Assignment 1

Question 1

1a)

The code is as follows to generate the matrix requested.

```
setwd("/Users/dylanchou/Desktop/MSiA400")
library(expm)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'expm'
```

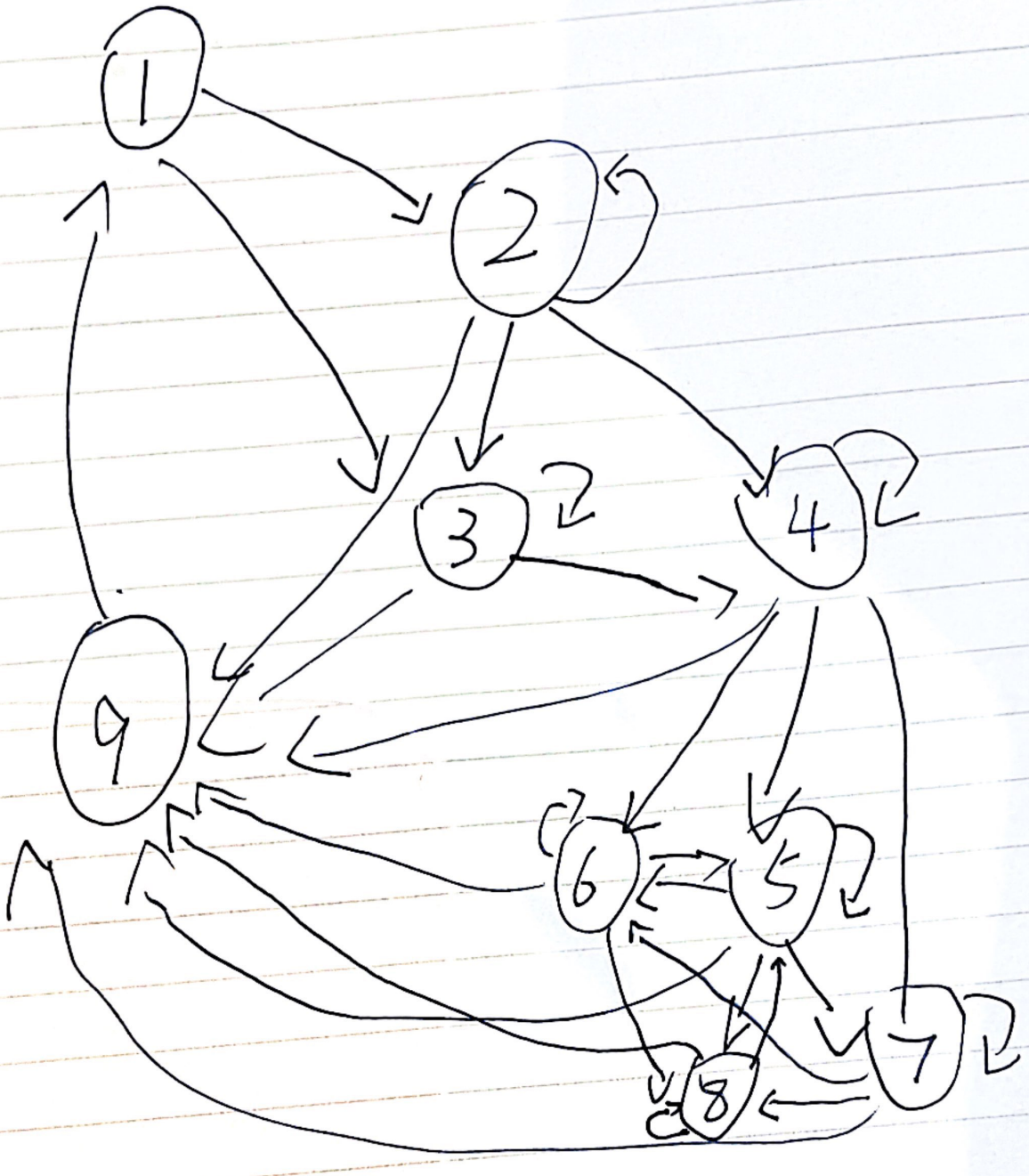
```
## The following object is masked from 'package:Matrix':
##
##     expm
```

```
webtraffic <- read.table("webtraffic.txt",header = TRUE)
kk<-colSums(webtraffic)
Traffic<-matrix(kk,nrow=9,ncol=9,byrow=TRUE)
Traffic[9,1] <- 1000
Traffic
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
##  [1,]    0  447  553    0    0    0    0    0    0
##  [2,]    0   23  230  321    0    0    0    0   63
##  [3,]    0  167   43  520    0    0    0    0   96
##  [4,]    0    0    0   44  158  312  247    0  124
##  [5,]    0    0    0    0   22   52   90  127  218
##  [6,]    0    0    0    0   67   21    0  294   97
##  [7,]    0    0    0    0    0   94    7  185   58
##  [8,]    0    0    0    0  262    0    0   30  344
##  [9,] 1000    0    0    0    0    0    0    0    0
```

