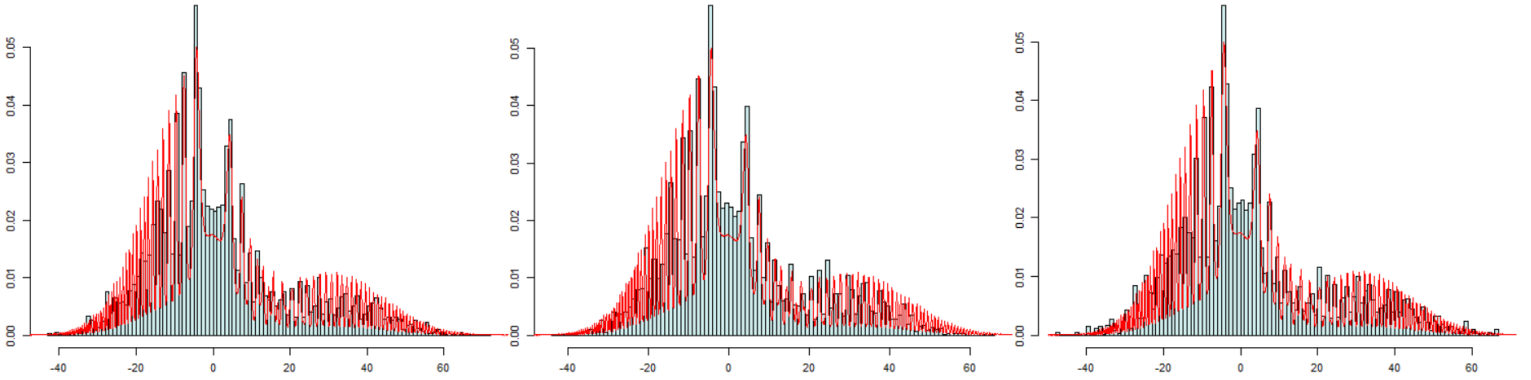
To compare with the objective, that is the $f(x)$, the empirical histogram of the samples resulting from the HMC algorithm for all values of ε and L were rendered (Figure 5.3). Qualitatively assessing the output, the empirical histograms appear to be samples from the target $f(x)$, that is the HMC algorithm was able to produce the correct samples, with $f(x)$ being it's invariant distribution.

ε = 0.2



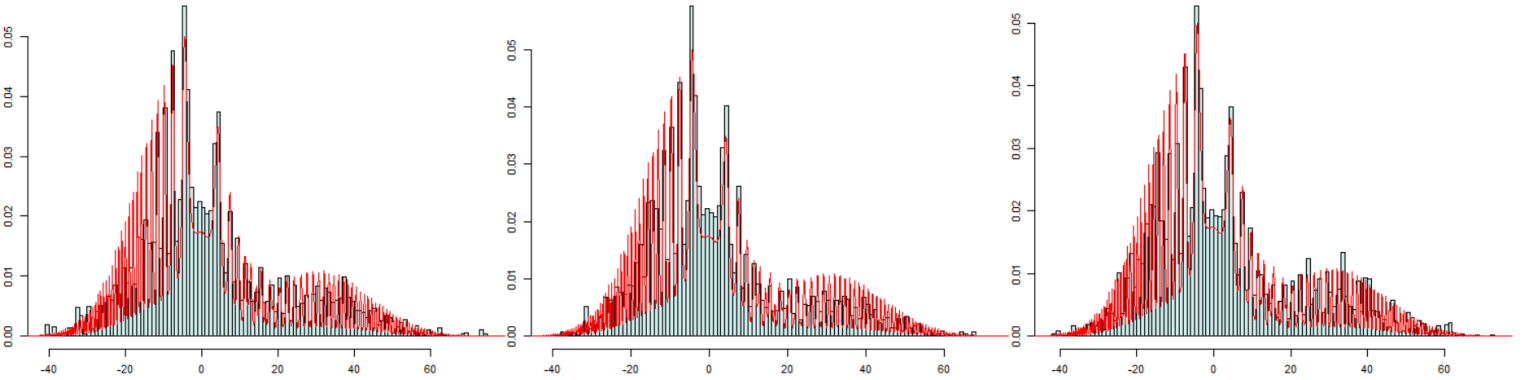
L = 15 *L = 25* *L = 40*

ε = 0.3



L = 15 *L = 25* *L = 40*

ε = 0.5



L = 15 *L = 25* *L = 40*

Figure 5.3. Empirical Hisograms (blue) vs True Density (red) for HMC

Part c: Modified Alpha – Barker Probability

i. *A valid modification*

The modification is valid, this form of the acceptance probability in a Metropolis Markov chain is known as the Barker Probability (Barker, 1965).

