HW4

Junhyuk Jang

5/11/2018

Problem 1 - Bayesian Infernce with conjugate priors

The Gamma distribution has a pdf of the form:

$$f(x; \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}$$

It has two shape parameters α and β. Many distributions (including the exponential and chi-squared distributions) can be written in the form of a gamma distribution. We will take a look at the gamma distribution because it serves as a conjugate-prior for many distributions.

When looking at the pdf of the gamma distribution, you can ignore the scary looking constant in the front $\frac{\beta^\alpha}{\Gamma(\alpha)}$, as its only purpose is to make sure the pdf integrates to 1.

The exponential distribution has the following pdf, defined by the rate parameter λ.

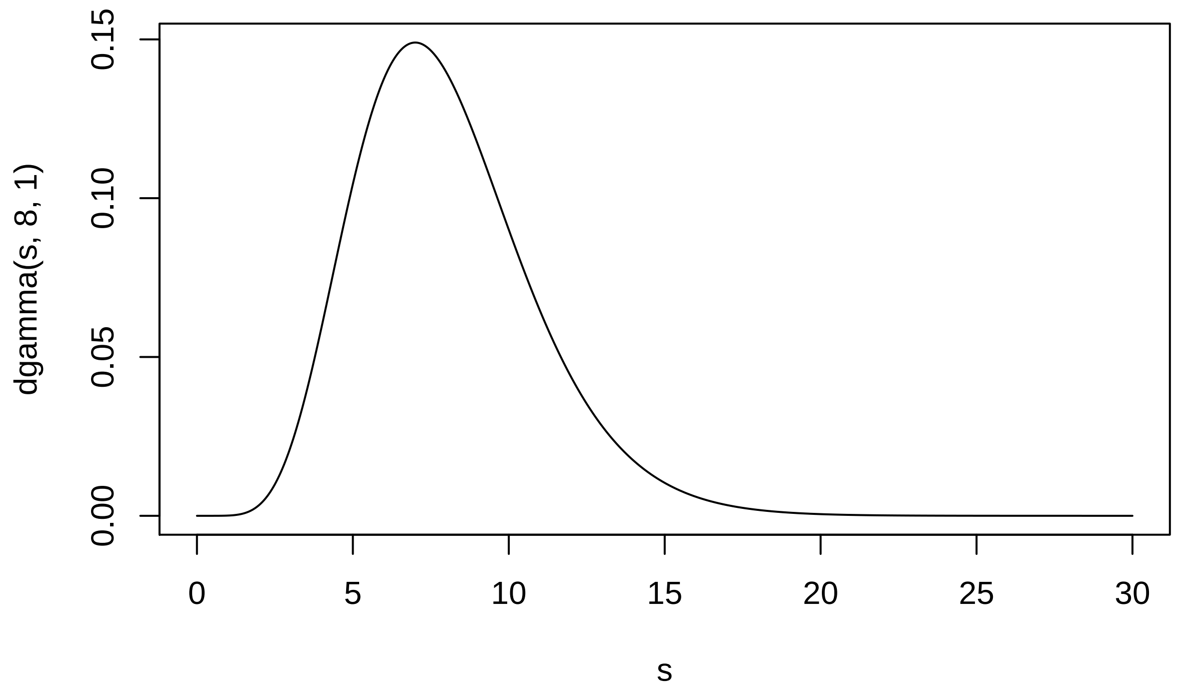
$$f(x; \lambda) = \lambda e^{-\lambda x}$$

The exponential distribution can be used to model the time between events, such as the time between customers entering a store. The λ parameter is the rate (the number of arrivals per time block). If we are trying to model customers entering a store each hour, and if $\lambda = 2$, that means the average rate is two arrivals per hour. The expected time between customers is $1/\lambda = 0.5$, meaning the mean time between customers is half an hour.

In this problem, we are trying to model customers entering a small business and will use Bayesian inference to create a distribution for the rate parameter. You talk to the business owner who tells you that sometimes the business gets busy and will see 20 customers in an hour. Other times, it's slow, and maybe only 3 or 4 customers come. But overall, the owner estimates the average is something like 8 customers per hour, give or take a few.

