

Local scale adaptation for Random Walk Metropolis

Supervisor: Prof. Jeffrey Rosenthal

Student: Xin Wang

Summer 2011

Contents

Abstract

1. Introduction

- 1.1 introduction
- 1.2 Metropolis-Hastings algorithm
- 1.3 optimal scaling
- 1.4 optimal acceptance rate
- 1.5 adaptive MCMC
- 1.6 motivation

2. Test of local acceptance rate $\alpha(X)$

- 2.1 idealist adaptation algorithm
- 2.2 methods to test $\alpha(X)$
- 2.3 Example 1: normal distribution in \mathcal{R}^1
 - 2.3.1 Output
 - 2.3.2 How $\alpha(x)$ changes with parameters
 - 2.3.3 How to improve the algorithm speed
- 2.4 Example 2: mixture of two normal distributions in \mathcal{R}^1
 - 2.4.1 How to avoid numerical error
 - 2.4.2 How to improve the algorithm speed
 - 2.4.3 Output
 - 2.4.4 How to make $\alpha(5)$ smaller
- 2.5 Example 3: mixture of three normal distributions in \mathcal{R}^1
 - 2.5.1 How to avoid numerical error
 - 2.5.2 Output
 - 2.5.3 How to choose γ

2.6 Example 4: normal distribution in \mathcal{R}^2

2.6.1 How to learn from Example 1

2.6.2 How to improve the algorithm speed

2.6.3 Output

3. Efficiency comparison of adaptive and non-adaptive algorithm

3.1 efficiency measures to evaluate Markov Chain

3.1.1 varfact

3.1.2 variance/ standard error

3.1.3 average squared jump distance

3.2 Example 1: normal distribution in \mathcal{R}^1

3.2.1 varfact

3.2.2 variance/ standard error

3.2.3 average squared jump distance

3.3 Example 2: mixture of two normal distributions in \mathcal{R}^1

3.3.1 varfact

3.3.2 variance/ standard error

3.3.3 average squared jump distance

3.4 Example 3: mixture of three normal distributions in \mathcal{R}^1

3.4.1 varfact

3.4.2 variance/ standard error

3.4.3 average squared jump distance

3.5 Example 4: normal distribution in \mathcal{R}^2

3.5.1 varfact

3.5.2 variance/ standard error

3.5.3 average squared jump distance

Conclusion

Reference

Appendix

1. Code

1.1 C code

1.1.1 C code to implement the adaptation algorithm and generate a Markov Chain (Example 1~3)

1.1.2 C code to implement the adaptation algorithm and generate a Markov Chain (Example 4)

1.2 R code

1.2.1 R code to plot the final $g(X)$ and $\sigma(X)$ (Example 1~3)

1.2.2 R code to plot the final $g(X)$ and $\sigma(X)$ (Example 4)

1.2.3 R code to generate Markov Chain without adaption (Example 1)

1.2.4 R code to generate Markov Chain without adaption (Example 2)

1.2.5 R code to generate Markov Chain without adaption (Example 3)

1.2.6 R code to generate Markov Chain without adaption (Example 4)

1.2.7 R code to compute different efficiency measures (Example 1~3)

1.2.8 R code to compute different efficiency measures (Example 4)

1.2.9 R code to implement OLS in approximation of the final $\sigma(X)$

2. Output

2.1 Example 1: normal distribution in R^1

2.1.1 kernel function $K(x) = e^{-\frac{|x|^{\alpha_1}}{\alpha_2}}$

2.1.2 kernel function $K(x) = \frac{1}{1 + \alpha_1 |x|^{\alpha_2}}$

2.1.3 kernel function $K(x) = \begin{cases} 0, & |x| \geq 2 * width \\ -1 * height, & width < |x| < 2 * width \\ 1 * height, & |x| \leq width \end{cases}$

2.1.4 kernel function $K(x) = \begin{cases} 0, & |x| \geq 2 * width \text{ or } |x| \leq width \\ 1, & width < |x| < 2 * width \end{cases}$

2.1.5 Constant $\sigma(X) = C$

2.1.6 varfact comparison

2.1.7 variance comparison

2.1.8 comparison of average squared jump distance

2.1.9 OLS in approximation of the final $\sigma(X)$

2.2 Example 2: mixture of two normal distributions in R^1

2.2.1 kernel function $K(x) = e^{-\frac{|x|^{\alpha_1}}{\alpha_2}}$

2.2.2 kernel function $K(x) = \frac{1}{1 + \alpha_1 |x|^{\alpha_2}}$

2.2.3 kernel function $K(x) = \begin{cases} 0, & |x| \geq 2 * width \\ -1 * height, & width < |x| < 2 * width \\ 1 * height, & |x| \leq width \end{cases}$

2.2.4 kernel function $K(x) = \begin{cases} 0, & |x| \geq 2 * width \text{ or } |x| \leq width \\ 1, & width < |x| < 2 * width \end{cases}$

2.2.5 Constant $\sigma(X) = C$

2.2.6 varfact comparison

2.2.7 variance comparison

2.2.8 comparison of average squared jump distance

2.3 Example 3: mixture of three normal distributions in R^1

2.3.1 kernel function $K(x) = e^{-\frac{|x|^{\alpha_1}}{\alpha_2}}$

2.3.2 kernel function $K(x) = \frac{1}{1 + \alpha_1 |x|^{\alpha_2}}$

2.3.3 kernel function $K(x) = \begin{cases} 0, & |x| \geq 2 * width \\ -1 * height, & width < |x| < 2 * width \\ 1 * height, & |x| \leq width \end{cases}$

2.3.4 kernel function $K(x) = \begin{cases} 0, & |x| \geq 2 * width \text{ or } |x| \leq width \\ 1, & width < |x| < 2 * width \end{cases}$

2.3.5 Constant $\sigma(X) = C$

2.3.6 varfact comparison

2.3.7 variance comparison

2.3.8 comparison of average squared jump distance

2.4 Example 4: normal distribution in \mathbb{R}^2

2.4.1 kernel function $K(x) = e^{-\frac{|x|^{\alpha_1}}{\alpha_2}}$

2.4.2 kernel function $K(x) = \frac{1}{1 + \alpha_1 |x|^{\alpha_2}}$

2.4.3 kernel function $K(x) = \begin{cases} 0, & |x| \geq 2 * width \\ -1 * height, & width < |x| < 2 * width \\ 1 * height, & |x| \leq width \end{cases}$

2.4.4 kernel function $K(x) = \begin{cases} 0, & |x| \geq 2 * width \text{ or } |x| \leq width \\ 1, & width < |x| < 2 * width \end{cases}$

2.4.5 Constant $\sigma(X) = C$

2.4.6 varfact comparison

2.4.7 variance comparison

2.4.8 comparison of average squared jump distance

Abstract

This report investigates the use of adaptation algorithm to update the parameters automatically during a Markov Chain. The target distribution we work on include normal distribution, the mixture of two or three normal distributions in one or two dimension space. Computer simulation shows that the adaptation algorithm makes a little improvement to the efficiency of Random Walk Metropolis than non-adaptation algorithm.

The report starts by an introduction to the proposed adaptation algorithm in chapter 1, followed by the test of local acceptance rate for some specific x in chapter 2. Then we will compare the efficiency of adaptation and non-adaptation algorithm in some different measures in Chapter 3.

1. Introduction

1.1 Introduction

Although MCMC algorithms such as the Metropolis-Hastings algorithm are widely used to sample from complicated target distribution, it has long been recognized that the choice of the proposal density $q(x, y)$ is crucial to the success of these algorithms. The most common case (which we will focus on here) involves a symmetric random-walk Metropolis algorithm (RMW), in which the proposal density is given by $Y_n = X_n + Z_n$, Where the increments Z_n are i.i.d from some fixed symmetric distribution (e.g. $N(0, \sigma^2 I_d)$). In this case, it is the crucial issue that how to choose the scale σ . If σ is too small, then the chain will move slowly; if it is too large, the proposal will always be rejected and then the chain always gets stuck. So we need a value of σ between the two extremes, thus achieving a reasonable-sized proposal moves together with a reasonable-high acceptance probability. Thus, the main issue is to find proper values for all the parameters in σ .

1.2 The Metropolis-Hastings Algorithm ^[1]

Suppose that our target distribution π has density with respect to some reference measure (usually d-dimensional Lebesgue measure). Then, given the current state X_n , a proposal value Y_n is generated from some pre-specified density (usually $Y_n \sim X_n + \sigma(X_n) * N(0, I_d)$) and accepted

with probability $\alpha(x, y) = \min \left\{ \frac{\pi(y)q(y, x)}{\pi(x)q(x, y)}, 1 \right\}$. If the proposed value is accepted, we set $X_{n+1} = Y_n$; otherwise we set $X_{n+1} = X_n$.

1.3 Optimal Scaling

The determining of proposal scale is both very important and very difficult. However, it is possible to use theory to estimate the optimal proposal scalings and/or adaptive algorithms to attempt to find good proposals automatically with little user intervention.

A simple way to avoid the extremes of the scale is to monitor the acceptance rate of the algorithm, that is, the fraction of the proposed moves which is accepted. If this fraction is very close to 1, this suggests very small σ and very small movements. If this fraction is very close to 0, this suggests large σ and the high probability of the chain getting stuck. But if the fraction is both far from 0 and from 1, then we have managed to avoid both extremes.

1.4 Optimal Acceptance Rate ^[1]

Roberts et al.(1997) has proved that under some restrictive assumptions, as $d \rightarrow \infty$, the optimal acceptance rate is 0.234. They considered RWM on R^d for very special target densities, of the form:

$$\pi(x_1, \dots, x_d) = \prod_{i=1}^d f(x_i)$$

Where f are some one-dimensional smooth density function. This means that we have to sample from i.i.d components.

This result are all asymptotic as $d \rightarrow \infty$. Numerical studies (Gelman et al.,1996; Roberts and Rosenthal,2001) indicate that the limiting results do seem to well approximate the finite-dimensional situation for d as small as 5. Also, numeric studies on normal distribution show that when $d = 1$, the optimal acceptance rate is approximately 0.44. I will try both these optimal acceptance rate in the following simulation for comparison.

1.5 Adaptive MCMC ^[2]

An alternative approach is adaptive MCMC, which "learns" better parameter choice "automatically".

Suppose $\{P_\alpha\}_{\alpha \in A}$ is a family of Markov Chains, each having stationary distribution π . An adaptive MCMC algorithm would randomly update the value of α at each iteration, in an attempt to find the best value.

Now see the scale σ as the index parameter α , we can σ update at each iteration to make the acceptance rate around the one we want.

It is known that adaptive MCMC will not preserve stationarity of the target distribution. That means

$\|L(X_n) - \pi(\bullet)\| \rightarrow 0$, where $\|L(X_n) - \pi(\bullet)\| \equiv \sup_{A \subseteq \mathcal{X}} |P(X_n \in A) - \pi(A)|$. However, there are several conditions under which they will still converge. One is like the adaptation is done under regeneration times, others are under various technical procedures. Roberts and Rosenthal^[3] proved ergodicity of adaptive MCMC under conditions which doesn't require the adaptive parameters to converge; they state that an adaptive scheme will converge if it satisfies diminishing adaptation and bounded convergence conditions.

Theorem: suppose an adaptive scheme updates X_n to X_{n+1} using the kernel P_{Γ_n} , where each fixed kernel has the stationary distribution π , but where Γ_n are random indices chosen iteratively from some collection ψ based on past output. The scheme will converge if it satisfies *diminishing adaptation* and *bounded convergence conditions*:

$$(1) \text{ (diminishing adaptation) } \overline{\lim}_{n \rightarrow \infty} \sup_{x \in \mathcal{X}} \|P_{\Gamma_{n+1}}(x, \bullet) - P_{\Gamma_n}(x, \bullet)\| = 0 \text{ in probability.}$$

$$(2) \text{ (bounded convergence conditions) } \{M_\varepsilon(X_n, \Gamma_n)\}_{n=0}^\infty \text{ is bounded in probability, } \varepsilon > 0$$

$$\text{Where } M_\varepsilon(x, \gamma) = \inf \{n \geq 1 : \|P_\gamma^n(x, \bullet) - \pi(x, \bullet)\| \leq \varepsilon\}$$

1.6 Motivation

Let $\alpha(X, Y)$ denote $P(\text{accept proposal } Y | \text{current state is } X)$, the acceptance probability from X to Y , and $\alpha(X)$ denote $E[\alpha(X, Y) | X]$, the local acceptance probability for X , then global acceptance probability α is $\alpha = E[\alpha(X)] = E[E[\alpha(Y | X) | X]]$.

$$\alpha = \overline{\lim}_{n \rightarrow \infty} \frac{\# \text{accepted moves}}{n}$$

By ergodicity theorem, we know that $\alpha = \overline{\lim}_{n \rightarrow \infty} \frac{\# \text{accepted moves}}{n}$. So we can estimate the global acceptance probability by the global acceptance rate, i.e. the fraction of accepted movements among all the proposals. The mentioned optimal acceptance rate choice in section 1.4 is to make the global acceptance rate α around 0.234.

If the global acceptance is around 0.234, then the "average" of all the local acceptance rate $\alpha(X)$ for the different X is 0.234. That indicates $\alpha(X)$ is sometimes higher than 0.234 for some X and sometimes lower than 0.234 for some X . So one may wonder whether it is a "even more optimal choice" to make the the local acceptance rate $\alpha(X)$ around 0.234 for all the different X . We are curious that: if we make this happen or during the procedure to do this, will the Markov Chain has better efficiency, say having smaller varfact or bigger average squared jump distance?

So we will first try to achieve this goal(i.e. making $\alpha(X)$ around 0.234 for all the different X) by using adaptation algorithm in chapter 2 and then do some efficiency comparison of adaptation and non-adaptation algorithm in chapter 3.

2. Test of Local Acceptance Rate

In this chapter, we will show some simulation results with the purpose of making $E[\alpha(X, Y) | X]$ around 0.234 for all the different X by using the adaptation algorithm. Simulation results show that

it is "impossible" to make $\alpha(X)$ around 0.234 for all the different X by using the adaptation algorithm proposed in 2.1. Although we cannot achieve this goal in the end, we can still check whether the adaptation algorithm improves the efficiency of RWM during the efforts to make around 0.234. The comparison of adaptation and non-adaptation algorithms will be shown in chapter 3.

2.1 Idealist Adaptation Algorithm

The procedure of the adaptation algorithm for a general target distribution is as follows:

Procedure
<p>Step 1 set the initial value x_0.</p>
<p>Step 2 given the 1th, ..., nth value x_0, \dots, x_n, update $\sigma_n(x)$,</p> $\sigma_n(x) = h_n(x) \exp\{g_n(x)\},$ <p>Where $g_n(x) = \begin{cases} g_{n-1}(x) + \frac{\eta_n}{b_n} K\left(\frac{ x-x_n }{b_n}\right) & \text{if } x_n = y_n \\ g_{n-1}(x) - \frac{\eta_n}{b_n} K\left(\frac{ x-x_n }{b_n}\right) & \text{if } x_n \neq y_n \end{cases}$.</p>
<p>Step 3 generate y_{n+1} with proposal density : $y_{n+1} \sim N(x_n, \sigma_n^2(x_n))$.</p>
<p>Step 4 generate an independent $U_{n+1} \sim U(0,1)$.</p> <p style="text-align: center;">Set $x_{n+1} = y_{n+1}$ if $U_{n+1} < \alpha(x_n, y_{n+1}) = \frac{\pi(y_{n+1})q(x_n; y_{n+1})}{\pi(x_n)q(y_{n+1}; x_n)}$,</p> <p style="text-align: center;">Otherwise set $x_{n+1} = x_n$.</p>
<p>Step 5 repeat step 2~4 M times</p>

In this algorithm, we will update $h_n(x)$ in $\sigma_n(x) = h_n(x) \exp\{g_n(x)\}$ as

$$h_n(x) = e^{\beta_n} * (1 + C |x - c|)^{\gamma}$$

Where β_n is updated by adaptive rules:

$$\beta_n = \begin{cases} \beta_{n-1} + \frac{1}{\sqrt{n+5}} * (1-0.234) & \text{if } x_n = y_n \\ \beta_{n-1} - \frac{1}{\sqrt{n+5}} * 0.234 & \text{if } x_n \neq y_n \end{cases}$$

This suggests that if accepting the previous proposal y_n , we make β_n bigger and thereby bigger σ_n ; if rejecting the previous proposal, we make β_n smaller and thereby smaller σ_n .

In the second term of $h_n(x)$, $(1+C|x-c|)^\gamma$, the part $C|x-c|$ is to modify the scale according to how far X is from the center $c=0$. The far away X from the center, the bigger σ_n . This comes from the intuition that the density is very small around the center. We will try different choice of C, γ to make $\alpha(X)$ around 0.234 for all different X .

We will update g_n in this way:

$$g_n(x) = \begin{cases} g_{n-1}(x) + \frac{\eta_n}{b_n} K\left(\frac{|x-x_n|}{b_n}\right) & \text{if } x_n = y_n \\ g_{n-1}(x) - \frac{\eta_n}{b_n} K\left(\frac{|x-x_n|}{b_n}\right) & \text{if } x_n \neq y_n \end{cases}$$

b_n is the bandwidth of the update. We can choose it as a fixed constant or decreasing with respect

$$b_n = \frac{1}{n^{\frac{1}{5}}}$$

to n . One possible choice is $\frac{1}{n^{\frac{1}{5}}}$. We will test which is better, fixed or decreasing.

η_n is the speed of adaptation. It is naturally chosen as: $\eta_n = \frac{1}{n^a}, a \in (\frac{1}{2}, 1]$. In the code, we choose

it as $\eta_n = \frac{1}{\sqrt{n+5}}$. We will also try other η_n with other $a \in (\frac{1}{2}, 1]$ to check whether they give better performance.

$K(x)$ is the kernel function, usually chosen as a polynomial or exponential function:

$$K(x) = \frac{1}{1 + \alpha_1 |x|^{\alpha_2}}$$

$$K(x) = e^{-\frac{|x|^{\alpha_1}}{\alpha_2}}$$

$$K(x) = \begin{cases} 0, & |x| \geq 2 * width \\ -1 * height, & width < |x| < 2 * width \\ 1 * height, & |x| \leq width \end{cases}$$

$$K(x) = \begin{cases} 0, & |x| \geq 2 * width \quad \text{or} \quad |x| \leq width \\ 1, & width < |x| < 2 * width \end{cases}$$

2.2 Methods to Test $\alpha(X)$

Suppose that we get the final scale function as $\sigma(X)$ from the above adaptation algorithm. We can estimate $\alpha(X) = E[\alpha(Y | X) | X]$ in two different ways:

Method 1 to test $\alpha(X)$

step 1 generate n samples Y_1, \dots, Y_n from proposal density: $Y \sim N(X, \sigma_n^2(X))$

step 2 estimate $\alpha(X)$ as $\frac{1}{n} \sum_{i=1}^n \alpha(Y_i | X)$

This method works because it's actually computing the target expectation by Monte Carlo Method.

Method 2 to test $\alpha(X)$

step 1 generate n samples Y_1, \dots, Y_n from proposal density: $Y \sim N(X, \sigma_n^2(X))$

step 2 generate n independent U_1, \dots, U_n from $U(0,1)$

step 3 estimate $\alpha(X)$ as $\frac{1}{n} \sum_{i=1}^n 1(U_i \pi \alpha(Y_i | X))$

This method works because $1(U_i \pi \alpha(Y_i | X))$ are independent distributed and $E[E[1(U_i \pi \alpha(Y_i | X))]] = E[P(U_i \pi \alpha(Y_i | X))] = E[\alpha(Y_i | X)] = \alpha(X)$. So by Strong Law of

Large Numbers $\frac{1}{n} \sum_{i=1}^n 1(U_i \pi \alpha(Y_i | X))$ will converge almost everywhere to $\alpha(X)$.

Another more intuitive way to prove the validation of this method is:

$$\begin{aligned} \alpha(X) &= E[\alpha(Y | X) | X] \\ &= E[\text{accept moving from } X \text{ to } Y | X] \end{aligned}$$

$$= \lim_{n \rightarrow \infty} \frac{1}{n} \sum_{i=1}^n \mathbb{1}(\text{accept moving from } X \text{ to } Y_i)$$

where Y_1, \dots, Y_n are sampled from the proposal density.

2.3 Example 1: Normal Distribution in R^1

Let's start with one easy case: standard normal distribution in R^1 : $N(0,1)$ with density function:

$$f(x) = C_1 * e^{-\frac{x^2}{2}}$$

To avoid numerical error, I will compute the logarithm of the un-normalized density and acceptance probability instead of themselves directly.

The proposal density becomes:

$$\sigma_n(x) = h_n(x) \exp\{g_n(x)\}$$

Where

$$h_n(x) = e^{\beta_n} * (1 + C |x|)^\gamma$$

$$g_n(x) = \begin{cases} g_{n-1}(x) + \frac{\eta_n}{b_n} K\left(\frac{|x-x_n|}{b_n}\right) & \text{if } x_n = y_n \\ g_{n-1}(x) - \frac{\eta_n}{b_n} K\left(\frac{|x-x_n|}{b_n}\right) & \text{if } x_n \neq y_n \end{cases}$$

2.3.1 Output

I try different choices for parameters and kernel functions. You can refer to Appendix 2.1 for all the 66 cases I try. Appendix 2.1 includes the plot of final $g(x), \sigma(x)$, estimation of $\alpha(x)$ for some specific x , and some Markov Chain generated using adaptation.

For the 66 cases, they all have reasonable trace plots and final $g(x), \sigma(x)$ plot. The global acceptance rate is around 0.234, but none satisfies that $\alpha(0), \alpha(2), \alpha(5), \alpha(10), \alpha(20), \alpha(30), \alpha(50)$ are around 0.234.

Among them, case 30:

$$\text{Kernel function: } K(x) = \frac{1}{1 + \alpha_1 |x|^{\alpha_2}},$$

$$\eta_n = \frac{1}{(n+5)^{0.5}}, \alpha_1 = 1, \alpha_2 = 1, C = 1, \gamma = 2 \text{ with decreasing bandwidth } b_n = \frac{1}{n^{0.2}}$$

gives the closest $\alpha(x)$ to 0.234 for these specific x :

```
alpha(0.000000) is 0.269674
alpha(2.000000) is 0.249986
alpha(5.000000) is 0.293759
alpha(10.000000) is 0.307941
alpha(20.000000) is 0.279936
alpha(50.000000) is 0.176204
alpha(100.000000) is 0.109100
alpha(150.000000) is 0.075300
```

Table 1 $\alpha(x)$ for some specific x in case 30

$\alpha(x)$ is around 0.26 when $x = 0$, first increases to around 0.3 when $x = 10$, and then decreases to 0 as goes to infinity.