1b)

The graph is hand-drawn and from this graph we can see, all states are aperiodic as the greatest common divisor of the steps with non-zero probablity to return to any state is 1. Also, all states communicate with each other. So it is irreducible and recurrent for all states.


1
2
3
4
9
6
5
8
7

1c)

The probality can be eaisly calculated

```
T_P<-Traffic/rowSums(Traffic)
T_P
```

```
##       [,1]       [,2]       [,3]       [,4]      [,5]       [,6]       [,7]
##  [1,]    0 0.44700000 0.55300000 0.00000000 0.0000000 0.00000000 0.00000000
##  [2,]    0 0.03610675 0.36106750 0.50392465 0.0000000 0.00000000 0.00000000
##  [3,]    0 0.20217918 0.05205811 0.62953995 0.0000000 0.00000000 0.00000000
##  [4,]    0 0.00000000 0.00000000 0.04971751 0.1785311 0.35254237 0.27909605
##  [5,]    0 0.00000000 0.00000000 0.00000000 0.0432220 0.10216110 0.17681729
##  [6,]    0 0.00000000 0.00000000 0.00000000 0.1398747 0.04384134 0.00000000
##  [7,]    0 0.00000000 0.00000000 0.00000000 0.0000000 0.27325581 0.02034884
##  [8,]    0 0.00000000 0.00000000 0.00000000 0.4119497 0.00000000 0.00000000
##  [9,]    1 0.00000000 0.00000000 0.00000000 0.0000000 0.00000000 0.00000000
##              [,8]      [,9]
##  [1,] 0.00000000 0.0000000
##  [2,] 0.00000000 0.0989011
##  [3,] 0.00000000 0.1162228
##  [4,] 0.00000000 0.1401130
##  [5,] 0.24950884 0.4282908
##  [6,] 0.61377871 0.2025052
##  [7,] 0.53779070 0.1686047
##  [8,] 0.04716981 0.5408805
##  [9,] 0.00000000 0.0000000
```

1d)

By the formula, we can easily calculated the answer by the following code

```
a <- c(1,rep(0,8))
Five<-a%*%(T_P %^%5)
Five[5]
```

```
## [1] 0.1315178
```

1e)

```
Q <- t(T_P) - diag(9)
Q[9,] <- rep(1, 9)
rhs <- c(rep(0, 8), 1)
Pi <- solve(Q, rhs)
Pi
```

```
## [1] 0.15832806 0.10085497 0.13077897 0.14012033 0.08058898 0.07583914 0.0544648
5
## [8] 0.10069664 0.15832806
```

1f)

By modifying the recursion we can get the following: the total spent time first_time exit is 14.563

```
B <- T_P[1:8, 1:8]
Q <- diag(8) - B
rhs <- c(0.1, 2, 3, 5, 5, 3, 3, 2)
m <- solve(Q, rhs)
m[1]
```

```
## [1] 14.563
```

2

2a)

The sample size is calculated related to lambda n= $1000^{21001/(\text{lambda}2)}$ For example when lambda =1: n=100000000

```
lambda=1
n<- 1000^2*100*1/(lambda^2)
n
```

```
## [1] 1e+08
```

2b)

