STA3127 Statistical Computing

Jeong Geonwoo

DUE Friday, September 15

Q1. A statistical mixture model is widely used for clustering and density estimation. In particular, a d-dimensional Gaussian mixture model has a density $p : \mathbb{R}^d \to (0, \infty)$ of the form

$$p(\cdot) = \sum_{k=1}^{K} \pi_k \phi(\cdot; \mu, \Sigma),\ \sum_{k=1}^{K} \pi_k = 1,\ \pi_k > 0,\ k = 1, ..., K,$$

where $\phi(\cdot; \mu, \Sigma)$ is the density of d-dimensional multivariate Gaussian distribution with mean μ and covariance matrix Σ, given by

$$\phi(x; \mu, \Sigma) = (2\pi)^{-d/2}(\Sigma)^{-1/2} exp(-\frac{1}{2}(x-\mu)'\Sigma^{-1}(x-\mu)),\ x \in \mathbb{R}^d$$

Consider a specific Gaussian mixture model with the following density:

$$p(x) = 0.4\phi(x; \mu_1, \Sigma_1) + 0.6\phi(x; \mu_2, \Sigma_2),\ x \in \mathbb{R}^2$$

where

$$\mu_1 = \begin{pmatrix} 1 \\ 0.5 \end{pmatrix},\ \Sigma_1 = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1.5 \end{pmatrix},\ \mu_2 = \begin{pmatrix} -1 \\ -1 \end{pmatrix},\ \Sigma_2 = \begin{pmatrix} 2 & -0.2 \\ -0.2 & 1 \end{pmatrix}$$

Proceed with the following tasks without using any R package.

i) Evaluating $log\ p(x)$ at $x \in -10 + 2(i-1)/10, i = 1, 2, ..., 101^2$, draw a contour plot of the log density on $[-10, 10]^2$.

ii) Evaluating $log\ p(x)$ at $x \in -50 + (i-1)/10, i = 1, 2, ..., 101^2$, draw a contour plot of the log density on $[-50, 10]^2$.

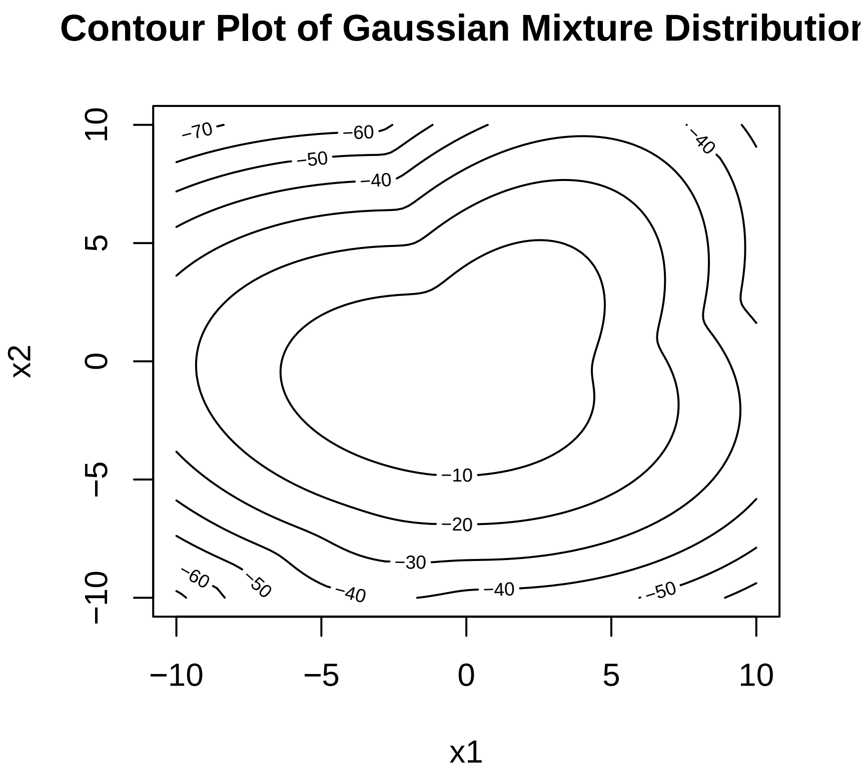
[Note 1: The grid points are defined with 101 points along each coordinate axis, resulting in a total of 10,201 points in two-dimensional space.]