For case 30, the final $g(x), \sigma(x)$ looks like:

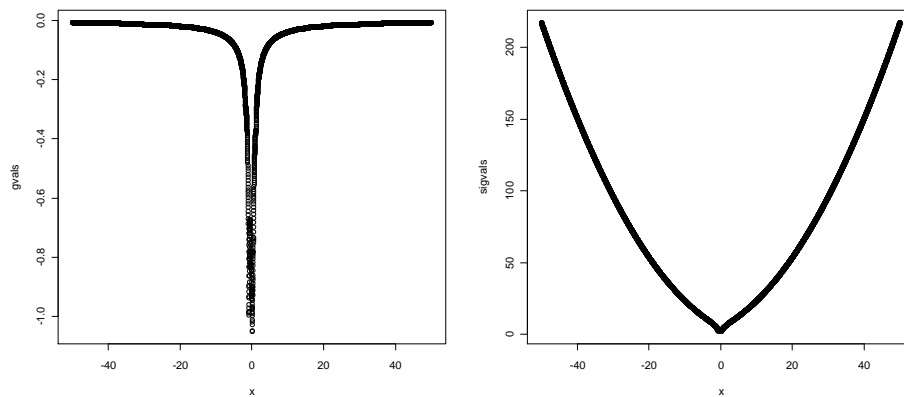


Figure 1 plot of $g(x), \sigma(x)$ in case 30

They seem reasonable. They are symmetric around 0. $\sigma(x)$ seems smooth and is the smallest around 0. This is good because we want a small scale at the right mode so that the sampled random variables will be more likely to lay around it, where the target distribution also reaches the maximum.

It is difficult to give the local acceptance rate for all x . So we will calculate $\alpha(x)$ for just some specific x , like $x = 0, 2, 5, 10, 20, 50, 100, 150$. The values of $\alpha(x)$ at these discrete points can provide us with a general idea of how $\alpha(x)$ changes with x over the real line. I compute $\alpha(x)$ for only positive x for it is symmetric around 0 and takes the same value for x and $-x$. $\alpha(x)$ for

some specific x is as Table 1. In this case, $\alpha(x)$ is around $0.2 \sim 0.4$ for $|x| \pi$ around 5 and always stays around 0.5, not 0.234, for $|x| \geq$ around 5.

The output of all the different 66 cases suggests that however we choose the parameters, we probably cannot make the local acceptance rate $\alpha(x)$ for some certain x , let alone for all the different x . But it remains to be tested that whether the adaptation algorithm with good parameter choices (like case 57) improves the efficiency of Markov Chain compared to non-adaptation algorithm.

2.3.2 How $\alpha(x)$ changes with parameters

If we only change the value of one factor and keep others the same, we can get a rough idea of how $\alpha(x)$ changes with parameters $C, \gamma, \alpha_1, b_n, \eta_n$ and different choice of kernel functions.

2.3.2.1 How $\alpha(x)$ changes with kernel function

If we want to make all the local acceptance rate around 0.234, then different kernel functions give almost the same good result. For the four kernel functions, the best cases are respectively case 18, 30, 34, 53:

Case 18: kernel function: $K(x) = e^{-\frac{|x|^{\alpha_1}}{\alpha_2}}$

$$\eta_n = \frac{1}{(n+5)^{0.5}}, \alpha_1 = 2, \alpha_2 = 1, C = 0.1, \gamma = 2 \text{ with decreasing bandwidth } b_n = \frac{1}{n^{0.2}}$$

Case 30: kernel function: $K(x) = \frac{1}{1 + \alpha_1 |x|^{\alpha_2}}$

$$\eta_n = \frac{1}{(n+5)^{0.5}}, \alpha_1 = 1, \alpha_2 = 1, C = 1, \gamma = 2 \text{ with decreasing bandwidth } b_n = \frac{1}{n^{0.2}}$$

Case 34: kernel function:
$$K(x) = \begin{cases} 0, & |x| \geq 2 * \text{width} \\ -1 * \text{height}, & \text{width} < |x| < 2 * \text{width} \\ 1 * \text{height}, & |x| \leq \text{width} \end{cases}$$

$$\eta_n = \frac{1}{(n+5)^{0.5}}, \text{height} = 0.5, \text{width} = 0.5, C = 0.1, \gamma = 2 \text{ with fixed bandwidth } b_n = 1$$

Case 53: kernel function:
$$K(x) = \begin{cases} 0, & |x| \geq 2 * \text{width} \text{ or } |x| \leq \text{width} \\ 1, & \text{width} < |x| < 2 * \text{width} \end{cases}$$

$$\eta_n = \frac{1}{(n+5)^{0.5}}, \text{ width} = 0.5, C = 0.1, \gamma = 2 \text{ with fixed bandwidth } b_n = 1$$

For these four cases, we have local acceptance rate as table 2 and final $g(X)$ and $\sigma(X)$ as table 3:

Case 18	Case 30	Case 34	Case 53
alpha(0) is 0.205731	alpha(0) is 0.269674	Alpha(0) is 0.230635	alpha(0) is 0.186088
alpha(2) is 0.300010	alpha(2) is 0.249986	alpha(2) is 0.321085	alpha(2) is 0.255791
alpha(5) is 0.330864	alpha(5) is 0.293759	alpha(5) is 0.392833	alpha(5) is 0.322271
alpha(10) is 0.353506	alpha(10) is 0.307941	alpha(10) is 0.407148	alpha(10) is 0.361781
alpha(20) is 0.317601	alpha(20) is 0.279936	alpha(20) is 0.374904	alpha(20) is 0.319420
alpha(50) is 0.211646	alpha(50) is 0.176204	alpha(50) is 0.263909	alpha(50) is 0.215913
alpha(100) is 0.1366	alpha(100) is 0.1091	alpha(100) is 0.1645	alpha(100) is 0.1306
alpha(150) is 0.0894	alpha(150) is 0.0753	alpha(150) is 0.11870	alpha(150) is 0.0984

Table 2 $\alpha(x)$ for the best cases with different kernel functions

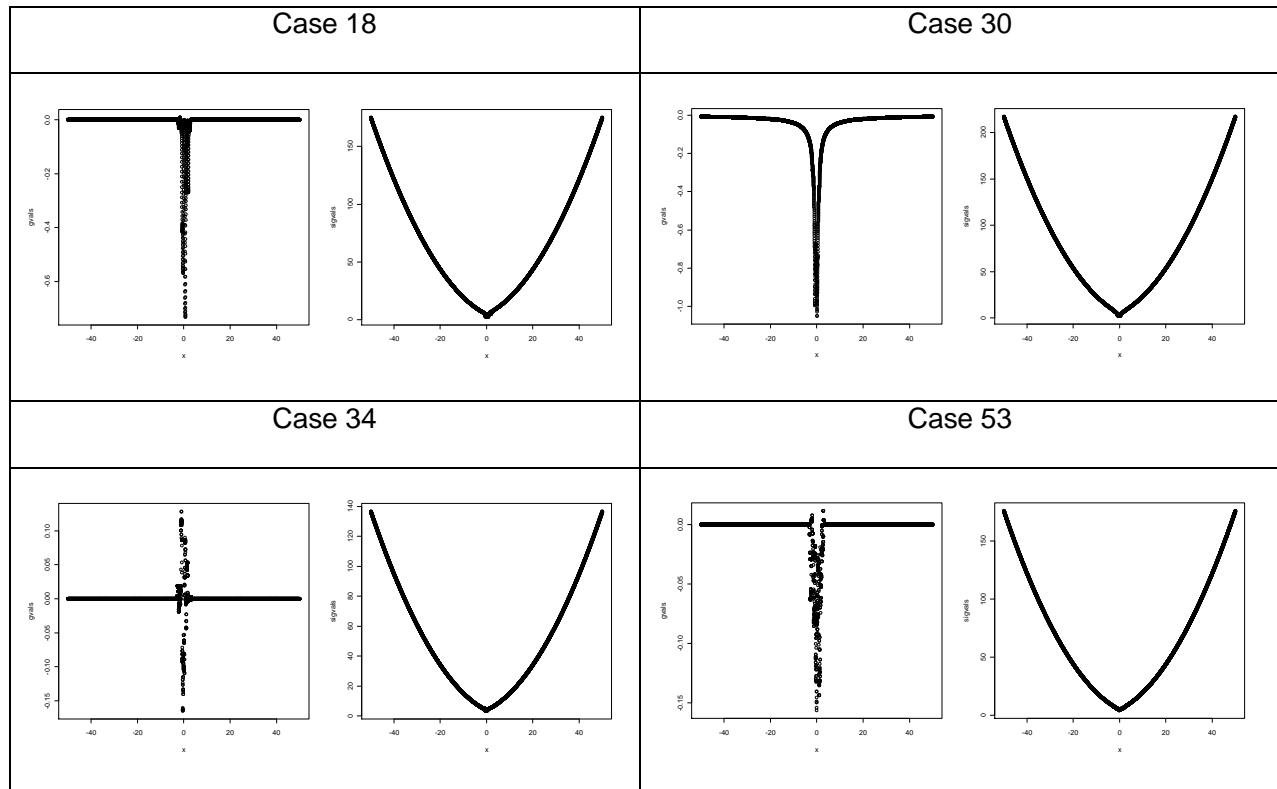


Table 3 plot of the final $g(X)$ and $\sigma(X)$ for the best cases with different kernel functions

From the above two tables, we can see that even the kernel function is different, but adaptation algorithm is still trying to make the final $\sigma(X)$ look some certain way. There is no strong support for

the choice of any kernel function. However, I prefer the first two: $K(x) = \frac{1}{1 + \alpha_1 |x|^{\alpha_2}}$ and $K(x) = e^{-\frac{|x|^{\alpha_1}}{\alpha_2}}$, for their final $g(x)$ and $\sigma(x)$ are smooth and this will facilitate our further discussion, like approximation of the final $\sigma(x)$.

2.3.2.2 How $\alpha(X)$ changes With C, γ

From Appendix 2.1, we can see that if $b_n = 1$, and $\eta_n = \frac{1}{(n+5)^{0.5}}$, then, $\alpha(X)$ has the same tendency as long as C, γ have the same values, however we choose other parameters.

1) When C, γ are big, $\alpha(X)$ peaks at $X = 0$, and decreases to 0 as $|X|$ goes to infinity. Among these cases are CASE 1, 2, 3, 4, 5, 6, 7, 9, 11. Parameter choices are like: $C \geq 0.5, \gamma > 1$; $C \geq 0.1, \gamma \geq 10$

2) When C, γ are small, $\alpha(X)$ reaches the global minimum at $X = 0$, and increases to 0.5 as $|X|$ goes to infinity. Among these cases are CASE 10, 12, 13, 14. Parameter choices are like: $C \leq 10, \gamma \leq 0.01$; $C \leq 0.1, \gamma \leq 1$

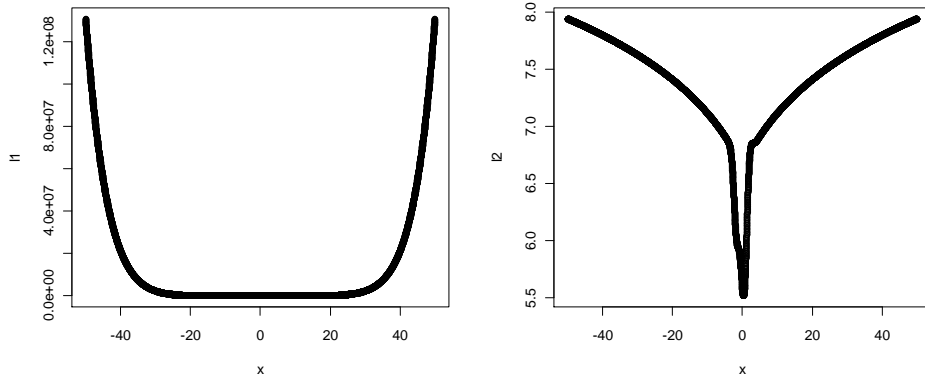
Although the below section 2.3.3.2.1 and 2.3.3.2.2 are based on kernel function: $K(x) = e^{-\frac{|x|^{\alpha_1}}{\alpha_2}}$, the similar results hold for other kernel functions, which you can check from the Appendix 2.1.

2.3.2.2.1 Why does $\alpha(X)$ decrease to 0 as $|X|$ goes to infinity when C, γ are big?

C, γ determines how big the effect the second term $(1 + C|x|)^\gamma$ has on $\sigma(x) = e^{\beta + g(x)}(1 + C|x|)^\gamma$. If they are big, then the second term has a big influence on $\sigma(X)$. Approximately, we can assume that $\sigma(x) \propto (1 + C|x|)^\gamma$, so $\sigma(X)$ grows very quickly as $|X|$ increases from 0 to infinity. It grows so quickly that it becomes the most important factor to determine the value of $\alpha(X)$. The bigger $\sigma(X)$, the smaller acceptance rate $\alpha(X)$. So $\alpha(X)$ decreases and goes to 0 as $|X|$ goes to infinity.

Figure 2 shows the different plot of final $\sigma(X)$ when C, γ are big and small. They are from case 11 ($\alpha_1 = 2, \alpha_2 = 1, C = 0.1, \gamma = 10$ with fixed bandwidth $b_n = 1$), and case 14

($\alpha_1 = 2, \alpha_2 = 1, C = 0.1, \gamma = 0.1$ with fixed bandwidth $b_n = 1$), They have the same values for all the parameters except γ . You can see how quickly $\sigma(X)$ grows in CASE 11, in which C, γ are big. For $X \approx 50$, $\sigma(X)$ takes the value around 10^8 in CASE 11, much bigger than 8 in CASE 14.



CASE 11 ($C = 0.1, \gamma = 10$)

CASE 14 ($C = 0.1, \gamma = 0.1$)

Figure 2 the plot of final $\sigma(X)$ for three different cases

2.3.2.2 Why does $\alpha(X)$ increase to 0.5 as $|X|$ goes to infinity when C, γ are small?

If C, γ are both small, then the second term has a small effect on $\sigma(X)$. So approximately, we can assume that $\sigma(x) \propto e^{g(x)}$. Unfortunately, we cannot get any conclusion about the tendency of $\sigma(X)$ as $|X|$ goes to infinity from this rough approximation because we don't know the express of $g(X)$. We need some more precise analysis.

Let's go back to the definition of the local acceptance probability:

$$\alpha(x) = E(\alpha(x, y)) = E\left(\min\left\{\frac{\pi(y)q_y(x)}{\pi(x)q_x(y)}, 1\right\}\right)$$

Since $q_x(y)$ is symmetric at x , the reason that $\alpha(x)$ takes the value of 0.5 for big enough x is

probably that $\min\left\{\frac{\pi(y)q_y(x)}{\pi(x)q_x(y)}, 1\right\} \approx 1$ for y on one side of x and $\min\left\{\frac{\pi(y)q_y(x)}{\pi(x)q_x(y)}, 1\right\} \approx 0$ for y on the other side of x . Now let's try to check whether it is true.

For $q_x(y) = \frac{1}{\sigma(x)} e^{-\frac{(y-x)^2}{2\sigma(x)^2}}$, we have $\frac{\pi(y)q_y(x)}{\pi(x)q_x(y)} = \frac{\exp(-\frac{y^2}{2})\exp\left(-\frac{(x-y)^2}{2\sigma(y)^2}\right)\sigma(x)}{\sigma(y)\exp(-\frac{x^2}{2})\exp\left(-\frac{(x-y)^2}{2\sigma(x)^2}\right)}$

$$= \frac{\sigma(x)}{\sigma(y)} \exp\left(\frac{(x-y)^2}{2\sigma(x)^2} + \frac{x^2}{2} - \frac{(x-y)^2}{2\sigma(y)^2} - \frac{y^2}{2}\right).$$

Since $\sigma(X)$ takes the form of recursion and the exact mathematical express of $\sigma(X)$ is very complicated, it is better to use some approximation function to reflect the tendency of $\sigma(x)$, like exponential function or polynomial function. Below I will use CASE 14 ($C = 0.1, \gamma = 0.1$) as an example to explain how to approximate $\sigma(x)$ and check $\alpha(x)$. Other cases of this kind of parameter choice follow the similar idea.

Let's firstly check the plot of final $\sigma(x)$.

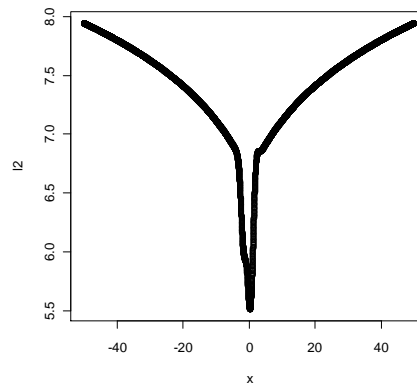


Figure 3 the plot of final $\sigma(X)$ for CASE 14

From Figure 3, we guess maybe a mixture of two exponential functions centered at 0 can be used to

approximate the final $\sigma(x)$, like $\hat{\sigma}(x) \approx \begin{cases} C_1 * |x|^{C_2} + C_3, |x| < 3 \\ C_4 * |x|^{C_5} + C_6, |x| \geq 3 \end{cases}$. I use R and implement OLS to get

the estimation as: $\hat{C}_1 \approx 0.3, \hat{C}_2 \approx 1.4, \hat{C}_3 \approx 5.572407, \hat{C}_4 \approx 0.1, \hat{C}_5 \approx 0.63, \hat{C}_6 \approx 6.773458$. I will call this as approximation I. We can believe this is a good approximation from Figure 4, the comparison

of the final $\sigma(x)$ and its approximation $\hat{\sigma}(x)$: they overlap a lot, especially when x is large.

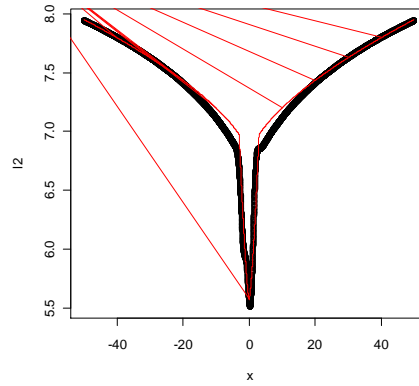


Figure 4 plot of the final $\sigma(x)$ and the approximation $\hat{\sigma}(x)$ (black: $\sigma(x)$; red: $\hat{\sigma}(x)$)

For big x, say 30,50,80,150,200,300, $\min \left\{ \frac{\pi(y)q_y(x)}{\pi(x)q_x(y)}, 1 \right\}$ has the tendency as Figure 5:

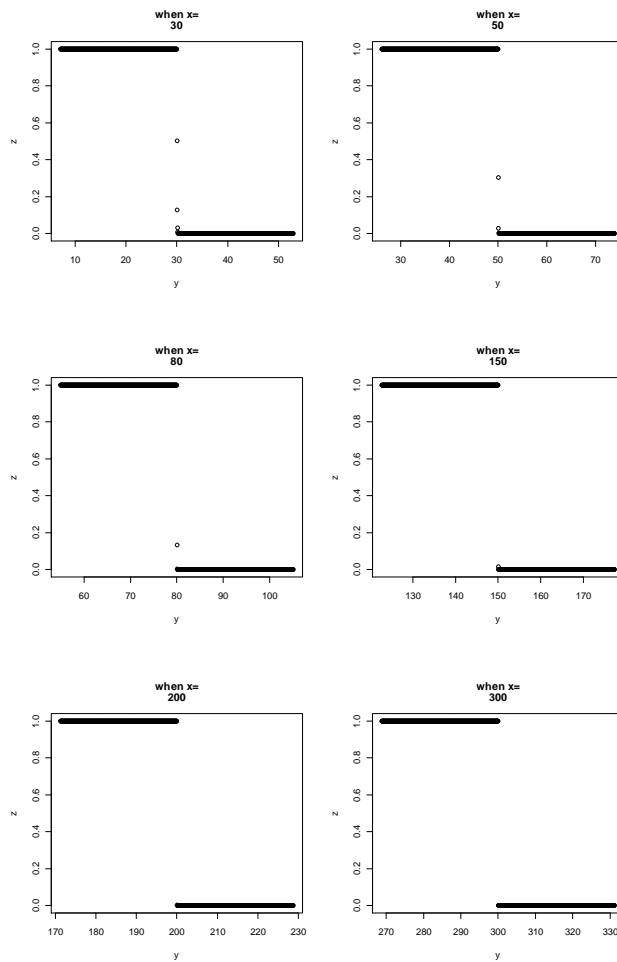


Figure 5 plot of $\min\left\{\frac{\pi(y)q_y(x)}{\pi(x)q_x(y)},1\right\}$ for big x

From Figure 5, for each fixed big x , $\min\left\{\frac{\pi(y)q_y(x)}{\pi(x)q_x(y)},1\right\} \approx 1$ for $y < x$ and $\min\left\{\frac{\pi(y)q_y(x)}{\pi(x)q_x(y)},1\right\} \approx 0$ for $y > x$. Since $q_x(y)$ is symmetric at x , $E\left[\min\left\{\frac{\pi(y)q_y(x)}{\pi(x)q_x(y)},1\right\}\right] \approx 0.5$. So $\alpha(x) = E(\alpha(x, y)) = E\left(\min\left\{\frac{\pi(y)q_y(x)}{\pi(x)q_x(y)},1\right\}\right) = 0.5$.

we can also use approximation of $\sigma(x)$ to see why when C, γ are between the two extremes, $E\left[\min\left\{\frac{\pi(y)q_y(x)}{\pi(x)q_x(y)},1\right\}\right] \approx 0$ for very big x, like case 16.

Comparing the local acceptance rate of the three situations, we find it better to choose C, γ between the two extremes, like $C = 0.1, \gamma = 2$. Because we want $\alpha(X)$ to neither decrease quickly to 0 nor always stay around 0.5. If we choose C, γ between the two extremes, then $\alpha(X)$ still converges to 0, but in a much slower speed; it stays around 0.2~0.4 for $|X| \leq 50$. This is the closet situation to our goal of making all the local acceptance around 0.234. But we are not sure whether it has better efficiency than the non-adaptation algorithm, which will be discussed in Chapter 3.

2.3.2.3 How $\alpha(X)$ Changes With α_1

If C, γ remain the same, then changes in the value of α_1 has little influence over the tendency of local acceptance rate.

In case 1~5, C, γ are fixed $C = 1, \gamma = 2$, bandwidth is fixed as $b_n = 1$, and $\eta_n = \frac{1}{(n+5)^{0.5}}$, while the value of α_1 varies from 0.1 to 10. The tendency of $\alpha(X)$ are very similar: starts from around 0.3 and then decreases to 0. Also, the plot of final $\sigma(X)$ have the same shape.

Case 1	Case 2	Case 3	Case 4	Case 5
$\alpha_1 = 1$	$\alpha_1 = 2$	$\alpha_1 = 10$	$\alpha_1 = 0.1$	$\alpha_1 = 0.5$
alpha(0) is 0.294400	alpha(0) is 0.275100	alpha(0) is 0.287000	alpha(0) is 0.263300	alpha(0) is 0.309200

alpha(2) is 0.146900	alpha(2) is 0.123300	alpha(2) is 0.146700	alpha(2) is 0.135800	alpha(2) is 0.150900
alpha(5) is 0.079300	alpha(5) is 0.065000	alpha(5) is 0.067300	alpha(5) is 0.070300	alpha(5) is 0.076500
alpha(10) is 0.047600	alpha(10) is 0.036300	alpha(10) is 0.042100	alpha(10) is 0.041900	alpha(10) is 0.045300
alpha(20) is 0.022500	alpha(20) is 0.020400	alpha(20) is 0.022000	alpha(20) is 0.021200	alpha(20) is 0.022600
alpha(50) is 0.009100	alpha(50) is 0.008500	alpha(50) is 0.008600	alpha(50) is 0.008200	alpha(50) is 0.010200
alpha(100) is 0.00580	alpha(100) is 0.00370	alpha(100) is 0.0055	alpha(100) is 0.00560	alpha(100) is 0.00330
alpha(150) is 0.00260	alpha(150) is 0.00440	alpha(150) is 0.0025	alpha(150) is 0.00400	alpha(150) is 0.00340

Table 4 $\alpha(X)$ for case 1~5

Case 1	Case 2	Case 3	Case 4	Case 5
$\alpha_1 = 1$	$\alpha_1 = 2$	$\alpha_1 = 10$	$\alpha_1 = 0.1$	$\alpha_1 = 0.5$

Table 5 plot of the final $\sigma(X)$ for case 1~5

From the above two tables, we can see that the final $\sigma(X)$ is quite similar for difference choice of α_1 and thereby it doesn't make big changes to the tendency of $\alpha(X)$ when we change α_1 .

