Stats 102C, Homework 5 - Multivariate MCMC

Charles Liu (304804942)

Do not post your solutions online on a site like github. Violations will be reported to the Dean of Students.

Modify this file with your answers and responses.

Academic Integrity Statement

By including this statement, I, **Charles Liu**, declare that all of the work in this assignment is my own original work. At no time did I look at the code of other students nor did I search for code solutions online. I understand that plagiarism on any single part of this assignment will result in a 0 for the entire assignment and that I will be referred to the dean of students.

I did discuss ideas related to the homework with Josephine Bruin for parts 2 and 3, with John Wooden for part 2, and with Gene Block for part 5. At no point did I show another student my code, nor did I look at another student's code.

Problem 1

Read, understand, and use the deciphering R code to decode the following messages:

message1 <- "TFH ERQV HKYH VQVNUH QU KNRVXQUN MRYUMNR TNLYVN HKNXR ERXNUS HKNRN YRN WQVN HKXUMW ZQF LYUH WKYRN JXHKQFH NUSXUM FC BXPXUM NYLK QHKNR YUS PUQLPXUM QFH Y HJNBIN-EQQH VQFUHYXU HRQBB XW QUN QE HKNV"

message2 <- "Y CYLZ OZN JYSD ZFE SNGNJ XPHN OP HN Y CYLZ SPSN PM OZYL ZFE ZFWWN-SNE LP EP FVV CZP VYGN OP LNN LBXZ OYHNL ABO OZFO YL SPO MPJ OZNH OP ENXYEN FVV CN ZFGN OP ENXYEN YL CZFO OP EP CYOZ OZN OYHN OZFO YL DYGNS OP BL"

message3 <- "ZHR PKNYSKCT, HK YOT QMSKTY TSRYO, DSK OSA SMUSEN SNNJDTA YOSY OT USN DHRT PKYTMMPLTKY YOSK AHMQOPKN GTCSJNT OT OSA SCOPTBTA NH DJCO-YOT UOTTM, KTU EHRV, USRN SKA NH HK-UOPMNY SMM YOT AHMQOPKN OSA TBTR AHKT USN DJCV SGHJY PK YOT USYTR OSBPKL S LHHA YPDT. GJY CHKBTRNTME, YOT AH-MQOPKN OSA SMUSEN GTMPTBTA YOSY YOTE UTRT ZSR DHRT PKYTMMPLTKY YOSK DSK-ZHR QRTCPNTME YOT NSDT RTSNHKN"

Some punctuation (e.g. apostrophes) have been removed from the text.

You may need to modify a few parameters in the code.

One is the starting seed. Based on where you randomly start, it is possible to get "stuck" in a region that delivers nonsense results.

Another is the weight used to combine the probability values, in the line of code:

```
weight <- 0.07 # tuneable parameter # line 157
cur_combined_loglike <- cur_2letter_prob + weight * cur_word_prob
```

Another is the value to return if a word does not appear in the lexicon. This value is in the function one_gram_prob. It is currently set to 1e-10, but can be modified to another value.

```
one_gram_prob <- function(word, lex) {
  row <- which(lex$word == word)
  if (length(row) == 0) {
    return( 1e-10 ) ## tuneable parameter # line 128
  }
```

Set the working directory & set up

```
setwd(getwd())
load("tm.Rdata")
load(file.path("lex.Rdata"))
```

Create the messages

```
message1 <- "TFH ERQV HKYH VQVNUH QU KNRVXQUN MRYUMNR TNLYVN HKNXR ERXNUS HKNRN YRN WQVN HKXUMW ZQF LYU

message2 <- "Y CYLZ OZN JYSD ZFE SNGNJ XPHN OP HN Y CYLZ SPSN PM OZYL ZFE ZFWWNSNE LP EP FVV CZP VYGN O

message3 <- "ZHR PKNYSKCT, HK YOT QMSKTY TSRYO, DSK OSA SMUSEN SNNJDTA YOSY OT USN DHRT PKYTMMPLTKY YOS
```

Decode function

```
decode <- function(mapping, coded) {
  coded <- toupper(coded)
  decoded <- coded
  for (i in 1:nchar(coded)) {
    if (substring(coded, i, i) %in% LETTERS) {
      substring(decoded, i, i) <- LETTERS[mapping == substring(coded, i, i)]
    }
  }
  decoded
}
```

2-letter log-probability function

```
two_letter_logp <- function(text) {
  logprob <- 0
  last_letter <- ""
  for (i in 1:nchar(text)) {
    cur_letter <- substring(text, i, i)
    row <- which(rownames(trans_mat) == last_letter)
    col <- which(colnames(trans_mat) == cur_letter)
    if (cur_letter %in% toupper(letters)) {
      logprob <- logprob + log(trans_prob_mat[row, col])
      last_letter <- cur_letter
    } else if (last_letter != "") {
        logprob <- logprob + log(trans_prob_mat[row, 27])
        last_letter <- ""
    }
  }
  row <- which(rownames(trans_mat) == last_letter)
  if (last_letter != "") {
    logprob <- logprob + log(trans_prob_mat[row, 27])
    last_letter <- ""
  }
  logprob
}
```

one-gram probability function

```
one_gram_prob <- function(word, lex) {
  row <- which(lex$word == word)
  if (length(row) == 0) {
    return( 1e-10 ) ## tuneable parameter # line 128
  } else {
    return(lex[[row, 2]])
  }
}
```

word log-probability function

```
log_prob_words <- function(text, lex) {
  logprob <- 0
  text <- tolower(text)
  text <- gsub('[[:punct:] ]+',' ', text)
  text <- unlist(strsplit(text, " "))
  for (string in text) {
    logprob <- logprob + log(one_gram_prob(string, lex))
  }
  logprob
}
```