If we consider the two forms

$$\alpha_{M-H}(X|X^{t-1}) := \min\ \{1, \quad \frac{f(X)\cdot q(X^{(t-1)}|X)}{f(X^{(t-1)})\cdot q(X|X^{(t-1)})}\ \}$$

And

$$\alpha_{G-M} = \ \alpha_{Barker} = \frac{f(X)\cdot q(X^{(t-1)}|X)}{f(X)\cdot q(X^{(t-1)}|X) + f(X^{(t-1)})\cdot q(X|X^{(t-1)})}$$

In the case of α_{Barker}, we can see that this value will never be greater than 1, and therefore it is a valid probability. However given it's form, it is inherent that less proposals will be accepted than in the case of MH, i.e there will be less cases where the computed α will be greater than $u \sim U(0,1)$. This reduced efficiency is an inherent and known trait of the Barker probability, however it is still useful in certain specific cases. For example, in cases where it is not feasible to determine $f(x)$, and therefore not feasible to calculate α directly, (neither α_{M-H} nor α_{Barker}), an estimate of α would therefore be required. Comparing the form of α_{M-H} with α_{Barker}, it would prove easier to determine the estimate of α_{Barker} than it would for α_{M-H}, as there is no requirement to determine the minimum, like in the case of α_{M-H}. Goncalves et al (2017) also make use of α_{Barker} in their MCMC scheme for sampling from intractable target densities.

ii. *Integration of modification with other Methodologies*

For the methodologies implemented in part b, the modification is valid for all algorithms that contain a Metropolis step, namely the RW Metropolis, Simulated Annealing, the Metropolised Slice Sampler and perhaps in the Metropolis step of the Hamiltonian MC.

RWM & Modification

The RWM will firstly be considered given that both the Simulated Annealing algorithm and the Metropolised Slice Sampler contain the a symmetric random walk proposal as an inherent step. The RWM algorithm was implemented with this modified acceptance probability and the function *rwm_alpha_modified* was written in R to implement this as below;

RWM Modified Algorithm

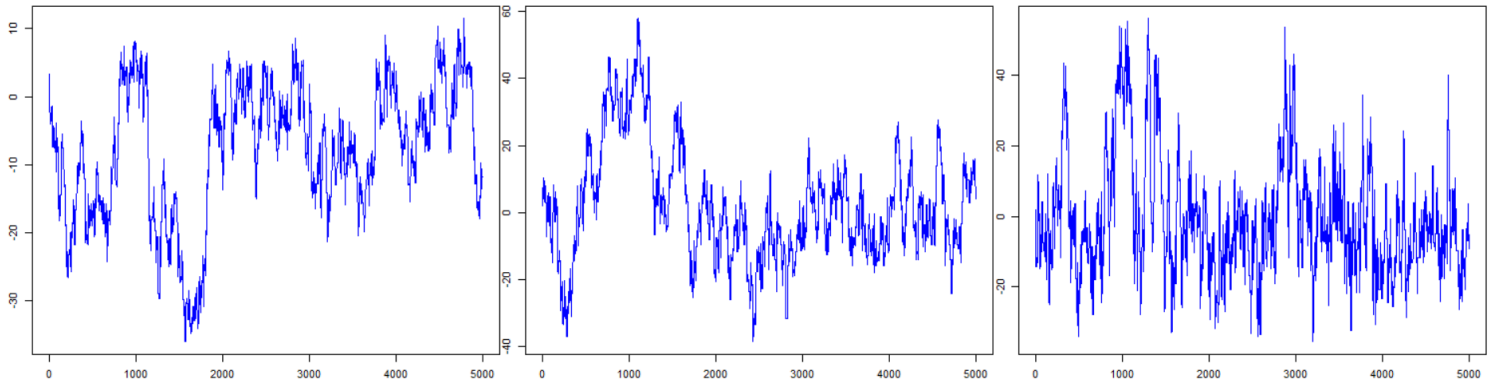
```
rwm_alpha_new = function(N, sigma, x0 = 2){

  #Vectors to store samples
  X = vector('numeric', N)
  X[1] = x0 #Initialise 1st sample in chain in order to start the chain
running
  count_accept = N #Count of number of accepted values
  #Loop
  for (t in 2:N){
    X[t] = X[t-1] + rnorm(1, mean = 0, sd= sigma) #Symmetric random walk
proposal
    u = runif(1)
    #Modified alpha
    alpha_new = get_fx(X[t])/(get_fx(X[t]) +get_fx(X[t-1]))

    if(u > alpha_new = get_fx(X[t])/(get_fx(X[t]) +get_fx(X[t-1]))) {
#Criterion for rejection
      X[t] = X[t-1] #If criterion for acceptance is not met the next sample
in chain is set to current sample
      count_accept = count_accept - 1
    }
  }
Print(count_accept/N)
  X #Return samples
}
```