The same result holds for the different choice of kernel functions, which you can check from the first several cases in Appendix 2.1.1~2.1.4.

2.3.2.4 How $\alpha(X)$ Changes With bandwidth b_n

Different b_n does not change the general tendency of $\alpha(X)$; it only affects the speed in which $\alpha(X)$ goes to 0 or 0.5.

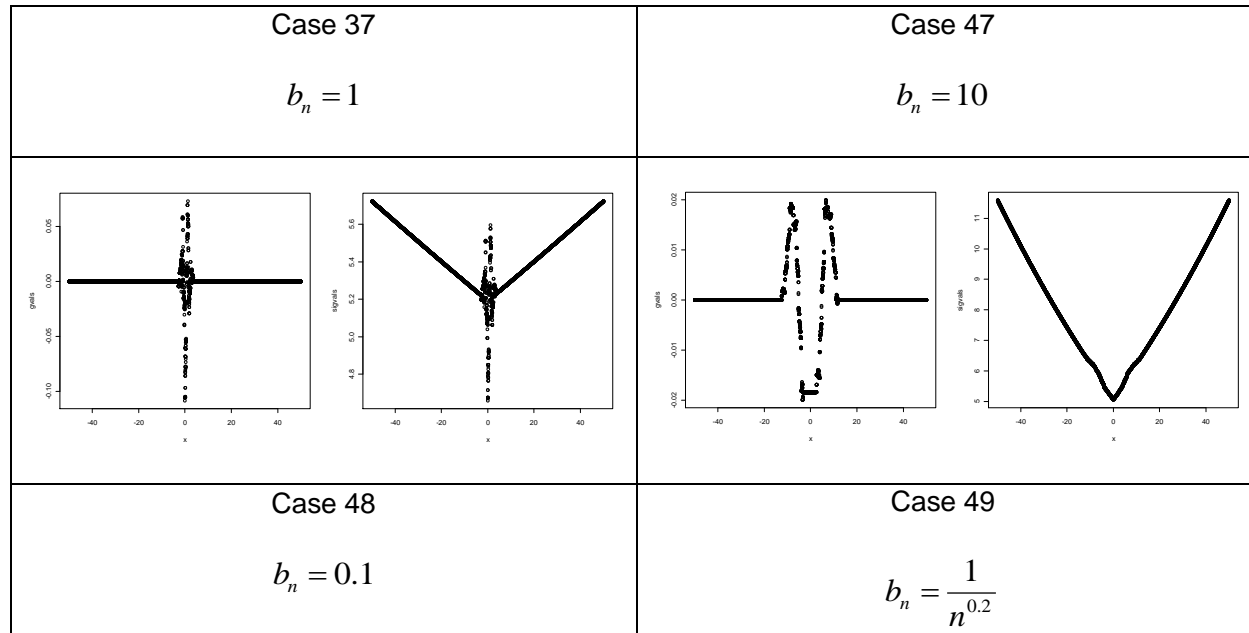
In case 37, 47, 48, 49, b_n is different and all the other parameters are the same

($\alpha_1 = 2, \alpha_2 = 1, C = 0.1, \gamma = 2$) kernel function is $K(x) = \begin{cases} 0, & |x| \geq 2 * width \\ -1 * heigth, & width < |x| < 2 * width \\ 1 * heigth, & |x| \leq width \end{cases}$. In

case 37 , 47 , 48, b_n are fixed as a constant 1, 10 , 0.1 respectively, while in case 49, I choose decreasing bandwidth $b_n = \frac{1}{n^{0.2}}$. However, $\alpha(X)$ has similar tendency and plot of final $\sigma(x)$ is also similar to each other. Table 6 and 7 shows the local acceptance rate and plot of $g(x)$ and $\sigma(x)$ for these four cases.

Case 37 $b_n = 1$	Case 47 $b_n = 10$	Case 48 $b_n = 0.1$	Case 49 $b_n = \frac{1}{n^{0.2}}$
alpha(0) is 0.194871	alpha(0) is 0.191540	alpha(0) is 0.198912	alpha(0) is 0.168346
alpha(2) is 0.334081	alpha(2) is 0.3355	alpha(2) is 0.328186	alpha(2) is 0.303514
alpha(5) is 0.494616	alpha(5) is 0.479633	alpha(5) is 0.474185	alpha(5) is 0.473132
alpha(10) is 0.501907	alpha(10) is 0.498527	alpha(10) is 0.494897	alpha(10) is 0.508745
alpha(20) is 0.503077	alpha(20) is 0.500063	alpha(20) is 0.509700	alpha(20) is 0.500702
alpha(50) is 0.499887	alpha(50) is 0.505558	alpha(50) is 0.493201	alpha(50) is 0.502543
alpha(100) is 0.49540	alpha(100) is 0.50062	alpha(100) is 0.50840	alpha(100) is 0.50144
alpha(150) is 0.49668	alpha(150) is 0.49368	alpha(150) is 0.49711	alpha(150) is 0.50043

Table 6 $\alpha(X)$ for different b_n



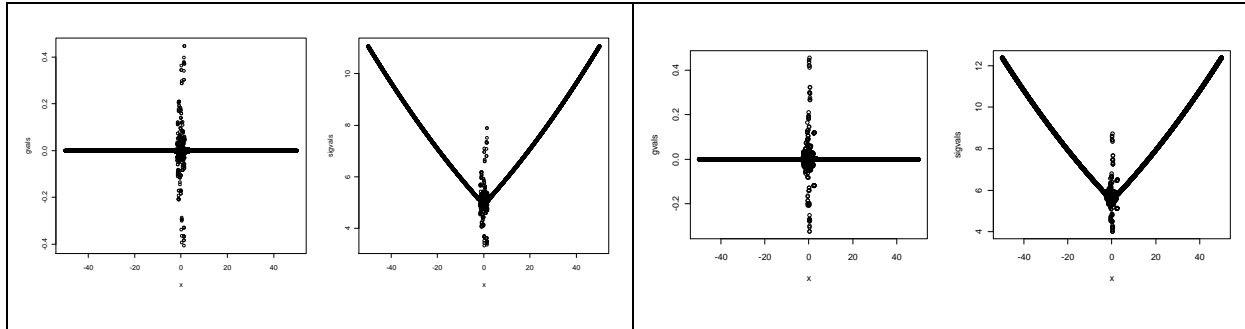


Table 7 plot of $g(x)$ and $\sigma(x)$ for different b_n

For other kernel functions, you can conclude the same statement.

2.3.2.5 How $\alpha(X)$ Changes With bandwidth η_n

In section 2.1, we discuss there are different choices of the power of η_n . We can try 0.5, and also some real value between 0.5 and 1. So I try 0.5 and 0.8 to compare which one is better. Simulation

indicates that there is no big difference in the power choice of $\eta_n = \frac{1}{(n+5)^{power}}$.

Let's choose case 27 and 31 for comparison:

Case 27: $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 1, \alpha_2 = 1, C = 0.1, \gamma = 2$ with fixed bandwidth $b_n = 10$

Case 31: $\eta_n = \frac{1}{(n+5)^{0.8}}$, $\alpha_1 = 1, \alpha_2 = 1, C = 0.1, \gamma = 2$ with fixed bandwidth $b_n = 10$

Case 27 $\eta_n = \frac{1}{(n+5)^{0.5}}$	Case 31 $\eta_n = \frac{1}{(n+5)^{0.8}}$
alpha(0.000000) is 0.201310	alpha(0.000000) is 0.221199
alpha(2.000000) is 0.301568	alpha(2.000000) is 0.320933
alpha(5.000000) is 0.366086	alpha(5.000000) is 0.391188
alpha(10.000000) is 0.380836	alpha(10.000000) is 0.401987
alpha(20.000000) is 0.352299	alpha(20.000000) is 0.378950
alpha(50.000000) is 0.239894	alpha(50.000000) is 0.264811
alpha(100.000000) is 0.155501	alpha(100.000000) is 0.163500
alpha(150.000000) is 0.105800	alpha(150.000000) is 0.120201

Table 8 $\alpha(X)$ for different η_n

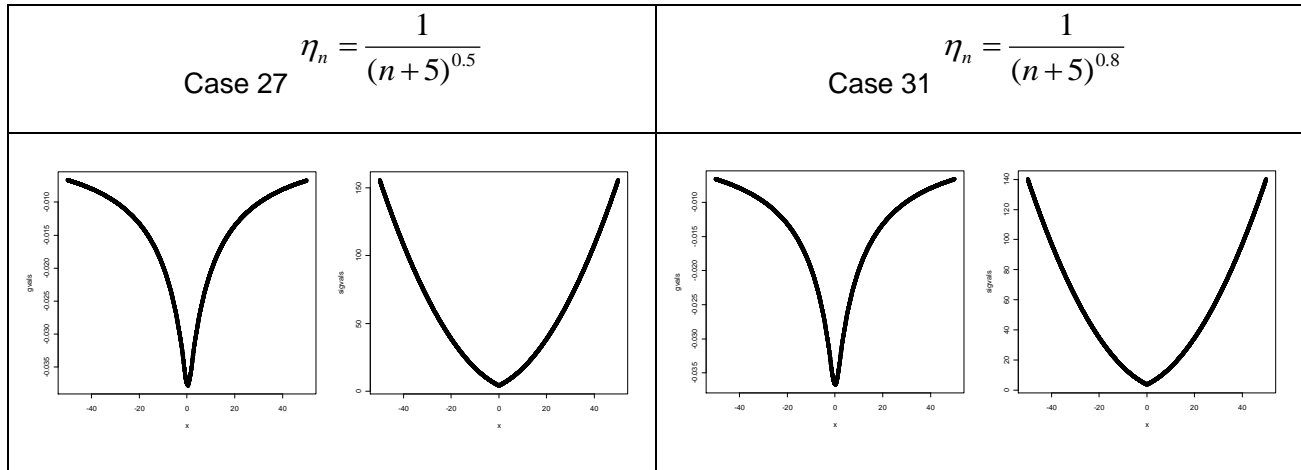


Table 9 plot of $g(x)$ and $\sigma(x)$ for different η_n

From the above tables, you can see that there is no big difference whether we choose $\eta_n = \frac{1}{(n+5)^{0.5}}$

or $\eta_n = \frac{1}{(n+5)^{0.8}}$.

So if we want to make $\alpha(x)$ close to 0.234, it is more important to find good values of C, γ than other parameters.

2.3.3 how to improve the algorithm speed

The current algorithm is running in a quite slow speed. It takes around 40 minutes to generate one Markov Chain and around 38 minutes to test the local acceptance for eight points. This is quite time-consuming. We need to explore some methods to improve the algorithm speed.

The reason why it runs so slow is that we use the exact value of final $\sigma(x)$ every time the Markov Chain moves to a different state x . So we need 10^4 times of iterations to compute each $g(x)$ and $\sigma(x)$ for every x and needs much time. One way to improve the speed is to use approximation instead of exact value. We know that good approximation of $\sigma(x)$ can significantly improve the algorithm speed.

We can divide the real area that the chain stays at most time (say, $x \in [-30, 30)$) into several smaller intervals (say, equally 6000 intervals) , then compute and save the values of final $\sigma(x)$ for every left point. That means we record $\sigma(x)$ for $x \in [-30, 30)$ with the separation of 0.01. For each $x \in [-30, 30)$ that is not the left point of any small interval, approximate $\sigma(x)$ by $\sigma(\bullet)$ at the left

point of the small interval that x lies in. For each $x \notin [-30,30]$, just use their exact value of $\sigma(x)$; this will not cost much time because for the random variable of the standard normal distribution will seldom go to the area $x \notin [-30,30]$.

I will call this approximation of $\sigma(x)$ as Approximation II. Now run the algorithm with Approximation II for case 1, and check the speed for comparison.

Without Approximation II	With Approximation II
it takes 218.947 seconds to run the adaptation algorithm	it takes 233.63636 seconds to run the adaptation algorithm
it takes 95.419 seconds to compute g and sigma	it takes 95.5646 seconds to compute g and sigma
it takes 2380.314 seconds to generate the first Markov Chain	it takes 21.525 seconds to generate the first Markov Chain
it takes 2367.3 seconds to generate the second Markov Chain	it takes 25.25 seconds to generate the second Markov Chain
it takes 2386.311 seconds to generate the third Markov Chain	it takes 23.8585 seconds to generate the third Markov Chain
it takes 2402.55 seconds to generate the fourth Markov Chain	it takes 21.156 seconds to generate the fourth Markov Chain
it takes 2402.489seconds to generate the fifth Markov Chain	it takes 26.6363 seconds to generate the fifth Markov Chain
it takes 2144.659 seconds to test the local acceptance	it takes 1564.99 seconds to test the local acceptance

Table 10 running time with and without Approximation II

From table 10, we can see that the speed is significantly faster: now it takes only around 20 seconds to generate a Markov Chain with Approximation II, while it needs around 40 minutes without Approximation II; and it takes around 25 minutes to test the local acceptance rate for 8 different X's, while it needs around 40 minutes without Approximation II.

The speed is good now; but will the algorithm with Approximation II give the similar result to that without it? Table 11 and 12 show $\alpha(x)$ and the plot of final $g(x)$ and $\sigma(x)$. They are similar to each other. So Approximation II is a good method to significantly improve the speed of the algorithm.

Without Approximation II	With Approximation II
alpha(0.000000) is 0.307000	alpha(0.000000) is 0.294400
alpha(2.000000) is 0.145700	alpha(2.000000) is 0.146900
alpha(5.000000) is 0.078100	alpha(5.000000) is 0.079300
alpha(10.000000) is 0.042800	alpha(10.000000) is 0.047600
alpha(20.000000) is 0.022500	alpha(20.000000) is 0.022500
alpha(50.000000) is 0.009400	alpha(50.000000) is 0.009100
alpha(100.000000) is 0.003800000	alpha(100.000000) is 0.0058

Table 11 $\alpha(x)$ for case 1 with and without Approximation II

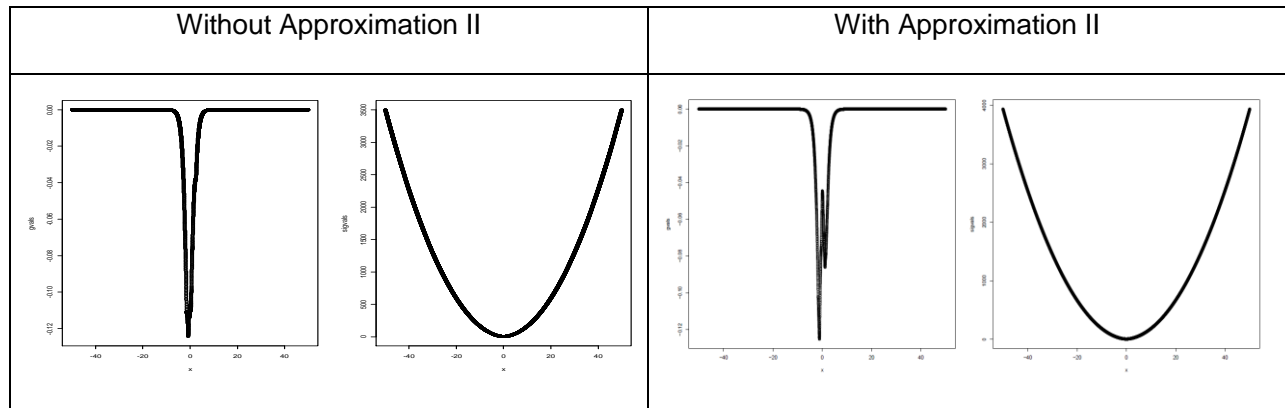


Table 12 plot of $g(x)$ and $\sigma(x)$ for case 1 with and without Approximation II

The speed to generate a Markov Chain is much faster now, about 100 times faster than without Approximation II. But the speed to test the local acceptance rate doesn't get as much improvement as generating. This is because when testing $\alpha(x)$ for big x , we need to compute $\sigma(x)$ for big x , which isn't approximated in Approximation I. One way to improve this is to use Approximation I instead, which gives approximation for every single x , not just a part of the real numbers.

Now run the algorithm with Approximation I for case 14, and check the speed for comparison.

With Approximation I	With Approximation II
alpha(0.000000) is 0.289099	alpha(0.000000) is 0.168239
alpha(2.000000) is 0.404383	alpha(2.000000) is 0.273334
alpha(5.000000) is 0.425934	alpha(5.000000) is 0.440324
alpha(10.000000) is 0.501186	alpha(10.000000) is 0.502495
alpha(20.000000) is 0.497216	alpha(20.000000) is 0.499843
alpha(50.000000) is 0.498001	alpha(50.000000) is 0.496880
alpha(100.000000) is 0.498768	alpha(100.000000) is 0.498992

Table 13 $\alpha(x)$ for case 14 with and without Approximation I

With Approximation I	With Approximation II
it takes 0.345 seconds to generate second Markov Chain	it takes 16.43 seconds to generate second Markov Chain
it takes 0.387 seconds to generate third Markov Chain	it takes 16.54 seconds to generate third Markov Chain
it takes 0.40 seconds to generate fourth Markov Chain	it takes 16.62seconds to generate fourth Markov Chain
it takes 0.354 seconds to generate fifth Markov Chain	it takes 16.72seconds to generate fifth Markov Chain
it takes 0.166 seconds to test the local acceptance	it takes 1005.606 seconds to test the local acceptance

Table 14 running time for case 14 with Approximation I and II

Now it is very quick to test the local acceptance rate. It takes about less than 1 second to test $\alpha(x)$ for 8 points, much faster than Approximate II, which needs around 1000 seconds. But from table 10, you, it has good estimation of $\alpha(x)$ for big x and bad estimation of $\alpha(x)$ for small x , which is untenable. So the advantage of Approximation I is to improve the speed of both generating Markov Chain and testing $\alpha(x)$ significantly, however big x is. But it doesn't give believable estimation of $\alpha(x)$ for small x . While Approximation II gives good estimation of $\alpha(x)$ for every x , although it is slow to test $\alpha(x)$ for big x . Another disadvantage of Approximation I is that we need to estimate the parameters in it before we can use it, like we do in section 2.3.2.2.1. While for Approximation II, we only need to let the computer record the values of $\sigma(x)$ for some x . In all the other cases, I will use Approximation II.

2.4 Example 2: Mixture of Two Normal Distributions in R^1

Now consider the target distribution as mixture of two normal distributions in R^1 : $N(0,1)$ and $N(10,1)$, so the density is

$$f(x) = C_4 * \left(e^{-\frac{x^2}{2}} + e^{-\frac{(x-10)^2}{2}} \right)$$

Figure 6 shows the un-normalized part of this density function. It reaches the maximum at 0 and 10.

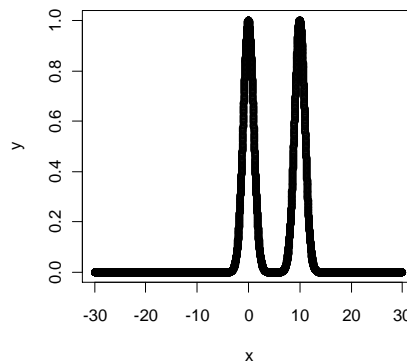


Figure 6 un-normalized part of the density

The proposal density will become :

$$\sigma_n(x) = h_n(x) \exp\{g_n(x)\}$$

Where

$$h_n(x) = e^{\beta_n} * (1 + C |x - 5|)^\gamma,$$

$$g_n(x) = \begin{cases} g_{n-1}(x) + \frac{\eta_n}{b_n} K\left(\frac{|x-x_n|}{b_n}\right) & \text{if } x_n = y_n \\ g_{n-1}(x) - \frac{\eta_n}{b_n} K\left(\frac{|x-x_n|}{b_n}\right) & \text{if } x_n \neq y_n \end{cases}$$

The same method to update parameters and test $\alpha(x)$ is conducted for this example.

2.4.1 How to Avoid Numeric Error

Numerical error may arise from the finite precision of computations. For example, to compute $\log(e^x)$ for $x < -900$. It is easy to get $\log(e^x) = x$ if we calculate it by ourselves. But if we let a computer to do this calculation, it first computes e^x for $x < -900$ and gets a value very close to 0, say y , and then compute $\log(y)$. y is usually so close to 0 that the computer calculates $\log(e^x)$ as $\log(0)$, which is infinity. So the computer will probably return NA or -INF. The same problem exists

in our target distribution: $\pi(x) \propto e^{-\frac{x^2}{2}} + e^{-\frac{(x-10)^2}{2}}$ when has big absolute value.

2.4.1.1 Why does the numerical error happen?

In our C code, we usually compute the logarithm of $\pi(x)$ instead of itself:

$$\log(\pi(x)) = \log\left(e^{-\frac{x^2}{2}} + e^{-\frac{(x-10)^2}{2}}\right)$$

when x is too big, say 50, then $e^{-\frac{x^2}{2}} = e^{-\frac{50^2}{2}} = e^{-1250}$ and $e^{-\frac{(x-10)^2}{2}} = e^{-\frac{40^2}{2}} = e^{-800}$, both very small to compute in C. C will first compute e^{-800} as a small value very close to 0 and e^{-1250} as an even smaller value much closer to 0, and then sum them to a small value close to 0, whose logarithm will be treated as $\log(0)$ in C. That's why C returns $-\infty$ for $\log(\pi(x))$ when x is too big. It's the same with negative x with big absolute value.

Table 15 gives the computed value of $\log(\pi(x))$ for some y in C:

when y=-50.000000, logpi(-50.000000)=-1.#INF00
when y=-30.000000, logpi(-30.000000)=-450.000000
when y=-10.000000, logpi(-10.000000)=-50.000000
when y=10.000000, logpi(10.000000)=0.000000
when y=30.000000, logpi(30.000000)=-200.000000
when y=50.000000, logpi(50.000000)=-1.#INF00

Table 15 $\log(\pi(x))$ for some y

2.4.1.2 How to approximate $\log(\pi(x))$?

Although for big x, $e^{-\frac{x^2}{2}}$ and $e^{-\frac{(x-10)^2}{2}}$ are very small, $e^{-\frac{x^2}{2}}$ is much smaller than $e^{-\frac{(x-10)^2}{2}}$:

$$\frac{e^{-\frac{x^2}{2}}}{e^{-\frac{(x-10)^2}{2}}} = e^{\frac{(x-10)^2 - x^2}{2}} = e^{-10x+50}$$

This inspires us to ignore the term $e^{-\frac{x^2}{2}}$ and approximate $\log(\pi(x))$

by $\log(\pi(x)) \approx \log(e^{-\frac{(x-10)^2}{2}}) = -\frac{(x-10)^2}{2}$ for big x. Similarly, for negative x with big absolute value,

although $e^{-\frac{x^2}{2}}$ and $e^{-\frac{(x-10)^2}{2}}$ are very small, $e^{-\frac{(x-10)^2}{2}}$ is much smaller than $e^{-\frac{x^2}{2}}$: $\frac{e^{-\frac{(x-10)^2}{2}}}{e^{-\frac{x^2}{2}}} = e^{\frac{x^2 - (x-10)^2}{2}} = e^{10x-50}$. Then we can ignore the term $e^{-\frac{(x-10)^2}{2}}$ and approximate

$\log(\pi(x))$ by $\log(\pi(x)) \approx \log(e^{-\frac{x^2}{2}}) = -\frac{x^2}{2}$ for small x.