Find the probabilities of the messages (rotated using the messages)

```
mapping <- sample(toupper(letters)) # initialize a random mapping
cur_decoded_text <- decode(mapping, message2) # rotate messages
cur_2letter_prob <- two_letter_logp(cur_decoded_text)
cur_word_prob <- log_prob_words(cur_decoded_text, lexical.database)

weight <- 0.07 # tuneable parameter
cur_combined_loglike <- cur_2letter_prob + weight * cur_word_prob

best_decode <- cur_decoded_text
max_combined_loglike <- cur_combined_loglike
```

Run the iterations (try with 100 times)

```
i <- 1
iters <- 10 # to make sure the code isn't long

# the metropolis algorithm

while (i <= iters) {
  # propose a new mapping by swapping two letters
  proposal <- sample(1:26, 2) # select 2 letters to switch
  proposed_mapping <- mapping
  proposed_mapping[proposal[1]] <- mapping[proposal[2]]
  proposed_mapping[proposal[2]] <- mapping[proposal[1]]

  # calculate the probability of the proposed mapping
  proposed_decoded_text <- decode(proposed_mapping, message2)
  proposed_2letter_prob <- two_letter_logp(proposed_decoded_text)
  proposed_word_prob <- log_prob_words(proposed_decoded_text, lexical.database)
  prop_combined_loglike <- proposed_2letter_prob + weight * proposed_word_prob

  # if u < p(propose)/p(current) accept
  # equivalent to: if log(u) < (logp(proposed) - logp(current))
  if (log(runif(1)) < (prop_combined_loglike - cur_combined_loglike)) {
    mapping <- proposed_mapping
    cur_decoded_text <- proposed_decoded_text
    cur_combined_loglike <- prop_combined_loglike

    # update the best decoding if it is the best
    if (cur_combined_loglike > max_combined_loglike) {
      max_combined_loglike <- cur_combined_loglike
      best_decode <- cur_decoded_text
    }

    # output to screen
    cat(i, cur_decoded_text, "\n")
    i <- i + 1
```

```
  }
}
```

```
## 1 G WGJB UBA QGYC BNZ YAIAQ PMDA UM DA G WGJB YMYA MS UBGJ BNZ BNKKAYAZ JM ZM NOO WBM OGIA UM JAA JV
## 2 G WGJB UBA QGYC BNZ YAIAQ PMDA UM DA G WGJB YMYA MS UBGJ BNZ BNKKAYAZ JM ZM NOO WBM OGIA UM JAA JV
## 3 G WGPB UBA QGYC BNZ YAIAQ JMDA UM DA G WGPB YMYA MS UBGP BNZ BNKKAYAZ PM ZM NOO WBM OGIA UM PAA PV
## 4 G WGPB UBA QGYC BNZ YAIAQ JDMA UD MA G WGPB YDYA DS UBGP BNZ BNKKAYAZ PD ZD NOO WBD OGIA UD PAA PV
## 5 G WGPB UBA QGYC BLZ YAIAQ JDMA UD MA G WGPB YDYA DS UBGP BLZ BLKKAYAZ PD ZD LOO WBD OGIA UD PAA PV
## 6 G WGPB UBA QGYC BLZ YAIAQ JDMA UD MA G WGPB YDYA DS UBGP BLZ BLVVAYAZ PD ZD LOO WBD OGIA UD PAA PK
## 7 G WGPB UBA QGYC BRZ YAIAQ JDMA UD MA G WGPB YDYA DS UBGP BRZ BRVVAYAZ PD ZD ROO WBD OGIA UD PAA PK
## 8 G WGPB UBA QGYC BRL YAIAQ JDMA UD MA G WGPB YDYA DS UBGP BRL BRVVAYAL PD LD ROO WBD OGIA UD PAA PK
## 9 G WGHB UBA QGYC BRL YAIAQ JDMA UD MA G WGHB YDYA DS UBGH BRL BRVVAYAL HD LD ROO WBD OGIA UD HAA HK
## 10 G TGHB UBA QGYC BRL YAIAQ JDMA UD MA G TGHB YDYA DS UBGH BRL BRVVAYAL HD LD ROO TBD OGIA UD HAA H
```

```
best_decode
```

```
## [1] "G TGHB UBA QGYC BRL YAIAQ JDMA UD MA G TGHB YDYA DS UBGH BRL BRVVAYAL HD LD ROO TBD OGIA UD HAA
```

Results

For each coded message, write what you believe to be the correct deciphering of the text.

You can use your `best_decode` result and the power of the Internet (these are somewhat famous quotes).

Important: Answer Prohibition

Do not post the answer to this question on Campuswire. Do not ask what the answer is. The problem does not ask you to include the code here, so I cannot verify if students actually did or did not attempt the problem on their own machine. As such, do not share what the answer is to this problem. Plus, it's fun to see the code run and I want you to experience that.

