Group 1 Member: PENG ZHICHAO, LI JINZHAO, WONG KI YAN, YUEN CHUN WING, CHEUNG SIU FUNG, WONG CHUN FAN

STAT3011 Computational Statistics Part 1

Group1 Final Report

The following report answer 4 questions that related to root finding, genetic algorithm, quadrature, Monte Carlo integration and Markov Chain Monte Carlo sampling. Detail steps to solve the problems as well as the result analysis will be provided.

1.1 Question 1:

The objective of question 1 is to find the maximal value of the $f(x) = \frac{\log(x+x^2)}{1+x^3}$. A function is at its maximum when the slope of its tangent is equal to zero and its graph caves downward, mathematically equivalent to $f'(x) = \frac{\frac{(1+2x)(1+x^3)}{x+x^2} - 3x^2 * \log(x+x^2)}{(1+x^3)^2} = 0$ and $f''(x) = \frac{(x(6(2x^3-1)*\log(x(x+1)) - 14x^3 + 8x^2 - 9x - 6) - 1) - 1)}{(x+1)^3(x^2-x+1)^3} < 0$. Hence it is reduced to a root-finding problem. Although several methods can be applied to solve the equation, bisection method and newton's method are the focus of this question.

1.2 Methods

1.2.1 Method 1: Bisection method

Bisection method can be applied to find the root. It is a popular method to estimate roots of an equation. The theory of the method works as follows: For a and b, such that f is continuous on [a, b] and $f(a) * f(b) < 0$, then by intermediate value theorem f(c) = 0 for some c in which c lies between a and b. let $c = \frac{a+b}{2}$. If $f(a) * f(c) < 0$, f(x) = 0 for some x for a < x < c, then reset b = c. If $f(c) * f(b) < 0$, similarly f(x) = 0 for some x for c < x < b, then reset a = c. Repeat the process until the error is small enough or a maximum number of iterations is reached if convergence is slow (Bisection method, 2018).

1.2.2 Method 2: Newton's method

Another method that is commonly used in root-finding is newton's method. It involved a more advanced theory as follows: Let $x_n = x_{n-1} + a_{n-1}$. Using Taylor's approximation, $f(x_n) = f(x_{n-1} + a_{n-1}) \approx f(x_{n-1}) + f'(x_{n-1})\, a_{n-1}$. Setting $f(x_n) = 0$ implies that $a_{n-1} = -\frac{f(x_{n-1})}{f'(x_{n-1})}$ and $x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$. Guess a starting point x_0 and calculate the next point x_1. The process is repeated until is the error is small enough or the maximum number of iteration is reached. For this case f'(x) = 0 is the target, $a_{n-1} = -\frac{f'(x_{n-1})}{f''(x_{n-1})}$ is modified accordingly (Newton's method, 2018).

1.3 Results and Discussion

To make an educated guess of the starting point(s), the graph of the function is plotted. It is clearly shown that the x corresponding to the maximal value lies between 0.5 and 1.5. Therefore, those points are used as starting points for bisection method and 1 is used for newton's method. Moreover, 10^{-10} is set as stopping criteria. The result is shown in the following table.

Table1.1

Method	Starting point(s)	x	$f(x)$	Number of iterations
Bisection method	(0.5, 1.5)	1.123312	0.3595798	30
Newton's method	1	1.123312	0.35957988	5
Error	N/A	10E-10	N/A	N/A

The efficiency of the methods is evaluated via examining the convergence of the methods. Since the runtime is abysmal and difficult to measure, the number of iterations needed to reach the same accuracy is measured instead. It should be noted that the starting point estimation will affect effectiveness and the comparison is not entirely fair. It is astonishing that both method is effective in solving this problem as only 5 and 30 iterations are needed to reach an accuracy of 10^{-10} for bisection method and newton's method respectively.

The notable difference between the two methods is that the latter is more difficult to implement as it requires the calculation of second derivative which, in general, can be complicated. Therefore, it is more convenient to use bisection for easy question. However, newton's method is still the more powerful and efficient method which is proved excellent in achieving high accuracy.

2.1 Question 2:

For question 2, a genetic algorithm is implemented to maximize the function with n=10.

$$f(\vec{x}) = \begin{cases} \left| \dfrac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|, & if\ (\forall i,\ 0 \le x_i \le 10)\ and\ (\prod_{i=1}^{n} x_i \ge 0.75) \\ 0, & Otherwise\ (i.e.not\ feasible) \end{cases}$$

Before entering the cycle of crossover, an initial population is generated. Function *ppcode* is used to generate an initial population. As binary encoding is used, *decimal* function is used to generate a series of 14 binary numbers for each x_i (C=14); *gfunction* function is used to convert the 14 binary numbers into a decimal number with 4 decimal places ranging from 0 to 10. *gfunction* is based on the formula,$a_1 + \frac{a_2 - a_1}{2^d - 1}\ decimal(b)$,where a_1=0, a_2=10, d=14. (Givens & Hoeting, 2013, p.79) By applying *ppcode* function, which consists of *decimal* and *gfunction*, the x1 and x10 in the population are ensured to satisfy the criteria that product of x1 to 10 are larger than or equals to 0.75, otherwise another new x_1 to x_{10} are generated. The population size is 21 (P=21), as we have to choose the population P to satisfy C≤P≤2C. (Givens & Hoeting, 2013, p.79) *ppcode(21)* gives a 210x14 matrix which consists of all the binary code of the initial population.

Next, we process to the selection of parents. Based on the idea behind genetic algorithm, "fitness" is calculated before the selection of parents. *fitness* is used to calculate the "fitness" by substituting x_1 to x_{10} into the formula before ranking. Built- in function rank is used. As function rank gives result in ascending order, the 21th is the one with the greatest fitness. Next, *rangroup* function and *rangroup6* function are performing the tournament selection. (Take x_1 as an example, all the x_1 in the whole population is randomly divided into 7 subgroups, among each subgroup, the greatest fitness will be selected, in this way, 7 parents are selected. This process is repeated for 6 times and get 42 parents in total. They are then randomly group in pair, i.e. 21 pairs. These 21 pairs can generate 21 new x_1. The process also applies to x_2, x_3 and so on. (Givens & Hoeting, 2013, p.81) Another method of selection is Roulette wheel selection, *Nextgen3*, greater the fitness, larger the probability to be selected. (Givens & Hoeting, 2013, p.80)

After that, *Nextgen* function is about the process of crossover, crossover is taken place within the pair selected before on the binary numbers. 2-cut points will be performed, one is between the 4^{th} and 5^{th} digits; another is between 10^{th} and 11^{th}. The central part (the 5^{th} to 10^{th} digits) of one of the parents is replaced by that of another parent. After having a new sequence of binary code, mutation is taken place. *Mutation* function assigns a random number to each binary digit and those smaller than 0.02, which is the mutation rate (r=0.02), will either change from 0 to 1, or from 1 to 0. *Finstep* function is used to check whether the next generation satisfy the criteria that products of x_1 to x_{10} are larger than or equals to 0.75, if the product of x_1 to x_{10} are less than 0.75, the individual will be disposed and a new individual will be generated by *Nextgen3*. Within *Finstep*, there is a *check* function, *check* is to check whether there is any missing value in the new generation as we encountered the problem of generating missing value sometimes. Until there is no missing value in the whole population matrix, the next generation is successfully generated. *GA* function is a huge loop to repeat the whole procedure starting from the selection of parents. For example, *GA(1000)* means performing the above procedures for 1000 generations. Within each generation, *retur* function is used to find out the maximum result and its corresponding x_1 to x_{10}, the values are then kept in another separate matrix for later use. *Result* function is used to return the maximum fitness value with corresponding x_1 to x_{10} among numbers of generations, which is as well as the result of the question.

2.2 Results and Discussion

The best result we got are shown below:

Table2.1

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
3.1636	3.1227	3.0025	2.9128	1.4649	0.3131	0.3577	0.4010	0.3516	0.3772
f(x) = 0.7441926									

Several problems appear when coding in R. (All the functions mentioned above in italic forms are user-defined functions.) The first one is that after the selection, some of the ranks of selected parents will be the same. In R, the rank will appear decimal number which cannot be recognized by the matrix function. Therefore, floor function is used here to convert the decimal rank into integer. Besides, the rank is no longer the non-repeated integers from 1 to 21, which requires us to redefine the set of ranks. The second problem is that the *mutation* function will return missing value from time to time for no reason. R studio is used to confirm the specific position of the problem since the temporary value of variables before crash can be seen. Mutation function is set to assign a probability to each digit to change from either 0 to 1 or from 1 to 0 and theoretically, all the digits should still be 0 or 1. However, somehow the mutation function returns a group of missing value frequently. Since the true reason is unknown, the *check* function is written to overcome this problem by finding the position of missing value in the outcome of *mutation* function and replace the part by conducting *mutation* to the original matrix again until no missing value appears. The last problem appears at *GA* function. Actually, *GA* is a large loop to repeat the whole procedure happened in one generation n times. One truth is that in the loop, once a variable is operated, only the new outcome will be kept unless we maintain the value of the variable first. (For example, C and D are two variables, if we write C = *function* (D), only C is stored where D becomes a name with no real meaning. Therefore, if both C and D are required, the coding should be: B = D; C= *function* (D) and now B contains the value of D and C has the operated result.)

One significant difference between our method and the method used by most students is that we use binary encoding, which means we need to operate two matrices meanwhile: one is the binary matrix, the other is the real-value matrix. We use binary matrix to do the crossover and mutation and use the real-value matrix to calculate the fitness in order to select parents. Because of the existence of two systems, the program runs relatively slowly. However, since the crossover and mutation are natural, the result obtained from the program is rather stable and can be gradually improved along the increment of the generations. (n=1000, the result is around 0.69; n=10000, around 0.71; n=100000, around 0.74) One obvious aspect which can be improved is the size of population, the choice taken now

is 1.5C, where C is the length of chromosome (in this case, C=14), actually larger n can improve the efficiency by obtaining the result with the same accuracy with fewer generations.

3.1 Question 3

Implement both a quadrature method and a Monte Carlo method to estimate the integral:

$$\int_0^{+\infty} \frac{|\cos x|}{x} e^{-(\log(x)-3)^2} dx$$

The function of the integral decreases as x increases. **(Graph 3.1)** We will estimate this integral by Monte Carlo Integration and two quadrature rules to examine the estimation difference and also analyze the error and running time so as to compare the performance of the two rules. **(Graph 3.2)**

3.2 Methods

3.2.1 Monte Carlo Integration

3.2.1.1 The idea behind Monte Carlo Integration

The big idea behind Monte Carlo Integration is the (weak) Law of Large Numbers which states that X_bar_n converges in probability towards the μ=E(X). (Carlberg, C., 2014) We take advantage of the convergence charateristic to sample randomly from a selected distribution, and then estimate the population mean which is the integral we would like to estimate by obtaining the sample average.

