Lab 3 - STAT40400

Q1

(i)

Write code for a Metropolis Hastings algorithm which monitors the rate at which the proposed values are accepted, for each of the three scenarios $\sigma = 20$, 0.2 and 3. Explain how this analysis can help to decide how to tune the proposal variance.

In order to monitor the rate at which the proposed values are accepted I implemented two counters. I counted both the number of accepted moves (c_A) and rejected moves (c_R). The rate at which proposed values are accepted is then given by

$$\frac{c_A}{c_A + c_R}$$

. My updated Metropolis-Hastings algorithm is defined below:

```
metropolis.hastings <- function(f, # the target distribution g,
                                g, # the proposal distribution
                                random.g, # a sample from the proposal distribution x0
                                x0, # initial value for chain, in R it is x[1]
                                sigma, # proposal st. dev.
                                chain.size=1e5){ # chain size
  x <- c(x0, rep(NA,chain.size-1)) # initialize chain
  count_accept = 0
  count_reject = 0
  for(i in 2:chain.size) {
    y <- random.g(x[i-1],sigma) # generate Y from g(.|xt) using random.g
    alpha <- min(1, f(y)*g(y,x[i-1],sigma)/(f(x[i-1])*g(x[i-1],y,sigma))) # Define alpha
    if( runif(1)<alpha){
      x[i] <- y    # Update current state to the candidate if candidate accepted
      count_accept = count_accept + 1  # Add 1 to the acceptance counter
    }
    else{
      x[i] <- x[i-1] # Set the current state to the old state if candidate rejected
      count_reject = count_reject + 1 # Add 1 to the rejection counter
    }
  }
  # Calculate rate of acceptance
  rate_of_acceptance = count_accept/(count_accept + count_reject)
  # Return both the chain and rate of acceptance
  list('x'=x, 'acc_rate' = rate_of_acceptance) }
```

We are asked to monitor the acceptance rate for varying σ values and explain how this analysis can help to decide on tuning the proposal variance. The intuition of the Metropolis-Hastings algorithm is that α, the probability of accepting a candidate move, is 1 if the candidate value lies in a higher density region of the target distribution ($\pi(x)$) than the current state. If, on the other hand, the candidate value lies in a lower density region of the target distribution than the current state of the chain, the probability of accepting the candidate will be inversely proportional to how much less likely the candidate value is than the current value of the chain.

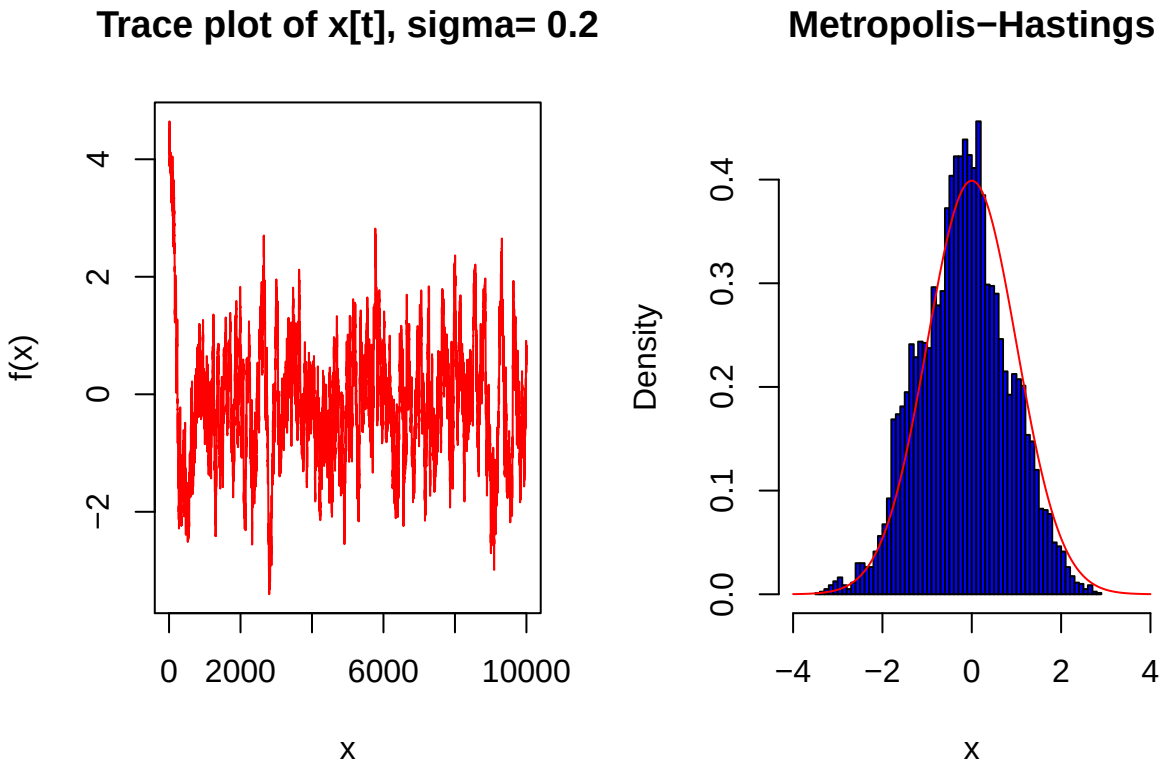
The proposal variance σ plays an important role. If σ is too high then the chain will generate too many candidates in the tails which will be rejected with high probability. If σ is too low then the chain will generate candidates very close to the current value of the chain and will take a long time moving through the parameter space.

