# Hw5_Stats_102C

*Junhyuk Jang*

*5/29/2018*

SID : 004 728 134

## Problem 1 (20 pts)

Understand and use the deciphering R code (available in Week 7) to decode the following messages:

message 1: "EFS O TPZS YBZS SP LP BCPZL CBT IQPINQ BNOJQ RQCBRVQT PA DPF JBZS AQNI SABS WBOT SAQ JBS YQRQ BNN CBT AQRQ OC CBT DPFRQ CBT APY TP DPF VZPY OC CBT WBOT BNOJQ DPF CFWS EQ WBOT SAQ JBS PR DPF YPFNTZS ABGQ JPCQ AQRQ"

message 2 (uses a different cipher from message 1): "Y CYLZ OZN JYSD ZFE SNGNJ XPHN OP HN Y CYLZ SPSN PM OZYL ZFE ZFWWNSNE 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"

All punctutation and non-letters (e.g. apostrophes) have been removed in the text.

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

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

```
weight = 0.10
cur.combined.loglike = cur.loglike + weight * cur.loglike.of.text
```

The other is the value to return if a word does not appear in the lexicon.

```
if (is.null(lexical.probability) || is.na(lexical.probability))
  {
    return(1e-10) # for words that don't exist in the lexicon, assign a small non-zero probability
  }
```

Unfortunately, the lexicon we are using has some significant shortcomings (and our code would be greatly improved if the lexicon were of higher quality). While the lexicon does assign probabilities to many words, it also omits some important words like contractions such as I'm, you're, along with plurals of nouns (a major problem). Extra credit to someone who figures out how to use word stemming (see library corpus function stemmer) to improve the deciphering code performance.

Adjusting the 'penalty' for a word not listed in the lexicon allows for words like Im, youre, but also allows for nonsense words.

### Results

For each coded message, display the "max.decode" result. This will not necessarily be the correct deciphering.

Based on your max.decode result, your reasoned deduction, and the power of the Internet (these are somewhat famous quotes), provide what you believe to be the correct deciphering of the text.

#### Important: Answer Prohibition

Do not post the answer to this question on Piazza. 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 kinda fun to see the code run, and I want you to experience that.

**Your answers:**

**Message 1**

max.decode: "BUT I DONT WANT TO GO AMONG MAD PEOPLE ALICE REMARKED OH FOU CANT HELP THAT SAID THE CAT WERE ALL MAD HERE IM MAD FOURE MAD HOW DO FOU KNOW IM MAD SAID ALICE FOU MUST BE SAID THE CAT OR FOU WOULDNT HAVE COME HERE"

true message: "But I don't want to go among mad people," Alice remarked. "Oh, you can't help that," said the Cat: "we're all mad here. I'm mad. You're mad." "How do you know I'm mad?" said Alice. "You must be," said the Cat, "or you wouldn't come here."

**Message 2**

max.decode: "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"

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.

# Problem 2: Multivariate Metropolis-Hastings Algorithm (30 pts)

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 $x_1'$ from $x_1 - 1$ to $x_1 + 1$, and sample $x_2'$ from $x_2 - 1$ to $x_2 + 1$.

Because our proposal distribution is symmetric, the acceptance ratio reduces to $\frac{p(x_1', x_2')}{p(x_1, x_2)}$ (and this is just a Metropolis algorithm).

Write the code to calculate the multivariate normal density yourself (consult wikipedia or your notes). Do not use `mvtnorm::dmvnorm()`, as the library mvtnorm would make sampling from the multivariate normal trivial.