Taking this into account, you decide to use a Gamma distribution with shape parameters $\alpha = 8$ and $\beta = 1$ as the prior distribution for the rate λ.

```r
s <- seq(0, 30, by = 0.01)
plot(s, dgamma(s, 8, 1), type = 'l')
```



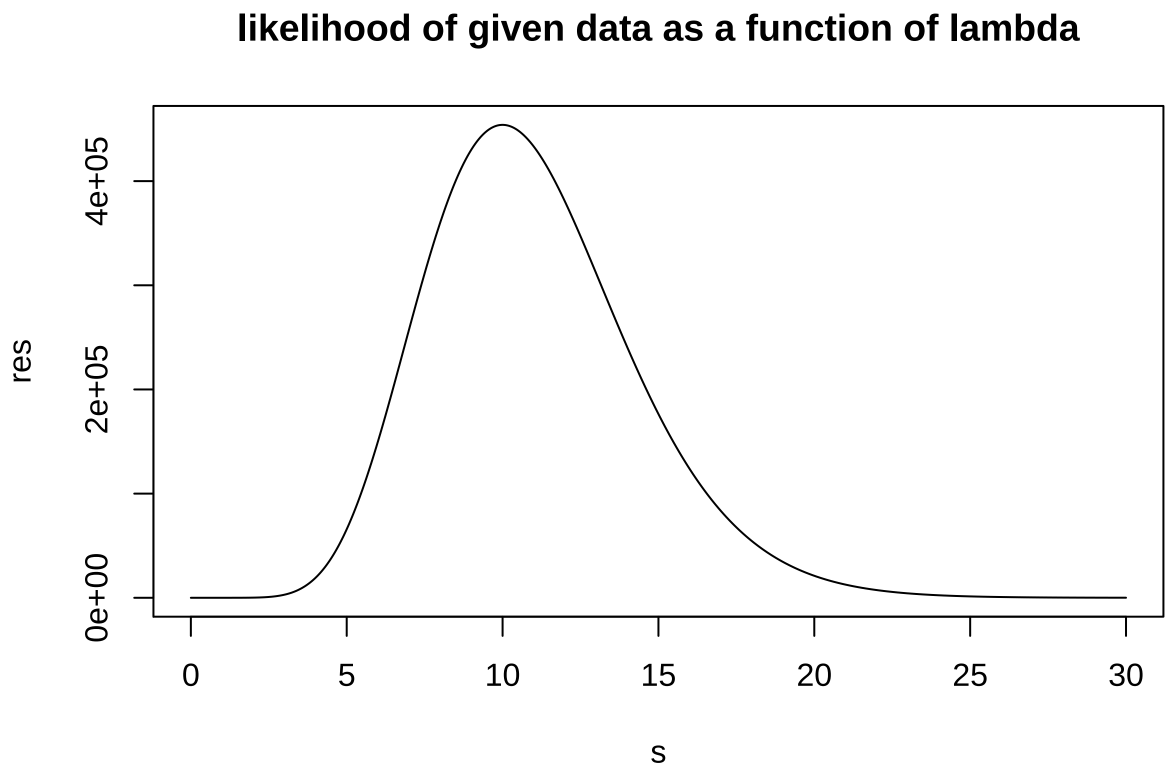
You decide to collect data by timing how long you wait between customer arrivals.

You gather the following values, measured in fractions of an hour:

```
y <- c(0.131, 0.078, 0.297, 0.024, 0.016, 0.057, 0.070, 0.148, 0.070, 0.109)
# after you started the stop watch, the first customer arrived after 7 minutes and 52 seconds (0.131 of
# the next customer came 4 minutes and 41 seconds after that (0.078 of an hour). etc. etc.
# You gathered values for 10 customers total.
# Conveniently, they add up to exactly one hour!
```

I have written a simple function `l()` to calculate the likelihood of the data for a given lambda. It simply takes the pdf of each data point and returns the product.

```
s <- seq(0, 30, by = 0.01)
l <- function(lambda){
  y <- c(0.131, 0.078, 0.297, 0.024, 0.016, 0.057, 0.070, 0.148, 0.070, 0.109)
  prod(lambda * exp(-lambda * y))
}

res <- rep(NA, length(s))
for(i in 1:length(s)){
  res[i] <- l(s[i])
}
plot(s, res, type = 'l', main = 'likelihood of given data as a function of lambda')
```



```
sum(y)
```

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

Calculate the likelihood function for lambda mathematically. The total likelihood of the data (which is assumed to be iid) is the product of each point's probability. You can take advantage of the fact that the sum of the y's is 1.