3.2.1.2 The Procedure of Monte Carlo Integration

The procedure of Monte Carlo Integration is firstly selecting a random variable X with p.d.f. f(x) with the domain equivalent to that of g(x) and stimulate n values from this favour distribution. In order to maximize the accuracy, we have select 5e6 as our sample size. In this case, we have chosen X following Lognormal distribution with mean 3 and variance 1 due to the domain from 0 to positive infinity and the p.d.f. similar to the integral.

Next, transform the integral I to involve f(x) inside the integral as the next show: $I: \int g(x)dx = \int \left[\frac{g(x)}{f(x)}\right] f(x)dx = E[\frac{g(x)}{f(x)}]$. Our transformation of the integral: $\int_0^{+\infty} \frac{|\cos x|}{x} e^{-(\log(x)-3)^2} dx =$ $\int_0^{+\infty} \sqrt{2\pi}|\cos x| e^{-\frac{1}{2}(\log(x)-3)^2-x} (\frac{1}{x\sqrt{2\pi}} e^{-\frac{(\log(x)-3)^2}{2}}) dx$ The final step is to calculate the expected value by sample mean to find out the integral estimation. As we mentioned above, the Law of the Large number has a stronger power as the sample size n increases. We can estimate the error of the standard deviation of the integral divided by the square root of the sample size (i.e. $\frac{Var(I)}{\sqrt{n}}$)

3.2.2 Quadrature

Since the function is too complicated to find the antiderivative in closed form, and the strong computing power allows to compute a series of sum and substitution in a short running time, we shall carry out numerical integration to find the integral. Quadrature is a skill to estimate the integral by calculating the sum of the area bound of every subinterval. We will apply the Composite Simpson's Rule and Composite Trapezoidal Rule in this question.

3.2.2.1 The Difference of the Two Rules

the Composite Simpson's Rule and Composite Trapezoidal Rule also compute the integral estimation by the sum of the subintervals. Yet, the area under the function in each subinterval is approximated by the area of the trapezoids while Simpson's Rule approximates the area in a quadratic polynomial. Simpson's Rule also considers

the area of the parabolic arc of each subinterval, different to the trapezoidal rule treats the line connecting $[(f(x_i), f(x_{i+1})]$ as a straight line. (Givens, G., & Hoeting, J., 2013) **(Graph 3.3)**

3.2.2.2 The Generalized formula of two Composite Rules

Composite Simpson's Rule: $\frac{h}{3}(\mathrm{f}(a) + 2\sum_{j=1}^{\binom{n}{2}-1} f(x_{2j}) + 4\sum_{j=1}^{\binom{n}{2}} f(x_{2j-1}) + f(x_n)),\ n\ is\ an\ even\ integer.$
(Graph 3.4) (Lindfield, G., & Penny, J., 2012).
Composite Trapezoidal Rule: $T(f,h) = \frac{h}{2}(f(a) + f(b)) + h\sum_{j=1}^{n-1} f(x_j)$ (**Graph 3.5**), where the domain is from a to b, n is the number of subintervals, h is the length of each subinterval (i.e. h= (a+b)/2).

3.2.2.3 The Error Analysis of the Two Rules

Composite Trapezoidal Rule: $\int_a^b f(x)\,dx - \frac{h}{2}\sum_{i=0}^{n-1}(f(x_i) + f(x_{i+1})) = \frac{-(b-a)*f"(Error)}{12}h^2$
Composite Simpson's Rule: $\int_a^b f(x)\,dx - S_n = \frac{(b-a)*f""(Error)}{2880}h^4$, where f'(Error) and f""(Error) equal to the average of the derivative of the subintervals. (Lindfield, G., & Penny, J., 2012)

From the formula, we found that the truncation error can be reduced by reducing the step size h and by using the higher-order integration formula of the order of $O(h_2)$, $O(h_4)$, and so on (i.e. Composite Simpson's Rule). For example, if the step size is reduced by half, the global truncation error of the composite trapezoidal rule is reduced by four.

3.2.2.4 Possible Problems and Our Approaches

The infinity bound is a possible problem in numerical integration, the statistical programmes will return to error for calculation of sum to infinity. We have chosen two approaches to solve this problem. Because of the decreasing y as x increases, the effect of y tends to 0 in the positive infinity bound, we compromise a little accuracy, maximize the bound according to the performance of the computer. The second approach is to transform the curve in order to change to a calculable interval by substitution. Here we choose to substitute u equaling to 1/(x+1). Both approaches will be conducted by the two rules in order to compare the difference in this function. We also find that setting x as the initial value may encounter error due to the cosine function involved. So, we add an extremely small adjustment value to the initial value x to prevent the error from occurring.

3.2.2.5 Parameters of Substitution & Maximization of the bound

The transformed function: $Let\ u = (x+1)^{-1}\ \int_0^{+\infty} \frac{|cos\,x|}{x} e^{-(log(x)-3)^2} dx = \int_0^1 \frac{|cos\frac{1-u}{u}|}{u(1-u)} e^{-(log\left(\frac{1-u}{u}\right)-3)^2} du$
Due to the computing power of our testing instruments, we set h to 0.005. The domain is from 0.005 to 1 after transformation **(Graph 3.6)**. For the method of maximization of the bound. we set h to 5000 and the bound is from 1e-7 to 1e7 **(Graph 3.9)**.

3.3 Results and Discussion

3.3.1 Interesting Finding in Monte Carlo Integration

The numerical result will be shown in the appendix. **(Table 3.1)** We find something interesting when we compare the result of exponential distribution and lognormal distribution. The result reveals that running time and error sampling from exponential distribution are much higher than that of lognormal distribution. Therefore, in this case exponential distribution is not a good method to deal with this function. The possible reason behind is exponential distribution performs bad as a sampling distribution in this function. The samples cannot simulate the function and fall into wrong data points under exponential distribution.

3.3.2 The Interesting Finding in Quadrature

Here we only discuss the fact behind the data. The numerical result is attached to the appendix of this report. **(Table 3.2 & Table 3.3)**

For the rate of convergence between the two methods, substitution apparently performs better than maximization of bound, whatever in integral estimation and error analysis by the two rules, in term of the running time. It is probably due to the complexity of the large number (10e7 numbers ranging from 10e-5 to 10e6) calculation. However, the high speed doesn't bring this method a better estimation. The error analysis reveals that substitution makes a larger error than that of maximization.

About the performance of the two rules in this question, we compare them by the running time & error. Both the tests show that the composite Simpson's rule has a relatively small efficiency and accuracy than the composite trapezoidal rule in maximization of the bound, but a reverse result shown in substitution, although the references suggest that the model of Simpson's rule is the improvement of trapezoidal rule.

The possible reason behind is due to the number of parabolic curve lie in each subinterval, in the function transformed by substitution, more parabolic curve lies in each subinterval **(Graph 3.7 & Graph 3.8)**, and this situation will be disadvantageous to use trapezoidal rule.

The higher error in the Composite Simpson's rule in maximization of bound is probably due to the significant decrease in x= 150. **(Graph 3.1)** The Simpson's rule estimates the area in consideration of the quadratic curve while the curve shows characteristics similar to a straight line which causes a higher error than trapezoidal rule which treats function as a line. The integral concentrating in x=0 to x=150 and tending to 0 afterwards while our first subinterval exceeds x=150 to x=5000. **(Graph 3.10)** The curve performs like a steep peak in the first subinterval. Simpson's Rule will estimate the parabolic arc of this subinterval in quadratic method which leads to a higher error than the Trapezoidal Rule does.

4.1 Question 4:

Calculate the correlation between x & y of the following distribution:

$$p(x,y) \propto sin(\pi x)sin^{20}(\pi x^2) + sin(\pi y)sin^{20}(2\pi y^2),\ 0 \le x,y \le 1$$

Intuitively, there is two ways to tackle the question. The first way is by using the definition of correlation combined with Monte Carlo Integration. By the definition of correlation, $Corr(X,Y) = \frac{Cov(X,Y)}{\sqrt{Var(X)Var(Y)}} = \frac{E(XY)-E(X)E(Y)}{\sqrt{(E(X^2)-E(X)^2)(E(Y^2)-E(Y)^2)}}$ (Pearson correlation coefficient, 2018). Therefore, one can simply calculate $E(XY), E(X), E(Y), E(X^2), E(Y^2)$ separately using Monte Carlo integration and obtain the correlation of x and y by the expanded formula. The second way of solving the problem is to sample points (x, y) from target distribution *p* and calculate correlation of the sampled points. Below are the some further elaboration of the methods and some brief recaps of the sampling methods.

4.2 Methods

4.2.1 Method 1: Use the definition of correlation

Let's denote $f(x,y) = sin(\pi x)sin^{20}(\pi x^2) + sin(\pi y)sin^{20}(2\pi y^2)$. Since *p(x, y)* is proportionate to the *f(x, y)*, it implies that *p(x, y) = c*f(x, y)*, where *c* is a constant such that c*$\int_0^1 \int_0^1 f(x,y)dxdy = 1$. To find c, we need to find the integral of $f(x,y)$. Denotes $f(x) = sin(\pi x)sin^{20}(\pi x^2)$ and $f(y) = sin(\pi y)sin^{20}(2\pi y^2)$. Notice that *f(x, y) = f(x) + f(y)* and hence $\int_0^1 \int_0^1 f(x,y)dxdy = \int_0^1 f(x)dx + \int_0^1 f(y)dy$. By expanding $E(XY), E(X), E(Y), E(X^2), E(Y^2)$, we have the followings:

E(XY) = c*$\int_0^1\int_0^1 xyf(x,y)dxdy$=c*($\int_0^1\int_0^1 xyf(x)dxdy + \int_0^1\int_0^1 xyf(y)dxdy$)= c*($\frac{1}{2}\int_0^1 xf(x)dx + \frac{1}{2}\int_0^1 yf(y)dy$)

E(X) = c*$\int_0^1\int_0^1 xf(x,y)dxdy$=c*($\int_0^1 xf(x)dx + \int_0^1 xdx\int_0^1 f(y)dy$)= c*($\int_0^1 xf(x)dx + \frac{1}{2}\int_0^1 f(y)dy$)

E(X^2)= c*$\int_0^1\int_0^1 x^2f(x,y)dxdy$=c*($\int_0^1 x^2f(x)dx + \frac{1}{3}\int_0^1 f(y)dy$) (Similar for *E(Y),* E(Y^2))

In summary, we need to calculate the following 6 integrals in order to compute *Corr(X, Y)*:

$$\int_0^1 f(x)dx\,,\int_0^1 f(y)dy,\int_0^1 xf(x)dx,\int_0^1 yf(y)dy,\int_0^1 x^2f(x)dx\,,\int_0^1 y^2f(y)dy$$

Since the range of the integration is from zero to one, these six integral can be easily calculated using Monte Carlo integration with a sampling distribution follows Uniform [0, 1].