Comparison of Outputs

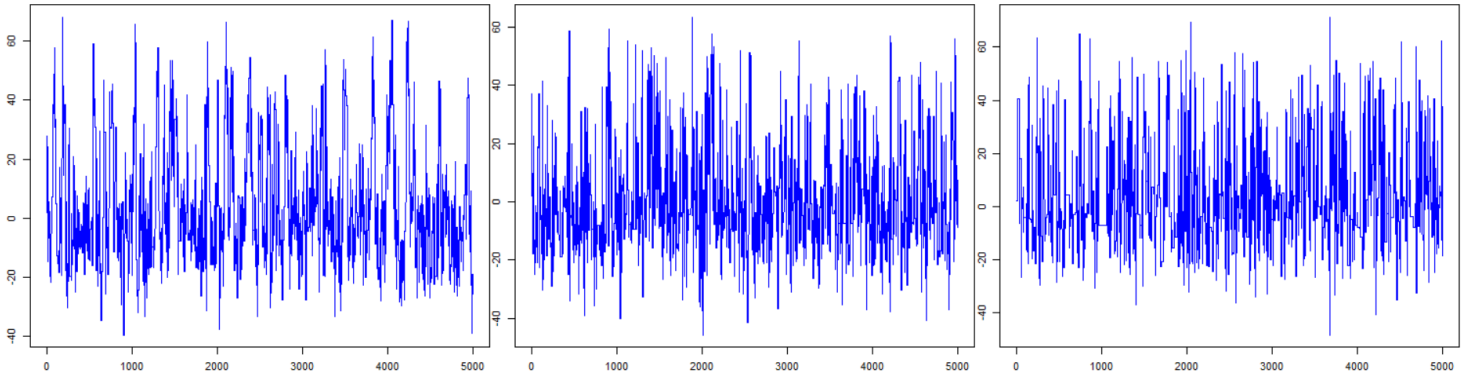
To compare the original RW algorithm with that of the modified version and determine whether the modification helps, their respective trace plots, diagnostics and ergodic averages were examined and compared. The results will first be presented and subsequently discussed.

Sample Paths of the HMC Algorithm (N = 5000)

$\sigma = 1.6$ $\sigma = 3.2$ $\sigma = 6.4$



$\sigma = 12.8$ $\sigma = 25.6$ $\sigma = 51.2$



$\sigma = 102.4$ $\sigma = 153.6$

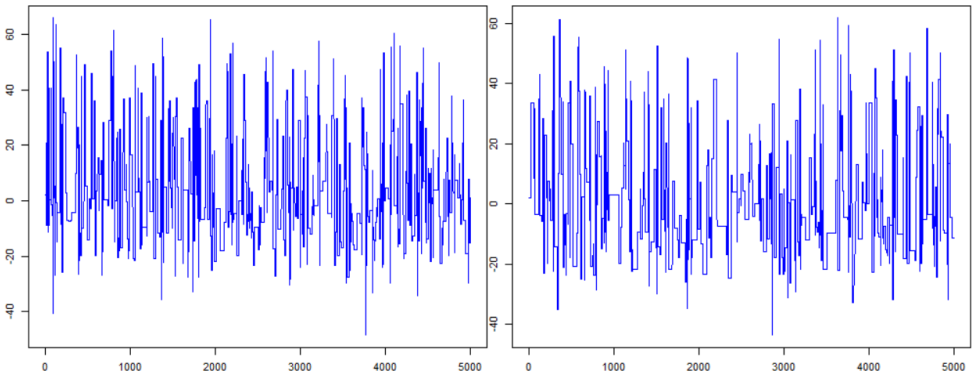


Figure 6.1 *Sample paths of X_t resulting from the modified RWM algorithm run for 5,000 iterations for* $\sigma \in \{1.6, 3.2, 6.4, 12.8\ 25.6, 51.2, 102.4, 153.6\}$

Diagnostics

Autocorrelation

σ	1.6	3.2	6.4	12.8	25.6	51.2	102.4	153.6
RWM original Autocorrelation (first)	0.997	0.991	0.966	0.901	0.776	0.713	0.797	0.853
RWM modified Autocorrelation (first)	0.999	0.994	0.977	0.931	0.842	0.794	0.853	0.894

Table 6.1 First autocorrelation of the chain for a given proposal scale σ

Acceptance Rate

σ	1.6	3.2	6.4	12.8	25.6	51.2	102.4	153.6	Mean
RWM original Acceptance Rate	0.713	0.686	0.653	0.578	0.447	0.296	0.165	0.110	**0.71**
RWM modified Acceptance Rate	0.433	0.427	0.412	0.373	0.299	0.200	0.113	0.766	**0.43**

Table 6.2 Acceptance Rate of the chain for a given proposal scale σ

Average Squared Jumping Distance

σ	1.6	3.2	6.4	12.8	25.6	51.2	102.4	153.6
RWM original ASJD	1.75	6.69	23.49	69.11	155.62	193.30	139.09	102.68
RWM modified ASJD	1.08	4.24	15.59	48.03	110.08	141.75	103.52	73.56

Table 6.3 The Average Squared Jumping Distance for a given proposal scale σ

Discussion – Comparison of Results – Does the modification help?

The difference in the acceptance rate is the most prominent difference that emerges in comparing the results of the RWM with that of the modified version using the Barker probability. Across all values of the proposal scale, the acceptance rate is 28% lower when α_{Barker}, a significant difference. A lower acceptance rate is expected and the Barker's probability is well-known to be slightly less efficient as described above. It

follows from the lower acceptance rate of the algorithm that the chain's mixing ability is also worse. As seen in Figure 6.1, for $\sigma \geq 51.2$, many samples are rejected and as a result the chain remains at a given state for a large number of iterations, and therefore the mixing is suboptimal. This starts to occur at a σ value of 51.2 in the case of the modified RWM but only becomes apparent at a σ value of 102.4 in the case of the original RWM. A further effect of the lower acceptance rate is that the autocorrelation is approximately 5% worse in the modified algorithm. Again this is because the chain remains at the same location for successive iterations more often when α_{Barker} than when α_{M-H} is implemented and thus the auto-correlation is higher.