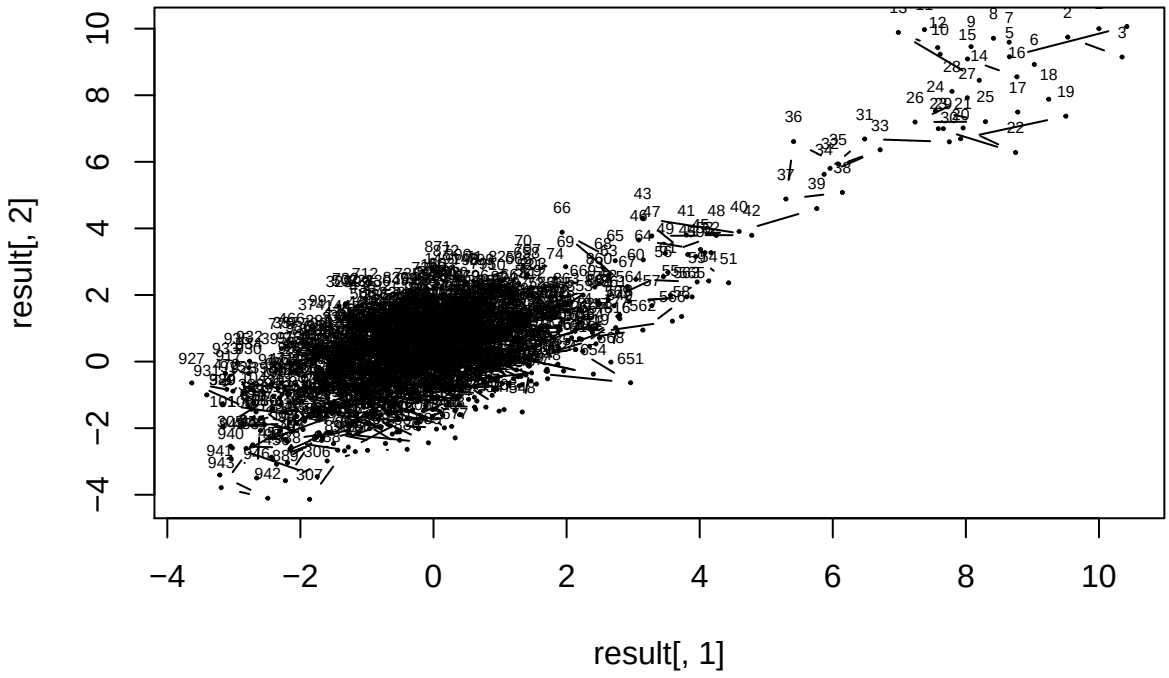
Start at the arbitrary location: (10, 10)

Do 1000 iterations of the Metropolis algorithm.

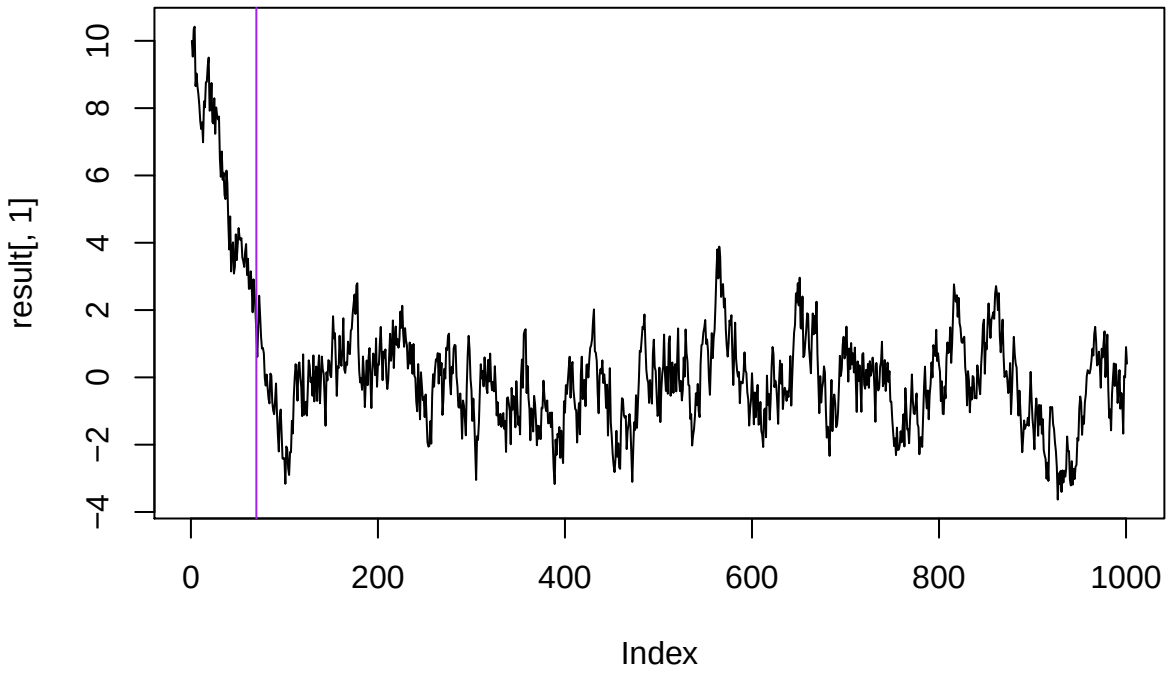
Create a plot of the results of your chain, and create another plot after removing the 'burn-in' values. (That is, we started in a terrible location and it took a little while for our chain to reach the 'appropriate' region. Remove those exploratory values.)

```
# write your code here
mu <- c(0,0)
cov_mat <- rbind(c(1,0.7),c(0.7,1))

normgen <- function(x1,x2) {
        c(runif(1, x1-1, x1+1), runif(1, x2-1, x2+1))
}
my_mvn <- function (x,mu,sig){
```