4.2.2 Method 2: Sample (x, y) by Two-dimensional Metropolis Algorithm

Two-dimensional Metropolis-Hasting Algorithm sample a pair x and y at the same time from a joint distribution. Proposed new points will be accepted with acceptance rate $\boldsymbol{A}(\boldsymbol{x}'|\boldsymbol{x}) = \frac{\boldsymbol{P}(\boldsymbol{x}')}{\boldsymbol{P}(\boldsymbol{x})}\frac{\boldsymbol{g}(\boldsymbol{x}|\boldsymbol{x}')}{\boldsymbol{g}(\boldsymbol{x}'|\boldsymbol{x})}$ (Givens & Hoeting, 2013). Notice that the aforementioned constant *c* appears on both $\boldsymbol{P}(\boldsymbol{x}')$ and $\boldsymbol{P}(\boldsymbol{x})$, so it cancels out itself and won't affect the sampling process. Meanwhile, $\frac{\boldsymbol{g}(\boldsymbol{x}|\boldsymbol{x}')}{\boldsymbol{g}(\boldsymbol{x}'|\boldsymbol{x})}$ can be canceled out when the proposal distribution is symmetric, and it is now called Metropolis Algorithm. Therefore, a bivariate normal distribution $\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})$, with $\boldsymbol{\mu} = \begin{bmatrix}x\\y\end{bmatrix}, \boldsymbol{\Sigma} = \begin{bmatrix}\mathbf{1} & \mathbf{0}\\ \mathbf{0} & \mathbf{1}\end{bmatrix}$ has been chosen as the proposal distribution. The variance-covariance matrix was chosen to ensure the acceptance rate is maintain at the level of around 20% to 30%. If the acceptance rate is too high, it means that the steps are too small which will slow down the sampling process; if acceptance rate is too low, it is hard for the points to converge to high probability region, which also slow down the process.

4.2.3 Method 3: Sample (x, y) by Gibbs sampling with rejection method.

Gibbs sampling use the idea of conditional distribution to reduce the high dimensional sampling problem to a one dimensional sampling problem. It samples one point at a time from conditional distribution, $p(x_j'|x_1,\dots,x_{j-1},x_{j+1},\dots,x_n)$, with acceptance rate equal to 1. Since $p(x_j'|x_1,\dots,x_{j-1},x_{j+1},\dots,x_n) = \frac{p(x_1,\dots,x_n)}{p(x_1,\dots,x_{j-1},x_{j+1},\dots,x_n)} = \frac{p(x_1,\dots,x_n)}{\int p(x_1,\dots,x_n)dx_j}$, again, c cancels out itself. The conditional distribution that we need to sample from can be further derived as $p(x|y) = \frac{p(x,y)}{\int_0^1 p(x,y)dx} = \frac{f(x)+f(y)}{\int_0^1 f(x)dx+f(y)}$, $p(y|x) = \frac{p(x,y)}{\int_0^1 p(x,y)dx} = \frac{f(x)+f(y)}{\int_0^1 f(y)dy+f(x)}$, where *f(x)* and *f(y)* are defined the same as in method 1. To sample from the one dimensional conditional distribution $p(x|y)$ and $p(y|x)$, rejection method is used in method 3. The idea of rejection method is to find a distribution with pdf g, which can be easily sampled from, such that $g \geq f$ for all x in the sampling domain. Sample x from g(x) with acceptance rate $a = \frac{f(x)}{g(x)}$, and the sampling distribution will approximate to *f(x)* (Givens & Hoeting, 2013). For a target distribution with maximum value *max(f(x)),* it can be generalized that the acceptance rate $a = \frac{f(x)}{max(f(x))*g(x)}$, where *g(x)* is a uniform distribution *U[a, b]* covering the sampling domain. *max(f(x))* is calculated using bisection method.

4.2.4 Method 4: Sample (x, y) by Gibbs sampling with Metropolis Algorithm.

Similar to method 3, method 4 utilize the idea of Gibbs sampling but with different approach to sample from the conditional distribution. Here, Metropolis Algorithm is used and the proposal distribution is uniform distribution from zero to one for simplicity.

4.3 Results and Discussion

Table 4.1 shows the results of constant c and 6 integrals calculated using Monte Carlo Integration comparing with the result obtained by the integral package "cubature" in R. To obtain the error of around 1E-05, number of sampled points from Uniform [0, 1] is set to be 1E8.

Table 4.1

Method	$\int_0^1 f(x)dx$	$\int_0^1 f(y)dy$	$\int_0^1 xf(x)dx$	$\int_0^1 yf(x)dy$	$\int_0^1 x^2f(x)dx$	$\int_0^1 y^2f(y)dy$	c
Monte Carlo Integration	0.099893	0.108082	0.069533	0.061786	0.048643	0.037361	0.208975
R Integration package	0.099885	0.10907	0.069536	0.061774	0.048654	0.037736	0.208956
Error (absolute)	9.9E-04	3.0E-06	1.2E-05	1.1E-05	3.8E-04	1.9E-05	9.9E-04

For Method 2 to 4, the length of each chain was set to be n=1E5, and we repeat the procedure for m=100 times to obtain 100 chains. b=100 points are burnt to make sure tha chain starts in a high probability region (burnt-in). Mean and standard deviation of the correlation of 100 chains and the total running time are recorded. Detailed scatter plots and autocorrelation plots can be found in the appendix 4. Decaying ACF indicates that the chains are stationary, which satisfy the condition of MCMC. Table 4.2 below summarize the results.

Table 4.2

Method		Mean	Standard Deviation	Error (absolute)	Running Time	Average Acceptance Rate
Correlation Definition	R Integral Package	-0.06096	N/A	True Value	1.503 secs	N/A
	Monte Carlo Integration	-0.06101	**0.000789**	5.0E-05	**3.487secs**	N/A
Two-dimensional Metropolis (thin=5)		-0.05990 (-0.06023)	0.005303 (0.006171)	1.1E-03 (7.3E-04)	5334.4secs	0.279
Gibbs Sampling	Rejection Method	-0.06105	<u>0.002222</u>	<u>9.0E-05</u>	3024.5secs	1
	Metropolis (thin=5)	-0.06042 (-0.06071)	0.002932 (0.005857)	5.4E-04 (2.5E-04)	<u>273.4secs</u>	

Note: 1. Assume the result obtained by R Integral Package is the true value.
2. The results in the blanket are corresponding to the same chain after thinning.
3. Bolded result is the best among all methods and the underlined result is the best among all MCMC methods.

From the table, it can be seen that the correlation definition approach are much faster in terms of running time. It's not surprising for this result because most of the effort lay in the hand computation to expand the correlation formula and derive the integrals that need to be calculated.

Observing the lower part of Table 2 where MCMC sampling methods are applied, it can be concluded that Gibbs Sampling is faster than Two-dimensional Metropolis Algorithm. The reason behind is that sampling directly from a high dimensional distribution might be slower than sampling from a low dimensional distribution. Gibbs Sampling with Rejection Method produced the smallest confidence interval among the three, implies that it gives a more accurate answer and indeed, its answer is the closest to the true value. However, Gibbs with Rejection is too slow. Among all the MCMC sampling methods, Gibbs sampling combining with Metropolis Algorithm is the most efficient one, which produced similar result as other with much less amount of time.

Especially, when inspecting the autocorrelation plot (see appendix 4), it can be noticed that Metropolis Algorithm has a higher autocorrelation than Rejection method. Autocorrelation can be reduced by thinning the chain by 5, i.e. select a point from every 5 points of the original chain to construct a new chain. The mean of correlation gets closer to the true value at the cost of a higher standard deviation.

Reference

Bisection method. (2018, March 3). Retrieved March 04, 2018, from https://en.wikipedia.org/wiki/Bisection_method

Carlberg, C. (2014). Statistical analysis : Microsoft Excel 2013.

Givens, G. H., & Hoeting, J. A. (2013). *Computational statistics*. Hoboken (New Jersey): Wiley.

Lindfield, G., & Penny, J. (2012). Numerical Methods: Using MATLAB (3rd ed.). Elsevier Science.

Newton's method. (2018, March 4). Retrieved March 04, 2018, from https://en.wikipedia.org/wiki/Newton%27s_method

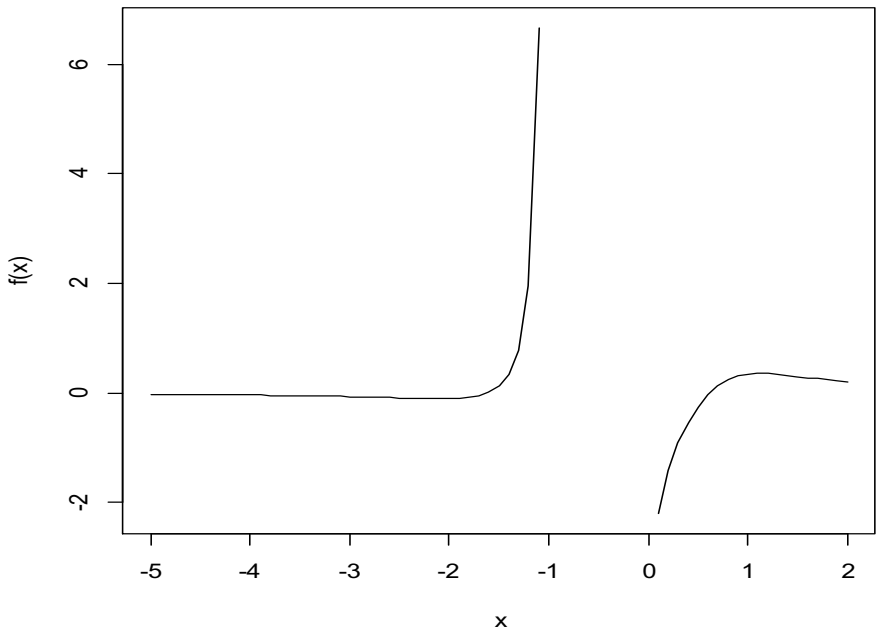
Pearson correlation coefficient. (2018, February 28). Retrieved March 04, 2018, from https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

Sherlock, C., & Roberts, G. (2009). Optimal scaling of the random walk Metropolis on elliptically symmetric unimodal targets. *Bernoulli,15*(3), 774-798. doi:10.3150/08-bej176