Your answers:

Message 1

true message: "BUT FROM THAT MOMENT ON HERMIONE GRANGER BECAME THEIR FRIEND THERE ARE MOST THINGS YOU CANT SHARE WITHOUT ENDING UP LIKING EACH OTHER AND KNOCKING OUT A TWELVE-FOOT MOUNTAIN TROLL IM ONE OF THEM" ("Harry Potter and the Sorcerer's Stone" by J.K. Rowling)

Message 2

true message: "I WISH THE RING HAD NEVER COME TO ME I WISH NONE OF THIS HAD HAPPENED SO DO ALL WHO LIVE TO SEE SUCH TIMES BUT THAT IS NOT FOR THEM TO DECIDE ALL WE HAVE TO DECIDE IS WHAT TO DO WITH THE TIME THAT IS GIVEN TO US" ("The Lord of the Rings" by J.R.R. Tolkien)

Message 3

true message: "FOR INSTANCE, ON THE PLANET EARTH, MAN HAD ALWAYS ASSUMED THAT HE WAS MORE INTELLIGENT THAN DOLPHINS BECAUSE HE HAD ACHIEVED SO MUCH-THE WHEEL, NEW YORK, WARS AND SO ON-WHILST ALL THE DOLPHINS HAD EVER DONE WAS MUCK ABOUT IN THE WATER HAVHIG A GOOD TIME. BUT CONVERSELY, THE DOLPHINS HAD ALWAYS BELIEVED THAT THEY WERE FAR MORE INTELLIGENT THAN MAN-FOR PRECISELY THE SAME REASONS" ("The Hitchhiker's Guide to the Galaxy" by Douglas Adams)

Problem 2: Acceptance ratios in higher dimensions

We will explore the effect of having higher dimensions on acceptance ratios $p_{move} = \frac{f(\mathbf{x}_{proposed})}{f(\mathbf{x}_t)}$

We will look at 3 target distributions in different dimensions.

- The first distribution will be a univariate standard normal distribution (mean 0 and sd 1).
  - Current location is $x_t = 4$.
  - The proposal distribution will be a uniform distribution with a width of 4. ($x_t \sim \text{Unif}[x_t \pm 2]$)
- The second distribution will be a 3D normal distribution (mean (0,0,0) and sigma = Identity matrix(3)).
  - Current location is $\mathbf{x}_t = (4, 4, 4)$.
  - The proposal distribution will be a uniform distribution with a width of 4 in all three directions. Each value can be sampled independently.
  - $x_{1p} \sim \text{Unif}[x_{1t} \pm 2]$, $x_{2p} \sim \text{Unif}[x_{2t} \pm 2]$, $x_{3p} \sim \text{Unif}[x_{3t} \pm 2]$
- The third distribution will be a 10D normal distribution (mean rep(0, 10) and sigma = Identity matrix(10)).
  - Current location is $\mathbf{x}_t = (4, \ldots, 4)$. (4s in all 10 dimenions)
  - The proposal distribution will be a uniform distribution with a width of 4 in all 10 directions. ($x_{dp} \sim \text{Unif}[x_{dt} \pm 2]$ for all d)

Because the distributions have diagonal covariance matrices, the joint PDF can be found using a product of univariate normal densities. There is no need to load a library like `mvtnorm`.

For each target distribution, estimate the proportion of proposed values that would be accepted if the current location is the one provided. You are not running a chain and updating the current location.

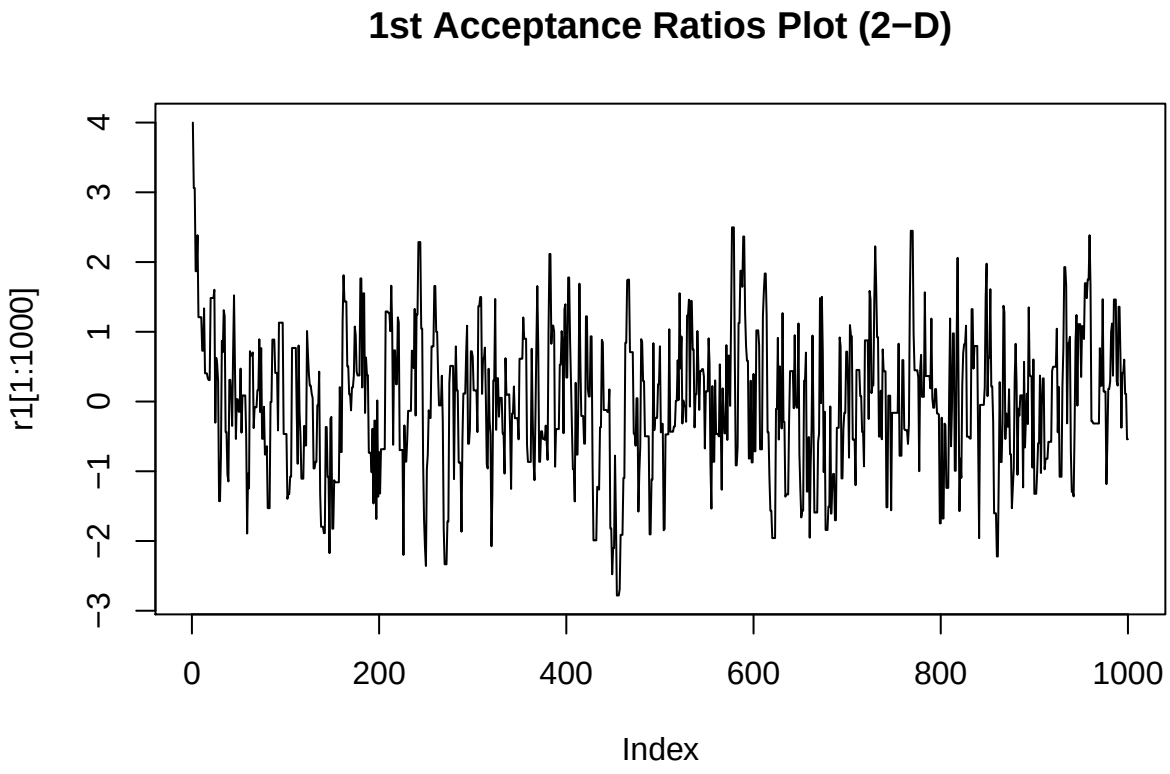
Estimate the acceptance rate by running 10,000 iterations. For each iteration, propose a value, find the acceptance probability p_{move} using the Metropolis algorithm, and then draw a random U to decide whether to accept or not. Keep track of the acceptances and after 10,000 iterations report the estimated acceptance rate.