A common approach to assist with tuning the choice of proposal variance is to examine the the rate at which candidates are accepted within the Metropolis-Hastings algorithm. The optimal acceptance rate depends on the target distribution, however; in general, an acceptance rate of 20%-40% is desirable.

$\sigma = 0.2$

If the proposal variance (σ) is low then the proposal distribution will generate candidates which are similar to the current value of the chain. This means the chain will take a long time to move through the parameter space and converge to the target distribution.

```
set.seed(123)
sigma=0.2
f <- function(x) dnorm(x,0,1)
random.g <- function(x,sigma) rnorm(1,x,sigma) # q(x,y)
g <- function(x,y,sigma) dnorm(y,x,sigma)
x0 <- 4
chain.size <- 1e4
output <- metropolis.hastings(f,g,random.g,x0,sigma,chain.size)
x = output$x

par(mfrow=c(1,2))
plot(x, xlab="x", ylab="f(x)", main=paste("Trace plot of x[t], sigma=", sigma),col="red", type='l')
xmin = min(x[(0.2*chain.size) : chain.size])
xmax = max(x[(0.2*chain.size) : chain.size])
xs.lower = min(-4,xmin)
xs.upper = max(4,xmax)
xs <- seq(xs.lower,xs.upper,len=1e3)
# Plot histogram
# Note: x[(0.2*chain.size) : chain.size] accounts for burn-in period
# by removing the first 20% of the chain
hist(x[(0.2*chain.size) : chain.size],50, prob=TRUE, xlim=c(xs.lower,xs.upper),col="blue",xlab="x", mai
lines(xs,f(xs),col="red", lwd=1)
```




```
print(paste0('The acceptance rate for sigma of ',sigma,' is : ',output$acc_rate))
```

```
## [1] "The acceptance rate for sigma of 0.2 is : 0.933093309330933"
```

In the case displayed above ($\sigma = 0.2$) we can see from the trace plot that the chain has not converged to stationarity. The histogram shows that there is a slight mismatch between the target density (the red line) and the samples generated from the Metropolis-Hastings algorithm.

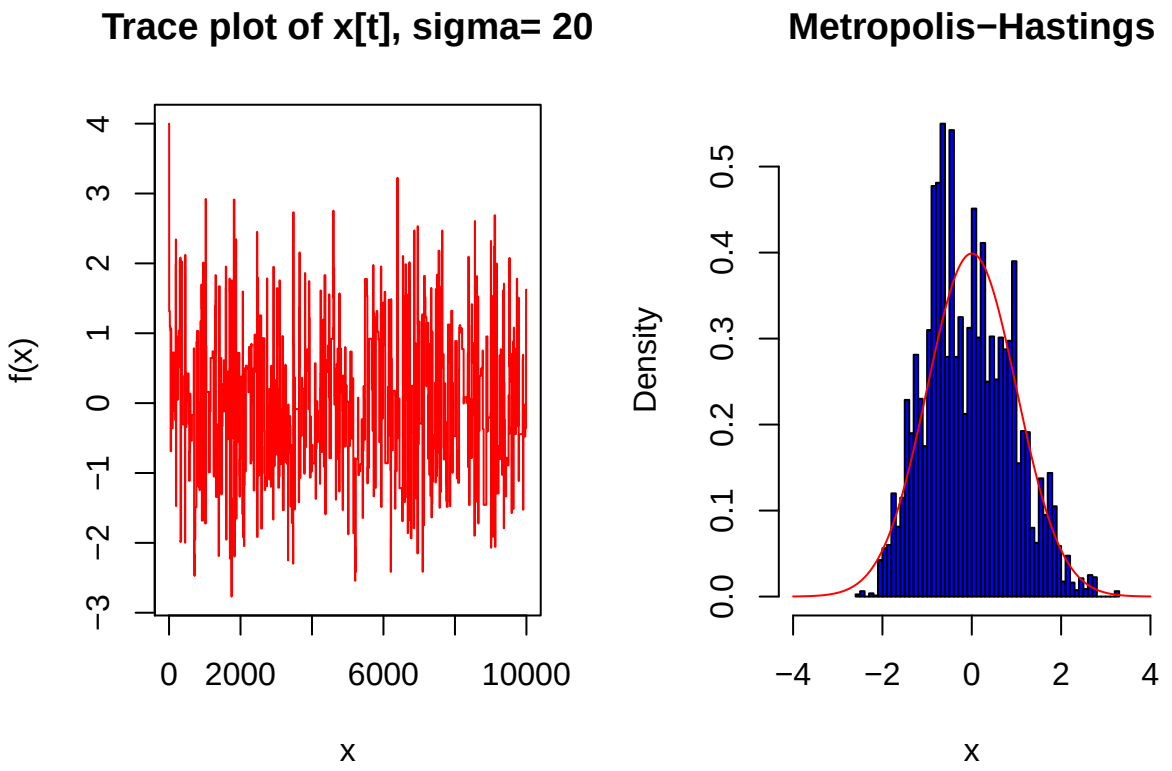
As we can see from the output, we get an acceptance rate of 0.93309. This means that 93.3% of the candidates proposed were accepted. This is much higher than the 20-40% we are looking for. This acceptance rate indicates that a higher value of σ could produce better performace from our algorithm.