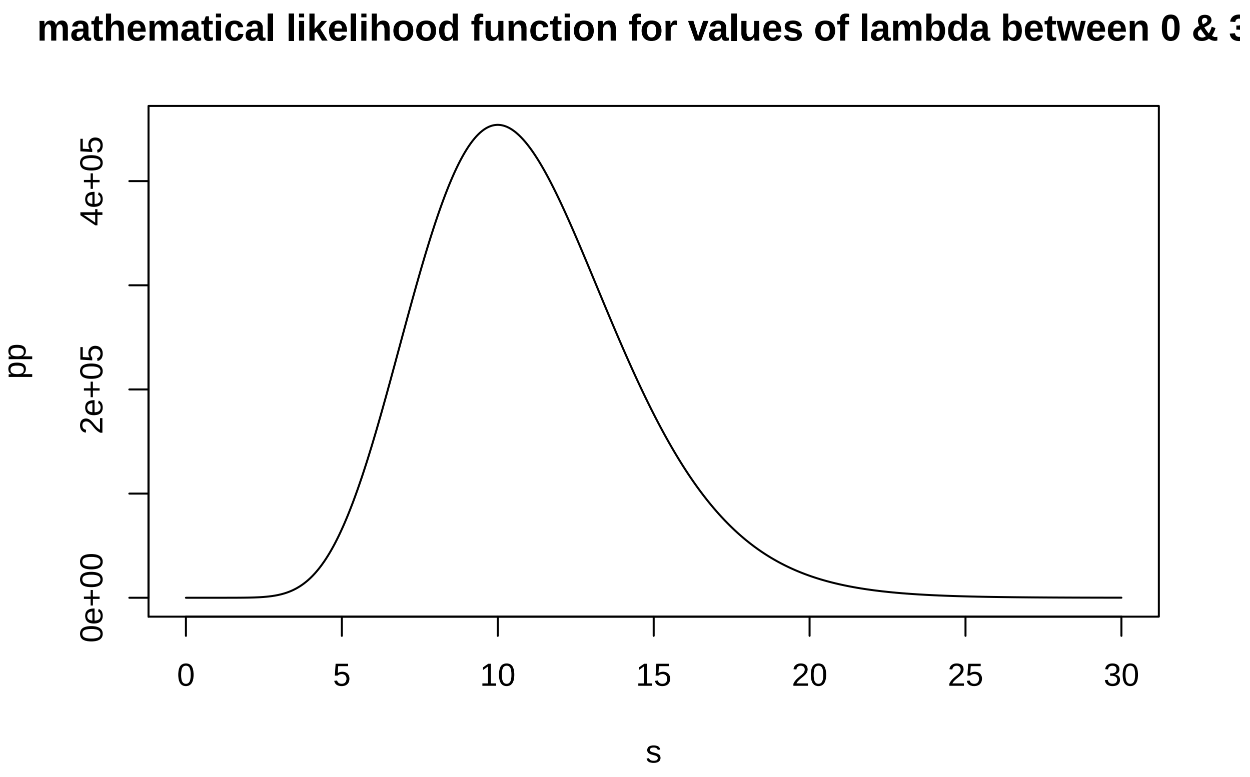
Write down your equation of the likelihood function.

$$p(data|\lambda) = \lambda e^{-\lambda * 0.131} * \lambda e^{-\lambda * 0.078} * \lambda e^{-\lambda * 0.297} * \lambda e^{-\lambda * 0.024} * \lambda e^{-\lambda * 0.016} * \lambda e^{-\lambda * 0.057} * \lambda e^{-\lambda * 0.070} * \lambda e^{-\lambda * 0.148} * \lambda e^{-\lambda * 0.070} * \lambda e^{-\lambda * 0.109}$$

$$likelihood(\lambda) = \lambda^{10} e^{-\lambda}$$

Create a plot of your mathematical likelihood function for values of lambda between 0 and 30. Is it identical to the plot I have provided above?

```
s <- seq(0, 30, by = 0.01)

lkl <- function(x){
        p <- c()
        for(i in 1: length(x)){
               p[i] <- x[i]^10 * exp(-x[i])
        }
        return(p)
}
pp <- lkl(s)
plot(s, pp, type = 'l' , main = 'mathematical likelihood function for values of lambda between 0 & 30' 
```



```r
# As we can see, it is identical to the plot that have provided above.
```

Mathematically, find the posterior distribution of lambda given the data.