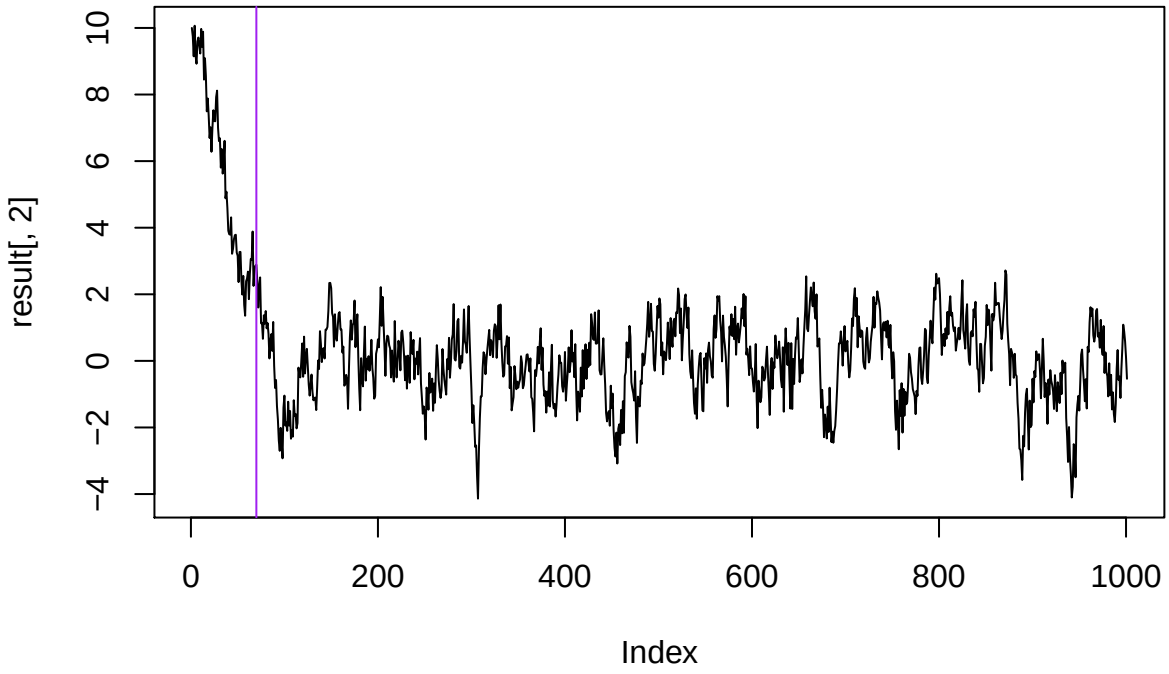
```
        return(det(2*pi*sig)^-0.5*exp(-0.5*t(x-mu) %*% solve(sig) %*% (x-mu)))
}

x <- c(10,10)
result <- x
set.seed(1)
for(i in 1:1000) {
        p1 <- x[1]
        p2 <- x[2]
        propose <- normgen(p1,p2)
        ratio <- min(1, my_mvn(propose,mu,cov_mat)/ my_mvn(x,mu,cov_mat))
        u <- runif(1)
        if(u < ratio) {
                x <- propose
        }
        result <- rbind(result,propose)
}
result <- as.matrix(result,nrow = 1000)
rownames(result) <- 1:1001
colnames(result) <- c('x1','x2')

# the plot command assuming the results are stored in a 1000 x 2 matrix called x
plot(result[,1],result[,2], type = 'b', pch = 19, cex = 0.21)
text(result[,1],result[,2], labels = row.names(result), pos = 3,cex = 0.5)
```