$\sigma = 20$

If the proposal variance (σ) is high then the proposal distribution will generate a lot of candidates which lie in areas where the target distribution has low density. This means that the value of α will be very low. Therefore, the chain will reject a high proportion of the candidate moves and will thus converge very slowly.

```
set.seed(1222)
sigma=20
f <- function(x) dnorm(x,0,1)
random.g <- function(x,sigma) rnorm(1,x,sigma) # q(x,y)
g <- function(x,y,sigma) dnorm(y,x,sigma)
x0 <- 4
chain.size <- 1e4
output <- metropolis.hastings(f,g,random.g,x0,sigma,chain.size)
x = output$x

par(mfrow=c(1,2))
```

```
plot(x, xlab="x", ylab="f(x)", main=paste("Trace plot of x[t], sigma=", sigma),col="red", type='l')
xmin = min(x[(0.2*chain.size) : chain.size])
xmax = max(x[(0.2*chain.size) : chain.size])
xs.lower = min(-4,xmin)

xs.upper = max(4,xmax)
xs <- seq(xs.lower,xs.upper,len=1e3)
# Plot histogram
# Note: x[(0.2*chain.size) : chain.size] accounts for burn-in period
# by removing the first 20% of the chain
hist(x[(0.2*chain.size) : chain.size],50, prob=TRUE, xlim=c(xs.lower,xs.upper),col="blue",xlab="x", mai
lines(xs,f(xs),col="red", lwd=1)
```




```
print(paste0('The acceptance rate for sigma of ',sigma,' is : ',output$acc_rate))
```

```
## [1] "The acceptance rate for sigma of 20 is : 0.0622062206220622"
```

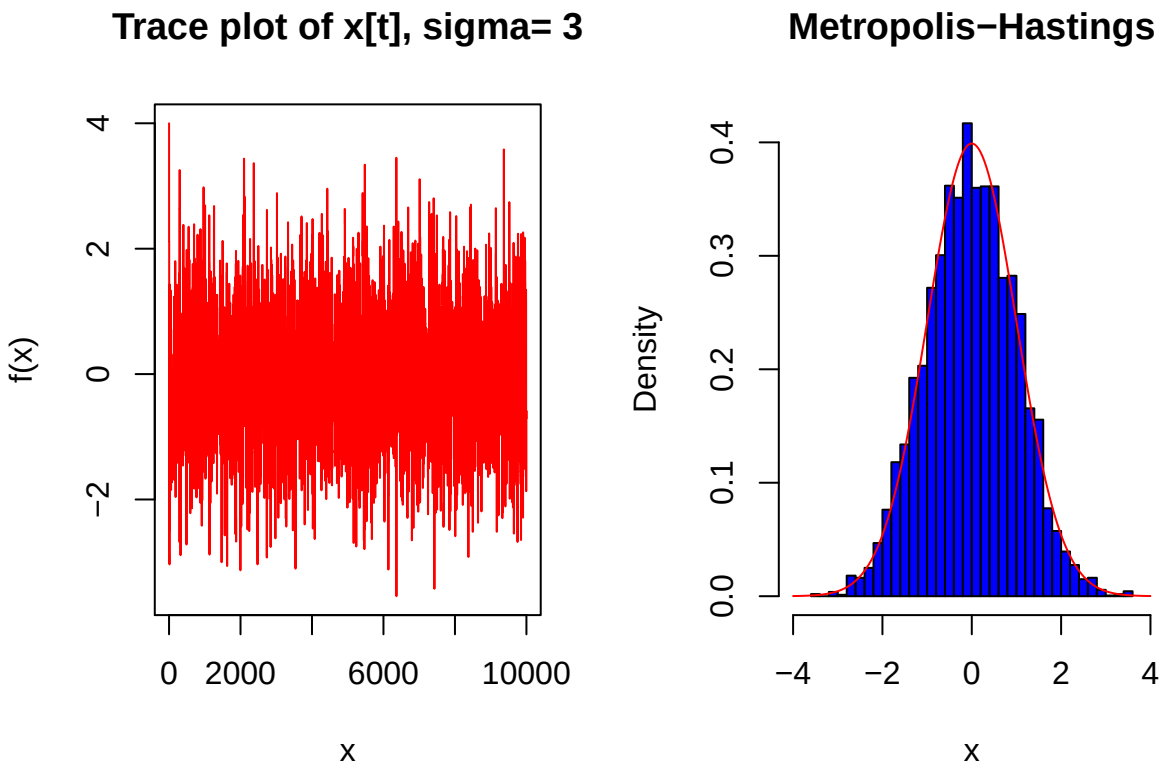
As we said before, a large σ will result in a very low acceptance rate which means the chain will not mix well. This can be seen in the trace plot above. We can also see that the histogram of sampled values is not representitive of the target density (a Normal(0,1)).