We can check the properness of this approximation by the table below:(the computation is in R)

	-25	-3	0	5	6	8	10	12	20
$e^{-\frac{x^2}{2}}$	1.9e-136	1.11e-2	1	3.73e-6	1.52e-8	1.7e-14	1.9e-22	5.3e-32	1.3e-87
$e^{-\frac{(x-10)^2}{2}}$	9.8e-267	2.0e-37	1.9e-22	3.73e-6	3.35e-4	1.35e-1	1	1.35e-1	1.9e-22
$e^{-\frac{x^2}{2}} + e^{-\frac{(x-10)^2}{2}}$	1.2e-136	1.11e-2	1	7.45e-6	3.35e-4	1.35e-1	1	1.35e-1	1.9e-22
$\log(e^{-\frac{x^2}{2}} + e^{-\frac{(x-10)^2}{2}})$	-312.5	-4.5	0	-11.8	-8.00	-2	0	-2	-50
$-\frac{x^2}{2}$	-312.5	-4.5	0	-12.5				-	
$-\frac{(x-10)^2}{2}$				-12.5	-8	-2	0	-2	-50

Table 16 some values for some x

The approximation returns the exact value of $\log(\pi(x))$ computed by R directly except $x \in [4,6]$.

The below is a plot of $\log(\pi(x))$ and its approximation for $x \in [-50,60]$. From table 16 and figure

7, we can see that this is a good approximation of $\log(\pi(x))$ for x with big absolute value.

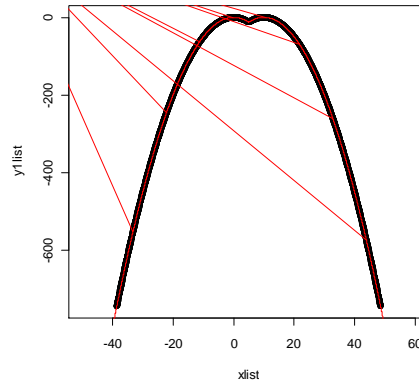


Figure 7 $\log(\pi(x))$ and its approximation (red: $\log(\pi(x))$; black: its approximation)

Let's see how the approximation avoid the numeric error. For case 67:

kernel function: $K(x) = e^{-\frac{|x|^{\alpha_1}}{\alpha_2}}$

$\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 1, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth $b_n = 1$

if we don't use approximation to avoid the numeric error, then the local acceptance rate is as following:

alpha(5.000000) is 0.981800
alpha(7.000000) is 0.615156
alpha(10.000000) is 0.212396
alpha(20.000000) is 0.188700
alpha(30.000000) is 0.137032
alpha(50.000000) is 1.000000
alpha(60.000000) is 1.000000
alpha(70.000000) is 1.000000

Table 17 $\alpha(x)$ for case 67 without approximation

while more accurate results without numeric error is:

alpha(5.000000) is 0.981209
alpha(7.000000) is 0.595073
alpha(10.000000) is 0.183040
alpha(20.000000) is 0.182034
alpha(30.000000) is 0.125577
alpha(50.000000) is 0.081246
alpha(70.000000) is 0.054118

Table 18 $\alpha(x)$ for case 67 with approximation

So in order to avoid numeric error, I use approximation $-\frac{x^2}{2}$ for $x \in (-\infty, 0]$, $-\frac{(x-10)^2}{2}$ for $x \in [10, +\infty)$ and compute the exact value of $\log(\pi(x))$ for $x \in (0, 10)$ in all the 73 cases.

2.4.2 How to Improve the Algorithm Speed

I try different parameter choice and kernel functions, which has 73 cases in total. For most cases, I use approximation II to approximate $\sigma(x)$:

1. Equally divide $[-50, 50)$ into 1000 parts, so we get 1000 small intervals: $[-50, -49.99]$, $[-49.99, -49.98)$ $[49.99, 50)$.
2. Compute and save the values of final $\sigma(x)$ for every left point: -50, -49.99, ..., 49.99.
3. Approximate $\sigma(x)$ by $\sigma(\bullet)$ at the left point of the small interval that x lies in.

This approximation is used to compute $\sigma(x)$ when we generate Markov Chain and test the local acceptance rate.

2.4.3 Output

You can find the output of all the 73 cases in Appendix 2.2. Among all the cases, if we judge from whether the local acceptance rate is close to 0.234, then the best case is probably case 74:

kernel function: $K(x) = e^{-\frac{|x|^{\alpha_1}}{\alpha_2}}$

$\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 2$, $\alpha_2 = 1$, $C = 0.1$, $\gamma = 2$ with fixed bandwidth $b_n = 1$

Check the local acceptance rate for some specific x in case 74.

```
alpha(5.000000) is 0.994617
alpha(7.000000) is 0.626367
alpha(10.000000) is 0.179817
alpha(20.000000) is 0.407404
alpha(30.000000) is 0.364035
alpha(50.000000) is 0.281372
alpha(70.000000) is 0.229441
alpha(120.000000) is 0.146200
alpha(150.000000) is 0.125800
```

Table 19 $\alpha(x)$ for case 74

$\alpha(x)$ is about 0.99, very big for x around the center 5. Then as x goes big, it first decreases to around 0.2 when $x \approx 10$; then increases to around 0.4 when $x \approx 20$; and decreases to 0 as x goes to infinity.

Check the plot of final $g(x)$ and $\sigma(x)$.

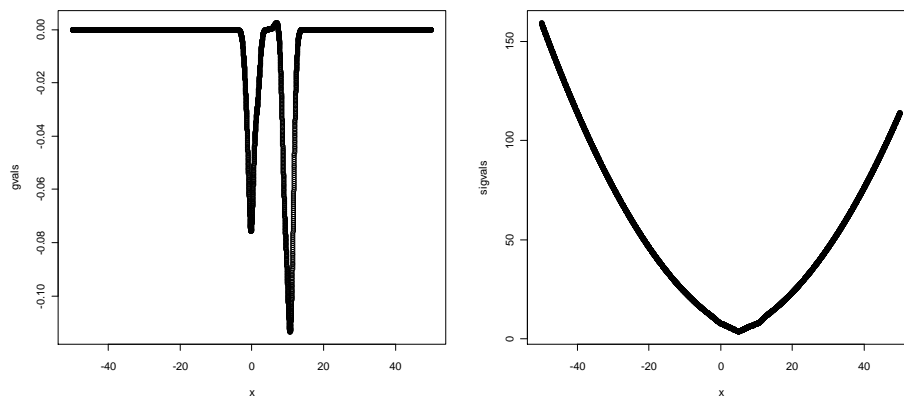


Figure 8 plot of final $g(x)$ and $\sigma(x)$ for case 74

it is reasonable: $g(x)$ has similar shape for $x < 5$ and $x > 5$; it reaches the local minimum at 0 and 10, where we want the sampled x to stay most frequently. $\sigma(x)$ is symmetric around 5. It increases to big values when x goes big. This is also good, because the target distribution is very small for big x , and thereby we don't want the sampled variable to go very big.

2.4.4 How to make $\alpha^{(5)}$ smaller

From table 19, we can see that the adaptation algorithm gives small scale for points around the center and thereby the acceptance rate is very high, nearly 1. So now our main problem we need to fix here is to make $\alpha^{(5)}$ smaller.

In order to make $\alpha^{(5)}$ smaller, we can try the below three changes:

2.4.4.1 change I

Change I:

$$\text{Use } g_n(x) = \begin{cases} g_{n-1}(x) + C_2 * (1 - 0.234) * \frac{\eta_n}{h_n} K\left(\frac{|x - x_n|}{h_n}\right) & \text{if } x_n = y_n \\ g_{n-1}(x) - C_2 * 0.234 * \frac{\eta_n}{h_n} K\left(\frac{|x - x_n|}{h_n}\right) & \text{if } x_n \neq y_n \end{cases}$$

Originally, we subtract the same value when the proposal is rejected as we add when it is accepted. Change I means, we subtract $(1-0.234)$ *some value when the proposal is rejected and add 0.234 *some value when it is accepted. For comparison, check the local acceptance rate for some specific x in case 74 with change I.

alpha(5.000000) is 0.999501
alpha(7.000000) is 0.648121
alpha(10.000000) is 0.191781
alpha(20.000000) is 0.443123
alpha(30.000000) is 0.419608
alpha(50.000000) is 0.338141
alpha(70.000000) is 0.278052
alpha(120.000000) is 0.177113

Table 20 $\alpha(x)$ for case 74 with change I

The local acceptance rates for these x are almost the same as that without change I. $\alpha^{(5)}$ is still too high. It seems that change I doesn't help a lot.

2.4.4.2 change II

Change II: Use $\sigma(x) = e^\beta e^{g(x)} = e^{\beta+g(x)}$

Professor J. Rosenthal suggest that maybe we can cancel the second term in the original $\sigma(x)$ and it becomes: $\sigma(x) = e^\beta e^{g(x)} = e^{\beta+g(x)}$. I call this as change II and implement it in case 74. Now $g(x)$ and $\sigma(x)$ looks like:

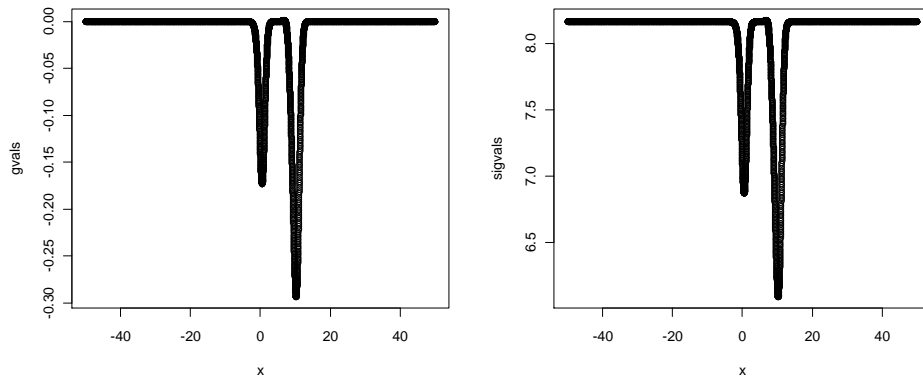


Figure 9 plot of $g(x)$ and $\sigma(x)$ for case 74 with change II

$g(x)$ and $\sigma(x)$ have the same shape now since they only differ in a constant. They are symmetric around 5, and smooth.

Check the local acceptance rate for some specific x in case 74 with change II.

```
alpha(5.000000) is 0.783000
alpha(7.000000) is 0.516100
alpha(10.000000) is 0.199000
alpha(20.000000) is 0.506500
alpha(30.000000) is 0.497500
alpha(50.000000) is 0.499700
alpha(70.000000) is 0.501400
alpha(120.000000) is 0.499400
alpha(150.000000) is 0.502400
```

Table 21 $\alpha(x)$ for case 74 with change II

Now the acceptance rate is around 0.78 for points around the center, smaller than the previous 0.99. But it is still too high if our goal is 0.234. It seems that change II cannot help us to achieve this.

Why does $\alpha(5)$ drop from around 1 to 0.8 ? $g(x)$ doesn't change a lot from comparison of case 74 with and without change II. Obviously, it is because that β is bigger with change II. Without change

II, $\beta \approx 1.366644$; while with change II , $\beta \approx 2.119721$. Why does β become bigger? Let's suppose that β remains the same , then for all the points except 5 $\sigma(x)$ will decrease since $g(x)$ has very small change and $\sigma(x)$ is originally $e^\beta (1+C|x-5|)^\gamma e^{g(x)}$ and now $e^\beta e^{g(x)}$. So the global acceptance rate now rises to a value bigger than 0.234. In order to ensure the global acceptance rate is around 0.234, β has to be pulled up.

2.4.4.3 change III

Although change II, which cancels the second term in $\sigma(x)$, doesn't make $\alpha(5)$ around 0.234, it makes it smaller anyway. This inspires us to fix our current problem by making some changes to the second term $(1+C|x-c|)^\gamma$.

Since I guess the approximate shape of $g(x)$ is reasonable, I want to add the second term as the one whose shape similar to the shape of $g(x)$ in order to keep the shape. So I guess the second term satisfies these:

- For $x=5$, it reaches local maximum to make a bigger scale and therefore a smaller acceptance rate.
- For $x=0,10$, it reaches local minimum to make a smaller scale and therefore a bigger acceptance rate.

I try the second term as:

$$C_1 |x-5|^{\alpha_1} (I(|x-5|>5) - I(|x-5|\leq 5)) + 2C_1 * 5^{\alpha_1} I(|x-5|\leq 5) + C_2$$

$$i.e. \begin{cases} -C_1 |x-5|^{\alpha_1} + 2C_1 * 5^{\alpha_1} + C_2 & \text{if } |x-5|\leq 5 \\ C_1 |x-5|^{\alpha_1} + C_2, & \text{if } |x-5|>5 \end{cases}$$

So now,

$$\sigma(x) = e^{\beta_n + g_n(x)} * (C_1 |x-5|^{\alpha_1} (I(|x-5|>5) - I(|x-5|\leq 5)) + 2C_1 * 5^{\alpha_1} I(|x-5|\leq 5) + C_2)$$

$$i.e. \sigma(x) = \begin{cases} e^{\beta_n + g_n(x)} * (-C_1 |x-5|^{\alpha_1} + 2C_1 * 5^{\alpha_1} + C_2) & \text{if } |x-5|\leq 5 \\ e^{\beta_n + g_n(x)} * (C_1 |x-5|^{\alpha_1} + C_2), & \text{if } |x-5|>5 \end{cases}$$

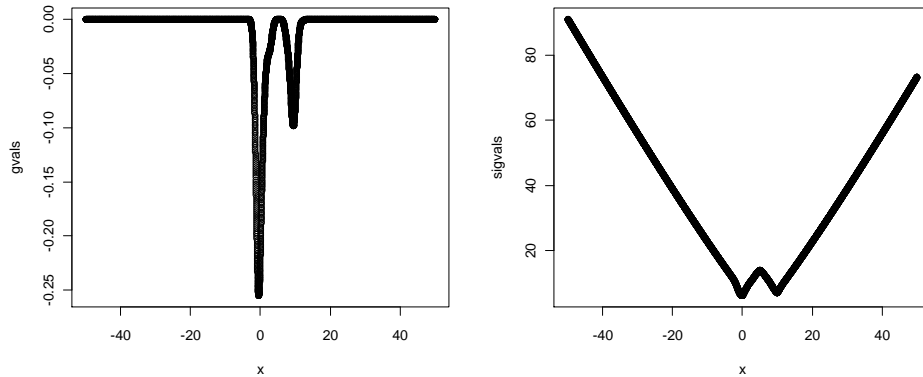
I try several choices of C_1 , C_2 , α_1 and finally choose $C_1 = 0.1$, $C_2 = 0.1$, $\alpha_1 = 1.1$. The current local acceptance rate is as table 22:

alpha(5.000000) is 0.550000
alpha(8.000000) is 0.302000
alpha(10.000000) is 0.176000
alpha(20.000000) is 0.411000
alpha(50.000000) is 0.379000
alpha(70.000000) is 0.389000

Table 22 $\alpha(x)$ for case 74 with change III

Now $\alpha(5)$ drops from 0.99 to around 0.55. Good. But it cannot make $\alpha(x)$ all around 0.234.

The final $g(x)$ and $\sigma(x)$ are as figure 10:

**Figure 10** $g(x)$ and $\sigma(x)$ for case 74 with change III

$g(x)$ have similar shape for $x < 5$ and $x > 5$; $\sigma(x)$ are symmetric around 5, and smooth. It reaches minimum at 0 and 10, where we want the sampled variable to stay most frequently. good.

The above three changes cannot make $\alpha(x)$ all round 0.234 for different x . It indicates that this adaptation algorithm might not achieve this goal for this target distribution, however we choose the parameters and functions.

2.5 Example 3: mixture of three normal distributions in \mathcal{R}^1

Let's try one more difficult target distribution:

$$\pi(x) = C * \left(0.3 * e^{-\frac{(x+10)^2}{2}} + 0.4 * e^{-\frac{x^2}{2}} + 0.3 * e^{-\frac{(x-10)^2}{2}} \right)$$

This is a mixture of three normal distributions: $N(0,1)$, $N(-10,1)$, $N(10,1)$.

Figure 11 shows the un-normalized part of this density. It is symmetric around 0, reaches the maximum at 0, and local maximum at 10 and -10.

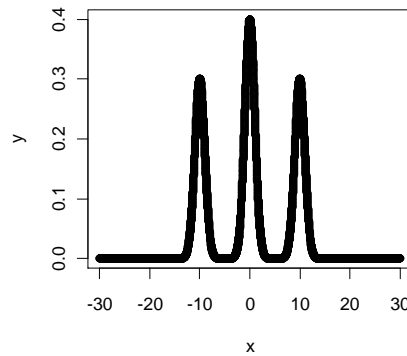


Figure 11 un-normalized part of the density

Now the proposed scale becomes: $\sigma(x) = e^{\beta+g(x)}(1+C*|x+5|*|x-5|)^{\gamma}$

Where the definition of parameters and functions are the same as the above examples.

2.5.1 How to Avoid Numerical Error

Similar to section 2.4.1, numerical error still exists if we compute $\log(\pi(x))$ directly for big x. In order to avoid this error, I will use approximation as the previous.

I will approximate it for x with big absolute value. For big x, $0.3 * e^{-\frac{(x+10)^2}{2}}$, $0.4 * e^{-\frac{x^2}{2}}$, $0.3 * e^{-\frac{(x-10)^2}{2}}$ will be small, and $0.3 * e^{-\frac{(x-10)^2}{2}}$ is much bigger than the other two:

$$\frac{0.3 * e^{-\frac{(x-10)^2}{2}}}{0.4 * e^{-\frac{x^2}{2}}} = 0.75 * e^{\frac{x^2}{2} - \frac{(x-10)^2}{2}} = 0.75 * e^{10x-50} \quad \text{and} \quad \frac{0.3 * e^{-\frac{(x-10)^2}{2}}}{0.3 * e^{-\frac{(x+10)^2}{2}}} = e^{\frac{(x+10)^2}{2} - \frac{(x-10)^2}{2}} = e^{20x}$$

So we can approximate it by $\log\left(0.3 * e^{-\frac{(x-10)^2}{2}}\right) = \log(0.3) - \frac{(x-10)^2}{2}$ for big x. Similarly, for negative x with big absolute value, we can approximate it by $\log\left(0.3 * e^{-\frac{(x+10)^2}{2}}\right) = \log(0.3) - \frac{(x+10)^2}{2}$. i.e.

$$\log(\pi(x)) = \begin{cases} \log(0.3) - \frac{(x+10)^2}{2} & \text{if } x < -12 \\ \log\left(0.3 * e^{-\frac{(x+10)^2}{2}} + 0.4 * e^{-\frac{x^2}{2}} + 0.3 * e^{-\frac{(x-10)^2}{2}}\right) & \text{if } -12 \leq x \leq 12 \\ \log(0.3) - \frac{(x-10)^2}{2} & \text{if } x > 12 \end{cases}$$

Below is the comparison of $\log(\pi(x))$ and its approximation for $x \in [-50, 50]$. They overlap very well. Good. In all the 65 cases I try, I will use this approximation to compute $\log(\pi(x))$ in order to avoid numerical error.

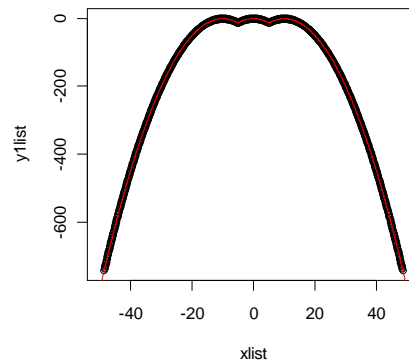


Figure 12 $\log(\pi(x))$ and its approximation

2.5.2 Output

Among the 65 cases, the best case to give the acceptance rate close to 0.234 is case 133:

$\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 1, \alpha_2 = 1, C = 10, \gamma = 0.5$ with fixed bandwidth $b_n = 1$. It has $\alpha(x)$ as table 23:

alpha(0.000000)	is 0.207800
alpha(5.000000)	is 0.696900
alpha(10.000000)	is 0.166300
alpha(15.000000)	is 0.354600
alpha(20.000000)	is 0.365400
alpha(50.000000)	is 0.357700
alpha(100.000000)	is 0.363600
alpha(150.000000)	is 0.363200

Table 23 $\alpha(x)$ for case 133

$\alpha(x)$ is around 0.2 at 0, first increases to around 0.7 when $x \sim 5$, then decreases to around 0.16 when $x \sim 10$, and then increases to around 0.36 when x goes to infinity.

The other 64 cases have similar tendency of $\alpha(x)$, which you check from Appendix 2.4.1~2.4.4.

Figure 13 is the plot of the final $g(x)$ and $\sigma(x)$.

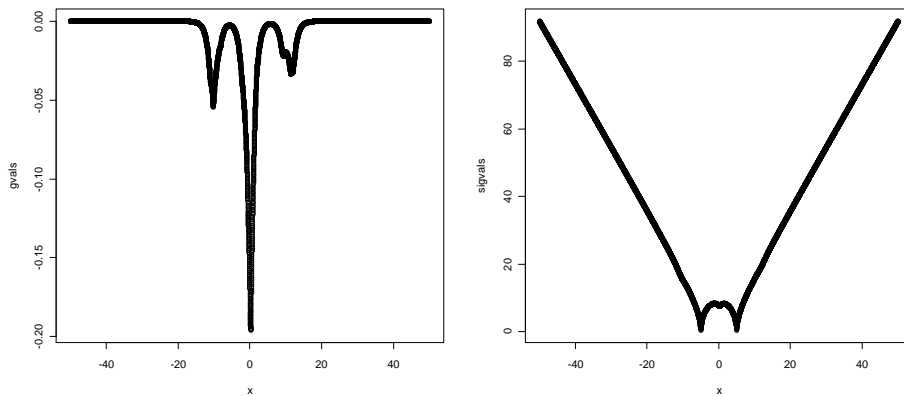


Figure 13 $g(x)$ and $\sigma(x)$ for case 133

$g(x)$ and $\sigma(x)$ are symmetric around 0. $\sigma(x)$ reaches the local minimum at 0, where we want the sampled variables to say around the most. Good.