[Note 2: You may encounter an underflow or overflow issue. If so, specify how you bypassed that issue.]

solution

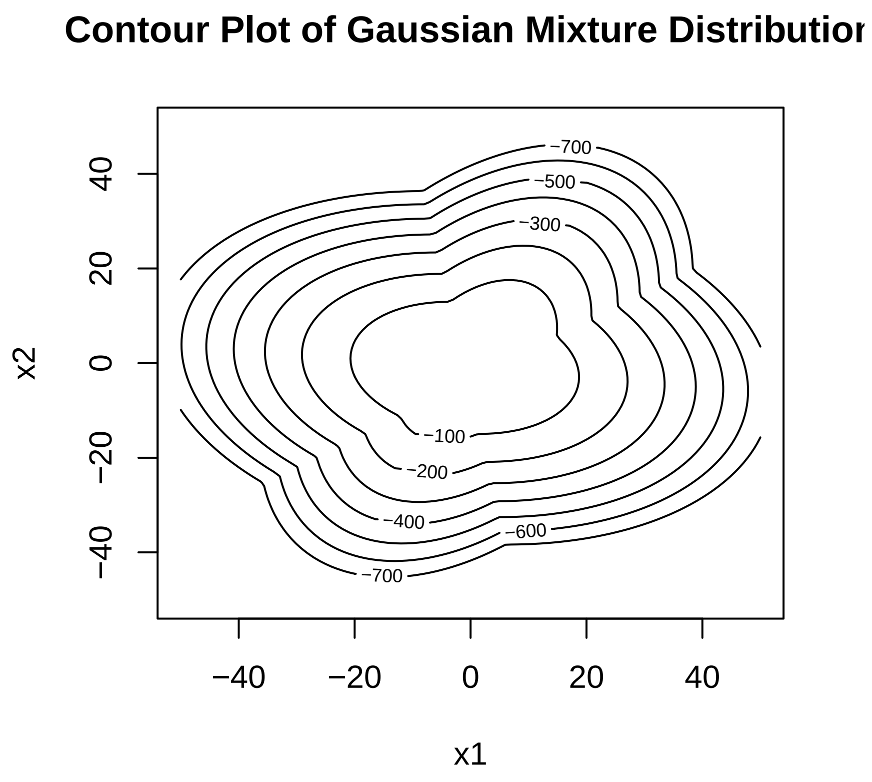
```r
rm(list=ls())
# Parameterization
mu1 <- c(1, 0.5)
mu2 <- c(-1, -1)
sigma1 <- matrix(c(1, 0.5, 0.5, 1.5), nrow = 2)
sigma2 <- matrix(c(2, -0.2, -0.2, 1), nrow = 2)
weights <- c(0.4, 0.6)
```

```r
##### [-10, 10] grid #####

# Setting the grid
data.grid <- expand.grid(x.1 = seq(-10, 10, length.out = 101),
                         x.2 = seq(-10, 10, length.out = 101))

# Defining multivariate normal function
multivariate_normal_pdf <- function(x, mean, cov) {
  d <- length(mean)
  det_cov <- det(cov)
  inv_cov <- solve(cov)
  constant <- 1 / ((2 * pi)^(d/2) * sqrt(det_cov))
  exponent <- -0.5 * t(x - mean) %*% inv_cov %*% (x - mean)
  pdf_value <- constant * exp(exponent)
  return (pdf_value)
}

# Compute Probability
prob1 <- apply(data.grid, 1, function(x) {
  multivariate_normal_pdf(x, mu1, sigma1)
})
prob2 <- apply(data.grid, 1, function(x) {
  multivariate_normal_pdf(x, mu2, sigma2)
})

prob = weights[1] * prob1 + weights[2] * prob2
log_prob = log(prob)

q.samp <- cbind(data.grid, prob = log_prob)
contour(x = unique(q.samp$x.1), y = unique(q.samp$x.2),
        z = matrix(q.samp$prob, ncol = 101),
        main = "Contour Plot of Gaussian Mixture Distribution",
        xlab = "x1", ylab = "x2")
```



```r
##### [-50, 50] grid #####

# Setting the grid
data.grid <- expand.grid(x.1 = seq(-50, 50, length.out = 101),
                         x.2 = seq(-50, 50, length.out = 101))

# Defining multivariate normal function
multivariate_normal_pdf <- function(x, mean, cov) {
  d <- length(mean)
  det_cov <- det(cov)
  inv_cov <- solve(cov)
  constant <- 1 / ((2 * pi)^(d/2) * sqrt(det_cov))
  exponent <- -0.5 * t(x - mean) %*% inv_cov %*% (x - mean)
  pdf_value <- constant * exp(exponent)
  return (pdf_value)
}

# Compute Probability
prob1 <- apply(data.grid, 1, function(x) {
  multivariate_normal_pdf(x, mu1, sigma1)
})
prob2 <- apply(data.grid, 1, function(x) {
  multivariate_normal_pdf(x, mu2, sigma2)
})
```

```r
prob = weights[1] * prob1 + weights[2] * prob2
log_prob = log(prob)

# Drawing the Contour
q.samp <- cbind(data.grid, prob = log_prob)
contour(x = unique(q.samp$x.1), y = unique(q.samp$x.2),
        z = matrix(q.samp$prob, ncol = 101),
        main = "Contour Plot of Gaussian Mixture Distribution",
        xlab = "x1", ylab = "x2")
```