There should be a pattern that the acceptance rates go down as the number of dimensions increase, though the differences in acceptance rates are likely not be orders of magnitude different.

1st Acceptance Ratios (2-D)

```
set.seed(1)
n <- 10^4
# 1st Target (2-D)
f1 <- function(x) {
  dnorm(x, mean = 0, sd = 1)
}
# 1st Proposal (2-D)
p1 <- function(x) {
  runif(1, min = (x - 2), max = (x + 2)) # width = 4
}

# Set up the MCMC loop (results)
r1 <- rep(NA, n)
r1[1] <- 4  # x_t = 4
```

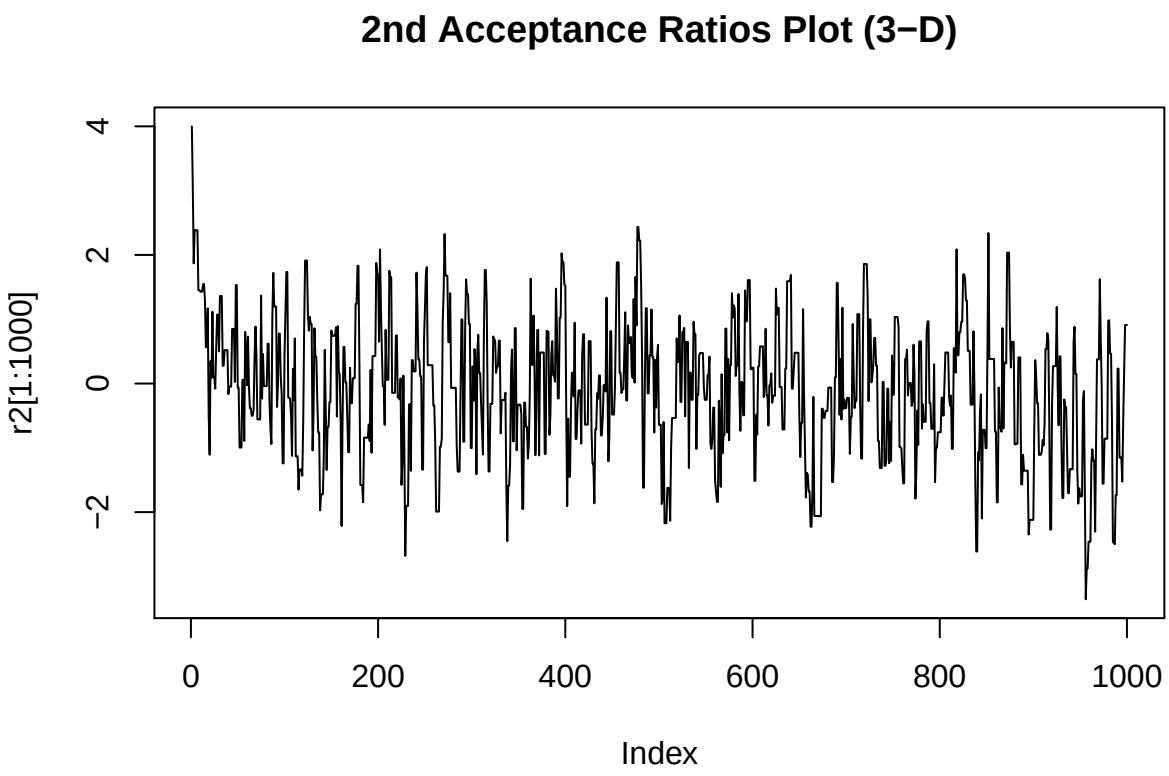
```r
# Perform the MCMC loop
for(i in 1:n){
  current <- r1[i]
  proposed <- p1(current)
  p_move <- f1(proposed) / f1(current)
  U <- runif(1)
  if(U < p_move){
    x_new <- proposed
  } else {
    x_new <- current
  }
  r1[i + 1] <- x_new
}

# create a plot of the first 1000 values in the MCMC
plot(r1[1:1000], type = "l", main = "1st Acceptance Ratios Plot (2-D)")
```



2nd Acceptance Ratios (3-D)

```r
set.seed(1)
n <- 10^4
# 2nd Target (3-D)
f2 <- function(x) {
```