2.5.3 How to choose γ

When $\gamma \leq 1$, the final $g(x)$ and $\sigma(x)$ is symmetric around 0. The generated Markov Chain has a good mixing. When $\gamma > 1$, the final $g(x)$ is not symmetric around 0, bad. The generated Markov Chain has small steps and always gets stuck at 0, or 10, or -10. Table 24 shows the plot of $g(x)$ and $\sigma(x)$ for $\gamma = 0.5$, $\gamma = 1$, $\gamma = 1.5$, $\gamma = 2$. In these four cases, Kennel

function: $K(x) = e^{-\frac{|x|^{\alpha_1}}{\alpha_2}}$, $\alpha_1 = 1, \alpha_2 = 1, C = 1, \gamma = 2$, height = 0.5, width = 0.5, $C = 1, \gamma = 2$ and $b_n = 1$. Table 25 shows the generated Markov Chain for the four cases

$\gamma = 0.5$	$\gamma = 1$
----------------	--------------

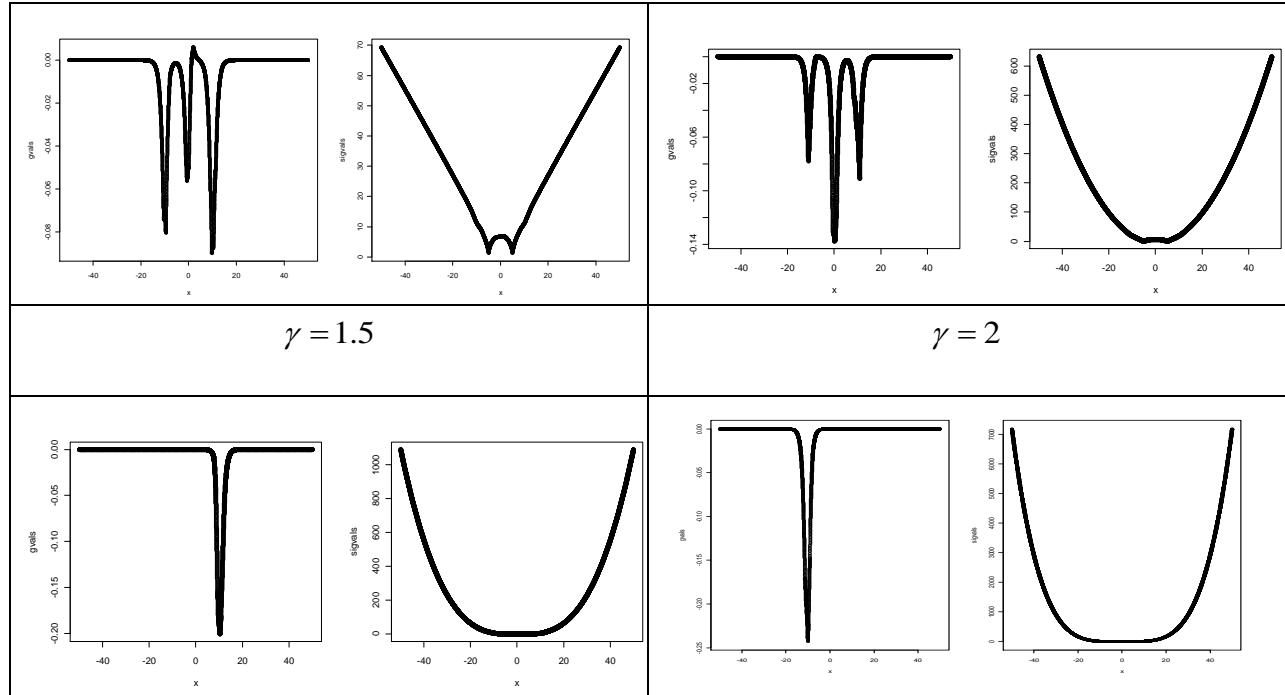


Table 24 plot of $g(x)$ and $\sigma(x)$ for the four cases

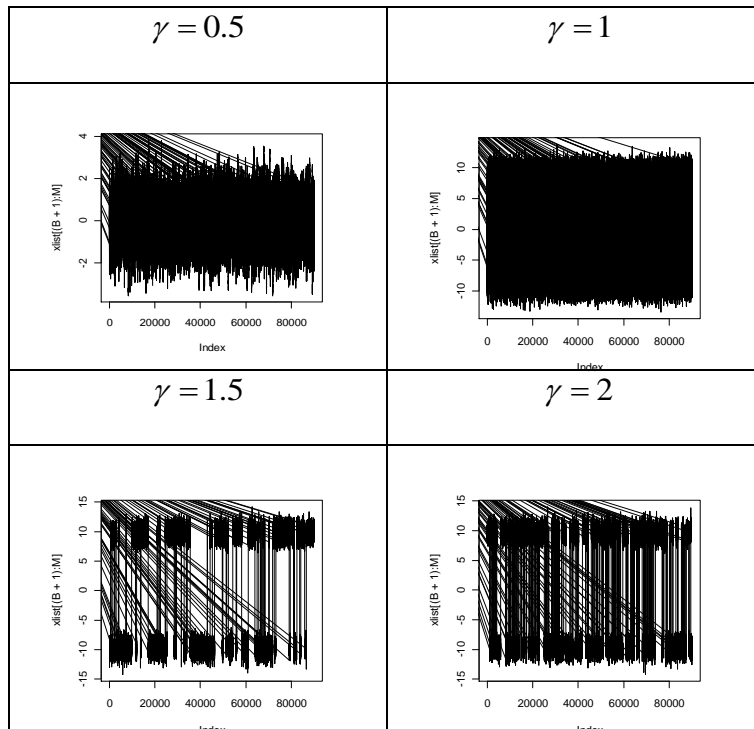


Table 25 generated Markov Chain for the four cases

From the above two tables, we can see that we need to choose $\gamma \leq 1$ to make the Markov Chain have a good mixing.

2.6 Example 4: normal distribution in R^2

Now consider the target distribution as :

$$f(x) = C * e^{-\frac{x^2}{2}} * e^{-\frac{y^2}{2}} \propto e^{-\frac{x^2 + y^2}{2}}$$

This is Standard Normal distribution in R^2 and the logarithm of the density is:

$$\log(f(x)) = -\frac{x^2}{2} - \frac{y^2}{2}$$

Now the adaptation algorithm updates scale as:

$$\sigma(X) = e^{\beta + g(X)} (1 + C |X - (0,0)|)^\gamma,$$

Where X is 2 dimensional vector: $X = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$, β , γ , $g(x)$ is updated adaptively. $|\bullet|$ is the distance in R^2 rather than absolute value.

2.6.1 How to learn from Example 1

Now our target distribution is Standard Normal distribution in R^2 , i.e. $X = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} \sim N\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\right)$.

This means $X_1 \sim N(0,1)$, $X_2 \sim N(0,1)$, and they are independent. So we can learn how to choose parameters from Example 1: Standard Normal distribution in R^1 . We can first consider all the good cases in Example 1, and then use the parameter choice in those cases to run the adaptation algorithm for this example.

From the following discussion in Chapter 3, we will know for Example 1, the best case (in judge of varfact, variance, and average squared jump distance) is case 57 and 65:

$$\text{Kernel function : } K(x) = \begin{cases} 0, & |x| \geq 2 * \text{width} \quad \text{or} \quad |x| \leq \text{width} \\ 1, & \text{width} < |x| < 2 * \text{width} \end{cases}$$

$$\text{Case 57: } \eta_n = \frac{1}{(n+5)^{0.5}}, \text{ width} = 5, C = 0.01, \gamma = 2 \text{ with fixed bandwidth } b_n = 1$$

I will try these two parameter choices . And also I will try the parameters in some other cases, which has the smallest varfact, variance, or biggest average squared jump distance for each kernel function.

You can check Appendix 2.4 for the output of all the 9 cases I try.

2.6.2 How to Improve the algorithm speed

We already see how slow C generates Markov Chain and test the local acceptance rate without approximation II. The speed will become much worse in two-dimensional space. So in order to improve the speed, I will use Approximation II again in this example:

1. Equally divide $[-30,30) * [-30,30)$ into 810000 parts, so we get 810000 small squares:

$$[-30, -30 + \frac{1}{15}) * [-30, -30 + \frac{1}{15}) \quad , \quad [-30, -30 + \frac{1}{15}) * [-30, -30 + \frac{2}{15}) \quad \dots$$

$$[-30, -30 + \frac{1}{15}) * [30, 30 - \frac{1}{15}) ,$$

$$[-30 + \frac{1}{15}, -30 + \frac{2}{15}) * [-30, -30 + \frac{1}{15}) \quad , \quad [-30 + \frac{1}{15}, -30 + \frac{2}{15}) * [-30, -30 + \frac{2}{15}) \quad \dots$$

$$[-30 + \frac{1}{15}, -30 + \frac{2}{15}) * [30, 30 - \frac{1}{15}) ,$$

....

$$[30 - \frac{1}{15}, 30) * [-30, -30 + \frac{1}{15}) , [30 - \frac{1}{15}, 30) * [-30, -30 + \frac{2}{15}) \quad \dots \quad [30 - \frac{1}{15}, 30) * [30, 30 - \frac{1}{15}) ,$$

2. Compute and save the values of final $\sigma(x)$ for every lower-left point:

$$(-30, -30), (-30, -30 + \frac{1}{15}), \dots (-30, 30 - \frac{1}{15}), \dots$$

$$(-30 + \frac{1}{15}, -30), (-30 + \frac{1}{15}, -30 + \frac{1}{15}), \dots (-30 + \frac{1}{15}, 30 - \frac{1}{15}),$$

.....

$$(30 - \frac{1}{15}, -30), (30 - \frac{1}{15}, -30 + \frac{1}{15}), \dots (30 - \frac{1}{15}, 30 - \frac{1}{15}) .$$

3. Approximate $\sigma(x)$ by $\sigma(\bullet)$ at the lower-left point of the small square that x lies in.

For case 194, I run the adaptation algorithm, generate Markov Chain and test $\alpha(x)$ both with and without approximation II. Table 26 shows the speed of both:

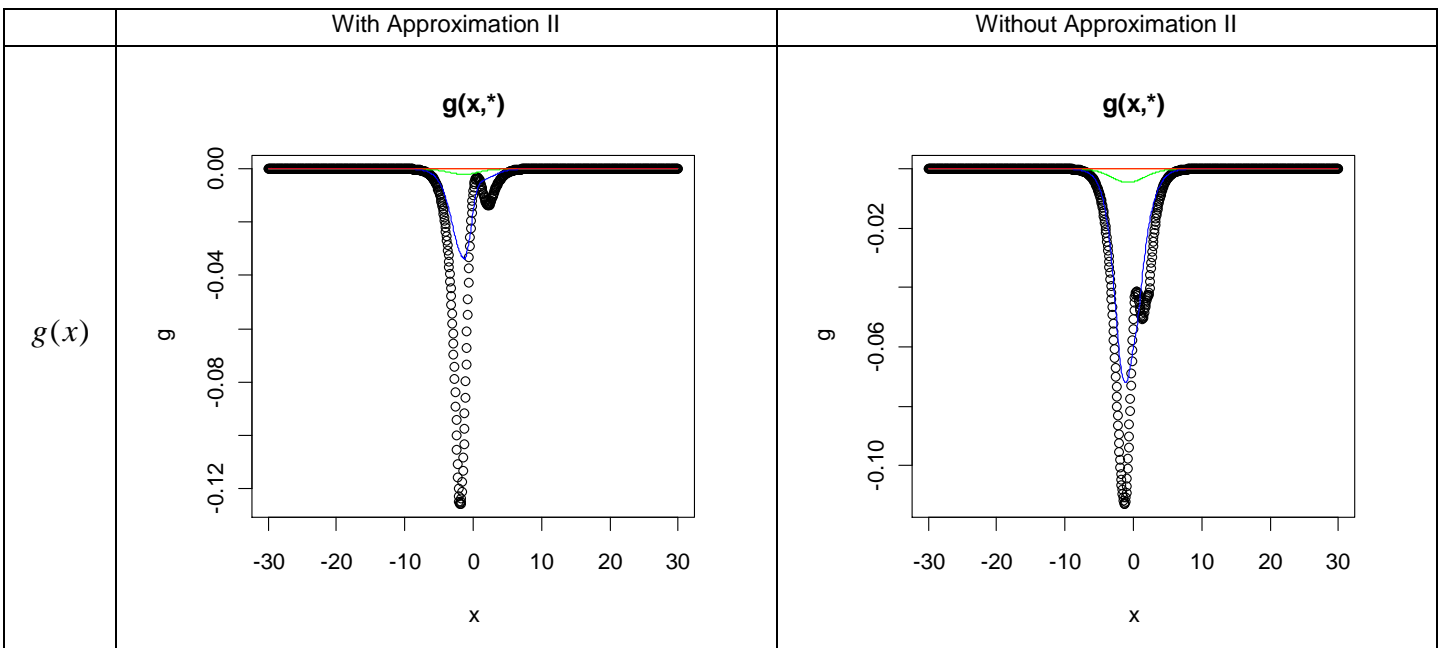
With Approximation II	Without Approximation II
-----------------------	--------------------------

it takes 310.314 seconds to run the adaptation algorithm	it takes 296.75 seconds to run the adaptation algorithm
it takes 9939.71 seconds to plot g and sigma	it takes 10001.9820 seconds to plot g and sigma
it takes 512.742 seconds to generate the first Markov Chain	it takes 3674.517 seconds to generate the first Markov Chain
it takes 471.932 seconds to generate second Markov Chain	it takes 3806.229 seconds to generate second Markov Chain
it takes 496.096 seconds to generate third Markov Chain	it takes 3813.156 seconds to generate the third Markov Chain
it takes 514.006 seconds to generate fourth Markov Chain	it takes 3638.899 seconds to generate fourth Markov Chain
it takes 468.624 seconds to generate the fifth Markov Chain	it takes 3640.485 seconds to generate the fifth Markov Chain
it takes 2030.625 seconds to test the local acceptance	it takes 2712.386 seconds to test the local acceptance

Table 26 running time for case 194 with and without Approximation II

It takes 3600~3800 seconds to generate a Markov Chain without Approximation II, while now it only needs around 500 seconds to do this with Approximation II, good. This approximation also improves the speed of testing the local acceptance rate by about 700 seconds.

Approximation II is good for this example too, for it gives similar plot of final $g(x)$ and $\sigma(x)$, and local acceptance rate for some specific x , which is shown in table 27. (black: $x=0$; blue: $x=-2$; green: $x=-5$; yellow: $x=-10$; red: $x=-30$). I will this approximation for the other cases.



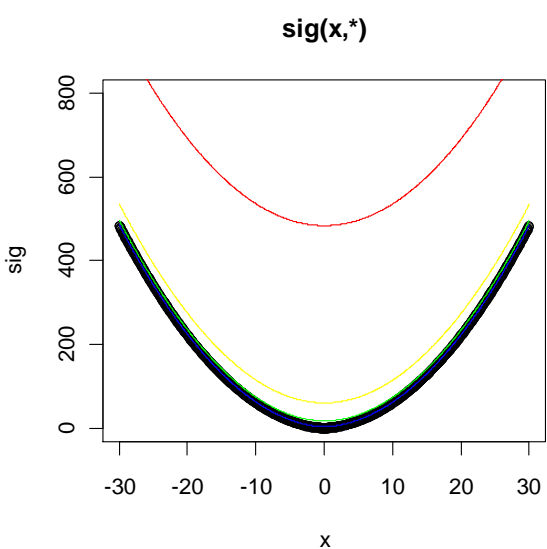
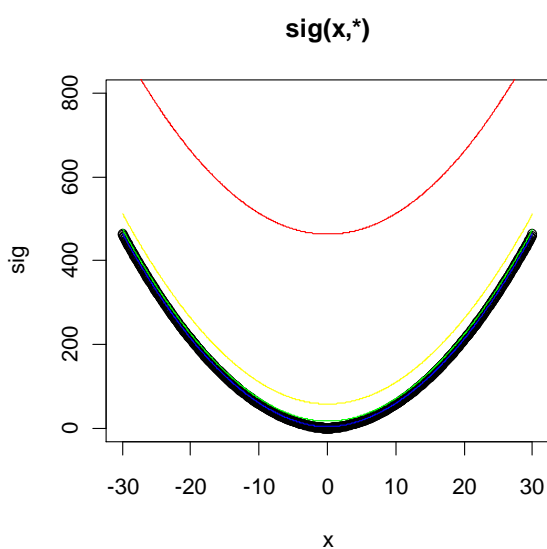
$\sigma(x)$		
$\alpha(x)$	<p>alpha(0.000000,0.000000) is about 0.677500 alpha(0.000000,2.000000) is about 0.137900 alpha(5.000000,5.000000) is about 0.025100 alpha(2.000000,10.000000) is about 0.011400 alpha(10.000000,10.000000) is about 0.007900 alpha(20.000000,50.000000) is about 0.000900 alpha(50.000000,50.000000) is about 0.000400</p>	<p>alpha(0.000000,0.000000) is about 0.758800 alpha(0.000000,2.000000) is about 0.138100 alpha(5.000000,5.000000) is about 0.024000 alpha(2.000000,10.000000) is about 0.015000 alpha(10.000000,10.000000) is about 0.007900 alpha(20.000000,50.000000) is about 0.000700 alpha(50.000000,50.000000) is about 0.000200</p>

Table 27 output for case 194 with and without Approximation II

2.6.3 Output

if judged from whether it makes $\alpha(x)$ around 0.234, then probably the best case is case 196 :

Kernel function: $K(x) = e^{-\frac{|x|^{\alpha_1}}{\alpha_2}}$

$\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 0.5, \alpha_2 = 1, C = 0.1, \gamma = 2$ with fixed bandwidth $b_n = 1$

It gives local acceptance rate as table 28 :

alpha(0.000000,0.000000) is about 0.215700
alpha(0.000000,2.000000) is about 0.285000
alpha(5.000000,5.000000) is about 0.328900
alpha(2.000000,10.000000) is about 0.335800
alpha(10.000000,10.000000) is about 0.319200
alpha(20.000000,50.000000) is about 0.157600
alpha(50.000000,50.000000) is about 0.124300

Table 28 $\alpha(x)$ **for case 197**

The tendency of $\alpha(x)$ is quite similar to 1-dimensional space. As the distance of x and 0 goes bigger, $\alpha(x)$ first increases from around 0.2 to 0.3, then decreases to 0. But all the simulations until now suggests that it may not be possible to make $\alpha(x)$ around 0.234 for different x .

The plot of final $g(x)$ and $\sigma(x)$ are as figure 14.

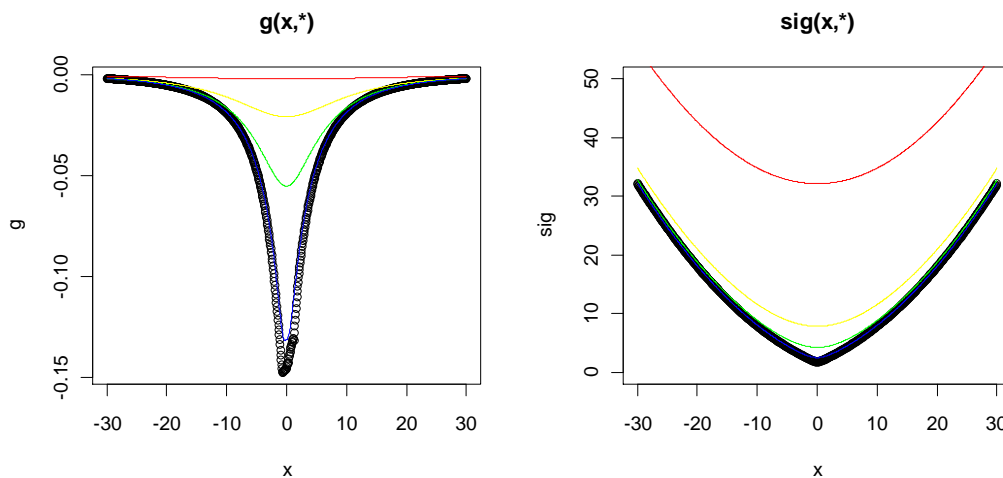


Figure 14 $g(x)$ and $\sigma(x)$ **for case 197**

(Black: $x=0$; blue: $x=-2$; green: $x=-5$; yellow: $x=-10$; red: $x=-30$)

Both are symmetric around 0 and smooth, good. $\sigma(x)$ is the smallest at 0, where we want the sampled variable to stay most frequently, good.

When x goes from 0 to -30, i.e. the black line to the red, $g(x)$ is more and more flat, and $\sigma(x)$ for each fixed point is increasing. This is good, because for every single variable, we want it to go to 0

More frequently than other values.

When x goes from 0 to -30, i.e. The black line to the red, $g(x)$ is more and more flat, and for each fixed point is increasing. This is good, because for every single variable, we want it to go to 0 more frequently than other values.

From Example 1~4, simulation studies indicates that probably it is impossible to make $\alpha(x)$ around 0.234 for different x .

Since we cannot make the local acceptance rate $\alpha(x)$ around 0.234 or all different x , so far we don't get any conclusion about whether this is a better idea than the current optimal choice. It still remains to be evaluated whether the adaptation algorithm improves the efficiency of RWM.

Now let's go one step back. Even if we cannot achieve the goal to make all $\alpha(x)$ around 0.234, but during our efforts to achieve this, the efficiency still improves in comparison without adaption algorithm. So I will use different measures to check this in Chapter 3.

3. Efficiency Comparison of Adaptive and Non-adaptive Algorithm

3.1 Efficiency Measures to Evaluate Markov Chain

There are many different measures to evaluate the efficiency of Markov Chain. Some usual measures are varfact, variance/ standard error, and average squared jump distance.

3.1.1 Varfact

3.1.1.1 Autocorrelation^[4]

Autocorrelation of a random process describes the correlation between values of the process at different points in time, as a function of the two times or of the time difference. Let X be some repeatable process, and i be some point in time after the start of that process. Then X_i is the value (or realization) produced by a given run of the process at time i . Suppose that the process is further known to have defined values for mean μ_i and variance σ_i^2 for all times i . Then the definition of the autocorrelation between times s and t is

$$R(s, t) = \frac{E((X_t - E(X_t)) * (X_s - E(X_s)))}{\sqrt{\text{var}(X_t) * \text{var}(X_s)}}$$

If the function R is well-defined, its value must lie in the range $[-1, 1]$, with 1 indicating perfect correlation and -1 indicating perfect anti-correlation.