The acceptance rate for this high value of σ was just 6.22% which is far from the range of 20-40% which we would hope for. This indicates that a lower value of σ should be used to try and obtain an acceptance rate in the range of 20-40%.

$\sigma = 3$

We wish to strike a balance between the high acceptance rate of $\sigma = 0.2$ and the low acceptance rate of $\sigma = 20$. Choosing $\sigma = 3$ achieves this as we will see.

```r
set.seed(1)
sigma=3
f <- function(x) dnorm(x,0,1)
random.g <- function(x,sigma) rnorm(1,x,sigma) # q(x,y)
g <- function(x,y,sigma) dnorm(y,x,sigma)
x0 <- 4
chain.size <- 1e4
output <- metropolis.hastings(f,g,random.g,x0,sigma,chain.size)
x = output$x

par(mfrow=c(1,2))
plot(x, xlab="x", ylab="f(x)", main=paste("Trace plot of x[t], sigma=", sigma),col="red", type='l')
xmin = min(x[(0.2*chain.size) : chain.size])
xmax = max(x[(0.2*chain.size) : chain.size])
xs.lower = min(-4,xmin)

xs.upper = max(4,xmax)
xs <- seq(xs.lower,xs.upper,len=1e3)
# Plot histogram
# Note: x[(0.2*chain.size) : chain.size] accounts for burn-in period
# by removing the first 20% of the chain
hist(x[(0.2*chain.size) : chain.size],50, prob=TRUE, xlim=c(xs.lower,xs.upper),col="blue",xlab="x", mai
lines(xs,f(xs),col="red", lwd=1)
```




```r
print(paste0('The acceptance rate for sigma of ',sigma,' is : ',output$acc_rate))
```

```
## [1] "The acceptance rate for sigma of 3 is : 0.368436843684368"
```

The trace plot for $\sigma = 3$ indicates the chain has converged and reached stationarity. Additionally, we can see that the histogram of the sample values closely follows the Standard Normal density (the red line).

The acceptance rate for $\sigma = 3$ is 36.8% which is within the desirable range of 20-40%.

Summary

We have now monitored the rate of acceptance for $\sigma = 0.2, 3, 20$. From our analysis we can see that the acceptance rate for $\sigma = 0.2$ is too high while the acceptance rate for $\sigma = 20$ is too low while the acceptance rate for $\sigma = 3$ strikes the right balance.

–>

(ii)

For each scenario, estimate the probability that $X > 2$, where $X \sim N(0,1)$ using the output from the Metropolis-Hastings algorithm. Provide an approximate standard error in each case. (Note that you can compare your estimates to the true value 0.02275)

We wish to estimate the probability that $X > 2$ where $X \sim N(0,1)$ using the Metropolis-Hastings algorithm applied to each of the cases from part (i). From the function defined in part (i) we get two outputs, the values of the chain and the acceptance rate. In order to roughly estimate $\mathbb{P}(X > 2)$ we could simply check the proportion of values in our chain which are greater than 2. However, there are some things we must consider before applying the Monte Carlo method directly to our chain as just described.

Burn-In Period

The MCMC technique requires that the Markov Chain has a stationary distribution equal to the target distribution. And so, before applying Monte Carlo methods to our Markov Chain it is necessary that the chain has converged to this stationary distribution. The easiest way to ensure this in practice is to discard the observations from some initial portion of the chain. This initial portion of the chain is called the burn-in period.

