STA3105 Bayesian Statistics

Jeong Geonwoo

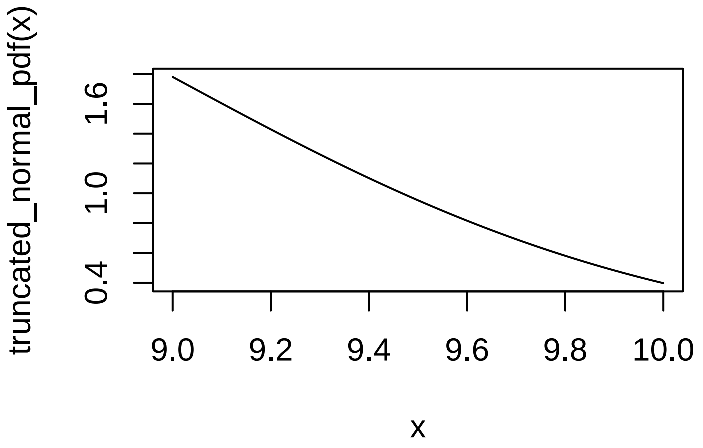
DUE Friday, October 6

Q1

```r
rm(list=ls())
set.seed(2018122062)
```

a) Implement the Metropolis-Hastings algorithm to generate 10,000 samples from the truncated normal distribution with $\mu = 8$, $\sigma = 1$ with $a = 9$, $b = 10$. You should tune the proposal distribution that have acceptance probability around 0.2-0.5. Report the trace plot, histogram, acceptance probability of your MCMC samples.

```r
mu = 8
sigma = 1
a = 9
b = 10
truncated_normal_pdf <- function(x) {
  dnorm_part <- dnorm((x-mu)/sigma, 0, 1)
  pnorm_part <- (pnorm((b-mu)/sigma, 0, 1) - pnorm((a-mu)/sigma, 0, 1))
  indicator_part <- 1*(a <= x & x <= b)
  z_value <- (1/sigma) * dnorm_part / pnorm_part * indicator_part
  return (z_value)
}
x <- seq(9, 10, 0.001)
plot(x, truncated_normal_pdf(x), type='l')
```



Below are the initial variables. **x_current** is $X^{(0)}$ (for now) which determines the distribution from which $X^{(1)}$ is sampled. **n_samples** is the size of sample generated from the truncated normal distribution. **x_samples** is an NA sequence to contain the generated samples. **acceptance** is a variable that records the count of proposed sample is accepted.

```
x_current = rnorm(1, mean=mu, sd=sigma)
n_samples = 10000
x_samples = rep(NA, n_samples)
acceptance = 0
```

If not $9 \leq X^{(0)} \leq 10$, we can not compute $\alpha = min(1, \frac{\pi(X')Q(X^{(0)}|X')}{\pi(X^{(0)})Q(X'|X^{(0)})})$ where $X^{(0)}$ is x_current (for now) and X' is proposed $X^{(1)}$. Therefore, we need to re-assign the initial value using a while loop.

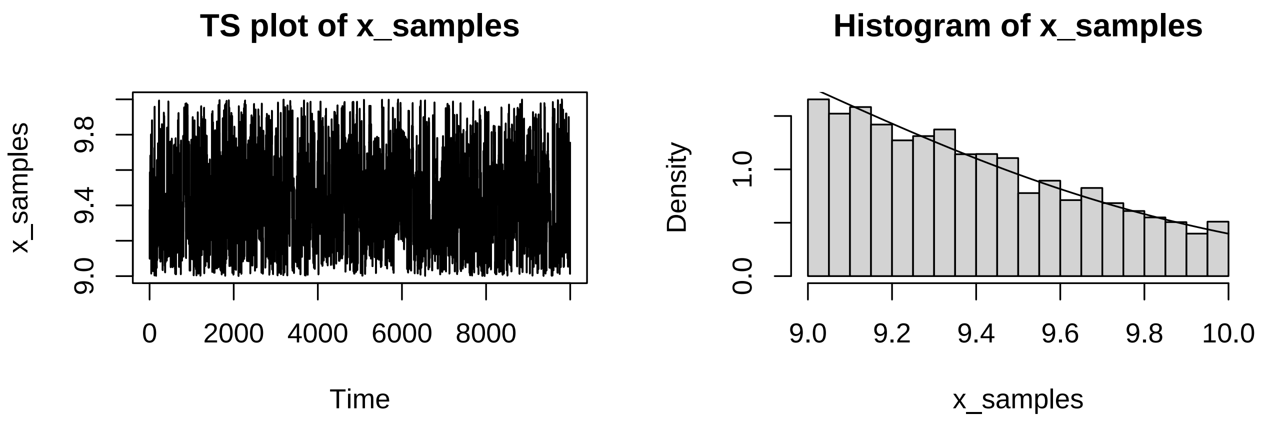
```
while (x_current < 9 | 10 < x_current){
  x_current = rnorm(1, mean=mu, sd=sigma)
}
```