If X_t is a second-order stationary process then the mean μ and the variance σ^2 are time-independent, and further the autocorrelation depends only on the difference between t and s : the correlation depends only on the time-distance between the pair of values but not on their position in time. This further implies that the autocorrelation can be expressed as a function of the time-lag, and that this

would be an even function of the lag $\tau = s - t$. This gives the more familiar form

$$R(k) = \frac{E((X_t - \mu)^*(X_{t+k} - \mu))}{\sigma^2} = \frac{E((X_0 - \mu)^*(X_k - \mu))}{\sigma^2} = \text{corr}(X_0, X_k)$$

3.1.1.2 Varfact ^[5]

Varfact is integrated autocorrelation time, defined as:

$$\text{varfact} = 1 + 2 \sum_{k=1}^{\infty} R(k) = 1 + 2 \sum_{k=1}^{\infty} R(0, k) = 1 + 2 \sum_{k=1}^{\infty} \text{corr}(X_0, X_k)$$

It is used to measure the autocorrelation of a Markov Chain. We can use it times the i.i.d variance to

estimate the variance($v = \text{var} \left(\frac{\sum_{i=B+1}^M h(X_i)}{M-B} \right)$), i.e. Uncertainty, where h is the function we are interested in.

$$\begin{aligned} v &\approx \mathbf{E}_{\pi} \left[\left(\left(\frac{1}{M-B} \sum_{i=B+1}^M h(X_i) \right) - \mathbf{E}_{\pi}(h) \right)^2 \right] = \mathbf{E}_{\pi} \left[\left(\frac{1}{M-B} \sum_{i=B+1}^M \bar{h}(X_i) \right)^2 \right] \\ &= \frac{1}{(M-B)^2} \left[(M-B) \mathbf{E}_{\pi}(\bar{h}(X_i)^2) + 2(M-B-1) \mathbf{E}_{\pi}(\bar{h}(X_i) \bar{h}(X_{i+1})) \right. \\ &\quad \left. + 2(M-B-2) \mathbf{E}_{\pi}(\bar{h}(X_i) \bar{h}(X_{i+2})) + \dots \right] \\ &\approx \frac{1}{M-B} \left(\mathbf{E}_{\pi}(\bar{h}^2) + 2 \mathbf{E}_{\pi}(\bar{h}(X_i) \bar{h}(X_{i+1})) + 2 \mathbf{E}_{\pi}(\bar{h}(X_i) \bar{h}(X_{i+2})) + \dots \right) \\ &= \frac{1}{M-B} \mathbf{E}_{\pi}(\bar{h}^2) \left(1 + 2 \text{Corr}_{\pi}(\bar{h}(X_i), \bar{h}(X_{i+1})) + 2 \text{Corr}_{\pi}(\bar{h}(X_i), \bar{h}(X_{i+2})) + \dots \right) \\ &\equiv \frac{1}{M-B} \text{Var}_{\pi}(h) (\text{varfact}) = (\text{iid variance}) (\text{varfact}), \end{aligned}$$

Usually, in order to compute varfact, we don't sum over all k, just, say, until $R(0, k) < 0.05$.

3.1.2 variance

Suppose P1 and P2 are two Markov Chains, each with the same stationary distribution $\pi(x)$. Then

we say P1 has smaller variance than P2 if $\text{var} \left(\frac{\sum_{i=1}^n h(X_i)}{n} \right)$ is smaller when $\{X_n\}$ follows P1 than it

follows P2.

If Markov Chain $\{X_n\}$ is stationary, then for large n , $\text{var}\left(\frac{\sum_{i=1}^n h(X_i)}{n}\right) \approx \frac{1}{n} \text{var}(h) * \text{varfact}(h)$,

3.1.3 average squared jump distance

Markov chain is better if it allows for faster exploration of the state space. We say P1 mixes faster than P2 if $E[(X_n - X_{n-1})^2]$ (average squared jump distance) is larger under P1 than P2. Of course,

$E[(X_n - X_{n-1})^2]$ can be estimated by $\frac{\sum_{i=B+1}^M (X_i - X_{i-1})^2}{M - B}$.

3.1.4 Relationship between the three measures

1) Since $\text{var}\left(\frac{\sum_{i=1}^n h(X_i)}{n}\right) \approx \frac{1}{n} \text{var}(h) * \text{varfact}(h)$ it is obvious that the bigger the varfact, the bigger variance

2) We can see that:

$$\begin{aligned}
 & \text{average squared jump distance} \\
 & \approx \frac{\sum_{i=B+1}^M (X_i - X_{i-1})^2}{M - B} \\
 & = \frac{\sum_{i=B}^M [(X_i - E(X_i)) - (X_{i-1} - E(X_{i-1}))]^2}{M - B} \\
 & = \frac{\sum_{i=B+1}^M [(X_i - E(X_i))]^2 + [(X_{i-1} - E(X_{i-1}))]^2 - 2[(X_i - E(X_i))][(X_{i-1} - E(X_{i-1}))]}{M - B} \\
 & = \text{const} \tan t - \frac{2}{M - B} \sum_{i=B+1}^M \text{corr}(X_i, X_{i-1}) \\
 & \approx \text{const} \tan t - \frac{\text{var fact} - 1}{M - B} \\
 & = \text{const} \tan t - \frac{\text{var fact}}{M - B}
 \end{aligned}$$

From this, we get the smaller varfact, the bigger average squared jump distance.

So for the efficiency of Markov Chain, we want smaller varfact, smaller variance, and bigger averaged jump distance. Now I will use the three measures to compare that whether the adaptation algorithm improves the efficiency of Markov Chain than non-adaption algorithm.

3.2 Example 1: Normal Distribution in R^1

For all the 66 different cases, I generate 5 independent Markov Chains for each case and then compute their varfact, variance and average squared jump distance. You can refer to Appendix 2.2.6~2.2.8 for output of each case. In the below discussion, I will list the best cases, i.e. Having the smallest varfact or variance, or having the biggest average squared jump distance.

3.2.1 varfact

For the four kernel functions, table 29 shows the best case who has smaller varfact than others with the same kernel function.

kernel function $K(x) = e^{-\frac{ x ^{\alpha_1}}{\alpha_2}}$										
α_1	C	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	
2	0.1	2	$\frac{1}{(n+5)^{0.5}}$	1	6.533223	6.42219	6.045298	6.83526	6.91115	
kernel function $K(x) = \frac{1}{1 + \alpha_1 x ^{\alpha_2}}$										
α_1	C	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	
1	0.1	1	$\frac{1}{(n+5)^{0.5}}$	1	7.185476	6.92573	6.877931	7.298966	7.185476	
kernel function $K(x) = \begin{cases} 0, & x \geq 2 * width \\ -1 * height, & width < x < 2 * width \\ 1 * height, & x \leq width \end{cases}$										
C	γ	height	width	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run
0.01	10	0.5	0.5	$\frac{1}{(n+5)^{0.5}}$	1	5.84989	6.229419	5.915948	6.139981	6.5259
Kernel function $K(x) = \begin{cases} 0, & x \geq 2 * width \text{ or } x \leq width \\ 1, & width < x < 2 * width \end{cases}$										
C	width	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	

0.0 1	5	2	$\frac{1}{(n+5)^{0.5}}$	1	5.72553	5.958359	5.350588	5.822084	5.500506
constant $\sigma(x) = C$					6.032679	6.428601	6.010923	6.143806	6.654467

Table 29 varfact

Among the four best cases, the last one, case 57: kernel function:

$$K(x) = \begin{cases} 0, & |x| \geq 2 * width \text{ or } |x| \leq width \\ 1, & width < |x| < 2 * width \end{cases}$$

$$\eta_n = \frac{1}{(n+5)^{0.5}}, \text{ width} = 5, C = 0.01, \gamma = 2 \text{ with fixed bandwidth } b_n = 1$$

has the smallest varfact, about 5.7~5.8. This is a little smaller than non-adaption algorithm, which is around 6.0.

3.2.2 variance/ standard error

Now compare the variance. For each kernel function, I choose the best one with the smallest variance from all the cases I try.

kernel function $K(x) = e^{-\frac{ x ^{\alpha_1}}{\alpha_2}}$										
α_1	C	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	
2	0.1	2	$\frac{1}{(n+5)^{0.5}}$	1	0.008631	0.008396	0.008139	0.008740	0.008697	
kernel function $K(x) = \frac{1}{1 + \alpha_1 x ^{\alpha_2}}$										
α_1	C	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	
1	0.1	10	$\frac{1}{(n+5)^{0.5}}$	1	0.008920	0.008758	0.00868	0.008971	0.009006	
kernel function $K(x) = \begin{cases} 0, & x \geq 2 * width \\ -1 * height, & width < x < 2 * width \\ 1 * height, & x \leq width \end{cases}$										
C	γ	height	width	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run

0.01	10	0.5	0.5	$\frac{1}{(n+5)^{0.5}}$	1	0.008052	0.008275	0.008178	0.008147	0.008477
Kernel function $K(x) = \begin{cases} 0, & x \geq 2 * width \text{ or } x \leq width \\ 1, & width < x < 2 * width \end{cases}$										
C	width	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	
0.01	5	2	$\frac{1}{(n+5)^{0.5}}$	1	0.007925	0.008171	0.007620	0.00800	0.007780	
constant $\sigma(x) = C$					0.008154	0.008419	0.008097	0.008276	0.008502	

Table 30 variance

Among the four best cases, case 57 still performs the best. It has the smallest variance, around 0.008, a little smaller than non-adaptation algorithm, which has the variance of about 0.0082~0.0084.

3.2.3 average squared jump distance

For each kernel function, I choose the best one with the biggest average squared jump distance from all the cases I try.

kernel function $K(x) = e^{-\frac{ x ^{\alpha_1}}{\alpha_2}}$										
α_1	C	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	
2	0.1	2	$\frac{1}{(n+5)^{0.5}}$	1	0.545549	0.525287	0.520330	0.535221	0.530106	
kernel function $K(x) = \frac{1}{1 + \alpha_1 x ^{\alpha_2}}$										
α_1	C	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	
1	0.1	0.1	$\frac{1}{(n+5)^{0.5}}$	10	0.561479	0.54956	0.584170	0.563489	0.569779	
kernel function $K(x) = \begin{cases} 0, & x \geq 2 * width \\ -1 * height, & width < x < 2 * width \\ 1 * height, & x \leq width \end{cases}$										
C	γ	height	width	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run
0.01	10	0.5	0.5	$\frac{1}{(n+5)^{0.5}}$	1	0.57572	0.56505	0.585138	0.546282	0.555975

Kernel function $K(x) = \begin{cases} 0, & x \geq 2 * width \text{ or } x \leq width \\ 1, & width < x < 2 * width \end{cases}$									
C	width	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run
0.0 1	5	2	$\frac{1}{(n+5)^{0.5}}$	$\frac{1}{n^{0.2}}$	0.611305	0.597235	0.604059	0.616791	0.610068
constant $\sigma(x) = C$					0.553845	0.554989	0.537098	0.56998	0.547339

Table 31 average squared jump distance

From table 12, we can see that case 65:

$$\eta_n = \frac{1}{(n+5)^{0.5}}, \text{ width} = 5, C = 0.01, \gamma = 2 \text{ with decreasing bandwidth } b_n = \frac{1}{n^{0.2}}$$

has the biggest average squared jump distance, about 0.60~0.61, a little bigger than 0.54~0.56.

So if measured in variance and varfact, case 57 is the best; if measured in average squared jump distance, case 65 is the best. Table 15 shows the comparison of case 57, 65 and non-adaptation algorithm.

	varfact	variance	average sq. distance
Case 57	5.4~5.8	0.0077~0.0080	0.54~0.55
Case 65	7.7~7.9	0.0092~0.0095	0.60~0.61
$\sigma(x) = C$	6.0~6.6	0.0081~0.0085	0.54~0.56

Table 32 comparison of the best three cases

This table shows that adaptive algorithm performs a little better than non-adaptation algorithm, but this advantage is not significant.

3.3 Example 2: Mixture of Two Normal Distributions in R^1

I try 62 kinds of parameter choice and kernel functions to implement the adaptation algorithm; and then generate 5 Markov Chains according to the final $\sigma(x)$. You can check the varfact, variance and average squared jump distance of them in Appendix 2.3.6~2.3.8.

3.3.1 varfact

For the four kernel functions, the below four cases have the smaller varfact than others with the same kernel functions.

kernel function $K(x) = e^{-\frac{ x ^{\alpha_1}}{\alpha_2}}$										
α_1	C	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	
2	0.1	2	$\frac{1}{(n+5)^{0.5}}$	10	12.86394	12.7031	11.99687	12.00228	12.88139	
kernel function $K(x) = \frac{1}{1 + \alpha_1 x ^{\alpha_2}}$										
α_1	C	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	
10	0.1	10	$\frac{1}{(n+5)^{0.5}}$	1	13.58733	12.41142	12.40737	13.24543	12.94998	
kernel function $K(x) = \begin{cases} 0, & x \geq 2 * width \\ -1 * height, & width < x < 2 * width \\ 1 * height, & x \leq width \end{cases}$										
C	γ	height	width	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run
0.1	0.1	1	0.1	$\frac{1}{(n+5)^{0.5}}$	$\frac{1}{n^{0.2}}$	11.79056	12.93844	12.33559	12.7613	11.16328
Kernel function $K(x) = \begin{cases} 0, & x \geq 2 * width \text{ or } x \leq width \\ 1, & width < x < 2 * width \end{cases}$										
C	width	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	
0.1	0.1	0.5	$\frac{1}{(n+5)^{0.5}}$	1	13.11484	13.85565	12.77984	13.13842	12.94547	
constant $\sigma(x) = C$					13.13618	13.87016	13.78088	13.15203	13.29707	

Table 33 varfact

Among the four best cases, the last one, case 116 :

$$\text{kernel function } K(x) = \begin{cases} 0, & |x| \geq 2 * width \\ -1 * height, & width < |x| < 2 * width \\ 1 * height, & |x| \leq width \end{cases}$$

$$\eta_n = \frac{1}{(n+5)^{0.5}}, C = 0.1, \gamma = 0.1, \text{height} = 1, \text{width} = 0.1, \text{with decreasing bandwidth } b_n = \frac{1}{n^{0.2}}$$

has the smallest varfact, about 11.2~12.9, while non-adaption algorithm has the varfact of about 13.2~13.8.

3.3.2 variance/ standard error

Now compare the variance. For each kernel function, I choose the best one with the smallest variance from all the cases I try.

kernel function $K(x) = e^{-\frac{ x ^{\alpha_1}}{\alpha_2}}$										
α_1	C	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	
2	0.1	2	$\frac{1}{(n+5)^{0.5}}$	10	0.060903	0.060613	0.05894	0.058761	0.061143	
kernel function $K(x) = \frac{1}{1 + \alpha_1 x ^{\alpha_2}}$										
α_1	C	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	
10	0.1	10	$\frac{1}{(n+5)^{0.5}}$	1	0.062730	0.059807	0.059948	0.06195	0.061115	
kernel function $K(x) = \begin{cases} 0, & x \geq 2 * width \\ -1 * height, & width < x < 2 * width \\ 1 * height, & x \leq width \end{cases}$										
C	γ	height	width	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run
0.1	2	1	0.1	$\frac{1}{(n+5)^{0.5}}$	$\frac{1}{n^{0.2}}$	0.058331	0.061151	0.059702	0.060674	0.056919
Kernel function $K(x) = \begin{cases} 0, & x \geq 2 * width \text{ or } x \leq width \\ 1, & width < x < 2 * width \end{cases}$										
C	width	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	
0.1	0.2	0.1	$\frac{1}{(n+5)^{0.5}}$	0.1	0.06208	0.062719	0.065429	0.062305	0.063206	
constant $\sigma(x) = C$					0.061665	0.063307	0.063017	0.061586	0.065461	

Table 34 variance

Among the four best cases, case 116 still performs the best. It has the smallest variance, around 0.057~0.061, a little smaller than non-adaptation algorithm, which has the variance of about 0.061~0.065.

3.3.3 average squared jump distance

For each kernel function, I choose the best one with the biggest average squared jump distance from all the cases I try.

kernel function $K(x) = e^{-\frac{ x ^{\alpha_1}}{\alpha_2}}$										
α_1	C	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	
2	0.1	2	$\frac{1}{(n+5)^{0.5}}$	10	7.728795	7.719727	8.015229	7.980781	7.709572	
kernel function $K(x) = \frac{1}{1 + \alpha_1 x ^{\alpha_2}}$										
α_1	C	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	
10	10	0.1	$\frac{1}{(n+5)^{0.5}}$	1	7.445668	7.493031	7.48323	7.579445	7.64341	
kernel function $K(x) = \begin{cases} 0, & x \geq 2 * width \\ -1 * height, & width < x < 2 * width \\ 1 * height, & x \leq width \end{cases}$										
C	γ	height	width	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run
0.01	10	0.5	0.5	$\frac{1}{(n+5)^{0.5}}$	1	8.03459	8.031316	7.766907	7.8188	8.158163
Kernel function $K(x) = \begin{cases} 0, & x \geq 2 * width \text{ or } x \leq width \\ 1, & width < x < 2 * width \end{cases}$										
C	width	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	
0.1	0.5	0.1	$\frac{1}{(n+5)^{0.5}}$	1	7.522012	7.63438	7.592666	7.63681	7.74818	
constant $\sigma(x) = C$					7.204595	7.282846	7.396178	7.31972	7.25605	

Table 35 average squared jump distance

From table 35 we can see that case 116 has the biggest average squared jump distance, about 7.7~8.1, a little bigger than 7.2~7.4.

So among all the 62 cases I try, case 116 gives the smallest variance and varfact, and biggest average squared jump distance, a little better than non-adaptation algorithm. You check check this from table 36.

	varfact	variance	average sq. distance
Case 116	11.2~12.9	0.057~0.061	7.7~8.1
$\sigma(x) = C$	13.2~13.8	0.061~0.065	7.2~7.4

Table 36 comparison of the best cases

3.4 Example 3: Mixture of Three Normal Distributions in R^1

As the first two examples, I try 65 kinds of parameter choice and kernel functions to implement the adaptation algorithm; and then generate 5 Markov Chains according to the final $\sigma(x)$. You can check the varfact, variance and average squared jump distance of them in Appendix 3.3.6~3.3.8. I will not include the cases that the Markov Chain has bad mixing, like cases when $\gamma = 2$.

3.4.1 varfact

For the four kernel functions, the below four cases have the smaller varfact than others with the same kernel functions.

kernel function $K(x) = e^{-\frac{ x ^{\alpha_1}}{\alpha_2}}$										
α_1	C	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	
1	10	0.5	$\frac{1}{(n+5)^{0.5}}$	1	12.64651	12.96737	12.65248	13.67789	12.82574	
kernel function $K(x) = \frac{1}{1 + \alpha_1 x ^{\alpha_2}}$										
α_1	C	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	
10	10	0.5	$\frac{1}{(n+5)^{0.5}}$	1	13.3392	12.24885	13.38762	13.53404	13.72235	
kernel function $K(x) = \begin{cases} 0, & x \geq 2 * width \\ -1 * height, & width < x < 2 * width \\ 1 * height, & x \leq width \end{cases}$										
C	γ	height	width	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run

0.1	0.5	0.1	2	$\frac{1}{(n+5)^{0.5}}$	10	12.76971	12.85744	12.76055	12.98399	12.9673
Kernel function $K(x) = \begin{cases} 0, & x \geq 2 * width \text{ or } x \leq width \\ 1, & width < x < 2 * width \end{cases}$										
C	width	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	
1	0.5	0.5	$\frac{1}{(n+5)^{0.5}}$	1	12.43654	12.98527	12.4032	12.94385	12.92349	
constant $\sigma(x) = C$					14.15845	15.31856	13.73391	15.47598	14.98496	

Table 37 varfact

Among the four best cases, the last two,

case 173:

$$\text{Kernel function: } K(x) = \begin{cases} 0, & |x| \geq 2 * width \\ -1 * height, & width < |x| < 2 * width \\ 1 * height, & |x| \leq width \end{cases}$$

$$\eta_n = \frac{1}{(n+5)^{0.5}}, \text{ height} = 0.1, \text{ width} = 2, C = 0.1, \gamma = 0.5 \text{ with fixed bandwidth } b_n = 10$$

case 182 :

$$\text{Kernel function: } K(x) = \begin{cases} 0, & |x| \geq 2 * width \text{ or } |x| \leq width \\ 1, & width < |x| < 2 * width \end{cases}$$

$$\eta_n = \frac{1}{(n+5)^{0.5}}, \text{ width} = 0.5, C = 1, \gamma = 0.5 \text{ with fixed bandwidth } b_n = 1$$

Have close performance. They have the smallest varfact, about 12.4~12.98, while non-adaption

algorithm has the varfact of about 14~15.4.

3.4.2 variance/ standard error

Now compare the variance. For each kernel function, I choose the best one with the smallest variance from all the cases I try.

kernel function $K(x) = e^{-\frac{ x ^{\alpha_1}}{\alpha_2}}$										
α_1	C	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	
1	10	0.5	$\frac{1}{(n+5)^{0.5}}$	1	0.092925 19	0.093307 61	0.092218 1	0.097322 62	0.093457 14	
kernel function $K(x) = \frac{1}{1 + \alpha_1 x ^{\alpha_2}}$										
α_1	C	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	
10	10	0.5	$\frac{1}{(n+5)^{0.5}}$	1	0.094868 4	0.090432 56	0.095753 66	0.096414 64	0.096500 97	
kernel function $K(x) = \begin{cases} 0, & x \geq 2 * width \\ -1 * height, & width < x < 2 * width \\ 1 * height, & x \leq width \end{cases}$										
C	γ	height	width	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run
0.1	0.5	0.1	2	$\frac{1}{(n+5)^{0.5}}$	$\frac{1}{n^{0.2}}$	0.093088 3	0.093172 7	0.093848	0.091052	0.090727
Kernel function $K(x) = \begin{cases} 0, & x \geq 2 * width \text{ or } x \leq width \\ 1, & width < x < 2 * width \end{cases}$										
C	width	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	
1	0.5	0.5	$\frac{1}{(n+5)^{0.5}}$	1	0.091608 58	0.093255 67	0.092064 86	0.093385 38	0.093535 89	
constant $\sigma(x) = C$					0.097670 42	0.101192 4	0.097137 88	0.098421 07	0.102212 6	

Table 38 varfact

Among the four best cases, case 173 and 182 still perform the best. They have the smallest variance, around 0.092~0.094, a little smaller than non-adaptation algorithm, which has the variance of about 0.097~0.10.

3.2.3 average squared jump distance

For each kernel function, I choose the best one with the biggest average squared jump distance from all the cases I try.

kernel function $K(x) = e^{-\frac{ x ^{\alpha_1}}{\alpha_2}}$										
---	--	--	--	--	--	--	--	--	--	--

α_1	C	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	
1	10	0.5	$\frac{1}{(n+5)^{0.5}}$	1	17.64668	17.55073	17.72917	17.68229	17.58393	
kernel function $K(x) = \frac{1}{1 + \alpha_1 x ^{\alpha_2}}$										
α_1	C	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	
10	10	0.5	$\frac{1}{(n+5)^{0.5}}$	1	17.2243	17.46561	17.20456	17.48782	17.49019	
kernel function $K(x) = \begin{cases} 0, & x \geq 2 * width \\ -1 * height, & width < x < 2 * width \\ 1 * height, & x \leq width \end{cases}$										
C	γ	height	width	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run
0.1	0.5	0.1	2	$\frac{1}{(n+5)^{0.5}}$	10	18.04347	18.16659	17.67161	17.90958	17.7375
Kernel function $K(x) = \begin{cases} 0, & x \geq 2 * width \text{ or } x \leq width \\ 1, & width < x < 2 * width \end{cases}$										
C	width	γ	η_n	b_n	First run	Second run	Third run	Fourth run	Fifth run	
1	0.5	0.5	$\frac{1}{(n+5)^{0.5}}$	1	17.93946	17.42754	17.92938	17.68764	17.78533	
constant $\sigma(x) = C$					15.54288	15.35994	15.284	15.54276	15.54288	

Table 39 average squared jump distance

From table 39, we can see that case 173 has the biggest average squared jump distance, about 17.7~18.1, a little bigger than 15.3~15.5.

	varfact	variance	average sq. distance
Case 173	12.4~12.98	0.092~0.094	17.7~18.1
$\sigma(x) = C$	14~15.4	0.097~0.10	15.3~15.5

Table 40 comparison of the best cases

So among all the 65 cases I try, case 173 gives the smallest variance and varfact, and biggest average squared jump distance, a little better than non-adaptation algorithm.