Underflow / Overflow issue ?

```r
prob[432:436]
```

```
## [1]  0.000000e+00  0.000000e+00  0.000000e+00 6.916919e-323 2.890778e-320
```

```r
log_prob[432:436]
```

```
## [1]      -Inf      -Inf      -Inf -741.8010 -735.7657
```

When we use $log\ p(x) = log(0.4\phi(x;\mu_1,\Sigma_1) + 0.6\phi(x;\mu_2,\Sigma_2))$, we encounter an underflow issue. That is, when we define $\phi(x;\mu_1,\Sigma_1)$ and $\phi(x;\mu_2,\Sigma_2)$, in a grid with **very small non-zero** probability values, the probability value is **represented as zero**. Then taking logarithm over p(x) where the underflow issue occurred will result in -Inf.

So we bypass the underflow issue by following the steps below. We assume that

$$a = 0.4 \times \phi(x; \mu_1, \Sigma_1), b = 0.6 \times \phi(x; \mu_2, \Sigma_2)$$

Then,

$$\begin{aligned} log(a) &= log(0.4) - \frac{d}{2}log(2\pi) - \frac{1}{2}log(det(\Sigma_1)) - \frac{1}{2}(x-\mu_1)'\Sigma_1^{-1}(x-\mu_1) \\ log(b) &= log(0.6) - \frac{d}{2}log(2\pi) - \frac{1}{2}log(det(\Sigma_2)) - \frac{1}{2}(x-\mu_2)'\Sigma_2^{-1}(x-\mu_2) \\ log(a+b) &= log(a(1+\frac{b}{a})) = log(a) + log(1+\frac{b}{a}) = log(a) + log(1+exp(log(b)-log(a))) \end{aligned}$$