If we set the initial values appropriately, we can generate the samples using Metropolis-Hastings algorithm. As the proposal distribution is symmetric, the each Q part in denominator and numerator do not need to be considered. Because the initial values is in [9, 10], the burn-in (discard first several samples) is not necessary.

```
for (i in 1:n_samples){
  x_proposal = rnorm(1, mean=x_current, sd=1)
  U <- runif(1, 0, 1)
  alpha <- truncated_normal_pdf(x_proposal) / truncated_normal_pdf(x_current)
  if (U < alpha){
    x_new <- x_proposal
    acceptance <- acceptance + 1
  } else {
    x_new <- x_current
  }
  x_samples[i] <- x_new
  x_current <- x_new
}
```

Trace plot and histogram of x_samples

```
par(mfrow=c(1,2))
ts.plot(x_samples, main="TS plot of x_samples")
hist(x_samples, freq = FALSE)
lines(seq(9, 10, 0.001), truncated_normal_pdf(seq(9, 10, 0.001)), type='l')
```



Acceptance probability

```
cat("Acceptance probability:", acceptance / n_samples)
```

```
## Acceptance probability: 0.2847
```

If we want to control the acceptance probability, we can adjust the variance of the distribution from which x_proposal is generated.

b) Compute the sample percentiles and compare with the truncated normal distribution ($\mu = 8$, $\sigma = 1$, a = 9, b = 10) percentiles. Especially, compare the 25th, 50th, 75th percentiles. You can use *rtruncnorm* function in package *truncnorm* to obtain the truncated normal distribution percentiles. Does the sample we generated follows the truncated normal distribution?

```r
library(truncnorm)
cat(
"Distribution 25-th percentiles:", qtruncnorm(0.25, a = 9, b = 10, mean = 8, sd = 1), "\n",
"Sample 25-th percentiles:", sort(x_samples)[25/100 * length(x_samples)]
)
```

```
## Distribution 25-th percentiles: 9.15191
##  Sample 25-th percentiles: 9.157538
```

```r
cat(
"Distribution 50-th percentiles:", qtruncnorm(0.50, a = 9, b = 10, mean = 8, sd = 1), "\n",
"Sample 50-th percentiles:", sort(x_samples)[50/100 * length(x_samples)]
)
```

```
## Distribution 50-th percentiles: 9.33644
##  Sample 50-th percentiles: 9.346725
```

```r
cat(
"Distribution 75-th percentiles:", qtruncnorm(0.75, a = 9, b = 10, mean = 8, sd = 1), "\n",
"Sample 75-th percentiles:", sort(x_samples)[75/100 * length(x_samples)]
)
```

```
## Distribution 75-th percentiles: 9.582862
##  Sample 75-th percentiles: 9.587321
```

From the fact that the percentiles of the true truncated normal distribution and the percentiles of the sample are very similar, we can conclude that the sample obtained by the Metropolis Hastings algorithm follows the truncated normal distribution as we intended.