STAT3011 Part 1
Group 1 Member: PENG ZHICHAO, LI JINZHAO, WONG KI YAN, YUEN CHUN WING, CHEUNG SIU FUNG, WONG CHUN FAN

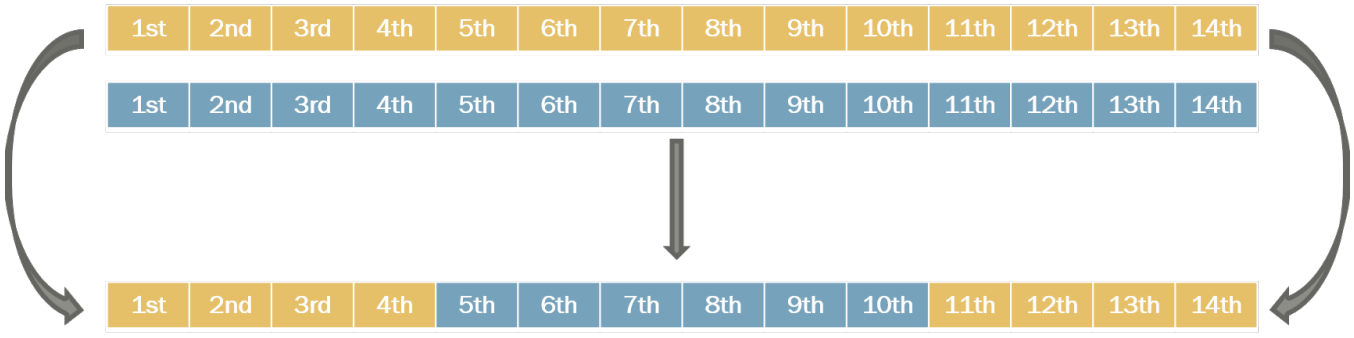
Appendix 1

1.1 Plot of f(x)

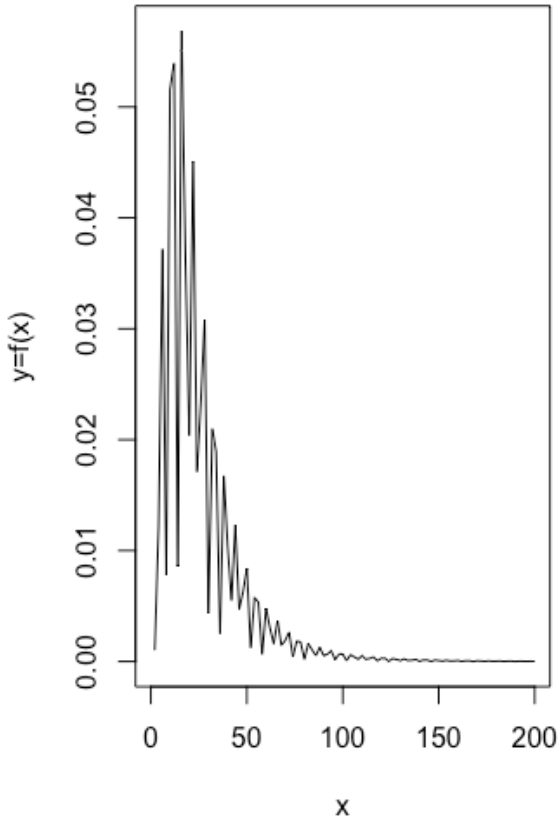


Appendix 2

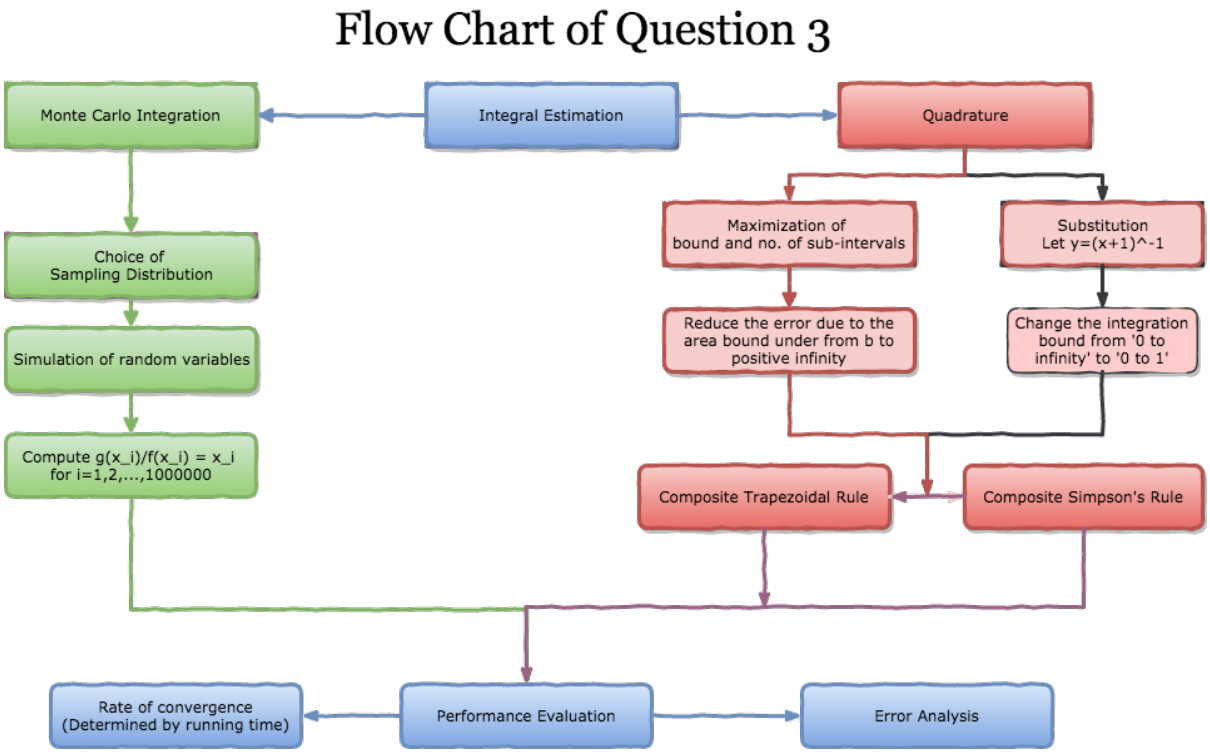
2.1 Crossover



Appendix 3



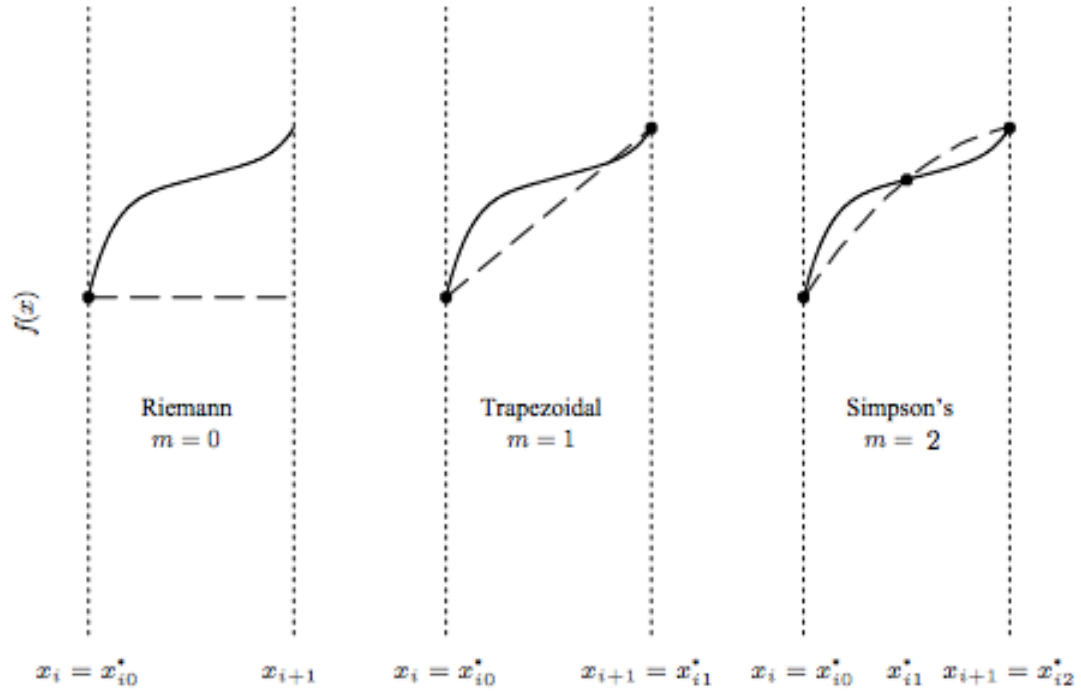
Graph 3.1 Function of Q3



Graph 3.2 Flow of Question 3

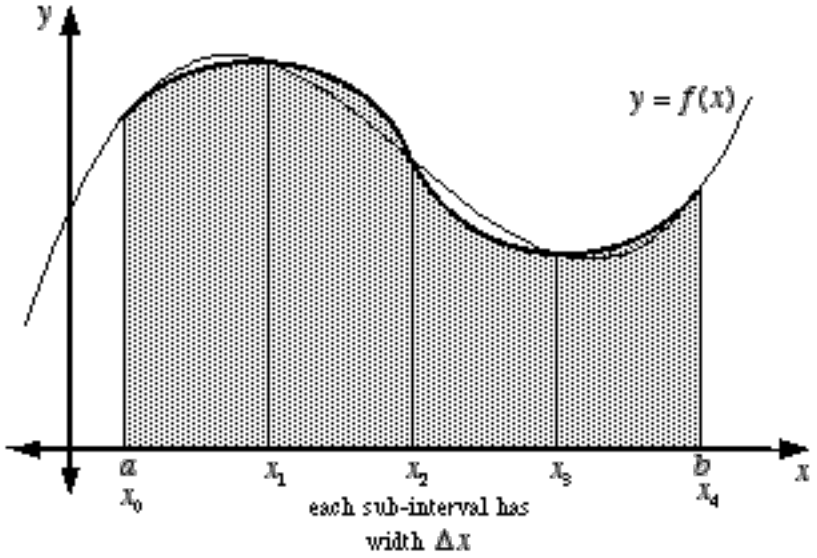
Monte Carlo Integration	**X~Exp(1)**	**X~Lognormal(3,1)**
Integral	0.9318097	1.128507
Error Estimate	0.3014	1.079433*10^(-7)
Running Time(seconds)	0.118232	0.03117204

Table 3.1 Result of the Monte Carlo Integration



Approximation (dashed) to f (solid) provided on the subinterval $[x_i, x_{i+1}]$, for the Riemann, trapezoidal, and Simpson's rules.