From the following, we can see in each case the diffference is very small. Within the given 0.001 error

```
exact <- function(lambda){
  res <- 1 / (1 + lambda^2)
  return(res)
}
lambda=1

n<- 1000^2*100*1/(lambda^2)
x1 <- runif(n,0,1)
x2<- sin(log(x1)*-1/lambda)/lambda
mean(x2)
```

```
## [1] 0.4999719
```

```
exact(1)
```

```
## [1] 0.5
```

```
exact(1)-mean(x2)
```

```
## [1] 2.813925e-05
```

```
lambda=2
n<- 1000^2*100*1/(lambda^2)
x1 <- runif(n,0,1)
x2<- sin(log(x1)*-1/lambda)/lambda
mean(x2)
```

```
## [1] 0.2000594
```

```
exact(2)
```

```
## [1] 0.2
```

```
exact(2)-mean(x2)
```

```
## [1] -5.942316e-05
```

```
lambda=4
n<- 1000^2*100*1/(lambda^2)
x1 <- runif(n,0,1)
x2<- sin(log(x1)*-1/lambda)/lambda
mean(x2)
```

```
## [1] 0.05878781
```

```
exact(4)-mean(x2)
```

```
## [1] 3.572315e-05
```

```
exact(4)
```

```
## [1] 0.05882353
```

3

3a)

For this question, we should use the metropolis_hasting algorithm. Because exponential distribution is not symmetric and in this problem, it's not a joint distribution. So, we can only use metropolis_hasting algorithm.

3b)

We use the following code to generate samples from Metropolis_hasting.