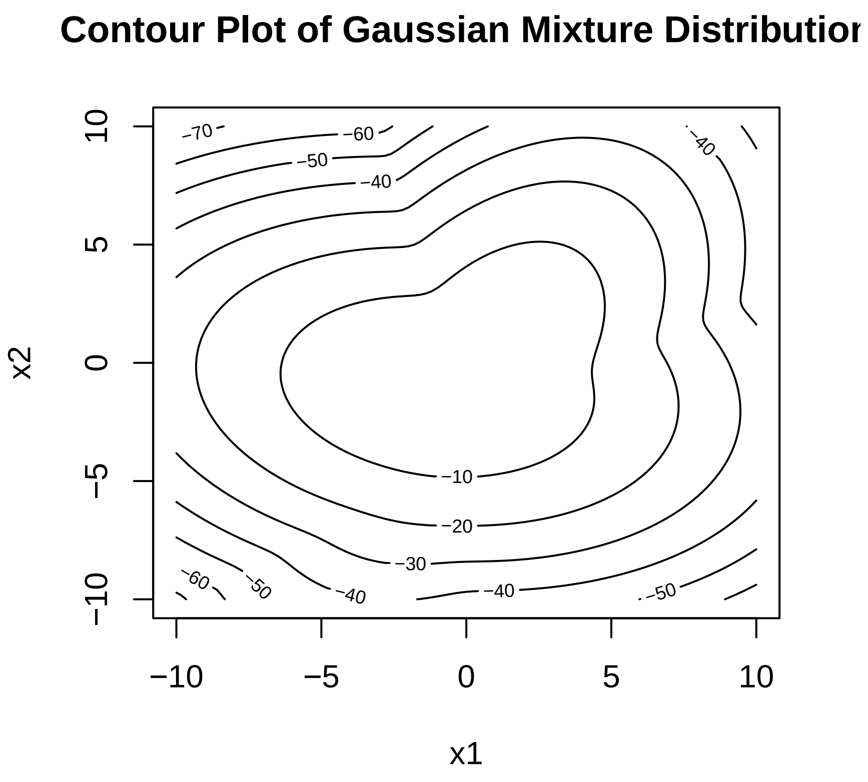
In the same context, since overflow can occur when $exp(log(b) - log(a))$ becomes large, we bypassed overflow by using the fact that a and b are interchangeable.

$$\therefore log(a+b) = \begin{cases} log(a) + log(1+exp(log(b)-log(a))), & \text{if } log(b) < log(a) \\ log(b) + log(1+exp(log(a)-log(b))), & \text{if } log(b) > log(a) \end{cases}$$

In conclusion, to bypass underflow, we mapped each point in the grid to a log-multivariate normal distribution (instead of mapping to a multivariate normal distribution and then taking the logarithm). Also, we did a min-max operation on exp in log(a+b) to bypass the overflow from exp.