Assesment & Comparison of the Output

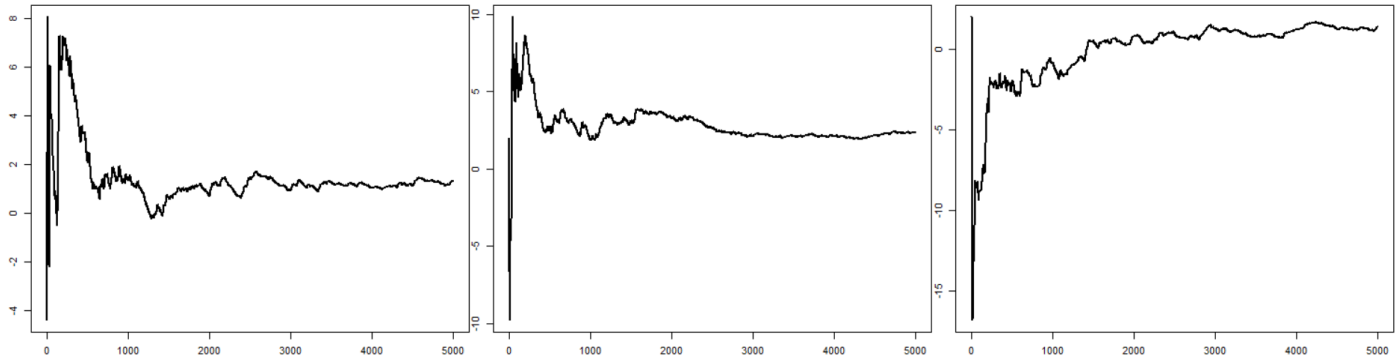
Ergodic Mean

The respective ergodic mean of the chains found using the two algorithms were compared to determine if one algorithm was generally better at converging to the invariant distribution $f(x)$ than the other. It was also necessary to determine whether the modified algorithm did infact converge to the correct value. The chain was run for only 5,000 iterations as in the limit it was expected that both chains would settle down to an equilibrium. Furthermore only the three best sigma values, in terms of their apparent mixing ability and autocorrelation, were considered for comparison namely;

$$\sigma \, \epsilon \, \{ 25.6, 51.2, 102.4\}$$

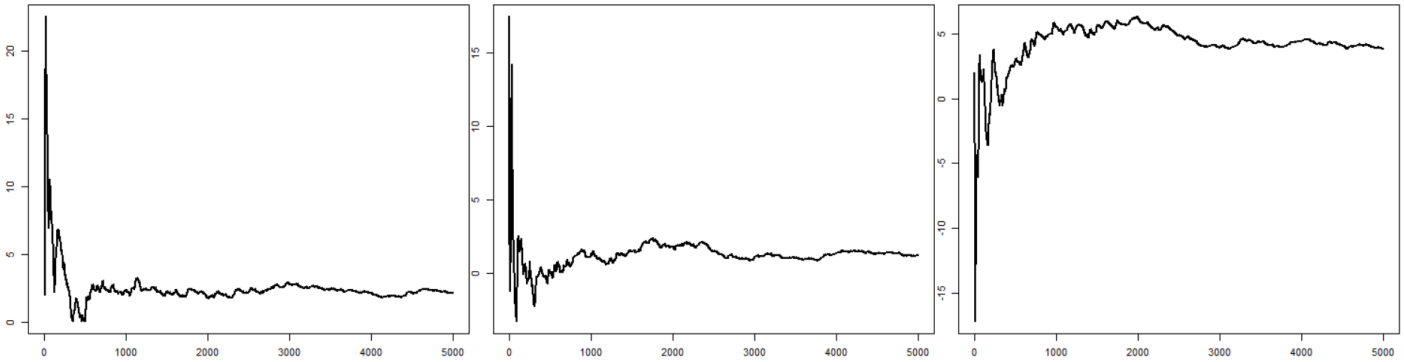
As shown in Figure 6.2 there is not a marked difference in the convergence of both chains. Both algorithms have more or less settled down to the correct value.

Original RWM Algorithm



$\sigma = 25.6$ $\sigma = 51.2$ $\sigma = 102.4$

Modified RWM Algorithm



$\sigma = 25.6$ $\sigma = 51.2$ $\sigma = 102.4$

Figure 6.2 Ergodic mean of the chains from both the original RWM algorithm and the modified version the random walk Metropolis algorithm run for 250,000 iterations for $\sigma \in \{25.6, 51.2, 102.4\}$