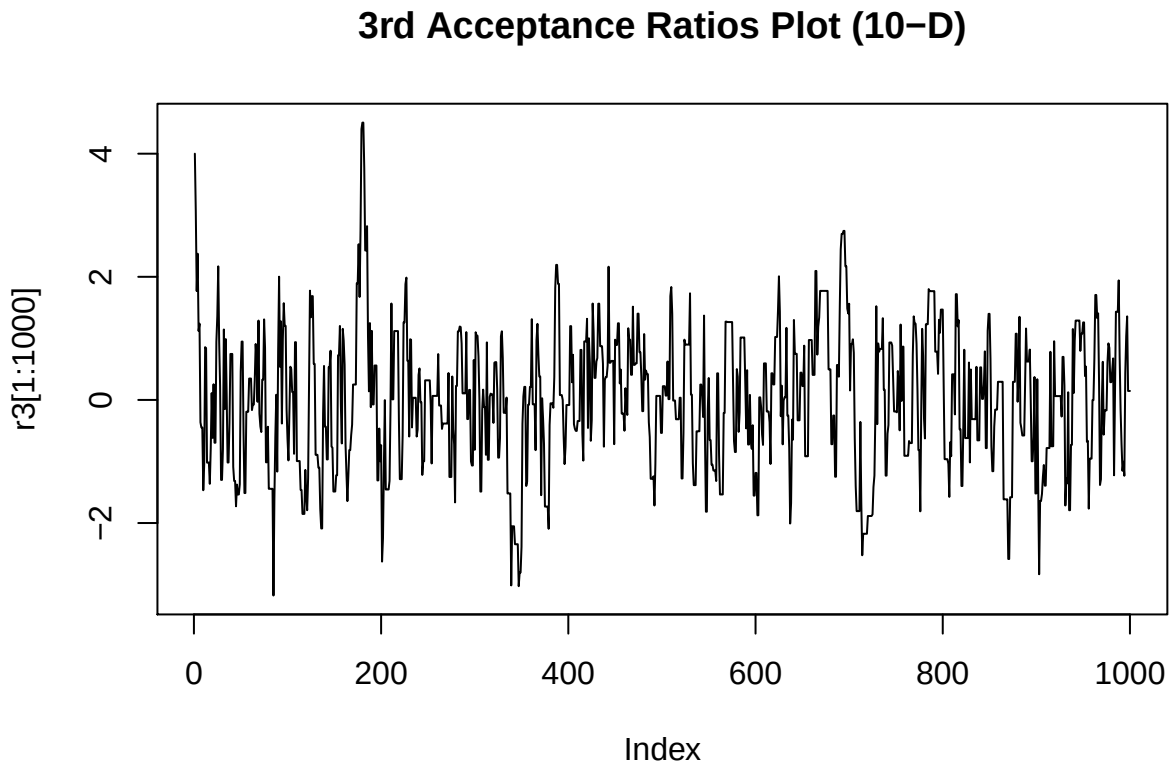
```r
  matrix(dnorm(x, mean = c(0,0,0), sd = diag(3)), nrow = 3)
}
# 2nd Proposal (3-D)
p2 <- function(x) {
  matrix(runif(1, min = (x - 2), max = (x + 2)), nrow = 3) # width = 4
}

# Set up the MCMC loop (results)
r2 <- rep(NA, n)
r2[1] <- c(4,4,4)  # x_t = 4

# Perform the MCMC loop
for(i in 1:n){
  current <- r2[i]
  proposed <- p2(current)
  p_move <- diag(f2(proposed)) / diag(f2(current))
  U <- runif(c(1,1,1))
  if(U < p_move){
    x_new <- proposed
  } else {
    x_new <- current
  }
  r2[i + 1] <- x_new
}

# create a plot of the first 1000 values in the MCMC
plot(r2[1:1000], type = "l", main = "2nd Acceptance Ratios Plot (3-D)")
```



3rd Acceptance Ratios (10-D)

```
set.seed(1)
n <- 10^4
# 3rd Target (10-D)
f3 <- function(x) {
  matrix(dnorm(x, mean = rep(0, 10), sd = diag(10)), nrow = 10)
}
# 3rd Proposal (10-D)
p3 <- function(x) {
  matrix(runif(1, min = (x - 2), max = (x + 2)), nrow = 10) # width = 4
}

# Set up the MCMC loop (results)
r3 <- rep(NA, n)
r3[1] <- rep(4, 10)  # x_t = 4

# Perform the MCMC loop
for(i in 1:n){
  current <- r3[i]
  proposed <- p3(current)
  p_move <- diag(f3(proposed)) / diag(f3(current))
  U <- runif(rep(1, 10))
  if(U < p_move){
```

```
    x_new <- proposed
  } else {
    x_new <- current
  }
  r3[i + 1] <- x_new
}

# create a plot of the first 1000 values in the MCMC
plot(r3[1:1000], type = "l", main = "3rd Acceptance Ratios Plot (10-D)")
```



COMMENTS: We can see the higher dimensions we use, the acceptance rates go down.

Problem 3: Multivariate Metropolis Hastings Algorithm

In the following two problems, we will compare two multivariate MCMC algorithms for sampling from a bivariate normal distribution.

The target distribution will be a bivariate normal distribution centered at (0,0) with covariance matrix `rbind(c(1, .7), c(.7, 1))`.

For the proposal distribution, use a multivariate uniform distribution centered at the current location (x1, x2), with a total span of 2. That is, sample the proposed x_{1p} from $x_{1t} - 1$ to $x_{1t} + 1$, and sample x_{2p} from $x_{2t} - 1$ to $x_{2t} + 1$.

The acceptance ratio p_{move} is:

$$p_{move} = \frac{f(\mathbf{x}_{proposed})}{f(\mathbf{x}_t)} \frac{g(\mathbf{x}_t|\mathbf{x}_{proposed})}{g(\mathbf{x}_{proposed}|\mathbf{x}_t)} = \frac{f(x_{1p}, x_{2p})}{f(x_{1t}, x_{2t})} \frac{g(x_{1t}, x_{2t}|x_{1p}, x_{2p})}{g(x_{1p}, x_{2p}|x_{1t}, x_{2t})}$$

Because our proposal distribution is symmetric, $\frac{g(x_{1t},x_{2t}|x_{1p},x_{2p})}{g(x_{1p},x_{2p}|x_{1t},x_{2t})} = 1$ and p_{move} reduces to $\frac{f(x_{1p},x_{2p})}{f(x_{1t},x_{2t})}$ (and the Metropolis-Hastings algoirthm reduces to the Metropolis algorithm).