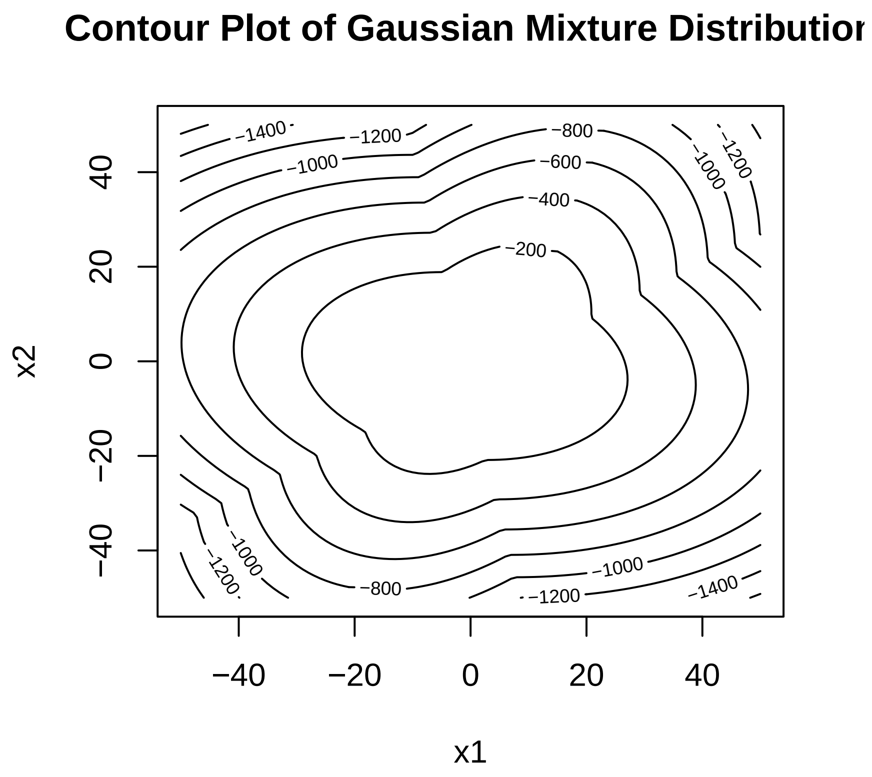
```r
##### [-10, 10] grid #####

# Setting the grid
data.grid <- expand.grid(x.1 = seq(-10, 10, length.out = 101),
                         x.2 = seq(-10, 10, length.out = 101))

# Defining Log-multivariate normal function
log_multivariate_normal_pdf <- function(x, mean, conv) {
 d <- length(mean)
 det_cov <- det(conv)
 inv_cov <- solve(conv)
 constant <- -d/2 * log(2*pi) - 1/2 * log(det_cov)
 exponent <- -0.5 * t(x - mean) %*% inv_cov %*% (x - mean)
 log_pdf_value <- constant + exponent
 return (log_pdf_value)
}

# Compute Probability
log_prob1 <- apply(data.grid, 1, function(x) {
  log_multivariate_normal_pdf(x, mu1, sigma1)
})
log_prob2 <- apply(data.grid, 1, function(x) {
  log_multivariate_normal_pdf(x, mu2, sigma2)
})
```

```r
log_weighted1 <- log(weights[1]) + log_prob1
log_weighted2 <- log(weights[2]) + log_prob2

log_prob <- numeric(length(log_weighted1))
for (i in 1:length(log_weighted1)) {
  if (log_weighted2[i] > log_weighted1[i]) {
    log_prob[i] <- log_weighted2[i] + log(1 + exp(log_weighted1[i] - log_weighted2[i]))
  } else {
    log_prob[i] <- log_weighted1[i] + log(1 + exp(log_weighted2[i] - log_weighted1[i]))
  }
}

# Drawing the Contour
q.samp <- cbind(data.grid, prob = log_prob)
contour(x = unique(q.samp$x.1), y = unique(q.samp$x.2),
        z = matrix(q.samp$prob, ncol = 101),
        main = "Contour Plot of Gaussian Mixture Distribution",
        xlab = "x1", ylab = "x2")
```



```r
##### [-50, 50] grid #####

# Setting the grid
data.grid <- expand.grid(x.1 = seq(-50, 50, length.out = 101),
                         x.2 = seq(-50, 50, length.out = 101))
```

```r
# Defining Log-multivariate normal function
log_multivariate_normal_pdf <- function(x, mean, conv) {
 d <- length(mean)
 det_cov <- det(conv)
 inv_cov <- solve(conv)
 constant <- -d/2 * log(2*pi) - 1/2 * log(det_cov)
 exponent <- -0.5 * t(x - mean) %*% inv_cov %*% (x - mean)
 log_pdf_value <- constant + exponent
 return (log_pdf_value)
}

# Compute Probability
log_prob1 <- apply(data.grid, 1, function(x) {
  log_multivariate_normal_pdf(x, mu1, sigma1)
})
log_prob2 <- apply(data.grid, 1, function(x) {
  log_multivariate_normal_pdf(x, mu2, sigma2)
})

log_weighted1 <- log(weights[1]) + log_prob1
log_weighted2 <- log(weights[2]) + log_prob2

log_prob <- numeric(length(log_weighted1))
for (i in 1:length(log_weighted1)) {
  if (log_weighted2[i] > log_weighted1[i]) {
    log_prob[i] <- log_weighted2[i] + log(1 + exp(log_weighted1[i] - log_weighted2[i]))
  } else {
    log_prob[i] <- log_weighted1[i] + log(1 + exp(log_weighted2[i] - log_weighted1[i]))
  }
}

# Drawing the Contour
q.samp <- cbind(data.grid, prob = log_prob)
contour(x = unique(q.samp$x.1), y = unique(q.samp$x.2),
        z = matrix(q.samp$prob, ncol = 101),
        main = "Contour Plot of Gaussian Mixture Distribution",
        xlab = "x1", ylab = "x2")
```


Contour Plot of Gaussian Mixture Distributior
x1
x2
−40
−20
0
20
40
−200
−400
−600
−800
−1000
−1200
−1400

Q2. Remove the first two digits '20' from your student ID number to create an 8-digit number, and use this as the initial value to generate pseudo-random numbers on (0,1) using the middle square method.

i) Specify a suitable value for the scaling factor that can transform the generated numbers into values on (0, 1), instead of remaining as 8-digit integers.
ii) Generate 100 pseudo random numbers on (0, 1) and draw a time series plot (**ts.plot** function) and a histogram (**hist** function).
iii) Generate 1000 pseudo random numbers on (0, 1) and draw a time series plot and a histogram.
iv) Generate 10000 pseudo random numbers on (0, 1) and draw a time series plot and a histogram.