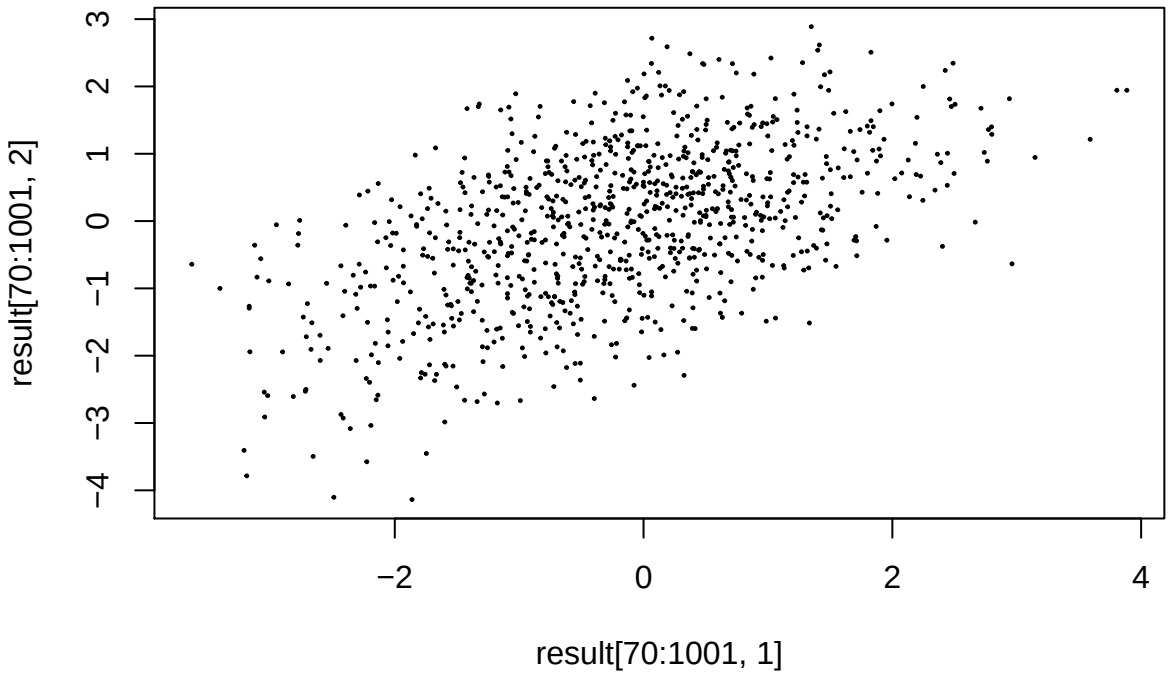

```
plot(result[,1], type = 'l')
abline(v = 70,col = 'purple')
```




```
plot(result[,2], type = 'l')
abline(v = 70,col = 'purple')
```




```
# we can see after 70 iterations,chain starts finishing 'burning-in'

# create another plot after removing the 'burn in values'
plot(result[70:1001,1],result[70:1001,2], pch = 19, cex = 0.2) # adjust as necessary, this assumes the
```




## Problem 3: The Gibbs Sampler (30 pts)

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

Using your notes as a guide, implement a (systematic) Gibbs sampler. In each iteration, you will generate each coordinate individually using the appropriate univariate conditional distribution.