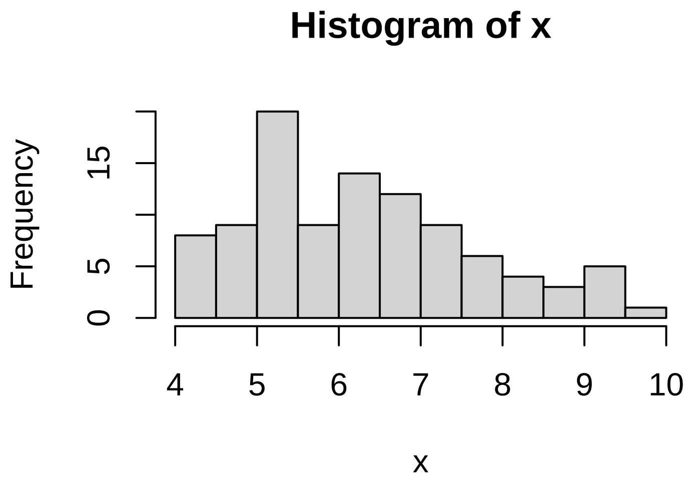
Q2

```r
rm(list=ls())
set.seed(2018122062)
```

a) Implement the Metropolis-Hastings algorithm to generate 10,000 samples of (μ, σ) from the posterior distribution. You should tune the proposal distribution that have acceptance probability around 0.2-0.5 for both μ and σ. Report the trace plots, histograms, acceptance probabilities of your MCMC samples.

```r
# loading the dataset from local directory
load(file="/Users/gunwoojung/Desktop/R_wd/Bayesian_Stat/hw02.RData")
```

```r
n <- length(x)
hist(x, breaks=10)
```



We can see that all the given data is distributed between 4 and 10.

```r
# setting the parameters of truncated normal distribution
a = 4
b = 10
```

$$\text{likelihood}: \ f(x;a,b,\mu,\sigma) = \prod_{i=1}^{n} \frac{1}{\sigma} \frac{\phi\left(\frac{x-\mu}{\sigma}\right)}{\Phi\left(\frac{b-\mu}{\sigma}\right) - \Phi\left(\frac{a-\mu}{\sigma}\right)}, \quad a \le x \le b$$

$$\text{likelihood (kernel)}: \ (\tfrac{1}{\sigma})^n \frac{exp(-\frac{1}{2}\sum(\frac{x_i-\mu}{\sigma})^2)}{(\Phi(\frac{b-\mu}{\sigma})-\Phi(\frac{a-\mu}{\sigma}))^n}$$

$$\text{log-likelihood (kernel)}: \ -n \times log(\sigma) - n \times log(\Phi\left(\frac{b-\mu}{\sigma}\right) - \Phi\left(\frac{a-\mu}{\sigma}\right)) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2$$

$$\text{prior of mu}: \ p(\mu) = \frac{1}{\sqrt{2\pi \times 10}} exp(-\frac{\mu^2}{2\times 10})$$

$$\text{prior of mu (kernel)}: \ exp(-\frac{\mu^2}{20})$$

$$\text{log-prior of mu (kernel)}: \ -\frac{\mu^2}{20}$$

$$\text{prior of sigma}: \ p(\sigma) = \frac{1}{30}\mathbf{1}(0 \le \sigma \le 30)$$