Estimating $\mathbb{P}(X > 2)$

To account for the burn-in period I chose to use the final 80% of the chain (ie. discard the first 20% of the chain to ensure convergence). Below is the code to estimate the $\mathbb{P}(X > 2)$ for the various σ values.

$\sigma = 0.2$

```
set.seed(23324)

x0 = 4
sigma = 0.2
output <- metropolis.hastings(f,g,random.g,x0,sigma,100000)
x = output$x
# Discard burn-in period
stationary_chain = x[(0.2*chain.size) : (1*chain.size)]
# apply indicator function
ind_samples = stationary_chain>2

s_squared = var(ind_samples)
sigma = sqrt(s_squared)
sterror = sigma/sqrt(length(ind_samples))

print(paste0('The estimate of P(X>2) from the samples is : ', mean(ind_samples)))
```

```
## [1] "The estimate of P(X>2) from the samples is : 0.0116235470566179"
```

```
print(paste0('The standard error of this estimate is : ', sterror))
```

```
## [1] "The standard error of this estimate is : 0.00119835513363321"
```

```
print(paste0('The 95% CI for P(X>2) is : (',mean(ind_samples)-1.96*sterror,' , ', mean(ind_samples)+1.9
```

```
## [1] "The 95% CI for P(X>2) is : (0.00927477099469684 , 0.013972323118539)"
```

$\sigma = 20$

```
set.seed(23324)

x0 = 4
sigma = 20
output <- metropolis.hastings(f,g,random.g,x0,sigma,100000)
x = output$x
# Discard burn-in period
stationary_chain = x[(0.2*chain.size) : (1*chain.size)]
# apply indicator function
ind_samples = stationary_chain>2

s_squared = var(ind_samples)
sigma = sqrt(s_squared)
sterror = sigma/sqrt(length(ind_samples))

print(paste0('The estimate of P(X>2) from the samples is : ', mean(ind_samples)))
```

```
## [1] "The estimate of P(X>2) from the samples is : 0.0256217972753406"
```

```
print(paste0('The standard error of this estimate is : ', sterror))
```

```
## [1] "The standard error of this estimate is : 0.00176654043187957"
```

```
print(paste0('The 95% CI for P(X>2) is : (',mean(ind_samples)-1.96*sterror,' , ', mean(ind_samples)+1.9
```

```
## [1] "The 95% CI for P(X>2) is : (0.0221593780288566 , 0.0290842165218245)"
```

$\sigma = 3$

```
set.seed(2322)

x0 = 4
sigma = 3
output <- metropolis.hastings(f,g,random.g,x0,sigma,100000)
x = output$x
# Discard burn-in period
stationary_chain = x[(0.2*chain.size) : (1*chain.size)]
# apply indicator function
ind_samples = stationary_chain>2

s_squared = var(ind_samples)
sigma = sqrt(s_squared)
sterror = sigma/sqrt(length(ind_samples))

print(paste0('The estimate of P(X>2) from the samples is : ', mean(ind_samples)))
```

```
## [1] "The estimate of P(X>2) from the samples is : 0.0257467816522935"
```

```
print(paste0('The standard error of this estimate is : ', sterror))
```

```
## [1] "The standard error of this estimate is : 0.00177073024790772"
```

```
print(paste0('The 95% CI for P(X>2) is : (',mean(ind_samples)-1.96*sterror,' , ', mean(ind_samples)+1.9
```

```
## [1] "The 95% CI for P(X>2) is : (0.0222761503663943 , 0.0292174129381926)"
```

The true theoretical value for $\mathbb{P}(X > 2)$ is 0.02275. As we can see above, with chain size of 100,000 we have a reasonable estimate for $\mathbb{P}(X > 2)$ from all three σ cases. Using the standard error calculated we can compute a 95% confidence interval and from this we can see that in all three cases we have that the true value lies within the confidence interval.

Of the three cases, $\sigma = 3$, has the best estimate with an estimate for $\mathbb{P}(X > 2)$ of 0.0224859 (this will change depending on the seed etc. I fixed the seed here so I could comment on a fixed value).

This is in line with what we would expect from part (i). In part (i) we saw that the $\sigma = 3$ case produced the histogram which most closely followed the normal density function and had a trace plot which indicated the chain had converged and mixed well.

***Extra Stuff* - Dependence of Samples and Markov Chain Central Limit Theorem**

I had this section typed up before asking in the tutorial and finding out we could disregard the covariance component when calculating the standard error for this assignment (ie. assume independence of elements in the chain). In the calculations above I have just applied the regular CLT as suggested however I kept this extra in as I already had it typed up.

Usually when we apply Monte Carlo methods we have that our samples are independent and identically distributed (iid). The central limit theorem underpins Monte Carlo methods but in order for the central limit theorem to apply we must have that our samples are iid. Now that our samples are coming from a markov chain, we no longer have iid samples. We know this from the definition of the Markov property (ie. that each sample depends on the previous state of the chain).