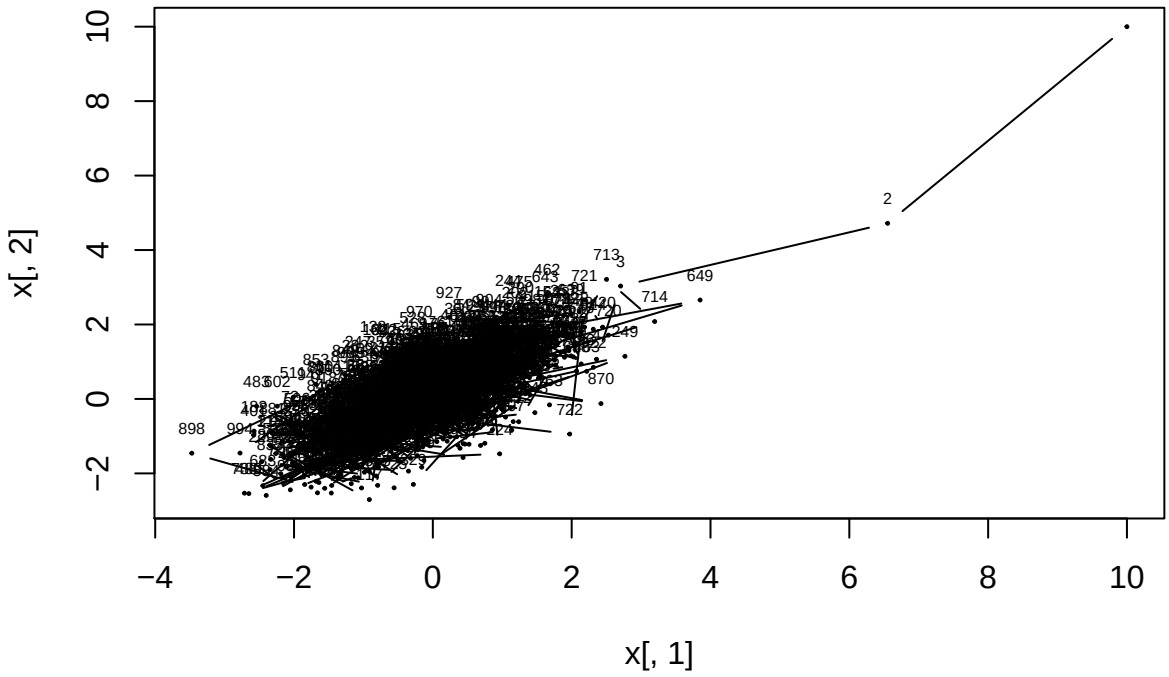
You are allowed to use the univariate `rnorm` function to generate random normal values.