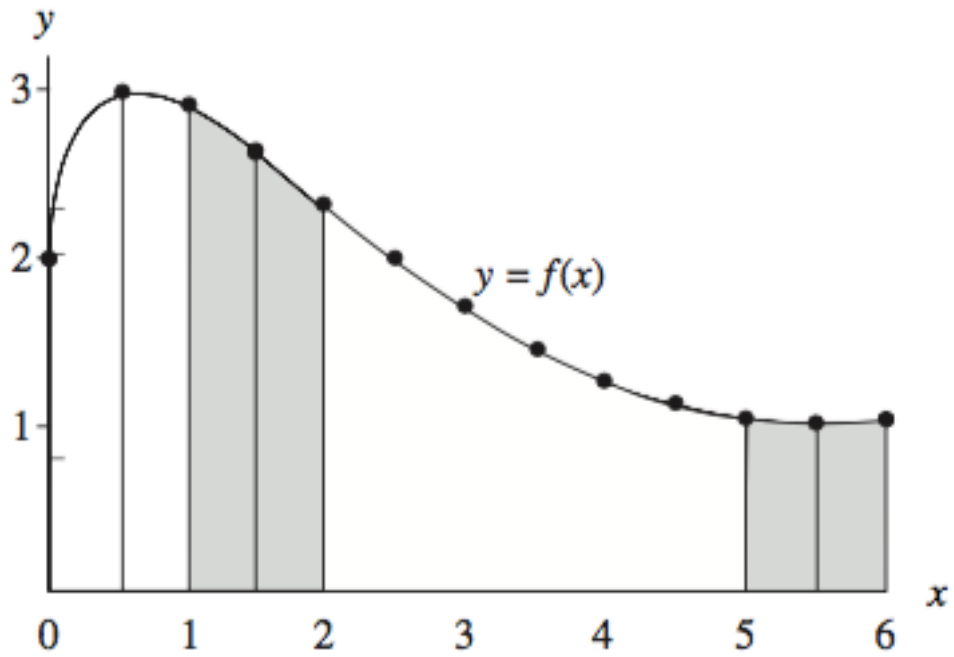
Graph 3.3 Difference between the Rules (Givens, G., & Hoeting, J., 2013)



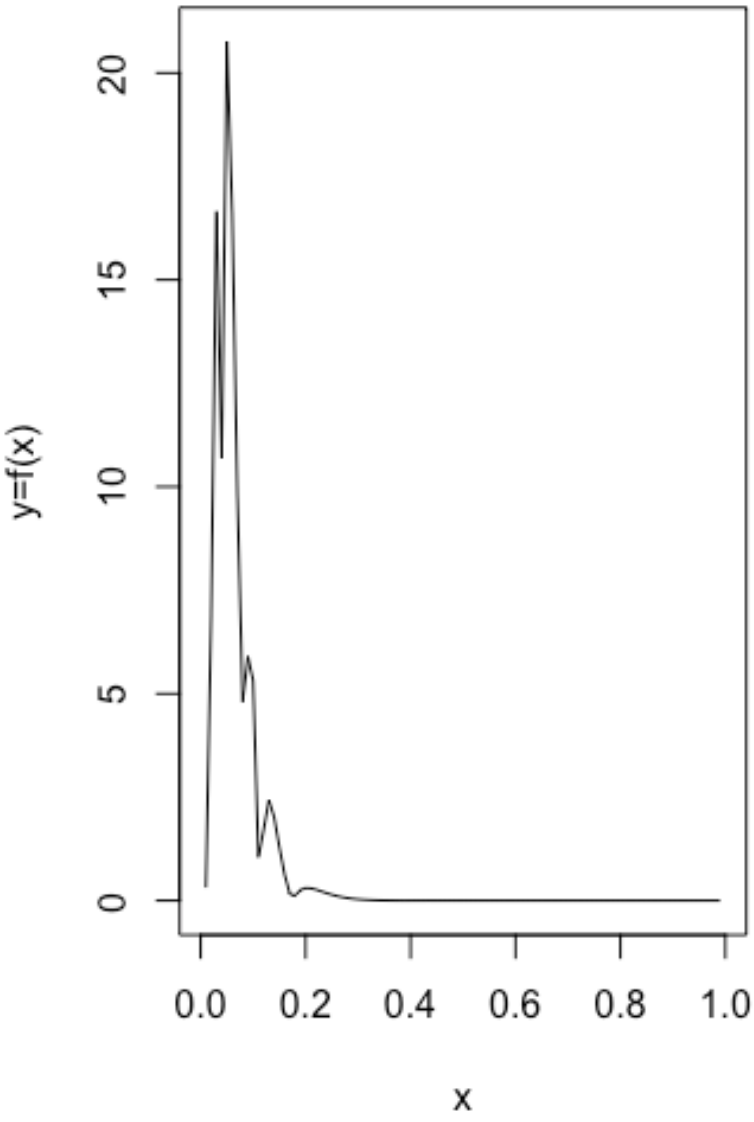
The shaded area bounded by the parabolas (the thicker curves) is approximately equal to the area bounded by $y = f(x)$.

$$\int_a^b f(x)dx \approx \frac{\Delta x}{3}\left[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + f(x_4)\right]$$

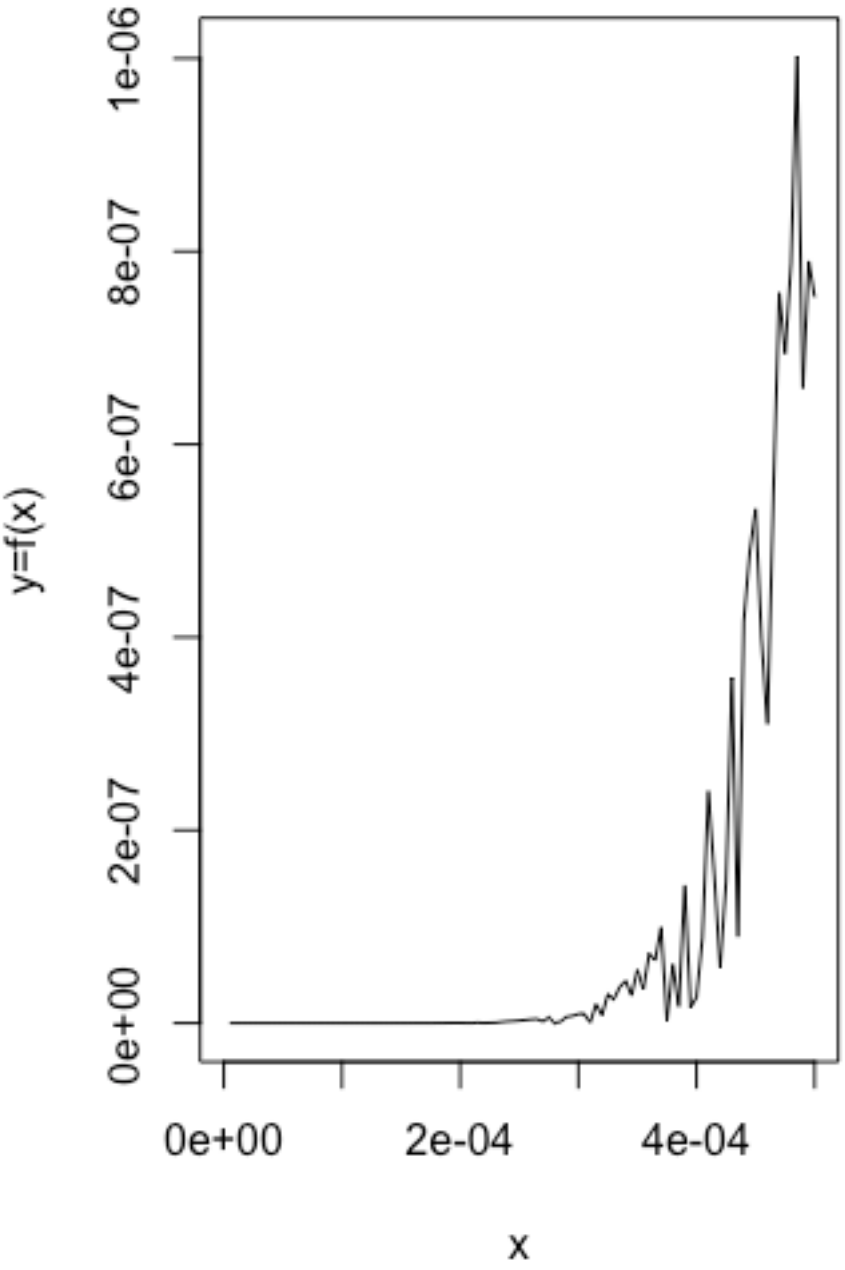
Graph 3.4 Formula and Illustaration of Composite Simpson's Rule



Graph 3.5 Formula and Illustaration of Composite Trapezoidal Rule



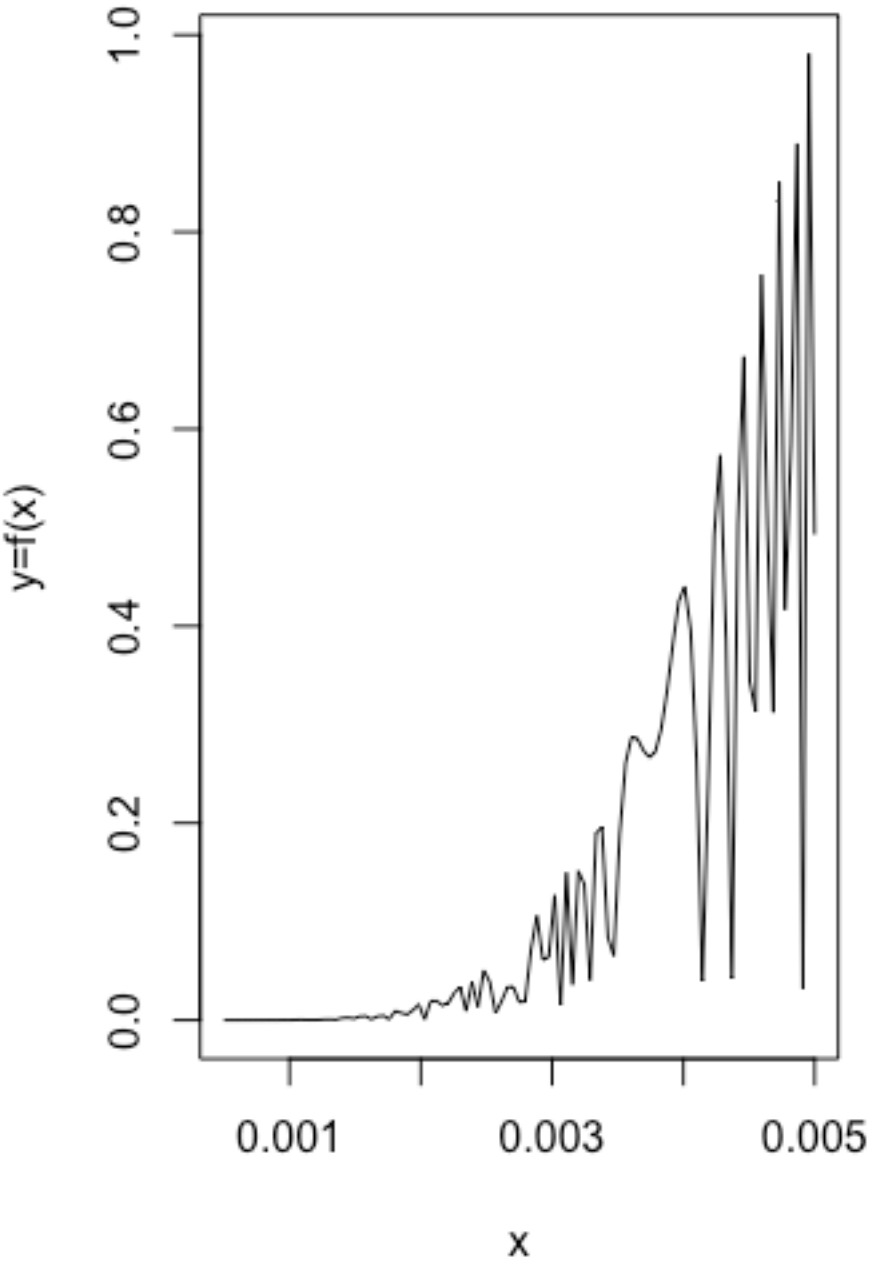
Graph 3.6 Transformed Function from 0 to 1