3.5 Example 4: Normal Distribution in R^2

I first learn some good parameter choices from Example 1: Normal Distribution in R^2 , then try these parameter choices to run the adaptation algorithm in this example. According to the final $\sigma(x)$ from the adaptation algorithm, I generate 5 Markov Chain for each case. I try 9 cases in total. The generated Markov Chain all have good mixing. Now I want to compare them with constant $\sigma(x)$ in the above measures.

3.5.1 varfact

First compare the varfact of these cases. Table 41 list the varfact of 5 Markov Chain generated for each case I try. The first line is the results for the list of x1, the first component of the random variance, and the second line is for that of x2, the second component.

Kernel function : $K(x) = e^{-\frac{ x ^{\alpha_1}}{\alpha_2}}$										
α_1	C	γ	b_n	η_n	First run	Second run	Third run	Fourth run	Fifth run	
0.5	0.1	2	1	$\frac{1}{(n+5)^{0.5}}$	8.01143	8.12074	8.02683	8.63622	7.84477	
					8.39018	8.68444	8.25877	8.12773	8.08205	
Kernel function : $K(x) = \frac{1}{1 + \alpha_1 x ^{\alpha_2}}$										
α_1	C	γ	b_n	η_n	First run	Second run	Third run	Fourth run	Fifth run	
1	0.1	1	1	$\frac{1}{(n+5)^{0.5}}$	9.01875	8.05305	7.61042	8.13736	8.28670	
					7.95014	8.35461	8.29565	8.04215	8.34854	
Kernel function : $K(x) = \begin{cases} 0, & x \geq 2 * width \\ -1 * height, & width < x < 2 * width \\ 1 * height, & x \leq width \end{cases}$										
height	width	C	γ	b_n	η_n	First run	Second run	Third run	Fourth run	Fifth run
0.5	0.5	0.01	10	1	$\frac{1}{(n+5)^{0.5}}$	7.74167	7.79554	8.50804	8.24295	8.0193
						8.51949	7.76981	8.37399	8.40765	8.05531
Kernel function : $K(x) = \begin{cases} 0, & x \geq 2 * width \text{ or } x \leq width \\ 1, & width < x < 2 * width \end{cases}$										
width	C	γ	b_n	η_n	First run	Second run	Third run	Fourth run	Fifth run	
5	0.01	2	1	$\frac{1}{(n+5)^{0.5}}$	9.5072	8.8876	8.8812	8.7593	8.6037	
					9.2312	8.9168	9.160	9.0149	9.0498	

constant $\sigma(x) = C$				
8.584404	8.331632	8.68917	8.07853	8.492082
8.08258	8.4440	8.334455	8.904434	8.6586

Table 41 varfact

For the adaptation algorithm, the best case is case 201 :

$$\text{Kernel function: } K(x) = \begin{cases} 0, & |x| \geq 2 * \text{width} \\ -1 * \text{height}, & \text{width} < |x| < 2 * \text{width} \\ 1 * \text{height}, & |x| \leq \text{width} \end{cases}$$

$$\eta_n = \frac{1}{(n+5)^{0.5}}, \text{ height} = 0.5, \text{ width} = 0.5, C = 0.01, \gamma = 10 \text{ with fixed bandwidth } b_n = 1$$

In case 201, the varfact is around 7.7~8.5 for x1 and 7.7~8.4 for x2. While non-adaptation algorithm

has the varfact of about 8.3~8.5 for x1 and 8.3~9.9 for x2. So adaptation algorithm works a little

better.

3.5.2 variance/ standard error

Table 42 list the standard error of 5 Markov Chain generated for each case I try. The first line is the results for the list of x1, and the second line is for that of x2.

Kernel function : $K(x) = e^{-\frac{ x ^{\alpha_1}}{\alpha_2}}$									
α_1	C	γ	b_n	η_n	First run	Second run	Third run	Fourth run	Fifth run
0.5	0.1	2	1	$\frac{1}{(n+5)^{0.5}}$	0.0099647	0.0101173	0.0099771	0.0104498	0.0099363
					0.0103432	0.0105912	0.0101549	0.0100481	0.0101184
Kernel function : $K(x) = \frac{1}{1 + \alpha_1 x ^{\alpha_2}}$									
α_1	C	γ	b_n	η_n	First run	Second run	Third run	Fourth run	Fifth run
1	0.1	1	1	$\frac{1}{(n+5)^{0.5}}$	0.0105313	0.0097265	0.0096214	0.0099982	0.0099314
					0.0097723	0.0099487	0.0098910	0.0098809	0.0100729

Kernel function : $K(x) = \begin{cases} 0, & x \geq 2 * width \\ -1 * height, & width < x < 2 * width \\ 1 * height, & x \leq width \end{cases}$										
height	width	C	γ	b_n	η_n	First run	Second run	Third run	Fourth run	Fifth run
0.5	0.5	0.01	10	1	$\frac{1}{(n+5)^{0.5}}$	0.0094858	0.0095293	0.0099355	0.0098088	0.0097593
						0.0100096	0.0095595	0.0099169	0.0098439	0.0099373
Kernel function : $K(x) = \begin{cases} 0, & x \geq 2 * width \text{ or } x \leq width \\ 1, & width < x < 2 * width \end{cases}$										
width	C	γ	b_n	η_n	First run	Second run	Third run	Fourth run	Fifth run	
5	0.01	2	1	$\frac{1}{(n+5)^{0.5}}$	0.01031	0.00998	0.01007	0.00984	0.00987	
					0.01010	0.01016	0.01006	0.01013	0.01010	
constant $\sigma(x) = C$										
0.00972953		0.0097068		0.009780478		0.00939504		0.009631347		
0.0094686		0.009739506		0.00954663		0.009958031		0.009772716		

Table 42 variance

The adaptation algorithm's standard error is quite close to non-adaptation algorithm. So if measured in variance/standard error, it is hard to say which one is better for this example by the cases I try.

3.5.3 average squared jump distance

Table 43 list the average squared jump distance of 5 Markov Chain generated for each case I try. The first line is the results for the list of x1, and the second line is for that of x2.

Kernel function : $K(x) = e^{-\frac{ x ^{\alpha_1}}{\alpha_2}}$										
α_1	C	γ	b_n	η_n	First run	Second run	Third run	Fourth run	Fifth run	
0.5	0.1	2	1	$\frac{1}{(n+5)^{0.5}}$	0.506856	0.504129	0.493900	0.491895	0.491927	
					0.506880	0.50175	0.495962	0.480626	0.496152	
Kernel function : $K(x) = \frac{1}{1 + \alpha_1 x ^{\alpha_2}}$										
α_1	C	γ	b_n	η_n	First run	Second run	Third run	Fourth run	Fifth run	
1	0.1	1	1	$\frac{1}{(n+5)^{0.5}}$	0.458961	0.461018	0.4751	0.480245	0.47135	
					0.466943	0.467047	0.4542	0.4680	0.45683	
Kernel function : $K(x) = \begin{cases} 0, & x \geq 2 * width \\ -1 * height, & width < x < 2 * width \\ 1 * height, & x \leq width \end{cases}$										

height	width	C	γ	b_n	η_n	First run	Second run	Third run	Fourth run	Fifth run
0.5	0.5	0.01	10	1	$\frac{1}{(n+5)^{0.5}}$	0.463001	0.46015	0.449489	0.465211	0.457632
						0.461200	0.465179	0.456853	0.452455	0.461372
Kernel function : $K(x) = \begin{cases} 0, & x \geq 2 * \text{width} \text{ or } x \leq \text{width} \\ 1, & \text{width} < x < 2 * \text{width} \end{cases}$										
width	C	γ	b_n	η_n	First run	Second run	Third run	Fourth run	Fifth run	
5	0.01	2	1	$\frac{1}{(n+5)^{0.5}}$	0.394871	0.406117	0.404594	0.396869	0.414291	
					0.40248	0.421151	0.404090	0.40975	0.413094	
constant $\sigma(x) = C$										
0.4365735			0.4408152		0.421703		0.4340354		0.4279432	
0.4396845			0.4289132		0.4226133		0.4327815		0.4182256	

Table 43 average squared jump distance

If measured in average squared jump distance, case 196 :

$$\text{Kernel function : } K(x) = e^{-\frac{|x|^{\alpha_1}}{\alpha_2}}$$

$$\eta_n = \frac{1}{(n+5)^{0.5}}, \alpha_1 = 0.5, \alpha_2 = 1, C = 0.1, \gamma = 2 \text{ with fixed bandwidth } b_n = 1$$

performs better than the non-adaptation algorithm and the other adaptation algorithm:

Its average squared jump distance is about 0.5, bigger than around 0.42~0.43 in non-adaptation algorithm. So if measured in average squared jump distance, non-adaptation algorithm performs a little better.

Conclusion

We explore the efficiency of adaptation algorithm for Random Walk Metropolis. Simulation studies indicate that probably it is impossible to make the local acceptance rate around 0.234 for all different x . But during the procedure to achieve this, the adaptation algorithm makes a little improvement to the efficiency of Markov Chain, if measured in variance, varfact, and average squared jump distance.

Reference

- [1] Steve Brooks, Andrew Gelman, Galin L. Jones and Xiao-Li Meng (2011), Handbook of Markov Chain Monte Carlo, Chapman and Hall/CRC.
- [2] Gareth O. Roberts and Jeffrey Rosenthal (2009), Example of Adaptive MCMC, J. Comp. Graph. Stat. 18:349-367, 2009
- [3] Gareth O. Roberts and Jeffrey Rosenthal (2007), Coupling and Ergodicity of Adaptive MCMC, J. Appl. Prob. 44 , 458–475, 2007
- [4] wikipedia: <http://en.wikipedia.org/wiki/Autocorrelation>
- [5] MCMC notes: <http://www.probability.ca/jeff/teaching/1011/sta3431/notes.pdf>

Appendix

1. Code

1.1 C code

1.1.1 C code to implement the adaptation algorithm and generate a Markov Chain (Example 1~3)

1.1.2 C code to implement the adaptation algorithm and generate a Markov Chain (Example 4)

1.2 R code

1.2.1 R code to plot the final $g(X)$ and $\sigma(X)$ (Example 1~3)

1.2.2 R code to plot the final $g(X)$ and $\sigma(X)$ (Example 4)

1.2.3 R code to generate Markov Chain without adaption (Example 1)

1.2.4 R code to generate Markov Chain without adaption (Example 2)

1.2.5 R code to generate Markov Chain without adaption (Example 3)

1.2.6 R code to generate Markov Chain without adaption (Example 4)

1.2.7 R code to compute different efficiency measures (Example 1~3)

1.2.8 R code to compute different efficiency measures (Example 4)

1.2.9 R code to implement OLS in approximation of the final $\sigma(X)$

2. Output

2.1 Example 1: normal distribution in \mathcal{R}^1

2.1.1 kernel function $K(x) = e^{-\frac{|x|^{\alpha_1}}{\alpha_2}}$

2.1.1.1 case 1 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 1, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.1.1.2 case 2 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 2, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.1.1.3 case 3 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 10, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.1.1.4 case 4 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 0.1, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.1.1.5 case 5 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 0.5, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.1.1.6 case 6 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 2, \alpha_2 = 1, C = 2, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.1.1.7 case 7 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 2, \alpha_2 = 1, C = 10, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.1.1.8 case 8 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 2, \alpha_2 = 1, C = 0.1, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.1.1.9 case 9 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 2, \alpha_2 = 1, C = 0.5, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.1.1.10 case 10 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 2, \alpha_2 = 1, C = 0.01, \gamma = 2$ with fixed bandwidth

$b_n = 1$

2.1.1.11 case 11 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 2, \alpha_2 = 1, C = 0.1, \gamma = 10$ with fixed bandwidth

$b_n = 1$

2.1.1.12 case 12 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 2, \alpha_2 = 1, C = 0.1, \gamma = 0.5$ with fixed bandwidth

$b_n = 1$

2.1.1.13 case 13 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 2, \alpha_2 = 1, C = 0.1, \gamma = 1$ with fixed bandwidth

$b_n = 1$

2.1.1.14 case 14 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 2, \alpha_2 = 1, C = 0.1, \gamma = 0.1$ with fixed bandwidth

$b_n = 1$

2.1.1.15 case 15 $\eta_n = \frac{1}{(n+5)^{0.8}}$, $\alpha_1 = 2, \alpha_2 = 1, C = 0.1, \gamma = 2$ with fixed bandwidth

$$b_n = 1$$

2.1.1.16 case 16 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 2, \alpha_2 = 1, C = 0.1, \gamma = 2$ with fixed bandwidth

$$b_n = 10$$

2.1.1.17 case 17 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 2, \alpha_2 = 1, C = 0.1, \gamma = 2$ with fixed bandwidth

$$b_n = 0.1$$

2.1.1.18 case 18 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 2, \alpha_2 = 1, C = 0.1, \gamma = 2$ with decreasing

bandwidth $b_n = \frac{1}{n^{0.2}}$

2.1.2 kernel function $K(x) = \frac{1}{1 + \alpha_1 |x|^{\alpha_2}}$

2.1.2.1 case 19 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 1, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.1.2.2 case 20 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 10, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.1.2.3 case 21 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 0.1, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth

$$b_n = 1$$

2.1.2.4 case 22 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 1, \alpha_2 = 1, C = 10, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.1.2.5 case 23 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 1, \alpha_2 = 1, C = 0.1, \gamma = 2$ with fixed bandwidth

$$b_n = 1$$

2.1.2.6 case 24 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 1, \alpha_2 = 1, C = 0.01, \gamma = 2$ with fixed bandwidth

$$b_n = 1$$

2.1.2.7 case 25 $\eta_n = \frac{1}{(n+5)^{0.5}}, \alpha_1 = 1, \alpha_2 = 1, C = 0.1, \gamma = 1$ with fixed bandwidth $b_n = 1$

2.1.2.8 case 26 $\eta_n = \frac{1}{(n+5)^{0.5}}, \alpha_1 = 1, \alpha_2 = 1, C = 0.1, \gamma = 10$ with fixed bandwidth

$$b_n = 1$$

2.1.2.9 case 27 $\eta_n = \frac{1}{(n+5)^{0.5}}, \alpha_1 = 1, \alpha_2 = 1, C = 0.1, \gamma = 2$ with fixed bandwidth

$$b_n = 10$$

2.1.2.10 case 28 $\eta_n = \frac{1}{(n+5)^{0.5}}, \alpha_1 = 1, \alpha_2 = 1, C = 0.1, \gamma = 2$ with fixed bandwidth

$$b_n = 0.1$$

2.1.2.11 case 29 $\eta_n = \frac{1}{(n+5)^{0.5}}, \alpha_1 = 1, \alpha_2 = 1, C = 0.1, \gamma = 0.1$ with fixed bandwidth

$$b_n = 10$$

2.1.2.12 case 30 $\eta_n = \frac{1}{(n+5)^{0.5}}, \alpha_1 = 1, \alpha_2 = 1, C = 1, \gamma = 2$ with decreasing bandwidth

$$b_n = \frac{1}{n^{0.2}}$$

2.1.2.13 case 31 $\eta_n = \frac{1}{(n+5)^{0.8}}, \alpha_1 = 0.1, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth

$$b_n = 10$$

2.1.3 kernel function
$$K(x) = \begin{cases} 0, & |x| \geq 2 * width \\ -1 * height, & width < |x| < 2 * width \\ 1 * height, & |x| \leq width \end{cases}$$

2.1.3.1 case 32 $\eta_n = \frac{1}{(n+5)^{0.5}}, height = 0.5, width = 0.5, C = 1, \gamma = 2$ with fixed

bandwidth $b_n = 1$

2.1.3.2 case 33 $\eta_n = \frac{1}{(n+5)^{0.5}}$, *height* = 0.5, *width* = 0.5, $C = 10, \gamma = 2$ with fixed

bandwidth $b_n = 1$

2.1.3.3 case 34 $\eta_n = \frac{1}{(n+5)^{0.5}}$, *height* = 0.5, *width* = 0.5, $C = 0.1, \gamma = 2$ with fixed

bandwidth $b_n = 1$

2.1.3.4 case 35 $\eta_n = \frac{1}{(n+5)^{0.5}}$, *height* = 0.5, *width* = 0.5, $C = 0.01, \gamma = 2$ with fixed

bandwidth $b_n = 1$

2.1.3.5 case 36 $\eta_n = \frac{1}{(n+5)^{0.5}}$, *height* = 0.5, *width* = 0.5, $C = 0, \gamma = 2$ with fixed

bandwidth $b_n = 1$

2.1.3.6 case 37 $\eta_n = \frac{1}{(n+5)^{0.5}}$, *height* = 0.5, *width* = 0.5, $C = 0.001, \gamma = 2$ with fixed

bandwidth $b_n = 1$

2.1.3.7 case 38 $\eta_n = \frac{1}{(n+5)^{0.5}}$, *height* = 0.5, *width* = 0.5, $C = 0.001, \gamma = 10$ with fixed

bandwidth $b_n = 1$

2.1.3.8 case 39 $\eta_n = \frac{1}{(n+5)^{0.5}}$, *height* = 0.5, *width* = 0.5, $C = 0.001, \gamma = 1$ with fixed

bandwidth $b_n = 1$

2.1.3.9 case 40 $\eta_n = \frac{1}{(n+5)^{0.5}}$, *height* = 0.5, *width* = 0.5, $C = 0.001, \gamma = 0.1$ with fixed

bandwidth $b_n = 1$

2.1.3.10 case 41 $\eta_n = \frac{1}{(n+5)^{0.5}}$, *height* = 1, *width* = 0.5, $C = 0.01, \gamma = 2$ with fixed

bandwidth $b_n = 1$

2.1.3.11 case 42 $\eta_n = \frac{1}{(n+5)^{0.5}}$, *height* = 10, *width* = 0.5, $C = 0.01, \gamma = 2$ with fixed

bandwidth $b_n = 1$

2.1.3.12 case 43 $\eta_n = \frac{1}{(n+5)^{0.5}}$, *height* = 0.1, *width* = 0.5, $C = 0.01, \gamma = 2$ with fixed

bandwidth $b_n = 1$

2.1.3.13 case 44 $\eta_n = \frac{1}{(n+5)^{0.5}}$, *height* = 0.5, *width* = 2, $C = 0.01, \gamma = 2$ with fixed

bandwidth $b_n = 1$

2.1.3.14 case 45 $\eta_n = \frac{1}{(n+5)^{0.5}}$, *height* = 0.5, *width* = 5, $C = 0.01, \gamma = 2$ with fixed

bandwidth $b_n = 1$

2.1.3.15 case 46 $\eta_n = \frac{1}{(n+5)^{0.5}}$, *height* = 0.5, *width* = 0.1, $C = 0.01, \gamma = 2$ with fixed

bandwidth $b_n = 1$

2.1.3.16 case 47 $\eta_n = \frac{1}{(n+5)^{0.5}}$, *height* = 0.5, *width* = 0.5, $C = 0.01, \gamma = 2$ with fixed

bandwidth $b_n = 10$

2.1.3.17 case 48 $\eta_n = \frac{1}{(n+5)^{0.5}}$, *height* = 0.5, *width* = 0.5, $C = 0.01, \gamma = 2$ with fixed

bandwidth $b_n = 0.1$

2.1.3.18 case 49 $\eta_n = \frac{1}{(n+5)^{0.5}}$, *height* = 0.5, *width* = 0.5, $C = 0.01, \gamma = 2$ with

decreasing bandwidth $b_n = \frac{1}{n^{0.2}}$

2.1.3.19 case 50 $\eta_n = \frac{1}{(n+5)^{0.8}}$, *height* = 0.5, *width* = 0.5, $C = 0.01, \gamma = 2$ with fixed

bandwidth $b_n = 1$

2.1.4 kernel function $K(x) = \begin{cases} 0, & |x| \geq 2 * \text{width} \text{ or } |x| \leq \text{width} \\ 1, & \text{width} < |x| < 2 * \text{width} \end{cases}$

2.1.4.1 case 51 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 0.5, $C = 1, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.1.4.2 case 52 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 0.5, $C = 10, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.1.4.3 case 53 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 0.5, $C = 0.1, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.1.4.4 case 54 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 0.5, $C = 0.01, \gamma = 2$ with fixed bandwidth

$b_n = 1$

2.1.4.5 case 55 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 0.5, $C = 0.001, \gamma = 2$ with fixed bandwidth

$b_n = 1$

2.1.4.6 case 56 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 1, $C = 0.01, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.1.4.7 case 57 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 5, $C = 0.01, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.1.4.8 case 58 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 0.1, $C = 0.01, \gamma = 2$ with fixed bandwidth

$b_n = 1$

2.1.4.9 case 59 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 8, $C = 0.01, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.1.4.10 case 60 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 5, $C = 0.01, \gamma = 5$ with fixed bandwidth $b_n = 1$

2.1.4.11 case 61 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 5, $C = 0.01, \gamma = 1$ with fixed bandwidth $b_n = 1$

2.1.4.12 case 62 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 5, $C = 0.01, \gamma = 0.1$ with fixed bandwidth

$$b_n = 1$$

2.1.4.13 case 63 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 5, $C = 0.01, \gamma = 2$ with fixed bandwidth

$$b_n = 10$$

2.1.4.14 case 64 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 5, $C = 0.01, \gamma = 2$ with fixed bandwidth

$$b_n = 0.1$$

2.1.4.15 case 65 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 5, $C = 0.01, \gamma = 2$ with decreasing bandwidth

$$b_n = \frac{1}{n^{0.2}}$$

2.1.4.16 case 66 $\eta_n = \frac{1}{(n+5)^{0.8}}$, width = 5, $C = 0.01, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.1.5 constant $\sigma(x) = C$