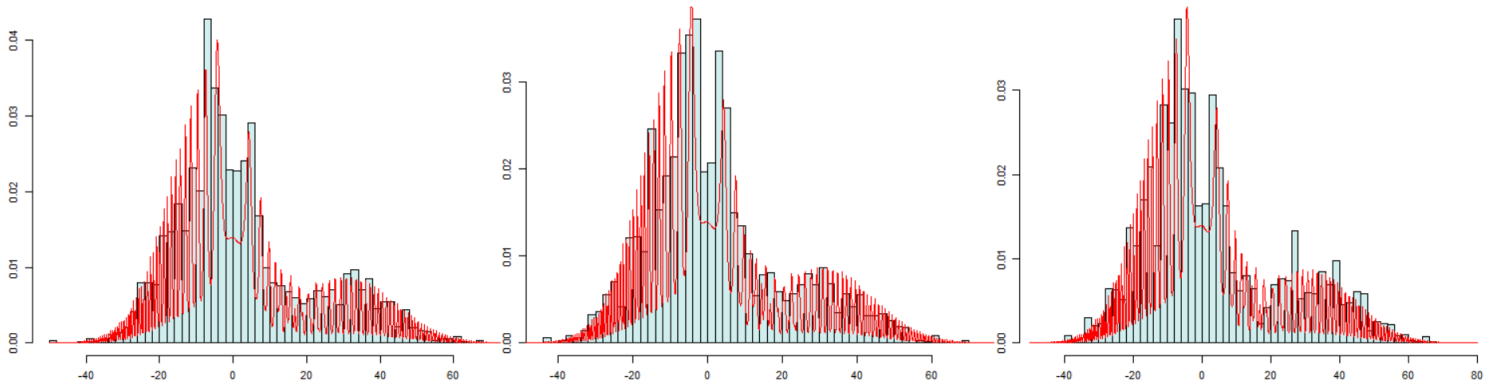
	Original RWM Algorithm			**Modified** RWM Algorithm		
σ (Proposal Scale)	**25.6**	**51.2**	**102.4**	**25.6**	**51.2**	**102.4**
Ergodic Mean (final 1000 samples)	1.203	2.174	1.359	2.134	1.364	4.191
Ergodic Mean (final 10 samples)	1.321	2.378	1.365	2.177	1.227	3.841

Table 6.4. The average of the chain across the last 1000 & last 10 samples for both the original RWM algorithm and the modified version. (5000 iterations of the chain)

Empirical Density/Histogram

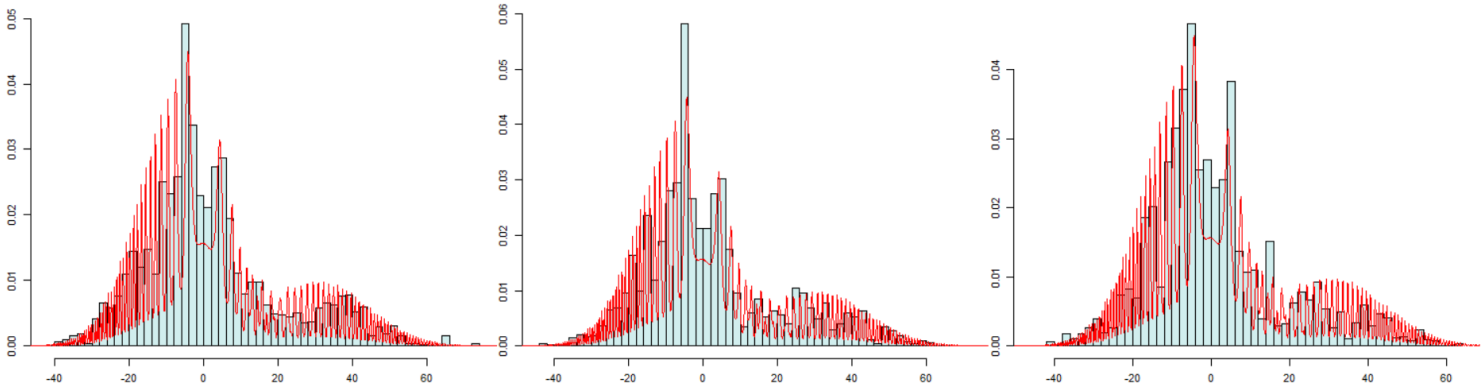
To determine whether the algorithm the did in fact produce the correct output, that is samples from f, the empirical histogram of the samples resulting from the modified RWM were rendered as shown in Figure 6.3 for $n = 5000$ iterations of the chain. The empirical histograms do appear to be samples from the target $f(x)$, that is the modified RWM algorithm was able to produce the correct samples, with $f(x)$ being it's invariant distribution. In comparing the original with the modified, there doesn't seem to be any marked difference in the empirical histograms.

Original RWM Algorithm



$\sigma = 25.6$ $\sigma = 51.2$ $\sigma = 102.4$

Modified RWM Algorithm



$\sigma = 25.6$ $\sigma = 51.2$ $\sigma = 102.4$

Figure 6.3 Empirical histogram of the chains for RWM and Metropolis modified with Barker probability

Simulated Annealing & Modification

The modification is also applicable in the case of Simulated Annealing. The function **sim_anneal_a_new** was written to implement this modification;

```
sim_anneal_a_new = function(N, x0 = 0, beta1 = 1, alpha = 1.001, sigma =
1){
  #Containers
  x = vector('numeric', N)
  fn = numeric(N)
  count_na = 0
  count_accept = 0

  #Variables current
  x_current = x0
  fx_current = get_fx(x_current)
  beta_current = beta1

  for (i in 1: N){
    x_new = x_current + rnorm(1)*sigma
    fx_new = get_fx(x_new)

    #Modifed acceptance probability
    alpha_accept_new = (fx_new^beta_current)/(fx_new^beta_current +
fx_current^beta_current)

    if (alpha_accept_new > runif(1)){
      x_current <- x_new
      fx_current <- fx_new
      count_accept = count_accept + 1
    }
    x[i] = x_current
    fn[i] = fx_current
    beta_current = beta_current*alpha


  }

  list(x, fn)
}
```

Does the Modification help?

To compare the original Simulated Annealing algorithm with that of the modified version and determine whether the modification helps, their diagnostics were determined and compared.