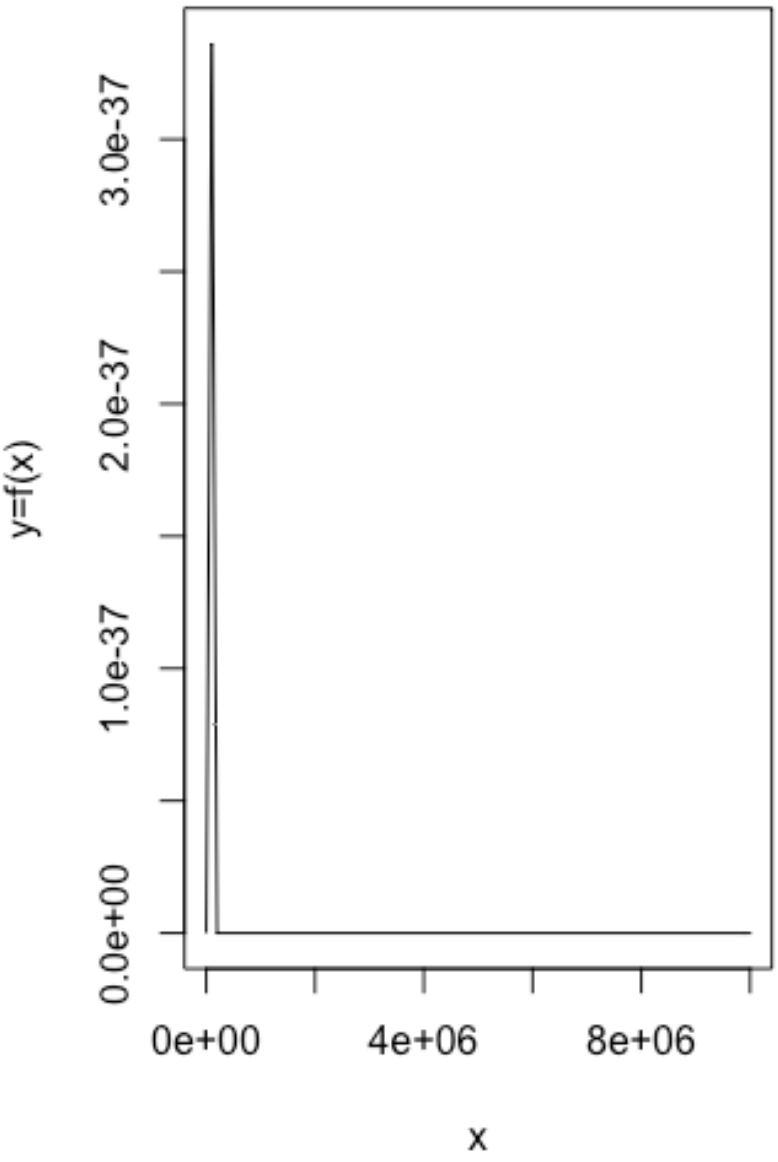
Graph 3.7 The first subinterval of the transformed Function

Substitution	**CTrape**	**CSimp**
Integral	1.123706	1.134448
Sign of the difference of the errors[Csimp-Ctrape]	Negative (i.e. $\text{Error}_{\text{Simpson's}} < \text{Error}_{\text{Trapezoidal}}$)	
Running Time(seconds) **[Integral Estimation]**	0.0001599789	0.0002520084
Running Time(seconds) **[Error Estimation]**	4.244604	7.770465
Running Time(seconds) [Sign of difference]	4.595622	

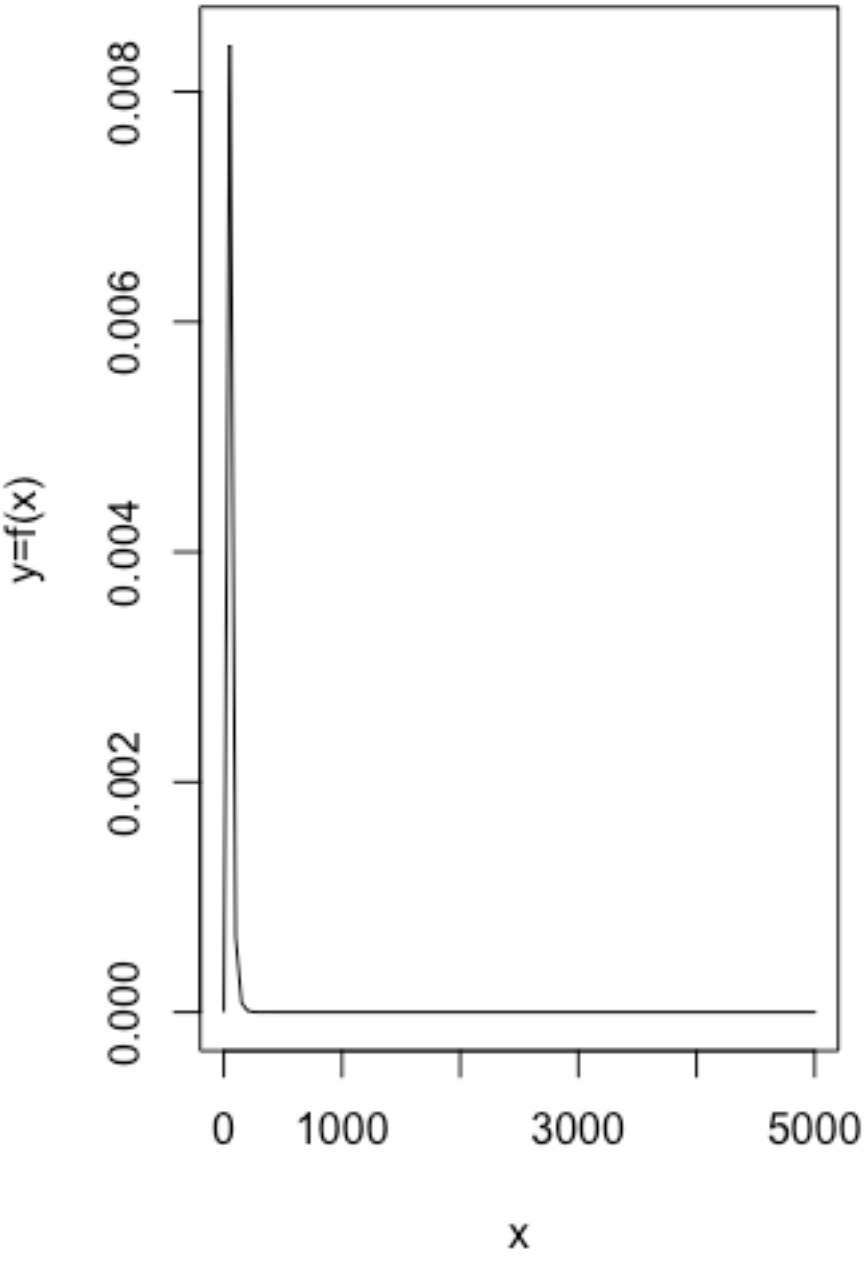
Table 3.2 Result of Quadrature by Substitution



Graph 3.8 The first 10 subintervals of the transformed Function



Graph 3.9 The interval of the function by using maximization of bound



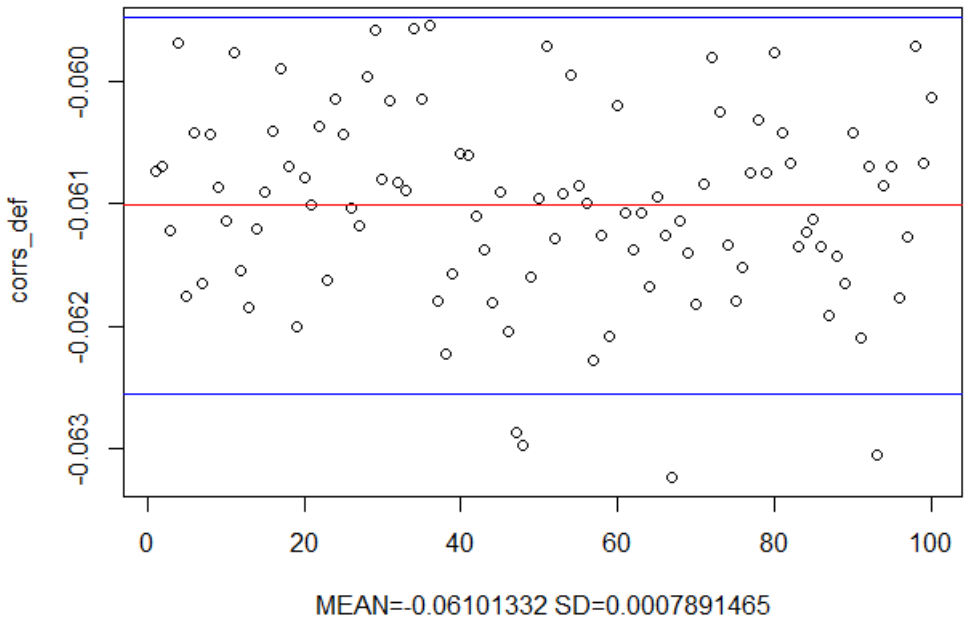
Graph 3.10 The first subinterval of the function by using maximization of bound

	CTrape	**CSimp**
Integral	1.128537	1.128537
Sign of the difference of the errors [Csimp-Ctrape]	Positive (i.e. $\text{Error}_{\text{Simpson's}} > \text{Error}_{\text{Trapezoidal}}$)	
Running Time(seconds)	0.8330648	1.493879
[Integral]		
Running Time(seconds) [Error estimation]	168.617438	278.313097
Running Time(seconds) [Sign of difference]	76.169147	

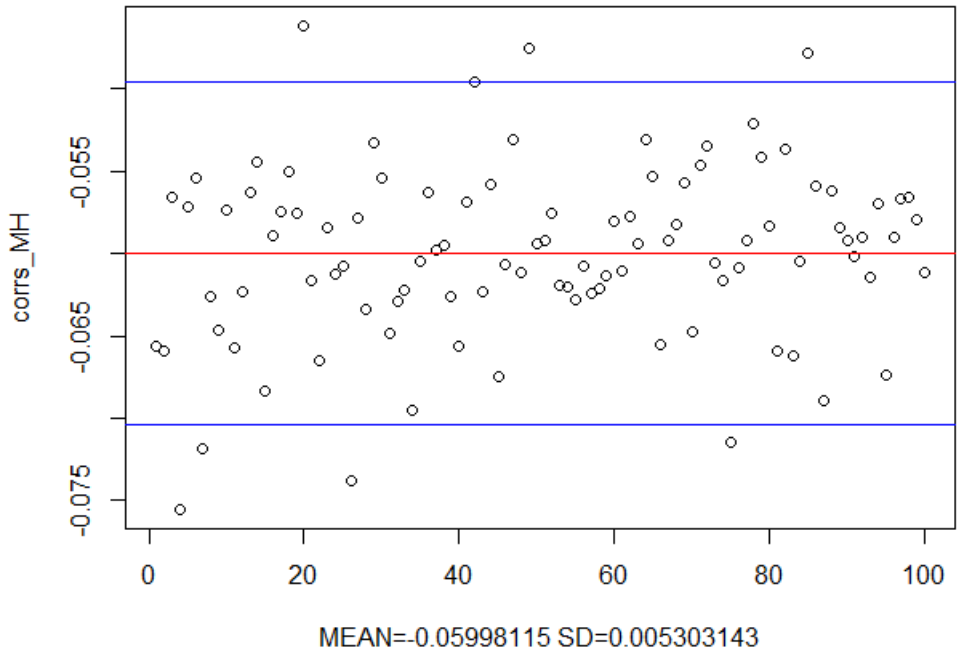
Table 3.3 Result of Quadrature by maximization of the bound