2.1.6 varfact comparison

2.1.7 variance comparison

2.1.8 comparison of average squared jump distance

2.1.9 OLS in approximation of the final $\sigma(x)$

2.2 Example 2: mixture of two normal distributions in \mathbb{R}^1

2.2.1 kernel function $K(x) = e^{-\frac{|x|^{\alpha_1}}{\alpha_2}}$

2.2.1.1 case 67 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 1, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.2.1.2 case 68 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 2, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.2.1.3 case 69 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 10, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.2.1.4 case 70 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 0.1, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth

$$b_n = 1$$

$$2.2.1.5 \text{ case 71 } \eta_n = \frac{1}{(n+5)^{0.5}}, \alpha_1 = 0.5, \alpha_2 = 1, C = 1, \gamma = 2 \text{ with fixed bandwidth}$$

$$b_n = 1$$

$$2.2.1.6 \text{ case 72 } \eta_n = \frac{1}{(n+5)^{0.5}}, \alpha_1 = 0.01, \alpha_2 = 1, C = 1, \gamma = 2 \text{ with fixed bandwidth}$$

$$b_n = 1$$

$$2.2.1.7 \text{ case 73 } \eta_n = \frac{1}{(n+5)^{0.5}}, \alpha_1 = 2, \alpha_2 = 1, C = 10, \gamma = 2 \text{ with fixed bandwidth}$$

$$b_n = 1$$

$$2.2.1.8 \text{ case 74 } \eta_n = \frac{1}{(n+5)^{0.5}}, \alpha_1 = 2, \alpha_2 = 1, C = 0.1, \gamma = 2 \text{ with fixed bandwidth}$$

$$b_n = 1$$

$$2.2.1.9 \text{ case 75 } \eta_n = \frac{1}{(n+5)^{0.5}}, \alpha_1 = 2, \alpha_2 = 1, C = 0.1, \gamma = 10 \text{ with fixed bandwidth}$$

$$b_n = 1$$

$$2.2.1.10 \text{ case 76 } \eta_n = \frac{1}{(n+5)^{0.5}}, \alpha_1 = 2, \alpha_2 = 1, C = 0.1, \gamma = 1 \text{ with fixed bandwidth}$$

$$b_n = 1$$

$$2.2.1.11 \text{ case 77 } \eta_n = \frac{1}{(n+5)^{0.5}}, \alpha_1 = 2, \alpha_2 = 1, C = 0.1, \gamma = 0.1 \text{ with fixed bandwidth}$$

$$b_n = 1$$

$$2.2.1.12 \text{ case 78 } \eta_n = \frac{1}{(n+5)^{0.5}}, \alpha_1 = 2, \alpha_2 = 1, C = 0.1, \gamma = 2 \text{ with fixed bandwidth}$$

$$b_n = 10$$

$$2.2.1.13 \text{ case 79 } \eta_n = \frac{1}{(n+5)^{0.5}}, \alpha_1 = 2, \alpha_2 = 1, C = 0.1, \gamma = 2 \text{ with fixed bandwidth}$$

$$b_n = 0.1$$

2.2.1.14 case 80 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 2, \alpha_2 = 1, C = 0.1, \gamma = 2$ with decreasing

bandwidth $b_n = \frac{1}{n^{0.2}}$

2.2.1.15 case 81 $\eta_n = \frac{1}{(n+5)^{0.8}}$, $\alpha_1 = 2, \alpha_2 = 1, C = 0.1, \gamma = 2$ with fixed bandwidth

$$b_n = 10$$

2.2.2 kernel function $K(x) = \frac{1}{1 + \alpha_1 |x|^{\alpha_2}}$

2.2.2.1 case 90 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 1, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.2.2.2 case 91 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 10, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.2.2.3 case 92 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 0.1, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth

$$b_n = 1$$

2.2.2.4 case 93 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 12, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.2.2.5 case 94 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 5, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.2.2.6 case 95 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 10, \alpha_2 = 1, C = 1, \gamma = 10$ with fixed bandwidth

$$b_n = 1$$

2.2.2.7 case 96 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 10, \alpha_2 = 1, C = 1, \gamma = 0.1$ with fixed bandwidth

$$b_n = 1$$

2.2.2.8 case 97 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 10, \alpha_2 = 1, C = 10, \gamma = 0.1$ with fixed bandwidth

$$b_n = 1$$

2.2.2.9 case 98 $\eta_n = \frac{1}{(n+5)^{0.5}}, \alpha_1 = 10, \alpha_2 = 1, C = 0.1, \gamma = 0.1$ with fixed bandwidth

$$b_n = 1$$

2.2.2.10 case 99 $\eta_n = \frac{1}{(n+5)^{0.5}}, \alpha_1 = 10, \alpha_2 = 1, C = 0.1, \gamma = 0.1$ with fixed bandwidth

$$b_n = 10$$

2.2.2.11 case 100 $\eta_n = \frac{1}{(n+5)^{0.5}}, \alpha_1 = 10, \alpha_2 = 1, C = 0.1, \gamma = 0.1$ with fixed bandwidth

$$b_n = 0.1$$

2.2.2.12 case 101 $\eta_n = \frac{1}{(n+5)^{0.5}}, \alpha_1 = 10, \alpha_2 = 1, C = 0.1, \gamma = 0.1$ with decreasing

bandwidth $b_n = \frac{1}{n^{0.2}}$

2.2.2.13 case 102 $\eta_n = \frac{1}{(n+5)^{0.8}}, \alpha_1 = 10, \alpha_2 = 1, C = 0.1, \gamma = 0.1$ with fixed bandwidth

$$b_n = 1$$

2.2.3 kernel function
$$K(x) = \begin{cases} 0, & |x| \geq 2 * width \\ -1 * height, & width < |x| < 2 * width \\ 1 * height, & |x| \leq width \end{cases}$$

2.2.3.1 case 103 $\eta_n = \frac{1}{(n+5)^{0.5}}, C = 1, \gamma = 2, height = 0.5, width = 0.5,$ with fixed

bandwidth $b_n = 1$

2.2.3.2 case 104 $\eta_n = \frac{1}{(n+5)^{0.5}}, C = 10, \gamma = 2, height = 0.5, width = 0.5,$ with fixed

bandwidth $b_n = 1$

2.2.3.3 case 105 $\eta_n = \frac{1}{(n+5)^{0.5}}, C = 0.1, \gamma = 2, height = 0.5, width = 0.5,$ with fixed

bandwidth $b_n = 1$

2.2.3.4 case 106 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $C = 0.1, \gamma = 10, \text{height} = 0.5, \text{width} = 0.5$, with fixed

bandwidth $b_n = 1$

2.2.3.5 case 107 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $C = 0.1, \gamma = 0.1, \text{height} = 0.5, \text{width} = 0.5$, with fixed

bandwidth $b_n = 1$

2.2.3.6 case 108 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $C = 0.1, \gamma = 2, \text{height} = 1, \text{width} = 0.5$, with fixed

bandwidth $b_n = 1$

2.2.3.7 case 109 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $C = 0.1, \gamma = 2, \text{height} = 5, \text{width} = 0.5$, with fixed

bandwidth $b_n = 1$

2.2.3.8 case 110 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $C = 0.1, \gamma = 2, \text{height} = 0.1, \text{width} = 0.5$, with fixed

bandwidth $b_n = 1$

2.2.3.9 case 111 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $C = 1, \gamma = 0.1, \text{height} = 1, \text{width} = 1$, with fixed

bandwidth $b_n = 1$

2.2.3.10 case 112 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $C = 0.1, \gamma = 0.1, \text{height} = 1, \text{width} = 5$, with fixed

bandwidth $b_n = 1$

2.2.3.11 case 113 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $C = 0.1, \gamma = 0.1, \text{height} = 1, \text{width} = 0.1$, with fixed

bandwidth $b_n = 1$

2.2.3.12 case 114 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $C = 0.1, \gamma = 0.1, \text{height} = 1, \text{width} = 0.1$, with fixed

bandwidth $b_n = 10$

2.2.3.13 case 115 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $C = 0.1, \gamma = 0.1, \text{height} = 1, \text{width} = 0.1$, with fixed

bandwidth $b_n = 0.1$

2.2.3.14 case 116 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $C = 0.1, \gamma = 0.1, \text{height} = 1, \text{width} = 0.1$, with

decreasing bandwidth $b_n = \frac{1}{n^{0.2}}$

2.2.3.15 case 117 $\eta_n = \frac{1}{(n+5)^{0.8}}$, $C = 0.1, \gamma = 0.1, \text{height} = 1, \text{width} = 0.1$, with

decreasing bandwidth $b_n = \frac{1}{n^{0.2}}$

2.2.4 kernel function $K(x) = \begin{cases} 0, & |x| \geq 2 * \text{width} \text{ or } |x| \leq \text{width} \\ 1, & \text{width} < |x| < 2 * \text{width} \end{cases}$

2.2.4.1 case 118 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $C = 1, \gamma = 2, \text{width} = 0.5$, with fixed bandwidth $b_n = 1$

2.2.4.2 case 119 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $C = 0.1, \gamma = 2, \text{width} = 0.5$, with fixed bandwidth

$b_n = 1$

2.2.4.3 case 120 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $C = 10, \gamma = 2, \text{width} = 0.5$, with fixed bandwidth

$b_n = 1$

2.2.4.4 case 121 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $C = 0.1, \gamma = 10, \text{width} = 0.5$, with fixed bandwidth

$b_n = 1$

2.2.4.5 case 122 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $C = 0.1, \gamma = 0.1, \text{width} = 0.5$, with fixed bandwidth

$b_n = 1$

2.2.4.6 case 123 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $C = 0.1, \gamma = 0.1, \text{width} = 1$, with fixed bandwidth

$$b_n = 1$$

2.2.4.7 case 124 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $C = 0.1, \gamma = 0.1, \text{width} = 2$, with fixed bandwidth

$$b_n = 1$$

2.2.4.8 case 125 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $C = 0.1, \gamma = 0.1, \text{width} = 5$, with fixed bandwidth

$$b_n = 1$$

2.2.4.9 case 126 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $C = 0.1, \gamma = 0.1, \text{width} = 0.2$, with fixed bandwidth

$$b_n = 1$$

2.2.4.10 case 127 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $C = 0.1, \gamma = 0.1, \text{width} = 0.2$, with fixed bandwidth

$$b_n = 10$$

2.2.4.11 case 128 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $C = 0.1, \gamma = 0.1, \text{width} = 0.2$, with fixed bandwidth

$$b_n = 0.1$$

2.2.4.12 case 127 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $C = 0.1, \gamma = 0.1, \text{width} = 0.2$, with decreasing

bandwidth $b_n = \frac{1}{n^{0.2}}$

2.2.4.13 case 128 $\eta_n = \frac{1}{(n+5)^{0.8}}$, $C = 0.1, \gamma = 0.1, \text{width} = 0.2$, with fixed bandwidth

$$b_n = 1$$

2.2.5 constant $\sigma(x) = C$

2.2.6 varfact comparison

2.2.7 variance comparison

2.2.8 comparison of average squared jump distance

2.3 Example 3: mixture of three normal distributions in \mathcal{R}^1

2.3.1 kernel function $K(x) = e^{-\frac{|x|^{\alpha_1}}{\alpha_2}}$

2.3.1.1 case 129 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 1, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth

$$b_n = 1$$

2.3.1.2 case 130 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 10, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth

$$b_n = 1$$

2.3.1.3 case 131 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 0.1, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth

$$b_n = 1$$

2.3.1.4 case 132 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 1, \alpha_2 = 1, C = 1, \gamma = 0.5$ with fixed bandwidth

$$b_n = 1$$

2.3.1.5 case 131 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 10, \alpha_2 = 1, C = 1, \gamma = 0.5$ with fixed bandwidth

$$b_n = 1$$

2.3.1.6 case 132 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 0.1, \alpha_2 = 1, C = 1, \gamma = 0.5$ with fixed bandwidth

$$b_n = 1$$

2.3.1.7 case 133 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 1, \alpha_2 = 1, C = 10, \gamma = 0.5$ with fixed bandwidth

$$b_n = 1$$

2.3.1.8 case 134 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 0.1, \alpha_2 = 1, C = 10, \gamma = 0.5$ with fixed bandwidth

$$b_n = 1$$

2.3.1.9 case 135 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 0.1, \alpha_2 = 1, C = 0.1, \gamma = 0.5$ with fixed bandwidth

$$b_n = 1$$

2.3.1.10 case 136 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 0.1, \alpha_2 = 1, C = 0.1, \gamma = 0.5$ with fixed

bandwidth $b_n = 10$

2.3.1.11 case 137 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 0.1, \alpha_2 = 1, C = 0.1, \gamma = 0.5$ with fixed

bandwidth $b_n = 0.1$

2.3.1.12 case 138 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 0.1, \alpha_2 = 1, C = 1, \gamma = 0.5$ with decreasing

bandwidth $b_n = \frac{1}{n^{0.2}}$

2.3.1.13 case 139 $\eta_n = \frac{1}{(n+5)^{0.8}}$, $\alpha_1 = 0.1, \alpha_2 = 1, C = 1, \gamma = 0.5$ with fixed bandwidth

$$b_n = 1$$

2.3.2 kernel function $K(x) = \frac{1}{1 + \alpha_1 |x|^{\alpha_2}}$

2.3.2.1 case 140 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 1, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.3.2.2 case 141 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 10, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth

$$b_n = 1$$

2.3.2.3 case 142 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 0.1, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth

$$b_n = 1$$

2.3.2.4 case 143 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 1, \alpha_2 = 1, C = 0.1, \gamma = 2$ with fixed bandwidth

$$b_n = 1$$

2.3.2.5 case 144 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 1, \alpha_2 = 1, C = 10, \gamma = 2$ with fixed bandwidth

$$b_n = 1$$

2.3.2.6 case 145 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 1, \alpha_2 = 1, C = 1, \gamma = 0.5$ with fixed bandwidth

$$b_n = 1$$

2.3.2.7 case 146 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 10, \alpha_2 = 1, C = 1, \gamma = 0.5$ with fixed bandwidth

$$b_n = 1$$

2.3.2.8 case 147 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 0.1, \alpha_2 = 1, C = 1, \gamma = 0.5$ with fixed bandwidth

$$b_n = 1$$

2.3.2.9 case 148 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 10, \alpha_2 = 1, C = 10, \gamma = 0.5$ with fixed bandwidth

$$b_n = 1$$

2.3.2.10 case 149 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 10, \alpha_2 = 1, C = 0.1, \gamma = 0.5$ with fixed bandwidth

$$b_n = 1$$

2.3.2.11 case 150 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 10, \alpha_2 = 1, C = 10, \gamma = 0.1$ with fixed bandwidth

$$b_n = 1$$

2.3.2.12 case 151 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 10, \alpha_2 = 1, C = 10, \gamma = 1$ with fixed bandwidth

$$b_n = 1$$

2.3.2.13 case 152 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 10, \alpha_2 = 1, C = 10, \gamma = 0.5$ with fixed bandwidth

$$b_n = 10$$

2.3.2.14 case 153 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 10, \alpha_2 = 1, C = 10, \gamma = 0.5$ with fixed bandwidth

$$b_n = 0.1$$

2.3.2.15 case 154 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 10, \alpha_2 = 1, C = 10, \gamma = 0.5$ with decreasing

bandwidth $b_n = \frac{1}{n^{0.2}}$

2.3.2.16 case 155 $\eta_n = \frac{1}{(n+5)^{0.8}}$, $\alpha_1 = 10, \alpha_2 = 1, C = 10, \gamma = 0.5$ fixed bandwidth

$$b_n = 1$$

2.3.3 kernel function
$$K(x) = \begin{cases} 0, & |x| \geq 2 * width \\ -1 * height, & width < |x| < 2 * width \\ 1 * height, & |x| \leq width \end{cases}$$

2.3.3.1 case 156 $\eta_n = \frac{1}{(n+5)^{0.5}}$, height = 0.5, width = 0.5, $C = 1, \gamma = 2$ with fixed

bandwidth $b_n = 1$

2.3.3.2 case 157 $\eta_n = \frac{1}{(n+5)^{0.5}}$, height = 0.5, width = 0.5, $C = 10, \gamma = 2$ with fixed

bandwidth $b_n = 1$

2.3.3.3 case 158 $\eta_n = \frac{1}{(n+5)^{0.5}}$, height = 0.5, width = 0.5, $C = 0.1, \gamma = 2$ with fixed

bandwidth $b_n = 1$

2.3.3.4 case 159 $\eta_n = \frac{1}{(n+5)^{0.5}}$, height = 2, width = 0.5, $C = 0.1, \gamma = 2$ with fixed

bandwidth $b_n = 1$

2.3.3.5 case 160 $\eta_n = \frac{1}{(n+5)^{0.5}}$, height = 0.5, width = 2, $C = 0.1, \gamma = 2$ with fixed

bandwidth $b_n = 1$

2.3.3.6 case 161 $\eta_n = \frac{1}{(n+5)^{0.5}}$, height = 0.5, width = 0.5, $C = 1, \gamma = 0.5$ with fixed

bandwidth $b_n = 1$

2.3.3.7 case 162 $\eta_n = \frac{1}{(n+5)^{0.5}}$, height = 0.5, width = 0.5, $C = 10, \gamma = 0.5$ with fixed

bandwidth $b_n = 1$

2.3.3.8 case 163 $\eta_n = \frac{1}{(n+5)^{0.5}}$, height = 0.5, width = 0.5, $C = 0.1, \gamma = 0.5$ with fixed

bandwidth $b_n = 1$

2.3.3.9 case 164 $\eta_n = \frac{1}{(n+5)^{0.5}}$, height = 2, width = 0.5, $C = 0.1, \gamma = 0.5$ with fixed

bandwidth $b_n = 1$

2.3.3.10 case 165 $\eta_n = \frac{1}{(n+5)^{0.5}}$, height = 5, width = 0.5, $C = 0.1, \gamma = 0.5$ with fixed

bandwidth $b_n = 1$

2.3.3.11 case 166 $\eta_n = \frac{1}{(n+5)^{0.5}}$, height = 10, width = 0.5, $C = 0.1, \gamma = 0.5$ with fixed

bandwidth $b_n = 1$

2.3.3.12 case 167 $\eta_n = \frac{1}{(n+5)^{0.5}}$, height = 0.1, width = 0.5, $C = 0.1, \gamma = 0.5$ with fixed

bandwidth $b_n = 1$

2.3.3.13 case 168 $\eta_n = \frac{1}{(n+5)^{0.5}}$, height = 0.1, width = 2, $C = 0.1, \gamma = 0.5$ with fixed

bandwidth $b_n = 1$

2.3.3.14 case 169 $\eta_n = \frac{1}{(n+5)^{0.5}}$, height = 0.1, width = 8, $C = 0.1, \gamma = 0.5$ with fixed

bandwidth $b_n = 1$

2.3.3.15 case 170 $\eta_n = \frac{1}{(n+5)^{0.5}}$, height = 0.1, width = 15, $C = 0.1, \gamma = 0.5$ with fixed

bandwidth $b_n = 1$

2.3.3.16 case 171 $\eta_n = \frac{1}{(n+5)^{0.5}}$, height = 0.1, width = 0.1, $C = 0.1, \gamma = 0.5$ with fixed

bandwidth $b_n = 1$

2.3.3.17 case 172 $\eta_n = \frac{1}{(n+5)^{0.5}}$, height = 0.1, width = 2, $C = 0.1, \gamma = 0.1$ with fixed

bandwidth $b_n = 1$

2.3.3.18 case 173 $\eta_n = \frac{1}{(n+5)^{0.5}}$, height = 0.1, width = 2, $C = 0.1, \gamma = 0.5$ with fixed

bandwidth $b_n = 10$

2.3.3.19 case 174 $\eta_n = \frac{1}{(n+5)^{0.5}}$, height = 0.1, width = 2, $C = 0.1, \gamma = 0.5$ with fixed

bandwidth $b_n = 0.1$

2.3.3.20 case 175 $\eta_n = \frac{1}{(n+5)^{0.5}}$, height = 0.1, width = 2, $C = 0.1, \gamma = 0.5$ with

decreasing bandwidth $b_n = \frac{1}{n^{0.2}}$

2.3.3.21 case 176 $\eta_n = \frac{1}{(n+5)^{0.8}}$, height = 0.1, width = 2, $C = 0.1, \gamma = 0.5$ with fixed

bandwidth $b_n = 1$

2.3.4 kernel function $K(x) = \begin{cases} 0, & |x| \geq 2 * width \text{ or } |x| \leq width \\ 1, & width < |x| < 2 * width \end{cases}$

2.3.4.1 case 177 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 0.5, $C=1, \gamma=2$ with fixed bandwidth $b_n = 1$

2.3.4.2 case 178 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 0.5, $C=10, \gamma=2$ with fixed bandwidth $b_n = 1$

2.3.4.3 case 179 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 0.5, $C=0.1, \gamma=2$ with fixed bandwidth $b_n = 1$

2.3.4.4 case 180 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 2, $C=1, \gamma=2$ with fixed bandwidth $b_n = 1$

2.3.4.5 case 181 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 0.1, $C=1, \gamma=2$ with fixed bandwidth $b_n = 1$

2.3.4.6 case 182 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 0.5, $C=1, \gamma=0.5$ with fixed bandwidth $b_n = 1$

2.3.4.7 case 183 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 0.5, $C=10, \gamma=0.5$ with fixed bandwidth $b_n = 1$

2.3.4.8 case 184 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 0.5, $C=0.1, \gamma=0.5$ with fixed bandwidth $b_n = 1$

2.3.4.9 case 185 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 2, $C=1, \gamma=0.5$ with fixed bandwidth $b_n = 1$

2.3.4.10 case 186 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 8, $C=1, \gamma=0.5$ with fixed bandwidth $b_n = 1$

2.3.4.11 case 187 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 15, $C=1, \gamma=0.5$ with fixed bandwidth $b_n = 1$

2.3.4.12 case 189 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 0.1, $C=1, \gamma=0.5$ with fixed bandwidth

$$b_n = 1$$

2.3.4.13 case 190 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 0.5, $C=1, \gamma=0.5$ with fixed bandwidth

$$b_n = 10$$

2.3.4.14 case 191 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 0.5, $C=1, \gamma=0.5$ with fixed bandwidth

$$b_n = 0.1$$

2.3.4.15 case 192 $\eta_n = \frac{1}{(n+5)^{0.5}}$, width = 0.5, $C=1, \gamma=0.5$ with decreasing bandwidth

$$b_n = \frac{1}{n^{0.2}}$$

2.3.4.16 case 193 $\eta_n = \frac{1}{(n+5)^{0.8}}$, width = 0.5, $C=1, \gamma=0.5$ with fixed bandwidth

$$b_n = 1$$

2.3.5 constant $\sigma(x) = C$

2.3.6 varfact comparison

2.3.7 variance comparison

2.3.8 comparison of average squared jump distance

2.4 Example 4: normal distribution in R^2

2.4.1 kernel function $K(x) = e^{-\frac{|x|^{\alpha_1}}{\alpha_2}}$

2.4.1.1 case 194 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 1, \alpha_2 = 1, C=1, \gamma=2$ with fixed bandwidth $b_n = 1$

2.4.1.2 case 195 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 0.5, \alpha_2 = 1, C=1, \gamma=2$ with fixed bandwidth

$$b_n = 1$$

2.4.1.3 case 196 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 0.5, \alpha_2 = 1, C = 0.1, \gamma = 2$ with fixed bandwidth

$$b_n = 1$$

2.4.1.4 case 197 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 2, \alpha_2 = 1, C = 0.1, \gamma = 2$ with fixed bandwidth

$$b_n = 1$$

2.4.2 kernel function $K(x) = \frac{1}{1 + \alpha_1 |x|^{\alpha_2}}$

2.4.2.1 case 198 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 1, \alpha_2 = 1, C = 1, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.4.2.2 case 199 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $\alpha_1 = 1, \alpha_2 = 1, C = 0.1, \gamma = 1$ with fixed bandwidth

$$b_n = 1$$

2.4.2.3 case 200 $\eta_n = \frac{1}{(n+5)^{0.8}}$, $\alpha_1 = 1, \alpha_2 = 1, C = 0.1, \gamma = 2$ with fixed bandwidth

$$b_n = 10$$

2.4.3 kernel function $K(x) = \begin{cases} 0, & |x| \geq 2 * width \\ -1 * height, & width < |x| < 2 * width \\ 1 * height, & |x| \leq width \end{cases}$

2.4.3.1 case 201 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $height = 0.5, width = 0.5, C = 0.01, \gamma = 10$ with fixed

bandwidth $b_n = 1$

2.4.4 kernel function $K(x) = \begin{cases} 0, & |x| \geq 2 * width \text{ or } |x| \leq width \\ 1, & width < |x| < 2 * width \end{cases}$

2.4.4.1 case 202 $\eta_n = \frac{1}{(n+5)^{0.5}}$, $width = 5, C = 0.01, \gamma = 2$ with fixed bandwidth $b_n = 1$

2.4.5 constant $\sigma(x) = C$

2.4.6 varfact comparison

2.4.7 variance comparison

2.4.8 comparison of average squared jump distance