So, can we apply Monte Carlo methods to a Markov Chain? Yes, provided the chain adheres to the conditions set out in the Markov Chain Central Limit Theorem. The Markov Chain Central Limit Theorem (CLT) can be used to justify Markov Chain Monte Carlo (MCMC) methods. From Wikipedia we have the definition of the Markov Chain CLT as follows:

Given the sequence $X_1, ..., X_n$ of random elements is a markov chain that has a stationary distribution, and that the initial distribution of the process, i.e. the distribution of X_1, is the stationary distribution, so that $X_1, X_2, X_3, \ldots$ are identically distributed. In the classic central limit theorem these random variables would be assumed to be independent, but here we have only the weaker assumption that the process has the Markov property. Consider g is some (measurable) real-valued function for which $Var(g(X_1)) < \infty$ then for:

$$\mu = \mathbb{E}(g(X_1))$$

$$\sigma^2 = Var(g(X_1)) + 2\sum_{k=1}^{\infty} Cov(g(X_1), g(X_{1+k}))$$

$$\hat{\mu}_n = \frac{1}{n}\sum_{k=1}^{n} g(X_k)$$

Then as $n \to \infty$ we have

$$\hat{\mu}_n \approx Normal(\mu, \frac{\sigma^2}{n})$$

or in other words

$$(\hat{\mu}_n - \mu) \to Normal(0, \frac{\sigma^2}{n})$$

This tells us that provided our chain is stationary *and* that the initial distribution of the chain is stationary we can indeed apply Monte Carlo methods to our Markov Chain and our estimator will tend to the true value as n gets large.

How would this affect our calculations?

The standard error of our estimate will not be the same as the standard error of a standard Monte Carlo estimator. It is still given by $\frac{\sigma}{\sqrt{n}}$, however, σ would be defined as above (ie. including a covariance component).

Also, we must ensure that the chain is stationary from the very start in order for the Markov Chain CLT to apply. We attempt to ensure this by discarding an initial portion of the chain know as the burn-in period.

The easiest way to roughly decide on the length of the burn in period is to eyeball the trace plot of the parameter values and attempt to observe when the chain has converged. For a more robust test of convergence one can use the Brooks-Gelman-Rubin technique. This technique involves running multiple chains, each from a different starting point, and then testing whether the chains end up being distributed significantly differently from eachother. The ANOVA technique is used to test this.

Q2

Consider sampling from Beta(2.5, 4.5) distribution using an independence sampler with a uniform proposal distribution. Provide R code to do this. Produce output from your code to illustrate the performance of your algorithm.

The key difference between the generic Metropolis-Hastings and the Independence Sampler is that in the Independence Sampler we draw the candidate observation independently from the current state of the chain.

For the general Metropolis-Hastings we had the acceptance probability as

$$\alpha(\theta_t, \phi) = min\{1, \frac{\pi(\phi)q(\phi, \theta_t)}{\pi(\theta_t)q(\theta_t, \phi)}\}$$