$$\text{prior of sigma (kernel)}: \ \mathbf{1}(0 \le \sigma \le 30)$$

$$\text{log-prior of sigma (kernel)}: \ log(1) = 0$$

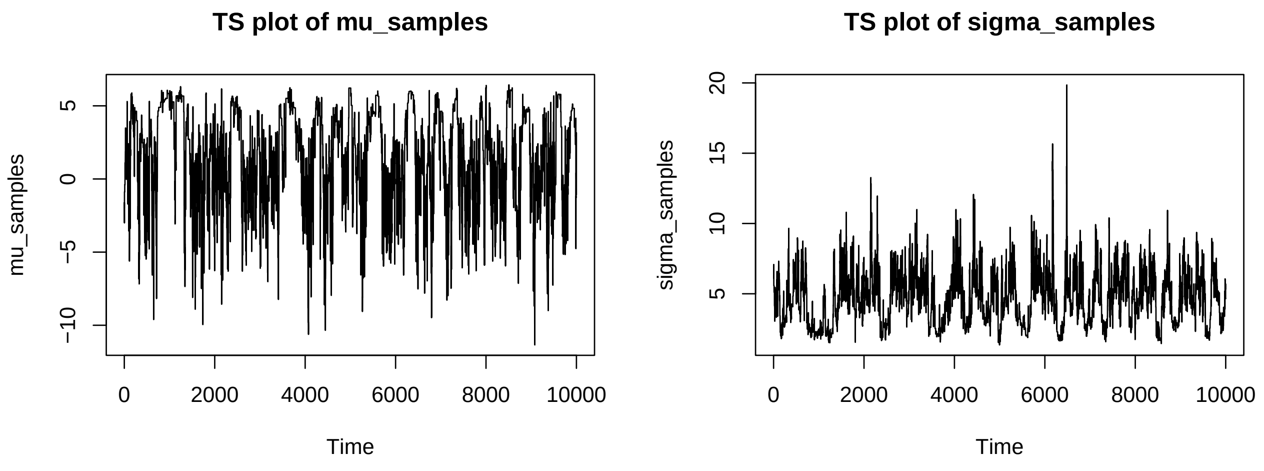
(We need to condition (if loop or while loop) so that the σ is between 0 and 30.)

$$\text{Therefore, the log-posterior} \propto -n \times log(\sigma) - n \times log(\Phi\left(\frac{b-\mu}{\sigma}\right) - \Phi\left(\frac{a-\mu}{\sigma}\right)) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2 - \frac{\mu^2}{20}$$