Write the code to calculate the multivariate normal density yourself (consult wikipedia https://en.wikipedia.org/wiki/Multivariate_normal_distribution). Do not use `mvtnorm::dmvnorm()`, as the library `mvtnorm` would make sampling from the multivariate normal trivial.

Start at the arbitrary location: (10, 10)

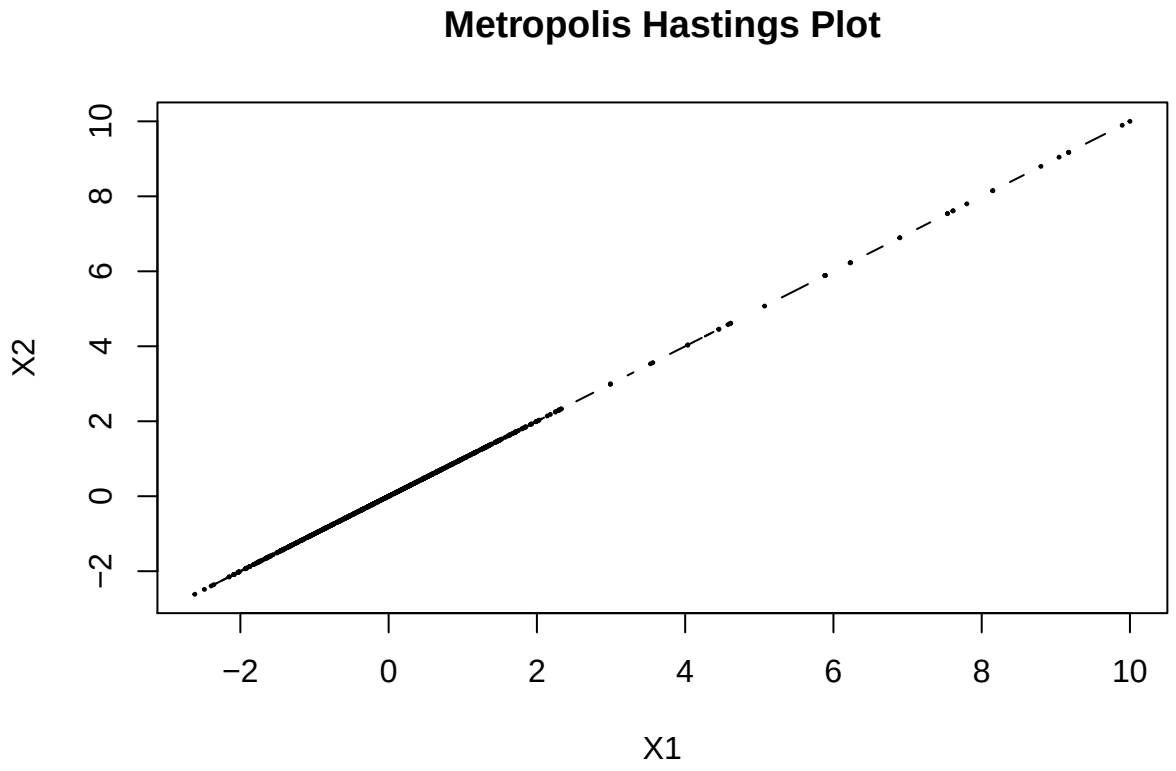
Do 1000 iterations of the Metropolis algorithm.

Create a plot of the results of your chain, and create another plot after removing the 'burn-in' values. (That is, we started in a terrible location and it took a little while for our chain to reach the 'appropriate' region. Remove those exploratory values.)