Hints: We know that the posterior distribution is proportional to the likelihood times the prior. We also know that the gamma distribution is the conjugate prior for the exponential distribution. This means that the posterior distribution of lambda will be a gamma distribution.

$$p(\lambda|y) \propto p(y|\lambda)p(y)$$

Start by multiplying the likelihood by the prior (a gamma distribution). Then, using algebra, rearrange terms so that the posterior is in the form of a gamma distribution with parameters α and β. If you temporarily ignore the normalizing constant in the gamma distribution, it is in the form $x^{\text{constant1}}e^{\text{-constant2}\cdot x}$

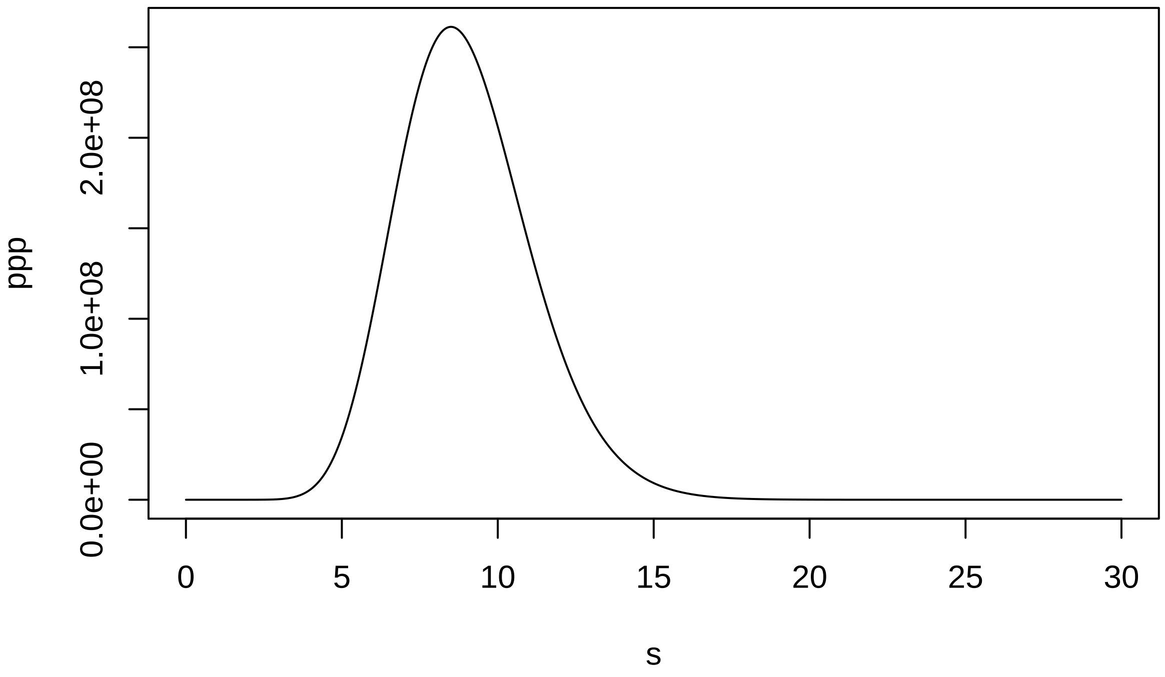
$$posteriordist \propto \lambda^7 e^{-\lambda} * \lambda^{10} e^{-\lambda} \propto Gamma(18, 2)$$

Your answer: The posterior distribution of lambda given the data is a gamma distribution with parameters

```r
# The posterior distribution of lambda given the data is
# a gamma distribution with shape parameters
# alpha = 18 and beta = 2
```

Graph the posterior distribution.

```r
s <- seq(0, 30, by = 0.01)

post <- function(x){
        a <- c()
        for(i in 1: length(x)){
              a[i] <- x[i]^17*exp(-2*x[i])
        }
        return(a)
}
ppp <- post(s)
plot(s, ppp, type = 'l')
```



Problem 2 - Transition Matrix and Stationary Distribution (Two state case)

Imagine a two-state Markov chain. With state 1 representing CA and state 2 representing TX.