Discuss your results of ii)–iv).

[Note 1: Your algorithm may become stuck at a specific value or enter a short rotation cycle.]

[Note 2: After squaring your number, you may not see all the digits due to floating-point representation. This is not necessarily a problem, and there are a few ways to address this potential issue.]

solution

```
set.seed(2018122062)
id = 18122062
```

i) Specify a suitable value for the scaling factor that can transform the generated numbers into values on (0, 1), instead of remaining as 8-digit integers.

```
options(digits=15)
scaling_factor <- 1e08
id_demical= id / scaling_factor
id_demical
```

```
## [1] 0.18122062
```

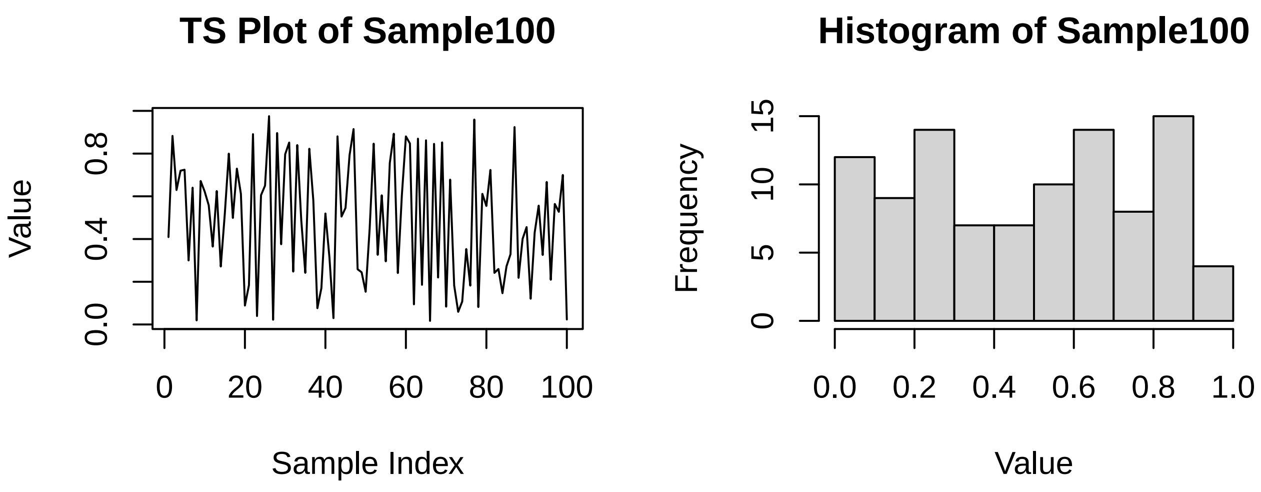
Functions used in the algorithm :

```
middle8 <- function(number){
  # input : 16-digit number
  # output : 8-digit number which is middle of input
  result = (number %% 1e12 - number %% 1e04) / 1e04
  return (result)
  # ex. middle8(1234567890123456)
  # = (0000567890123456 - 3456) / 1e04
  # = 567890120000 / 1e04 = 56789012
```

```r
}

Middle_Square <- function(sample_size){
  # input : sample size
  # output : samples generated by middle square method
  samples = c()
  for (i in 1:sample_size) {
    if (i == 1) {
      samples <- c(samples, middle8(id^2))
    } else {
      samples <- c(samples, middle8((samples[length(samples)])^2))
    }
  }
  samples <- samples / scaling_factor
  return (samples)
}
```

ii) Generate 100 pseudo random numbers on (0, 1) and draw a time series plot (**ts.plot** function) and a histogram (**hist** function).