Now, for the Independence Sampler we have the acceptance probability as

$$\alpha(\theta, \phi) = min\{1, \frac{w(\phi)}{w(\theta)}\}$$

where $w(\theta) = \frac{\pi(\theta)}{f(\theta)}$

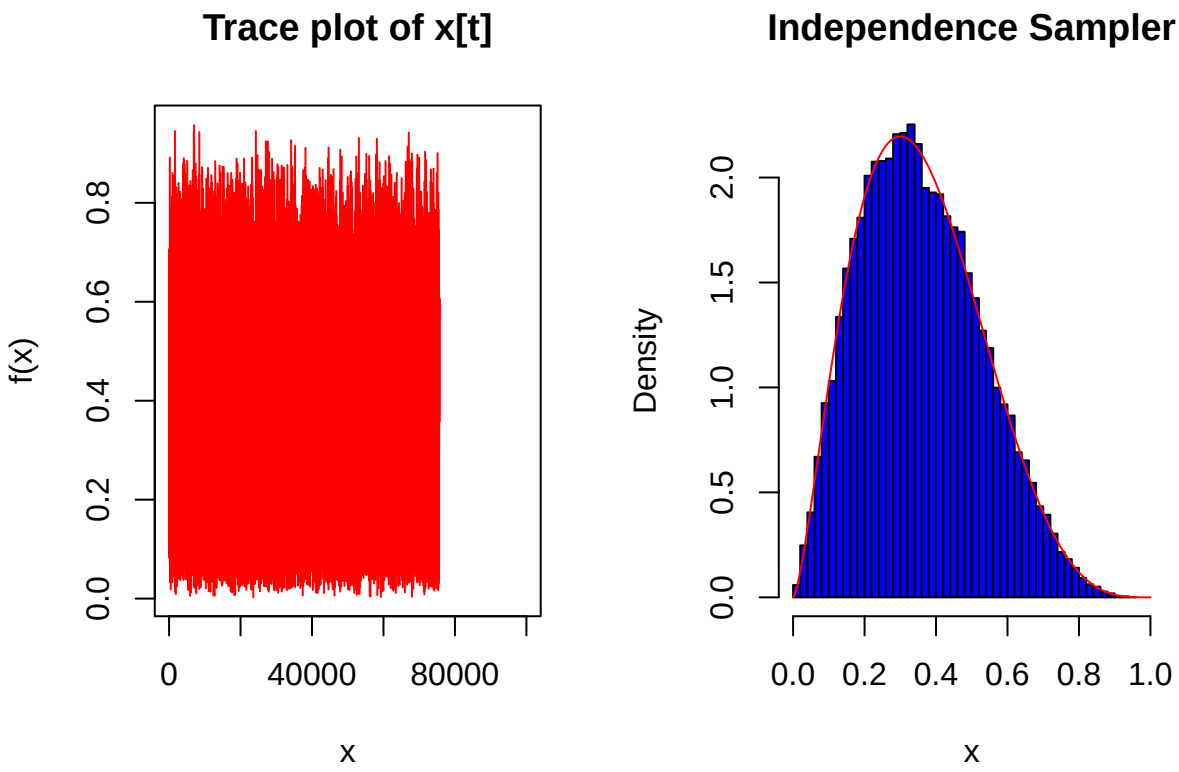
I applied this in the following code:

```
importance.sampler <- function(f, # the target distribution g, # the proposal distribution
                                g, # the proposal distribution
                                random.g, # a sample from the proposal distribution x0, # initial value
                                x0, # initial value for chain, in R it is x[1]
                                chain.size=1e5){ # chain size
  x <- c(x0, rep(NA,chain.size-1)) # initialize chain
  count_accept = 0
  count_reject = 0

  for(i in 2:chain.size) {
    y <- random.g(1) # generate Y
    # Define alpha for importance sampler
    alpha <- min(1, (f(y)/f(x[i-1]))/(g(y)/g(x[i-1])))

    # Check if candidate is accepted and update accordingly
    if(runif(1)<alpha){
      x[i] <- y
      count_accept = count_accept + 1
    }
    else{
      x[i] <- x[i-1]
      count_reject = count_reject + 1
    }
  }
  # Return rate of acceptance and the chain itself
  rate_of_acceptance = count_accept/(count_accept + count_reject)
  list('x'=x, 'acc_rate' = rate_of_acceptance) }
```

In order to sample from a Beta(2.5, 4.5) distribution using the independence sampler defined above using a uniform proposal distribution I ran the following code.

Note: I choose the mean of the distribution as the starting point of the chain since this will result in faster convergence. The mean of a Beta($\alpha = 2.5, \beta = 4.5$) is given by $\frac{\alpha}{\alpha+\beta} = \frac{2.5}{2.5+4.5}$

```r
f <- function(x) dbeta(x,2.5,4.5)
random.g <- function(x) runif(x)
g <- function(x) dunif(x)
x0 <- 2.5/(2.5 + 4.5)
chain.size <- 1e5
output <- importance.sampler(f,g,random.g,x0,chain.size)
x = output$x

par(mfrow=c(1,2))
options(scipen=10)
plot(x, xlab="x", ylab="f(x)", main=paste("Trace plot of x[t]"),col="red", type='l')
# Beta only defined on 0 to 1
xs <- seq(0,1,len=1e3)
hist(x[(0.2*chain.size) : chain.size],50, prob=TRUE, xlim=c(0,1),col="blue",xlab="x", main="Independenc
lines(xs,f(xs),col="red", lwd=1)
```



As can be seen from the trace plot the chain appears to have converged and reached stationarity. The histogram plot illustrates that the simulated values closely follow the beta density (red line) and thus seem to follow the target distribution.

We can check the acceptance rate of the algortihm using the code :