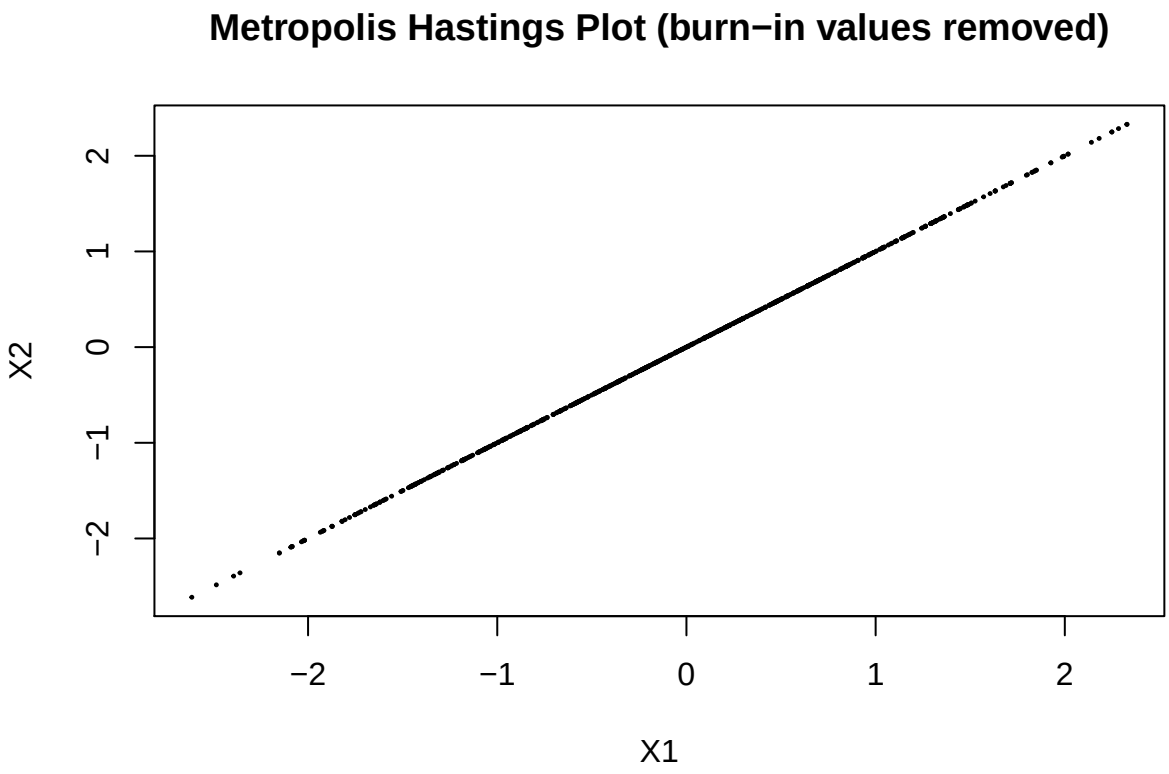
```
# set up the parameters
n <- 10^3

# create the target distribution
target <- function(x) {
  covmat <- rbind(c(1, 0.7), c(0.7, 1))
  matrix(dnorm(x, mean = c(0,0), sd = covmat), nrow = 2)
}

# create the proposal distribution -> width = 2
propose <- function(x) {
  x <- runif(1, min = (x - 1), max = (x + 1))
  return(c(x, x))
}

# Set up the MCMC loop (results)
X <- matrix(rep(NA, n*2), ncol = 2)
X[1,] <- cbind(c(10,10))  # x_t = 10

# Perform the MCMC loop
for(i in 1:n){
  current <- X[i,]
  proposed <- propose(current)
  p_move <- (target(proposed) / target(current))
  U <- runif(c(1,1))
  if(U < p_move){
    x_new <- proposed
  } else {
    x_new <- current
  }
  X[(i + 1),] <- x_new
}
```

```
## Error in `[<-`(`*tmp*`, (i + 1), , value = x_new): subscript out of bounds
```

```r
# the plot command assuming the results are stored in a 1000 x 2 matrix called X
plot(X[ , 1], X[ , 2], type = 'b', pch = 19, cex = 0.2,
     main = "Metropolis Hastings Plot", xlab = "X1", ylab = "X2")
```




```r
# create another plot after removing the 'burn in values'
# adjust as necessary, this assumes the first 100 values are burn-in
plot(X[100:1000, 1], X[100:1000, 2], pch = 19, cex = 0.2,
     main = "Metropolis Hastings Plot (burn-in values removed)",
     xlab = "X1", ylab = "X2")
```



Problem 4: The Gibbs Sampler

Again, the target distribution will be a bivariate normal distribution centered at (0,0) with covariance matrix `rbind(c(1, .7), c(.7, 1))`.

Implement a Gibbs sampler.

In each iteration, you will generate each coordinate individually using the appropriate univariate conditional distribution.

For help deriving conditioning a bivariate normal distribution, I point you to the following resource (See section A.2): http://www.math.chalmers.se/~rootzen/highdimensional/SSP4SE-appA.pdf

You are allowed to use the univariate `rnorm` function to generate random normal values.

Again, start at the arbitrary location: (10, 10)

Let the chain run for 1000 iterations.

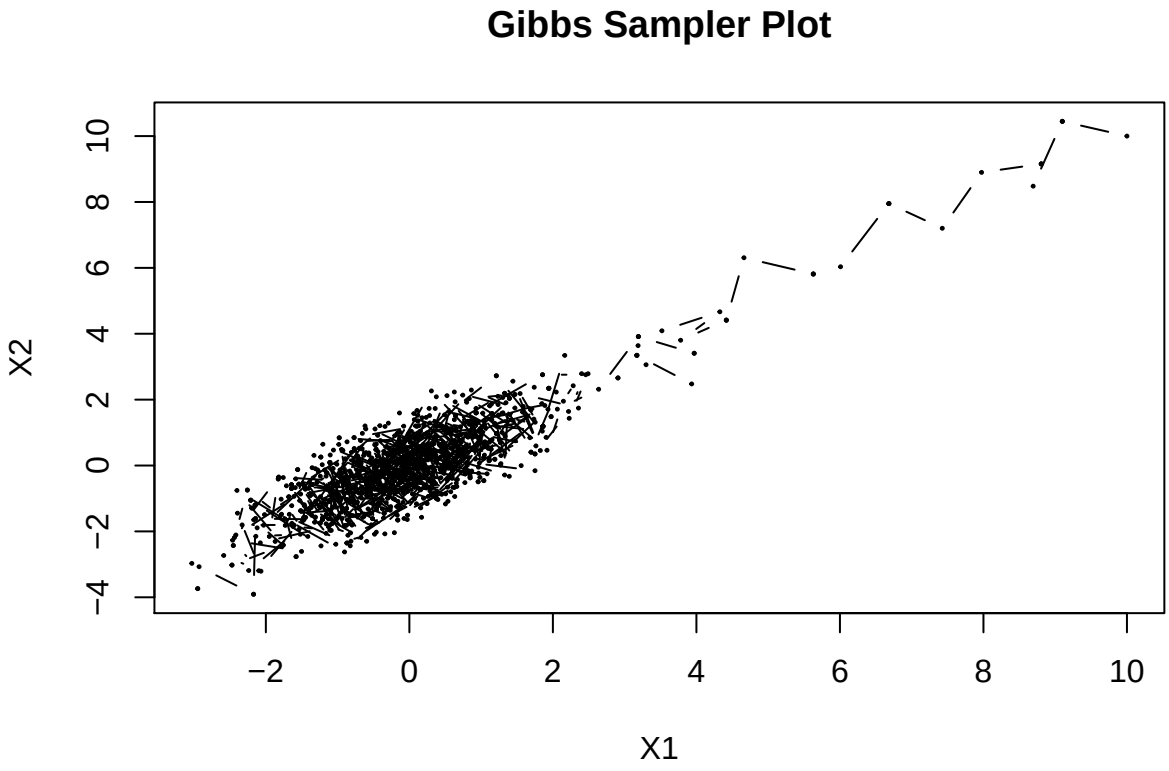
Create a plot of the results of your chain, and create another plot after removing the 'burn-in' values.

```
# set up the parameters
n <- 10^3

# create the target distribution
target <- function(x) {
  covmat <- rbind(c(1, 0.7), c(0.7, 1))
  matrix(dnorm(x, mean = c(0,0), sd = covmat), nrow = 2)
```