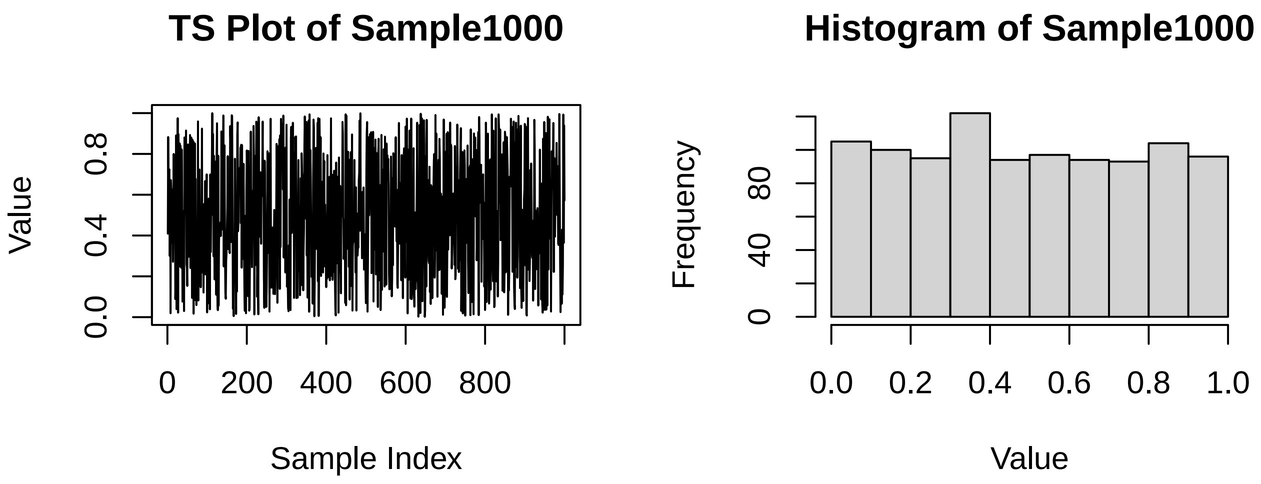
```r
par(mfrow=c(1,2))
Sample100 = Middle_Square(100)
ts.plot(Sample100, main = "TS Plot of Sample100",
        xlab = "Sample Index", ylab = "Value")
hist(Sample100, main = "Histogram of Sample100",
     xlab = "Value", ylab = "Frequency")
```



As sample size is too small(100), we can not say that we generate from **completely** random number on (0, 1).

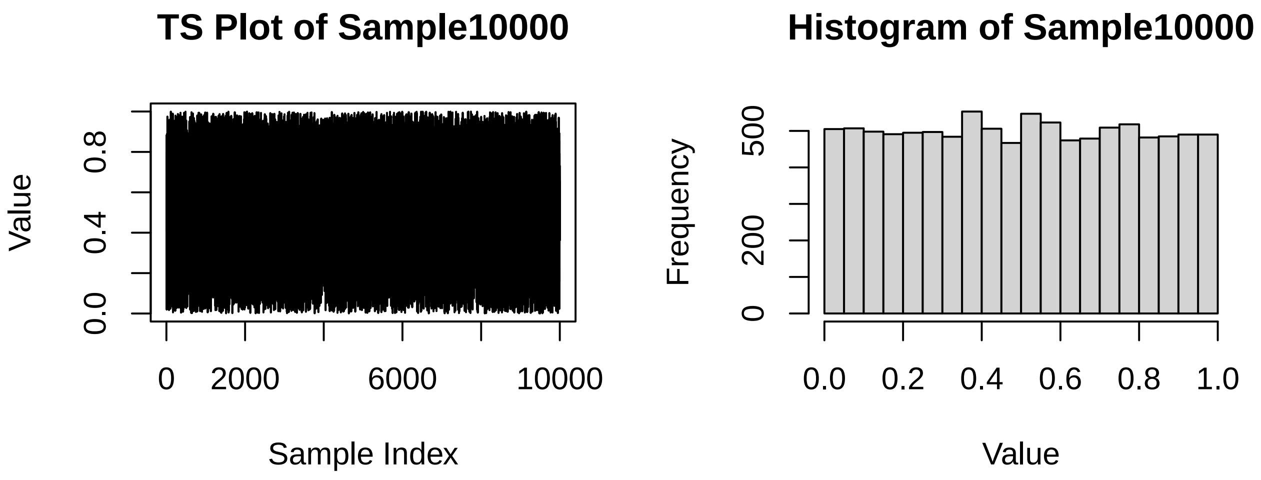
iii) Generate 1000 pseudo random numbers on (0, 1) and draw a time series plot and a histogram.

```r
par(mfrow=c(1,2))
Sample1000 = Middle_Square(1000)
ts.plot(Sample1000, main = "TS Plot of Sample1000",
        xlab = "Sample Index", ylab = "Value")
hist(Sample1000, main = "Histogram of Sample1000",
     xlab = "Value", ylab = "Frequency")
```