Autocorrelation

Although the values are quite high, the parameterisation of alpha that displayed the best mixing properties i.e $\alpha = 1.1$, also has the lowest autocorrelation

	$\sigma = 1$				
α	1.001	1.010	1.10	2.00	10.00
Original S.A Autocorrelation (first)	0.994	0.969	0.923	0.950	0.951
Modified S.A Autocorrelation (first)	0.973	0.969	0.954	0.972	0.953

Table 6.5

Acceptance Rate

	$\sigma = 1$				
α	1.001	1.010	1.10	2.00	10.00
Original S.A *Acceptance Rate*	0.152	0.173	0.174	0.165	0.179
Modified S.A *Acceptance Rate*	0.011	0.0152	0.0095	0.0103	0.009

Table 6.6

Average Squared Jumping Distance (ASJD)

	$\sigma = 1$				
α	1.001	1.010	1.10	2.00	10.00
Original S.A *ASJD*	0.121	0.228	0.194	0.226	0.221
Modified S.A *ASJD*	0.143	0.152	0.132	0.1737	0.1737

Table 6.7

Comparison of Results

The most prominent difference emerges when one compares the acceptance rate of the the original form of the Simulated Algorithm with that of the modified. The acceptance rate of the modified form is extremely low at just ~ 1%. Again this a result of the inherent inefficiency that comes with using the Barker Probability. In contrasting the respective autocorrleations and ASJD there is not a major difference in the results. The modified algorithm is slightly worse due to the fact that it rejects more proposals and so gets stuck at the same location for consecutive more often than in the case of the original algorithm

Modification for Other Algorithms

In considering the other algorithms, firstly, given that the Metropolised Slice Sampler contains a Metropolis step, I would suspect it would be possible to implement the Barker probability in that setting. While in the case of HMC, since the algorithm alternates the simple updates for the momenta with Metropolis updates, it may also be the case that the modification is valid there. However due to time constraints, it was not possible to investigate the correctness of the modification for these two algorithms. This could perhaps be investigated in a follow-on study.

Part d – Choosing a Single Methodology

To choose a single methodology to sample from the target distribution f, the results obtained in part b and c will be considered. Given that the modified versions of the algorithms using the Barker probability were less efficient than the original, they will not be considered. The algorithms for consideration in choosing one single methodology will be the RWM algorithm, the Slice Sampler (Metropolised) version, and the HMC algorithm. The algorithms will be compared across their operating speed, diagnostics such as their autocorrelation and acceptance rate and ability to converge to the equilibrium distribution. To simplify matters the algorithms will be compared using their respective 'optimal' configurations as determined in part b. For both the RWM and Slice Sampling algorithm a proposal scale σ of 51.2 was deemed an optimal proposal scale while for HMC the configuration of $\varepsilon = 0.5$, $L = 25$ was deemed a good option. In all cases, the chain at these configurations had the lowest or second to lowest autocorrelation, exhibited good mixing properties and it converged to a steady equilibrium distribution.

Operating Speed of Algorithms

The operating speed of each algorithm is firstly compared. In terms of practical implementation this is a very important feature, especially if the algorithm is required to be implemented on a larger scale. The code shown below was run in The execution time or operating speed was determined in R using the code shown below. Note this code determines the time taken to run the RWM algorithm, the same method applies for the other two algorithms by simply replacing the second line code with the given function call.

```
start_time = Sys.time()

X_rw = rwm(N, sigma = 51.2, x0 = 2) #Implement given algorithm

end_time = Sys.time()

time_elapsed = end_time - start_time
```

The results are given in table 7.1. Note that these are average results across three trials. Based on these results, it can be said that the execution speed of the HMC algorithm is half that of the RWM and Slice Sampler. Furthermore this is only a 1-dimension problem, and HMC is deemed superior option in cases of higher dimensions (Neal, 2011) so you would expect it to be even faster in those cases.

Algorithm	**RWM** $\boldsymbol{\sigma = 51.2}$	**Slice Sampler** $\boldsymbol{\sigma = 51.2}$	**HMC** ($\boldsymbol{\varepsilon}$ = 0.5, $\boldsymbol{L}$ = 25).
Execution time (N = 5000)	0.108 s	0.110 s	0.052 s

Table 7.1 Execution speed of the algorithms

Diagnostics – Autocorrelation

In terms of the autocorrelation the RWM does best as shown in Table 7.2. The autocorrelation of the HMC algorithm is high, however this is most likely due to it's efficient exploration of the state space in comparison to the diffusive behaviour of a random walk proposal.