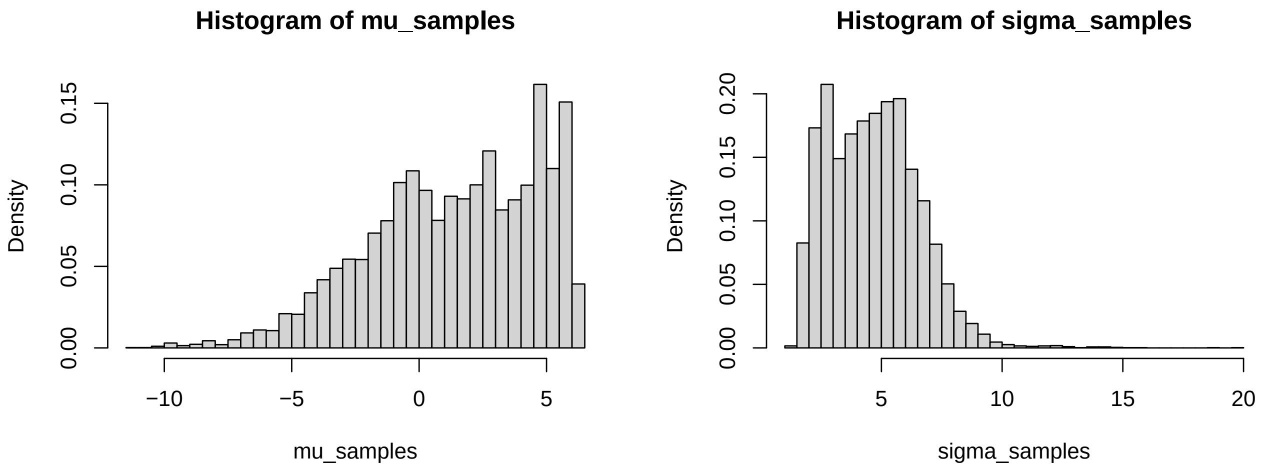
```r
# Defining the density function (log-posterior)
log_posterior <- function(mu, sigma){
  pnorm_part <- pnorm((b-mu)/sigma) - pnorm((a-mu)/sigma)
  dnorm_part <- - (1/sigma^2) * (sum(x^2)/2 -sum(x)*mu + (n/2)*mu^2)
  prior_part <- - (mu^2)/20
  z_value <- -n*log(sigma) -n*log(pnorm_part) + dnorm_part + prior_part
  return (z_value)
}
```

```r
### Sampling mu and sigma (Metropolis-Hastings algorithm)

# Sample sequence (empty)
n_samples = 10000
mu_samples = rep(NA, n_samples)
sigma_samples = rep(NA, n_samples)

# hyperparameters : standard deviation of distribution
# from which proposed mu and sigma are generated.
mu_proposal_sd <- 5
sigma_proposal_sd <- 2

# the number of acceptance
acceptance_mu <- 0
acceptance_sigma <- 0

# initial value of mu and sigma
mu_current <- rnorm(1, 0, 10)
sigma_current <- runif(1, 0, 30)

# Metropolis-Hastings algorithm
for (i in 1:n_samples){
  # mu
  mu_proposal <- rnorm(1, mean=mu_current, sd=mu_proposal_sd)
  log_alpha_mu <- (log_posterior(mu=mu_proposal, sigma=sigma_current)
                   - log_posterior(mu=mu_current, sigma=sigma_current))

  if (is.nan(log_alpha_mu) | abs(log_alpha_mu)==Inf){
    # If both log_posteriors are Inf (or -Inf): log_alpha is NaN
    # If one of the two log_posteriors is Inf (or -Inf) : log_alpha is Inf
    mu_new <- mu_current # reject

  } else if (log(runif(1)) < log_alpha_mu){
    # The condition to accept mu_proposal
    mu_new <- mu_proposal # accept
    acceptance_mu <- acceptance_mu + 1

  } else {
    mu_new <- mu_current # reject
  }
  mu_samples[i] <- mu_new
  mu_current <- mu_new

  # sigma
  sigma_proposal <- rnorm(1, mean=sigma_current, sd=sigma_proposal_sd)
  while ((sigma_proposal < 0 | 30 < sigma_proposal)){
    sigma_proposal <- rnorm(1, mean=sigma_current, sd=sigma_proposal_sd)
  }
  log_alpha_sigma <- (log_posterior(mu=mu_current, sigma=sigma_proposal)
                      - log_posterior(mu=mu_current, sigma=sigma_current))

  if (is.nan(log_alpha_sigma) | abs(log_alpha_sigma)==Inf){
    # If both log_posteriors are Inf (or -Inf): log_alpha is NaN
    # If one of the two log_posteriors is Inf (or -Inf) : log_alpha is Inf
    sigma_new <- sigma_current # reject

  } else if (log(runif(1)) < log_alpha_sigma){
    # The condition to accept sigma_proposal
    sigma_new <- sigma_proposal # accept
    acceptance_sigma <- acceptance_sigma + 1

  } else {
    sigma_new <- sigma_current # reject
  }
  sigma_samples[i] <- sigma_new
  sigma_current <- sigma_new
}
```

```
par(mfrow=c(1,2))
ts.plot(mu_samples, main="TS plot of mu_samples")
ts.plot(sigma_samples, main="TS plot of sigma_samples")
```



```
par(mfrow=c(1,2))
hist(mu_samples, freq = FALSE, breaks = 30)
hist(sigma_samples, freq = FALSE, breaks = 30)
```



Since the initial few values of the sample do not deviate much from the distribution of the entire sample, the burn-in (discard first several samples) is not necessary.

```r
cat("Acceptance probability of mu:", acceptance_mu / n_samples, "\n",
"Acceptance probability of sigma:", acceptance_sigma / n_samples)
```

```
## Acceptance probability of mu: 0.2869
##  Acceptance probability of sigma: 0.3531
```

b) Report the posterior means, 95% HPD intervals of μ and σ.

```r
library(coda)
cat("posterior mean of mu:", mean(mu_samples), "\n",
"posterior mean of sigma:", mean(sigma_samples))
```

```
## posterior mean of mu: 1.392211
##  posterior mean of sigma: 4.678051
```

```r
hpd_interval_mu <- HPDinterval(as.mcmc(mu_samples), prob = 0.95)
cat("95% HPD intervals of mu:", hpd_interval_mu)
```

```
## 95% HPD intervals of mu: -4.43971 6.232008
```

```r
hpd_interval_sigma <- HPDinterval(as.mcmc(sigma_samples), prob = 0.95)
cat("95% HPD interval of sigma:", hpd_interval_sigma)
```

```
## 95% HPD interval of sigma: 1.67053 7.860947
```