iv) Generate 10000 pseudo random numbers on (0, 1) and draw a time series plot and a histogram.

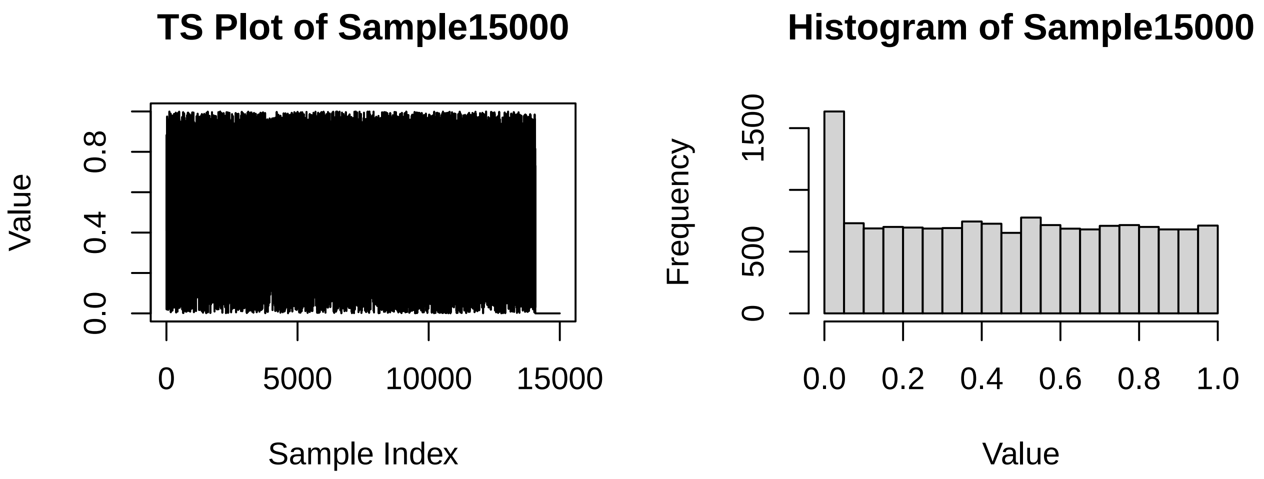
```r
par(mfrow=c(1,2))
Sample10000 = Middle_Square(10000)
ts.plot(Sample10000, main = "TS Plot of Sample10000",
        xlab = "Sample Index", ylab = "Value")
hist(Sample10000, main = "Histogram of Sample10000",
     xlab = "Value", ylab = "Frequency")
```



As the sample size increases, we can see the distribution of the sample becomes more like a uniform(0,1) distribution.

+ Generate 15000 pseudo random numbers on (0, 1) and draw a time series plot and a histogram.

```r
par(mfrow=c(1,2))
Sample15000 = Middle_Square(15000)
ts.plot(Sample15000, main = "TS Plot of Sample15000",
        xlab = "Sample Index", ylab = "Value")
hist(Sample15000, main = "Histogram of Sample15000",
     xlab = "Value", ylab = "Frequency")
```



We can see that we got stuck at 0 at around the 14000th sample.