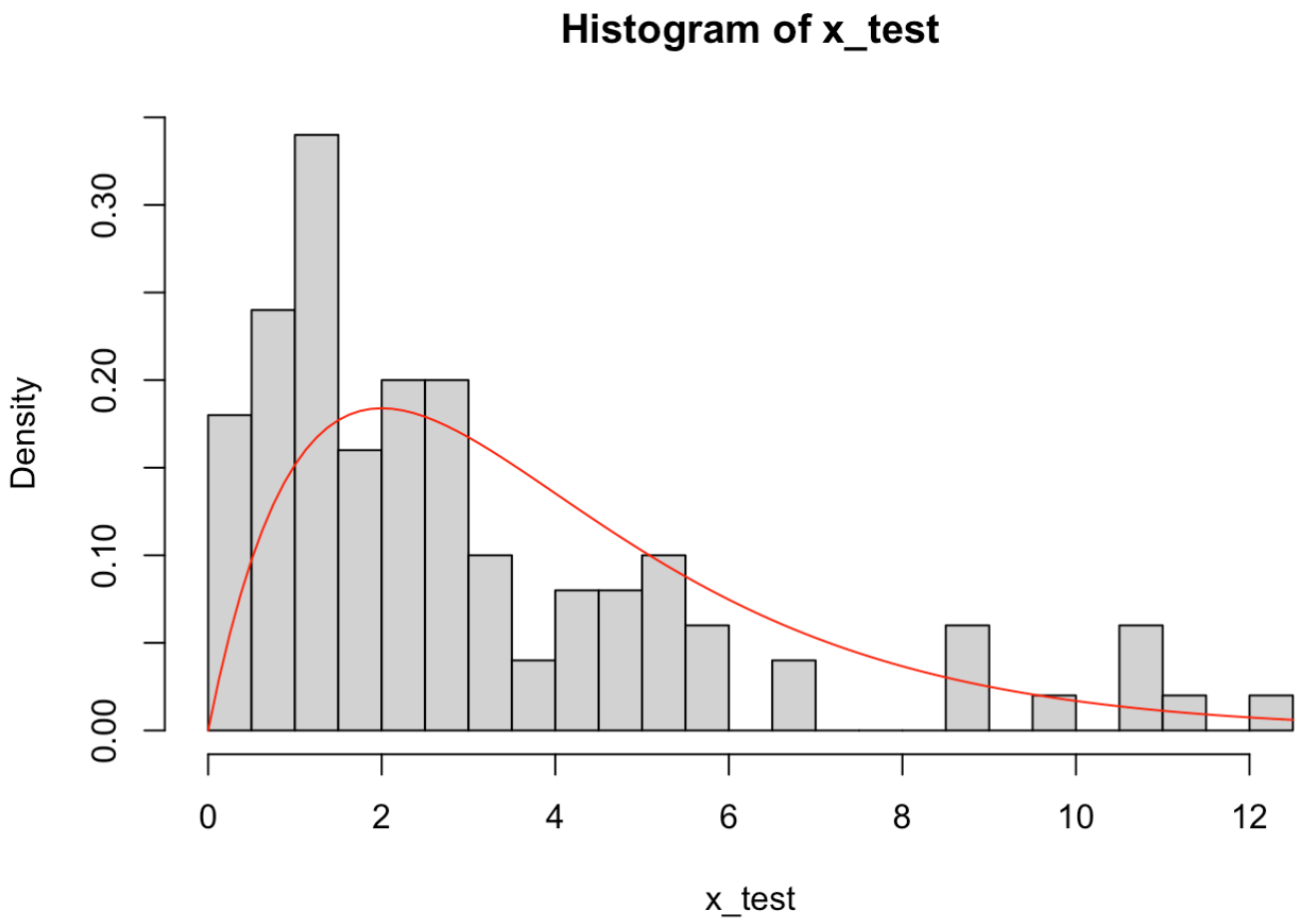
```
q_draw = function(lambda){ return(rexp(1, rate=lambda))
}
q <- function(x){
return(rexp(1, rate = x)) }
p <- function(x, k, theta){ return(x^(k-1)*exp(-1*x/theta))
}
sampler = function(t, k, thet){
x <- rep(NA, t)
x[1] <- 1
for(i in 1:t){
xp = q_draw(x[i])
alpha = (p(xp,k,thet)*q(x[i]))/(p(x[i],k,thet)*q(xp))
u = runif(1,0,1)
if(u<=alpha){
x[i+1] = xp } else{
x[i+1] = x[i] }
}
return(x) }
x <- sampler(15000, 2, 2)
x_test <- x[5001:15000]
x_test <- x_test[seq(1, length(x_test), 100)]
x_test
```

```
##   [1]  3.26500435  1.70295011 12.48247512 10.58701380 11.07753792  4.36742563
##   [7]  2.00327908  2.16702160  1.18410778  0.35728998  0.52807050  2.93252134
##  [13]  1.01725249  2.82714042  4.83154891  1.85200643  2.35116000  5.31148572
##  [19]  1.55814519 10.90243465  1.23810870  1.00381205  1.40175455  3.61892744
##  [25]  2.56789597  2.74051880  0.23419523  1.28655048  8.70007809  8.66782189
##  [31]  1.89065957  1.09393383  2.58579679  1.49790946  1.62862127  0.78566943
##  [37]  1.03589607  4.81737622  4.15699158  1.47182908  1.06966435  0.46668729
##  [43]  6.70253820  2.38546412  1.72124856  0.46156291  4.39874783  2.62295604
##  [49]  0.24875051  2.11608485  0.72807409  0.53265981  9.88993652  1.11427584
##  [55]  8.97606444  6.54562956  0.80705317  3.26164049  5.92926204  2.81106858
##  [61]  0.56952317  3.15933765  2.07693611  1.10203747  2.72974997  0.08405874
##  [67]  5.03336909  5.44020091  5.52020926 10.85439398  0.88538654  1.76435781
##  [73]  5.21944010  1.03207901  2.31065951  5.01708711  4.59824939  0.97377943
##  [79]  0.14660609  0.07205429  2.28918934  1.51922941  4.04628808  0.85809565
##  [85]  1.36306537  2.18584767  0.80314833  3.04322167  5.89993720  3.12808740
##  [91]  1.31530761  0.20021292  0.80928216  2.55701027  1.34765171  3.87401122
##  [97]  2.31915805  0.98367852  4.78218800  2.59149299
```

3C)