Appendix 4

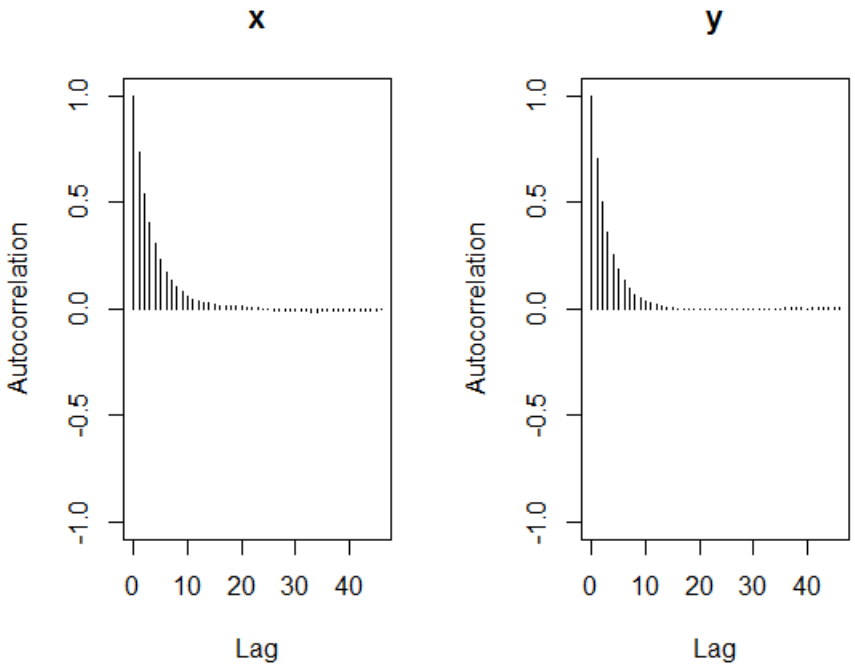
4.1 Scatter Plot for correlation result of Correlation Definition Method



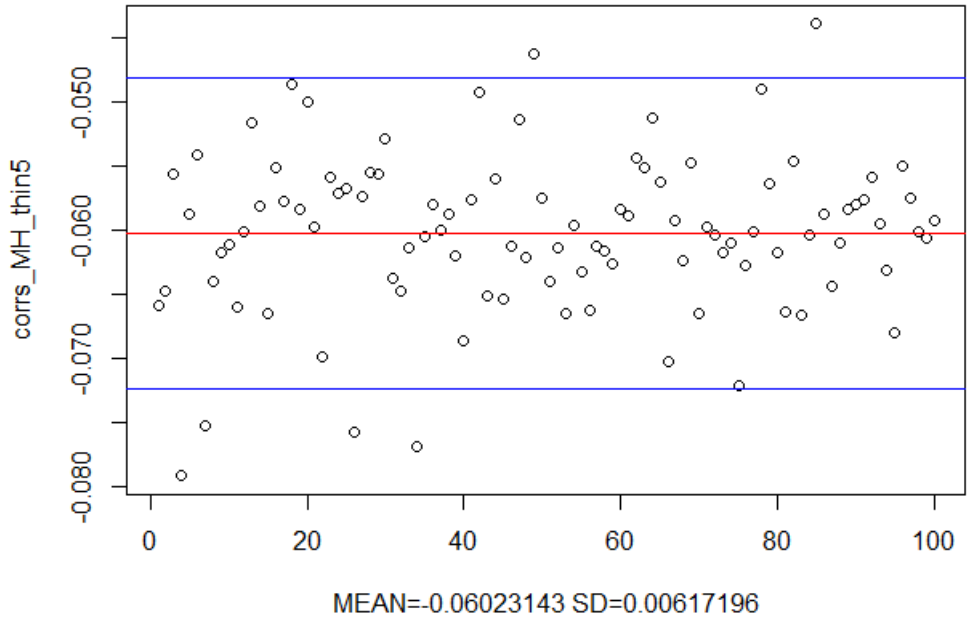
4.2.1.1 Scatter Plot for correlation result of Metropolis Algorithm



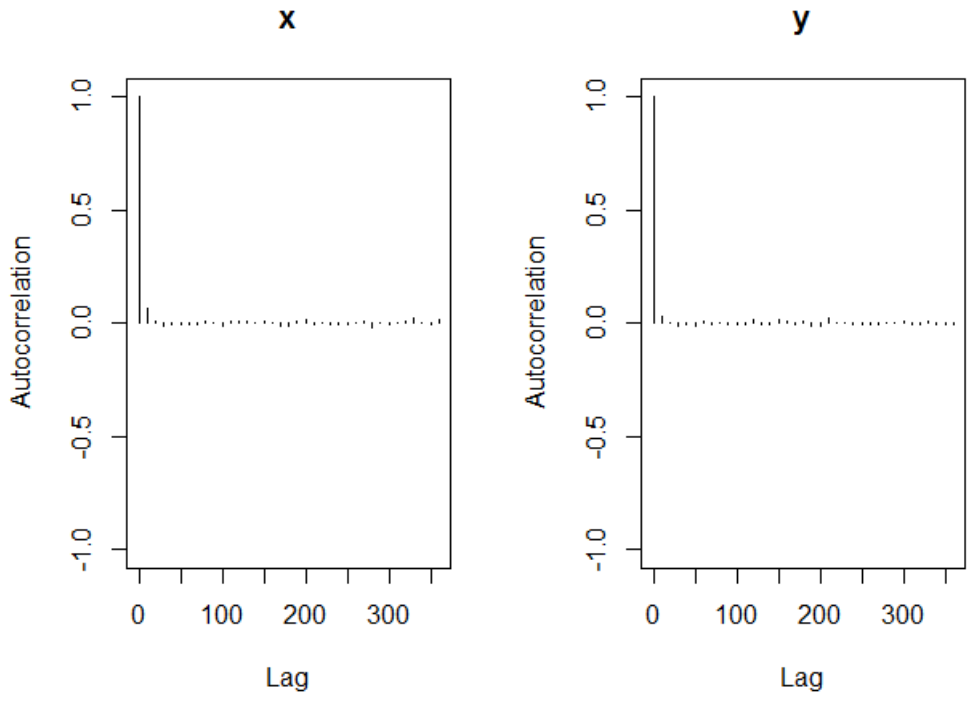
4.2.1.2 Autocorrelation Plot of sampled points from Metropolis Algorithm



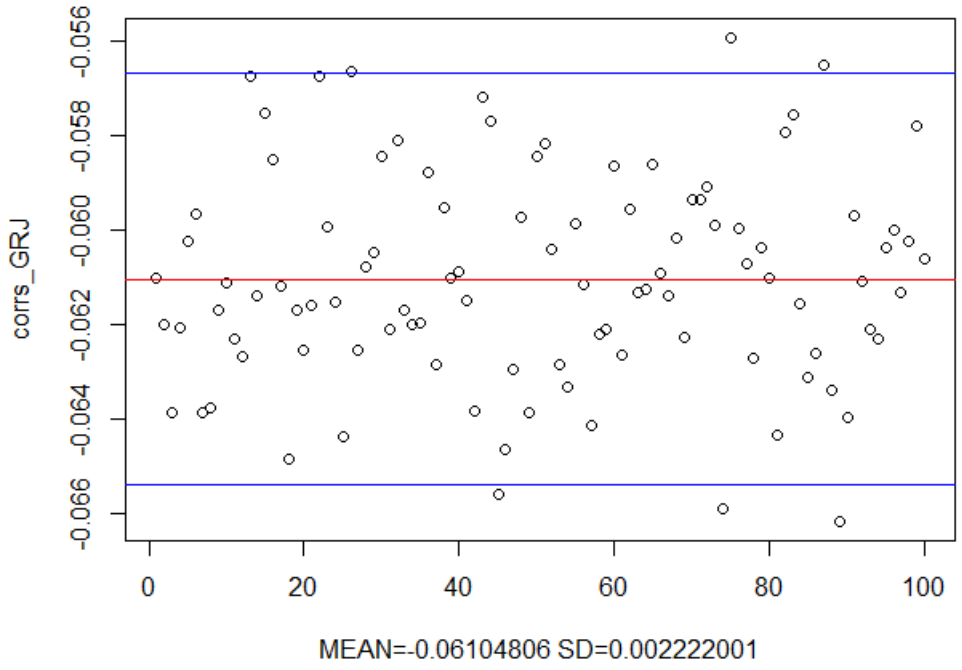
4.2.2.1 Scatter Plot for correlation result of Metropolis Algorithm (thin=5)



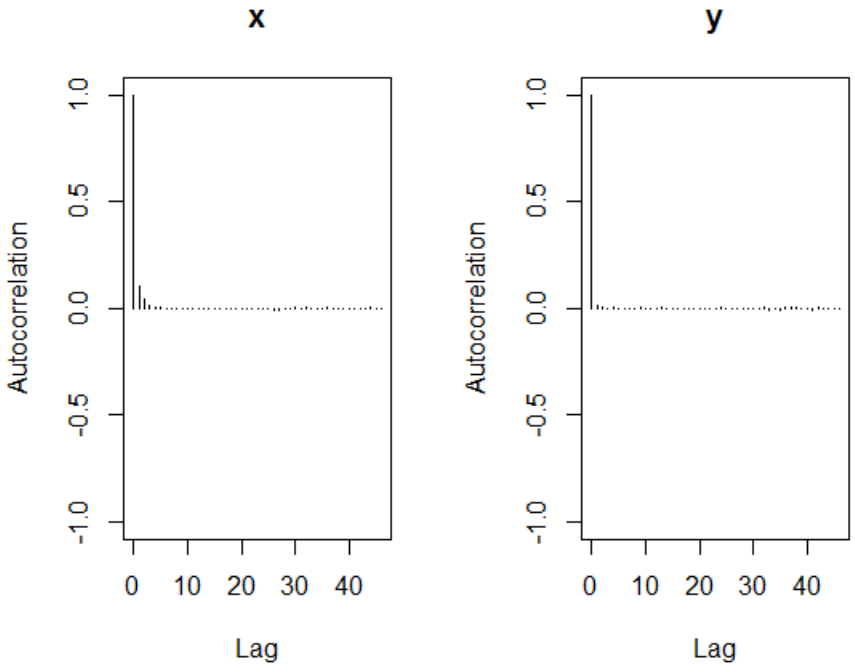
4.2.2.2 Autocorrelation Plot of sampled points from Metropolis Algorithm (thin=5)