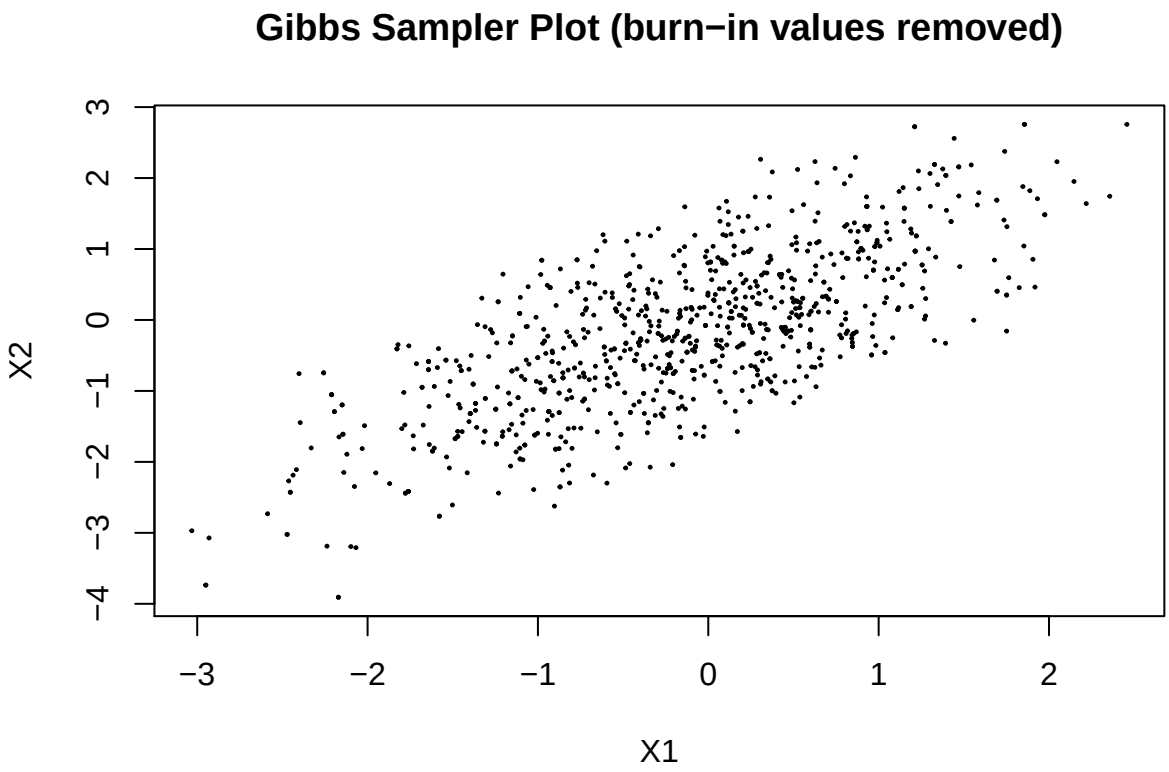
```
}

# create the proposal distribution -> width = 2
propose <- function(x) {
  x1 <- runif(1, min = (x - 1), max = (x + 1))
  x2 <- runif(1, min = (x - 1), max = (x + 1))
  return(c(x1, x2))
}

# Set up the Gibbs Sampler loop (results)
X <- matrix(rep(NA, n*2), ncol = 2)
X[1,] <- cbind(c(10,10))  # x_t = 10

# Perform the Gibbs Sampler loop
for(i in 1:n){
  current <- X[i,]
  proposed <- propose(current)
  p_move <- (target(proposed) / target(current))
  U <- runif(c(1,1))
  if(U < p_move){
    x_new <- proposed
  } else {
    x_new <- current
  }
  X[(i + 1),] <- x_new
}
```

```
## Error in `[<-`(`*tmp*`, (i + 1), , value = x_new): subscript out of bounds
```

```
# the plot command assuming the results are stored in a 1000 x 2 matrix called X
plot(X[ , 1], X[ , 2], type = 'b', pch = 19, cex = 0.2,
     main = "Gibbs Sampler Plot", xlab = "X1", ylab = "X2")
```




```
# create another plot after removing the 'burn in values'
# adjust as necessary, this assumes the first 100 values are burn-in
plot(X[100:1000, 1], X[100:1000, 2], pch = 19, cex = 0.2,
     main = "Gibbs Sampler Plot (burn-in values removed)",
     xlab = "X1", ylab = "X2")
```


Gibbs Sampler Plot (burn−in values removed)
X2
X1
−4
−3
−2
−1
0
1
2
3
−3
−2
−1
0
1
2