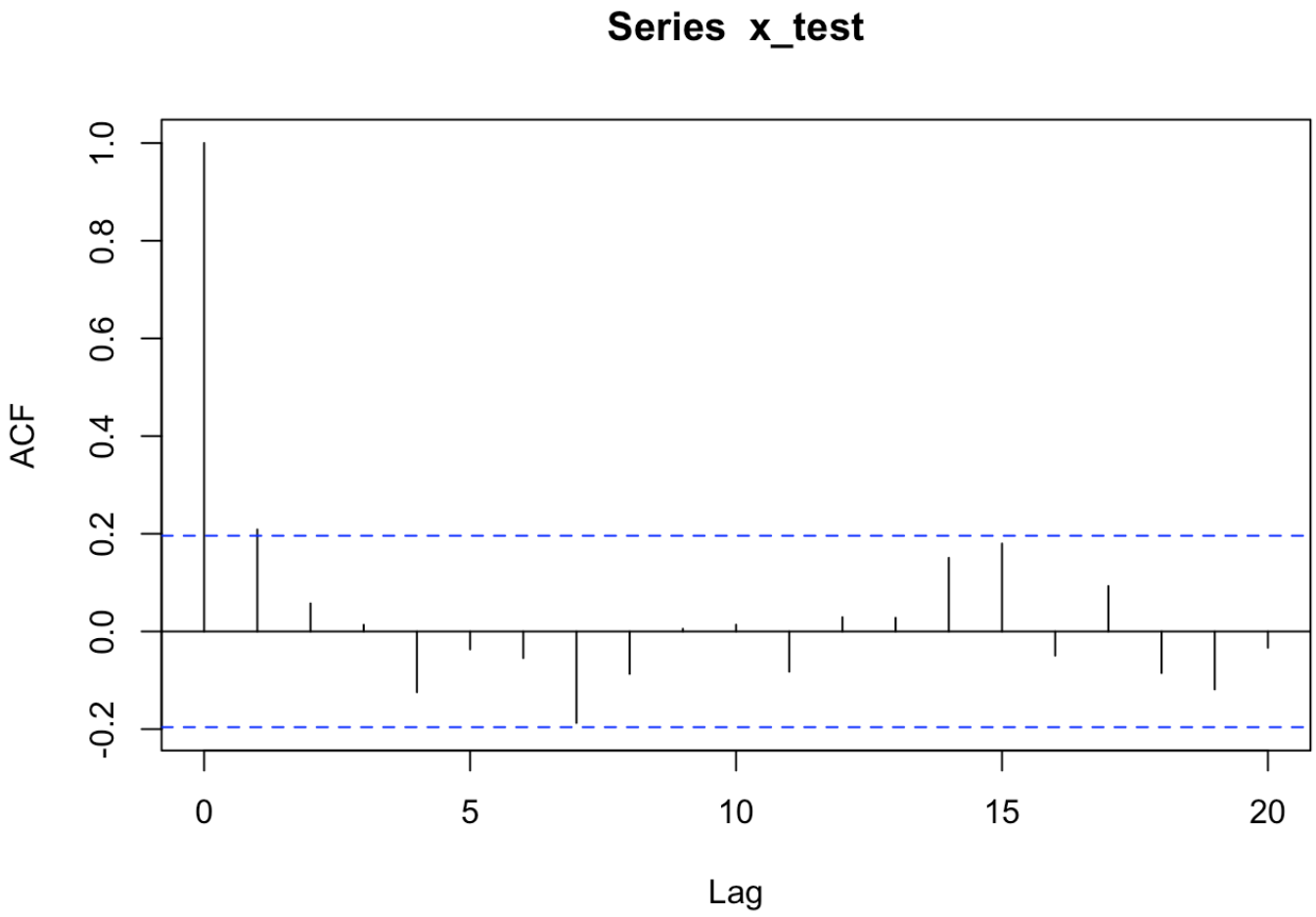
We generate histogram for the samples generated, it's clear gamma

```
hist(x_test, 20 ,prob=TRUE)
curve(dgamma(x, shape = 2, scale = 2), add=TRUE, col="red")
```



Also, we see the ACF from the series, it's totally random

```
acf(x_test)
```


Series x_test
ACF
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
0
5
10
15
20
Lag