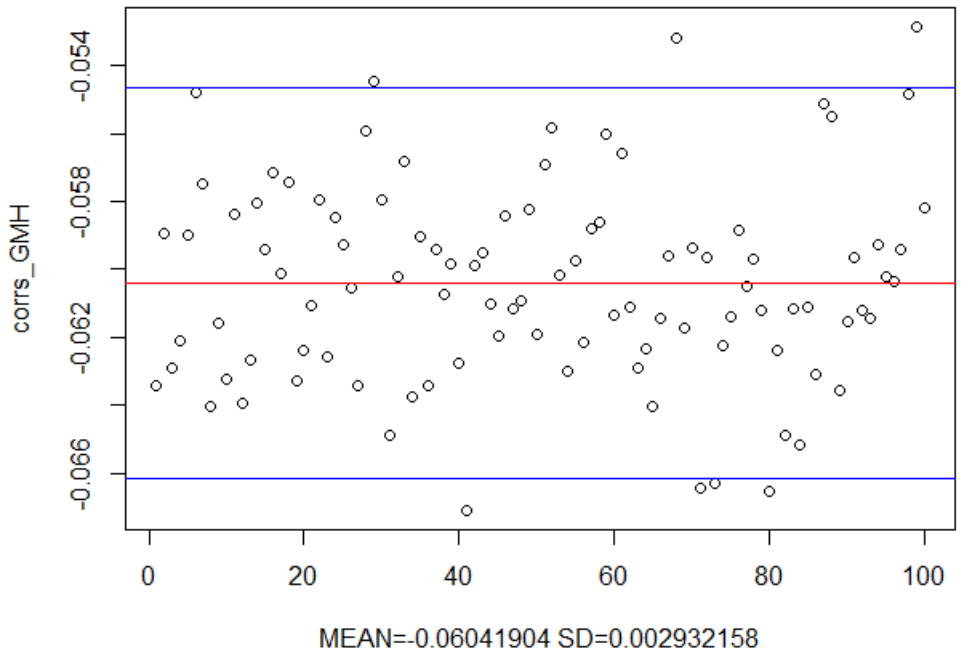
4.3.1 Scatter Plot for correlation result of Gibbs Sampling with Rejection Method



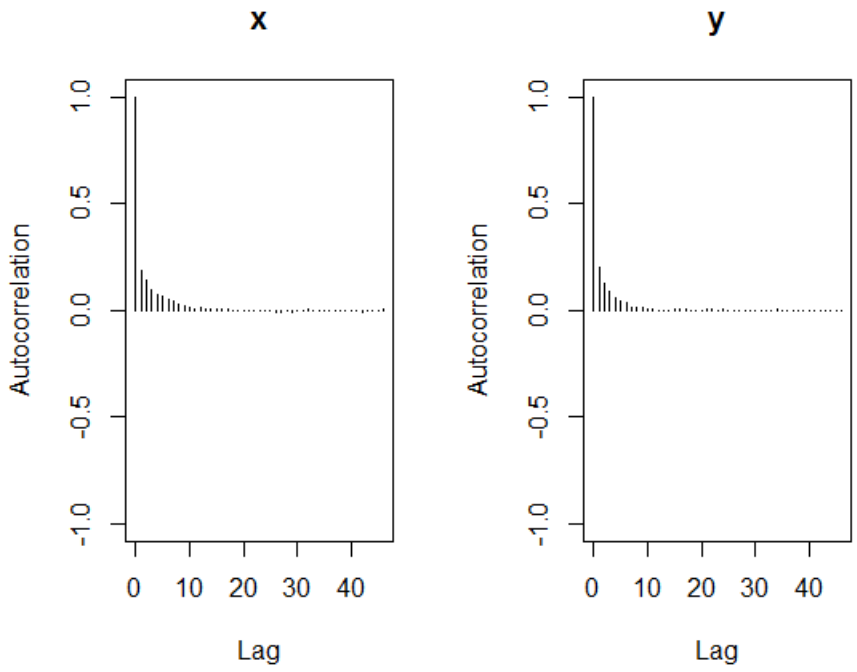
4.3.2 Autocorrelation Plot of sampled points from Gibbs Sampling with Rejection Method



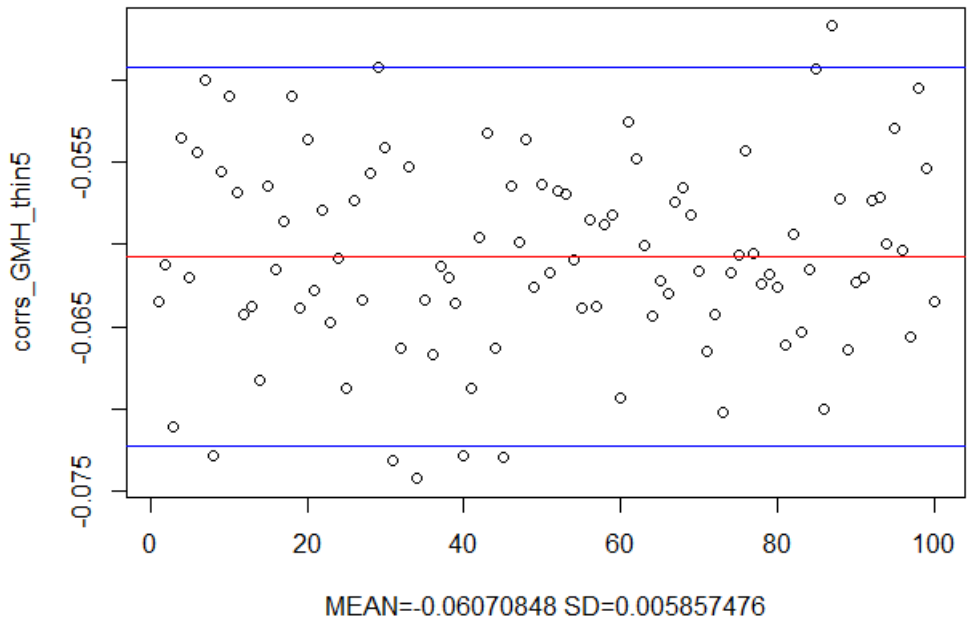
4.4.1.1 Scatter Plot for correlation result of Gibbs Sampling with Metropolis Algorithm



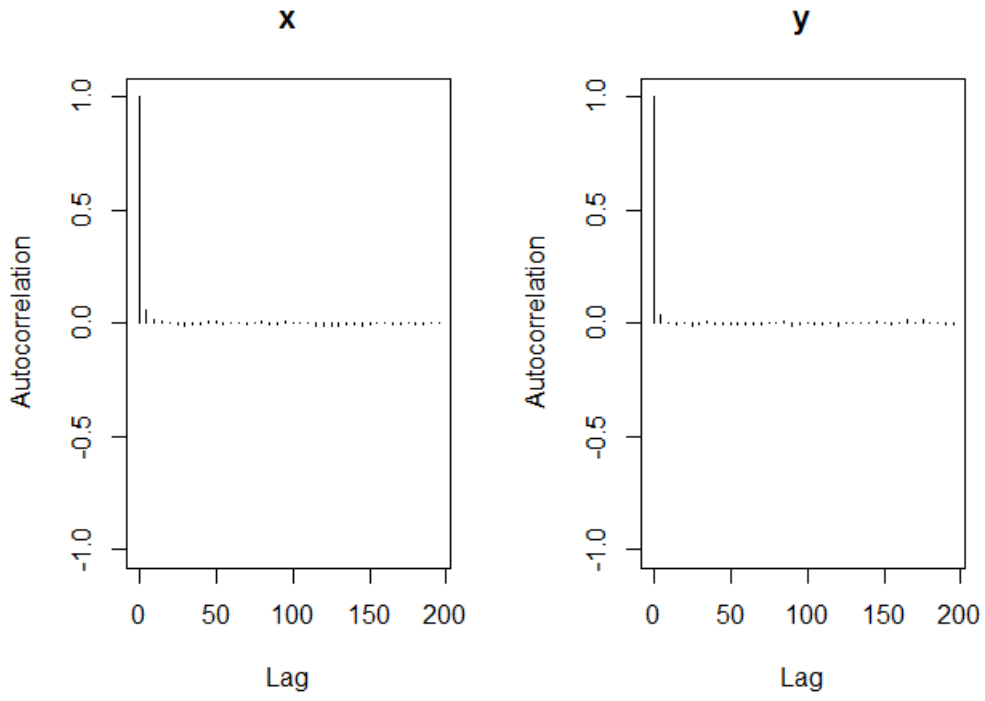
4.4.1.2 Autocorrelation Plot of sampled points from Gibbs Sampling with Metropolis Algorithm



4.4.2.1 Scatter Plot for correlation result of Gibbs Sampling with Metropolis Algorithm (thin=5)



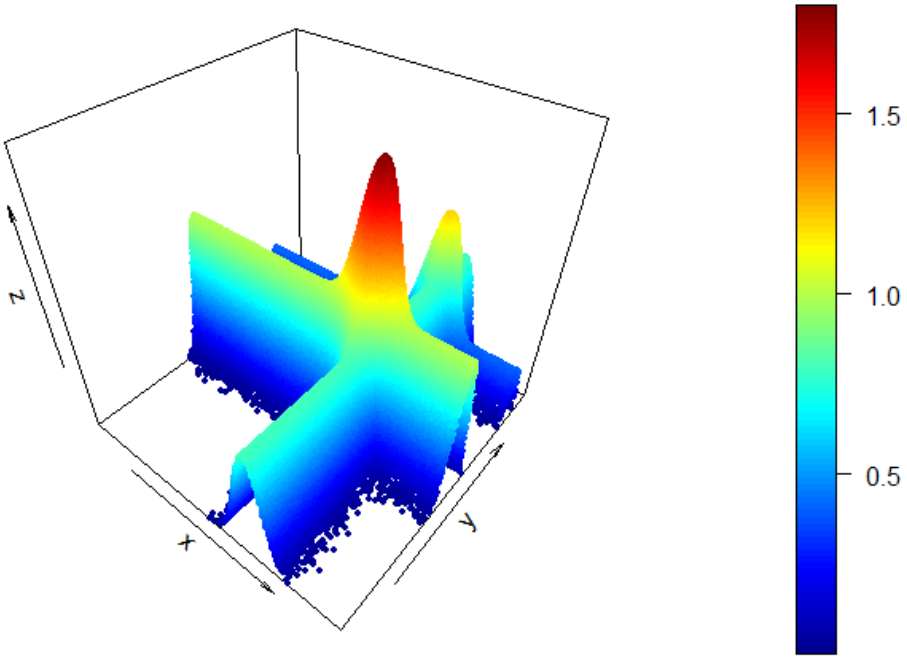
4.4.2.2 Autocorrelation Plot of sampled points from Gibbs Sampling with Metropolis Algorithm (thin=5)



STAT3011 Part 1

Group 1 Member: PENG ZHICHAO, LI JINZHAO, WONG KI YAN, YUEN CHUN WING, CHEUNG SIU FUNG, WONG CHUN FAN

4.5.1 3D Scatter Plot of Sampled Points



4.5.1 2D Density Plot of Sampled Points