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

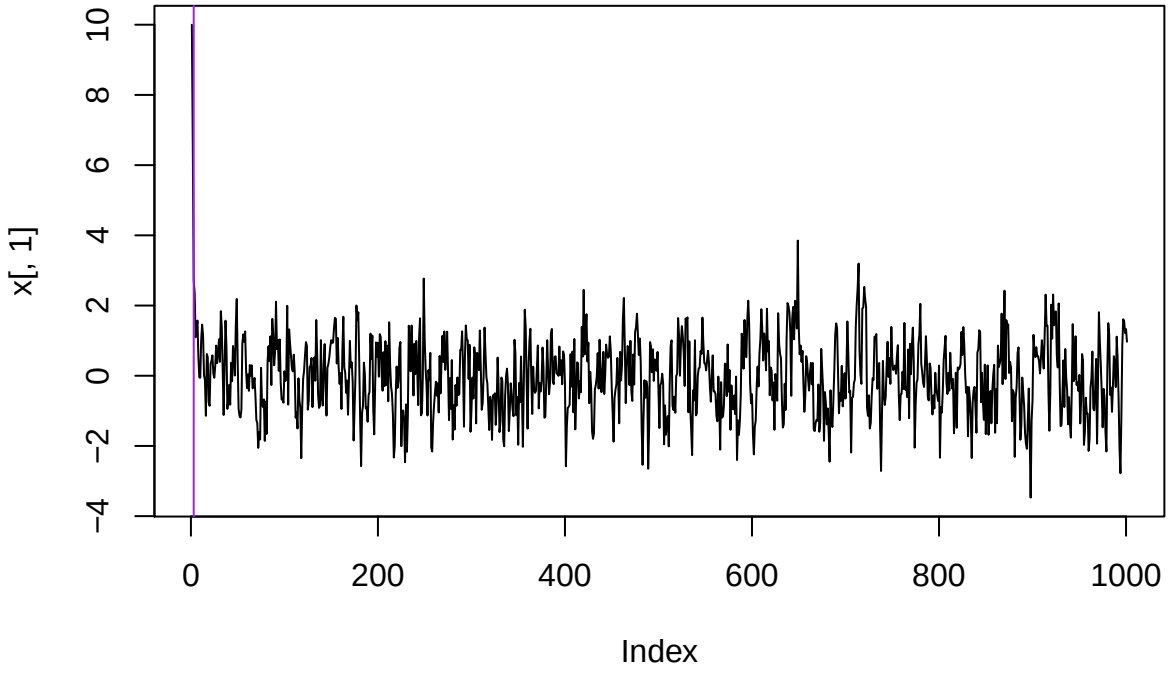
Let the chain run for 1000 iterations.

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

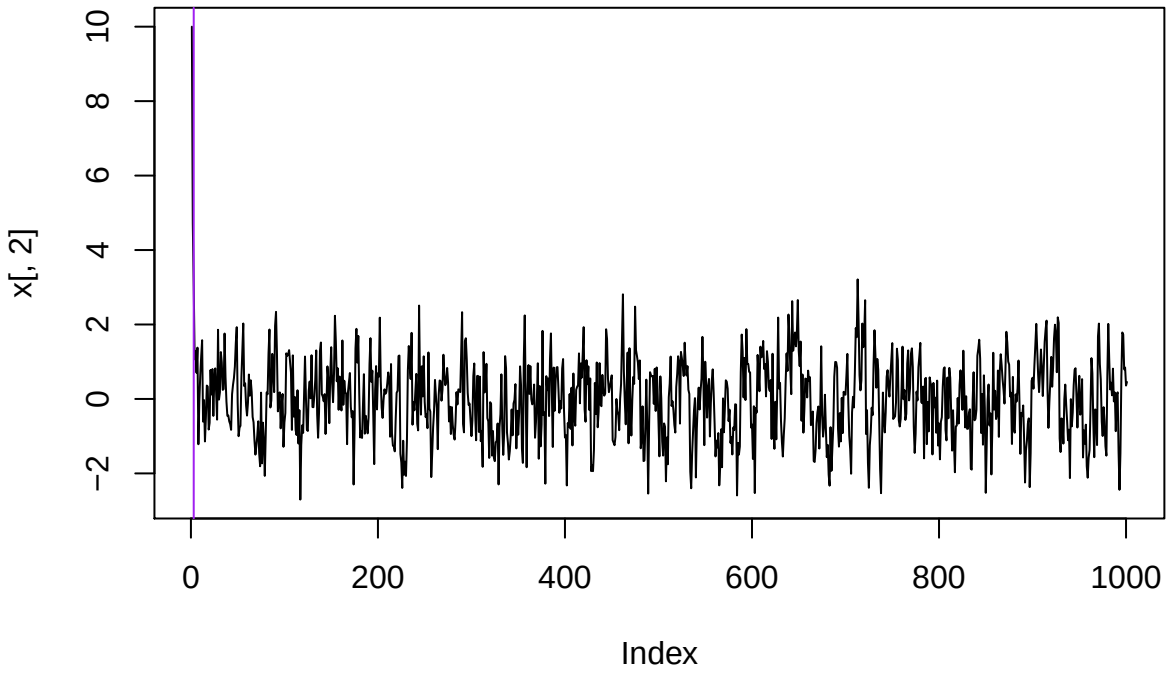
```
# write your code here
x <- matrix( nrow = 1001, ncol = 2)
x1 <- 10
x2 <- 10
x[1, ] <- c(x1, x2)
set.seed(1)
for(i in 1:1000) {
   x1 <- rnorm(1,0.7*x[i,2], sqrt(1-0.7^2))
   x2 <- rnorm(1,0.7*x1, sqrt(1-0.7^2))
   x[i+1, ] <- c(x1, x2)
}
x <- as.data.frame(x)
plot(x[,1],x[,2], type = 'b', pch = 19, cex = 0.2)
text(x[,1],x[,2], labels = row.names(x), pos = 3,cex = 0.5)
```