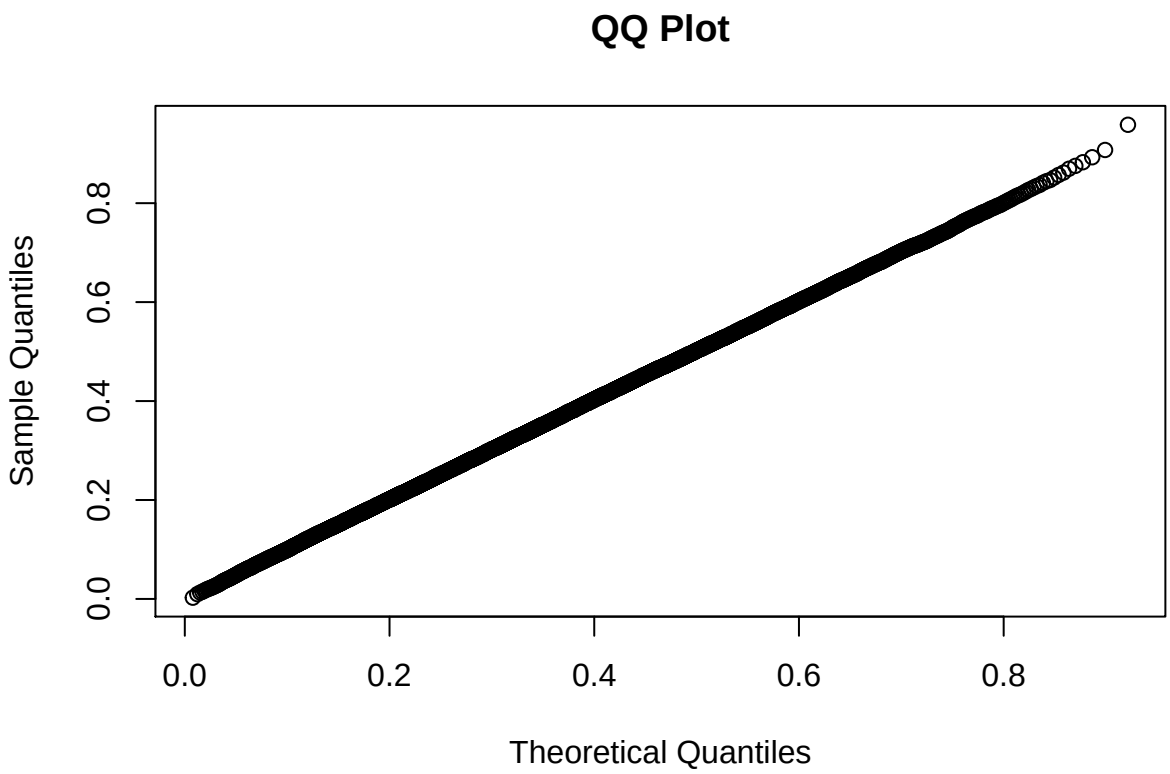
```r
output$acc_rate
```

```
## [1] 0.5430854
```

The acceptance rate is 0.544 or 54.4%. As we discussed in Q1, in general, we would like the acceptance rate to be between 20-40%. However, from the histogram and further tests below we can see that with an acceptance rate of 54.4% the algorithm still does a very good job at sampling for the Beta(2.5,4.5).

Another test we can use to illustrate the performance of this algorithm is to produce a QQ plot comparing the quantiles of the sample data to the quantiles of the target Beta(2.4,4.5) distribution. This is done below.

```
qqplot(qbeta(ppoints(5000),2.5,4.5),x,
       main = 'QQ Plot',
       xlab = 'Theoretical Quantiles',
       ylab = 'Sample Quantiles')
```



As we can see from this QQ plot, plotting the theoretical quantiles versus the sample quantiles results in a straight line along $y = x$. This indicates that the samples do indeed follow the proposed distribution of Beta(2.5,4.5).

Another check we can do is to compare the theoretical mean, variance, kurtosis and skewness to the theoretical values. If they are a close match this is a good indication of the performance of the algorithm. Using $\alpha = 2.5$ and $\beta = 4.5$ we get :

Theoretical Mean

$$\frac{\alpha}{\alpha+\beta} = 0.3571$$

Theoretical Variance

$$\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)} = 0.0306$$

Theoretical Skewness

$$\frac{2(\beta-\alpha)\sqrt{\alpha+\beta+1}}{(\alpha+\beta+2)\sqrt{\alpha\beta}} = 0.3748$$

Theoretical Kurtosis

$$\frac{6[(\alpha-\beta)^2(\alpha+\beta+1)-\alpha\beta(\alpha+\beta+2)]}{\alpha\beta(\alpha+\beta+2)(\alpha+\beta+3)} = -0.4104$$

```r
library(e1071)
print(paste0("Mean : ", mean(x)))
```

```
## [1] "Mean : 0.358794843382902"
```

```r
print(paste0("Variance : ", var(x)))
```

```
## [1] "Variance : 0.0287975321353793"
```

```r
print(paste0("Skewness : ", skewness(x)))
```

```
## [1] "Skewness : 0.361855802738422"
```

```r
print(paste0("Kurtosis : ", kurtosis(x)))
```

```
## [1] "Kurtosis : -0.422031410471739"
```

As we can see the theoretical quantities are very close to the sample quantities. This is another excellent sign that our algorithm is doing a good job at sampling from the Beta(2.5,4.5) distribution.