Algorithm	**RWM** $\boldsymbol{\sigma = 51.2}$	**Slice Sampler** $\boldsymbol{\sigma = 51.2}$	**HMC** ($\boldsymbol{\varepsilon}$ = 0.5, $\boldsymbol{L}$ = 25).
First Autocorrelation (N = 50,000)	0.713	0.791	0.928

Table 7.2 (First) Autocorrelation of the algorithms

Diagnostics – Acceptance Rate

There is a marked difference between the acceptance rate of the Markov chain algorithms which employ a symmetric random walk as proposal (I.e RWM and SS) and that of HMC. The efficiency of the HMC is a major advantage of the algorithm. This exemplifies a key property of HMC as described in b(iv) when implementing the HMC algorithm. That is, using Hamiltonian dynamics, distant proposals can be produced for the Metropolis algorithm eschewing the slow exploration of the state space by the chain that is inherent in the case of simple random-walk proposals given their diffusive behaviour. (Neal, 2011)

Algorithm	**RWM** $\boldsymbol{\sigma = 51.2}$	**Slice Sampler** $\boldsymbol{\sigma = 51.2}$	**HMC** ($\boldsymbol{\varepsilon}$ = 0.5, $\boldsymbol{L}$ = 25).
Acceptance Rate (N = 50,000)	0.292	0.289	0.521

Table 7.3 Execution speed of the algorithms

Comparison of Output

The ergodic averages shown in Figure 7.1. All the algorithms have converged to a steady equilibrium that is the invariant distribution f. While the empirical Histograms are plotted against the true density in Figure 7.2. In all cases, the empirical histograms appear to be samples from the target $f(x)$, that is the algorithms were able to produce the correct samples, with $f(x)$ being it's invariant distribution. Qualitatively speaking, they all appear to be a good fit.

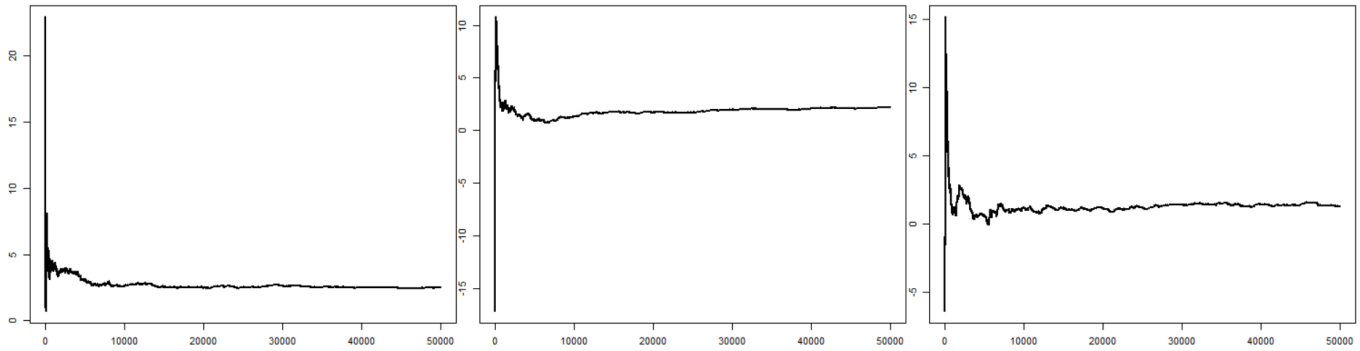


Figure 7.1. Ergodic averages for the RWM, the Slice Sampler and HMC Algorithm

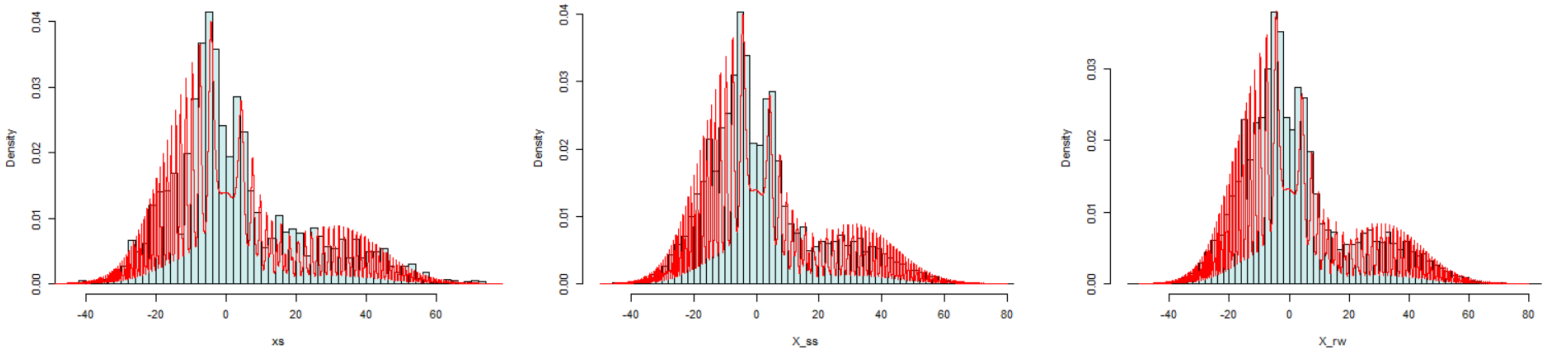


Figure 7.2 Empirical Histograms (blue) vs True Density (red) for the RWM, the Slice Sampler and HMC Algorithm

Conclude

To conclude, in choosing a single methodology to sample from the target distribution f, based on the overall results described above, the Hamiltonian MC algorithm would be chosen due to its efficiency. It had greater efficiency on two levels, firstly in terms of the operating speed of the algorithm, which was half that of the other two algorithms. Secondly it is much more efficient at generating samples in terms of its acceptance rate, which is 20% greater than that of the other two algorithms. Another major draw of this algorithm in general, is in it's application to distributions of higher dimensions for which it works particularly well (Neal, 2011). However one major draw-back of the algorithm is that it is limited to cases where the gradient exists and can be computed. However with this f as the invariant distribtion this was possible and overall, based on the results, it was deemed the best option of those considered.