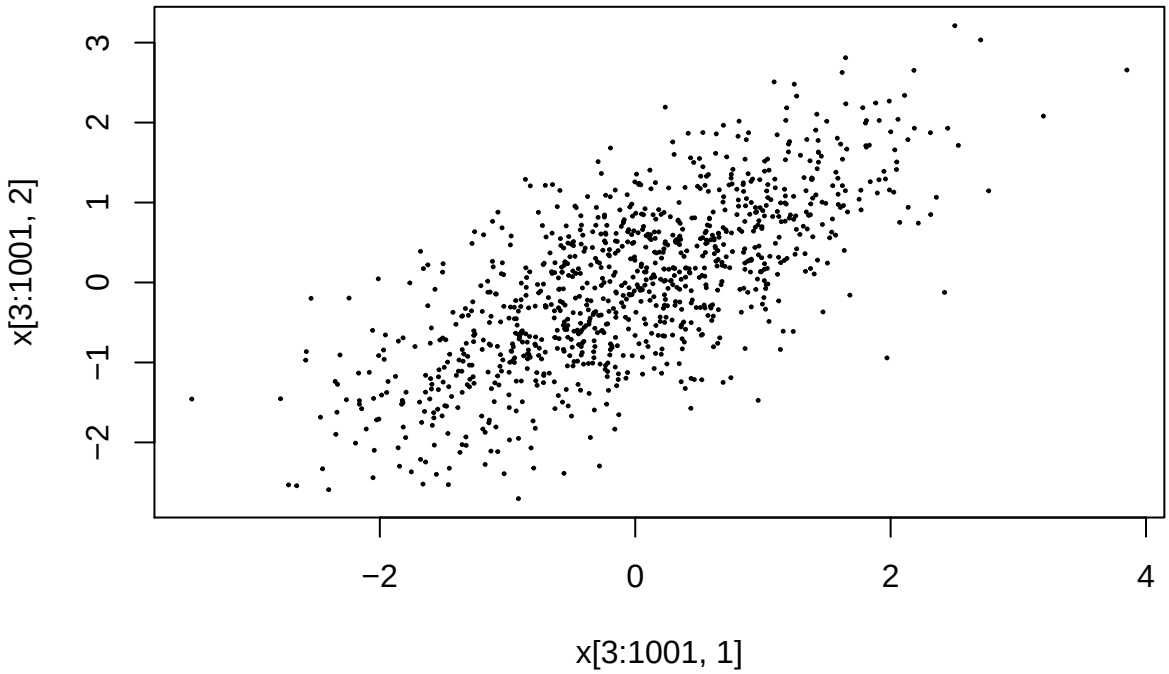

```
plot(x[,1], type = 'l', pch = 19, cex = 0.2)
abline(v = 3,col = 'purple')
```




```
plot(x[,2], type = 'l', pch = 19, cex = 0.2)
abline(v = 3,col = 'purple')
```




```
# we can see after 3 iterations,chain starts finishing 'burning-in'
plot(x[3:1001,1],x[3:1001,2], pch = 19, cex = 0.2)
```




```
# we can see that the gibbs sampler is more efficient than metropolis hasting algorithmm
```