References

Barker, A. A. (1965). Monte Carlo calculations of the radial distribution functions for a protonelectron plasma. *Australian Journal of Physics*, 18:119–133.

Betancourt, Michael, and Mark Girolami. 2013. “Hamiltonian Monte Carlo for Hierarchical Models.” *arXiv* 1312.0906.

Goncalves F.B, Latuszynski K., Roberts G.O, (2017), Barker's algorithm for Bayesian inference with intractable likelihoods. *Brazilian Journal of Probability and Statistics*

Johansen, A.M. (2018), ‘Monte Carlo Methods’ *Warwick Statistics*.

Livingstone, S. (2015) *Some contributions to the theory and methodology of Markov chain Monte Carlo*. Ph.D. thesis, University College London.

Neal, R. (2011) MCMC using Hamiltonian dynamics. *In Handbook of Markov Chain Monte Carlo* (eds. S. Brooks, A. Gelman, G. L. Jones and X.-L. Meng), 113–162. CRC Press.

Neal, Radford M. (2003). "Slice Sampling". *Annals of Statistics*. **31** (3): 705–767

Appendix

The 'pure' form of the Slice Sampling method as described by Neal (2003) was also investigated as an additional method for (b.iv). The function `slice_sampler2` was written and implemented in R. However due to time constraints, it was not feasible to do a full analysis of the results using this method.

```
slice_sampler2 = function(Ns, sigma, w = 0.5 x0 = 2){

  #Initialise variables
  xs = vector('numeric', Ns)
  x_current = x0

  #Iterations
  for (i in 1: Ns) {
    y = runif(1, 0, get_fx(x_current)) #uniform sample
    lb = x_current #Left bound
    rb = x_current #Right bound

    #interval w is randomly positioned around x0
    #Expanded in steps of size w until both ends are outside the slice
    while (y < get_fx(lb)) {
      lb = lb - w
    }
    while (y < get_fx(rb)) {
      rb = rb + w
    }

    #x _new - uniformly pick point from interval until a point inside the
slice is found
    x_new = runif(1, lb, rb)
    #Points picked that are outside the slice are used to shrink the
interval.
    if (y > get_fx(x_new)) {
      if (abs(x_new - lb) < abs(x_new - rb)) {
        lb = x_new
      } else {
        lb = y
      }
    }
    else {
      x_current = x_new #Set new value of x to current value

    }
    xs[i] = x_current    #Include current x as a sample
  }

  xs
}
```

The output for the following configuration is shown in Figure I. Qualitatively speaking, it is a very good match to the objective *f*

Configuration;

```
w = 0.5 #Initial guess for interval width
slice_sampler2(Ns = 5000, w, x0 = 2)
```

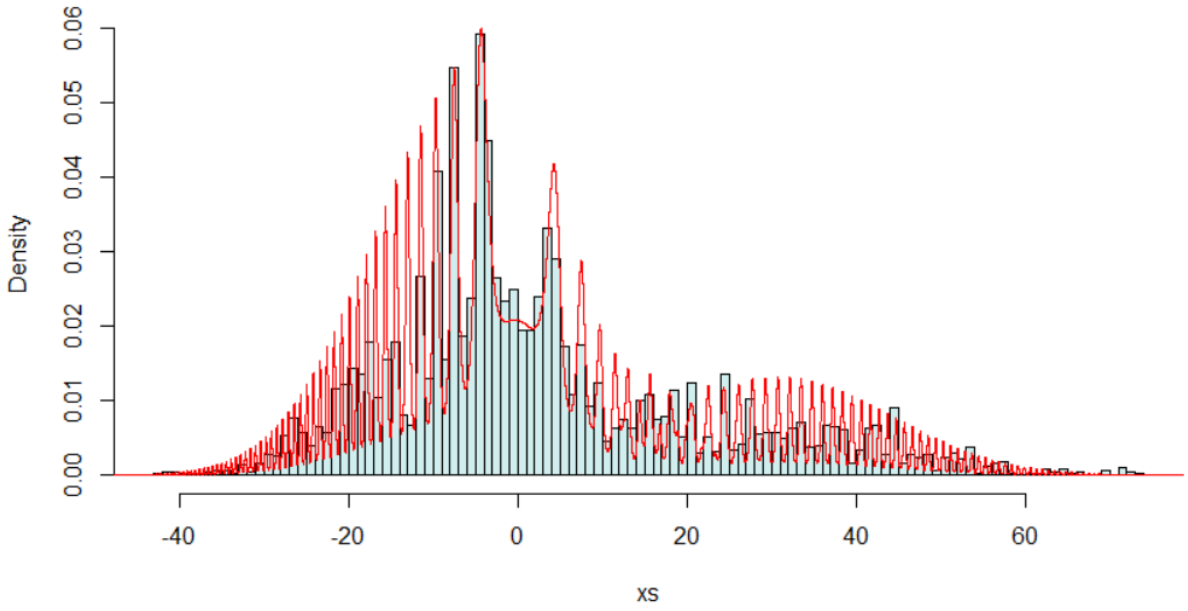


Figure I. True density (red) vs the empirical density found using the 'pure' Slice Sampling algorithm.