Let's pretend that each year, 9% of Californians move to TX and that 12% of Texans move to CA.

Create and display a 2x2 transition matrix **P** to represent the transition probabilities.

Calculate and display the stationary distribution **w**, so that $\mathbf{wP} = \mathbf{w}$.

```r
P <- matrix(c(0.91,0.09,0.12,0.88), byrow = TRUE, nrow = 2)
P
```

```
##      [,1] [,2]
## [1,] 0.91 0.09
## [2,] 0.12 0.88
```

```r
# write your code here
(w <- matrix(c(P[2,1]/(P[1,2]+P[2,1]), P[1,2]/(P[1,2]+P[2,1])), nrow = 1))
```

```
##           [,1]      [,2]
## [1,] 0.5714286 0.4285714
```

```r
identical(w %*% P,w)
```

```
## [1] TRUE
```

```r
# We can see, w %*% p = w
```

The form $\mathbf{wP} = \mathbf{w}$ is very similar to the definition of an eigenvector. Except with eigenvectors, we would write $\mathbf{Ax} = \lambda\mathbf{x}$. (The multiplication is the matrix times the vector, rather than the vector times the matrix.)

If we take transposes, $(\mathbf{wP})^T = \mathbf{w}^T$, we get $\mathbf{P}^T\mathbf{w}^T = \mathbf{w}^T$. Thus, the transpose of the transition matrix has an eigenvector with an eigenvalue of 1 that is equal to the stationary distribution.

Find the eigenvectors of P transpose. Show that your stationary distribution is a scalar multiple of the positive eigenvector.

```
# The eigenvectors of P transpose.
(eigen(t(P)))
```

```
## eigen() decomposition
## $values
## [1] 1.00 0.79
##
## $vectors
##      [,1]       [,2]
## [1,]  0.8 -0.7071068
## [2,]  0.6  0.7071068
```

```
# stationary distribution is a scalar multiple of the positive eigenvector.
(eigenv <- eigen(t(P))$vectors[,1])
```

```
## [1] 0.8 0.6
```

```
(w /eigenv)
```

```
##           [,1]      [,2]
## [1,] 0.7142857 0.7142857
```

```
0.7142857 * eigenv
```

```
## [1] 0.5714286 0.4285714
```

```
as.numeric(w)
```

```
## [1] 0.5714286 0.4285714
```

```
all.equal(0.7142857 * eigenv,as.numeric(w),tolerance = 1e-7)
```

```
## [1] TRUE
```

Problem 3 - Transition Matrix and Stationary Distribution (7 island example)

Look at the example with the politician visiting the island chain in chapter 7 of the textbook, Doing Bayesian Data Analysis.

Create and display the full 7 x 7 transition matrix P. Populate the matrix with actual decimal values, and not symbols.

Start with w = c(0,0,0,1,0,0,0)

Multiply w by P 6 times and show the results after each iteration. (for example, after the first multiplication, w should equal c(0,0, 0.375, 0.125, 0.5, 0, 0))

Find the eigenvectors of the transpose of the transition matrix. Show that it is a scalar multiple of the stationary (target) distribution specified by the example.

```
P <- matrix(c(0.5,0.5,0,0,0,0,0,0.25,0.25,0.5,0,0,0,0,0,1/3,1/6,0.5,0,0,0,0,
             0,0.375,0.125,0.5,0,0,0,0,0,0.4,0.1,0.5,0,0,0,0,0,0.4166667,
             0.08333333,0.5,0,0,0,0,0,0.4285714,0.5714286),nrow=7,byrow=T)
P
```

```
##      [,1]      [,2]      [,3]  [,4]      [,5]       [,6]      [,7]
## [1,] 0.50 0.5000000 0.0000000 0.000 0.0000000 0.00000000 0.0000000
## [2,] 0.25 0.2500000 0.5000000 0.000 0.0000000 0.00000000 0.0000000
## [3,] 0.00 0.3333333 0.1666667 0.500 0.0000000 0.00000000 0.0000000
## [4,] 0.00 0.0000000 0.3750000 0.125 0.5000000 0.00000000 0.0000000
## [5,] 0.00 0.0000000 0.0000000 0.400 0.1000000 0.50000000 0.0000000
## [6,] 0.00 0.0000000 0.0000000 0.000 0.4166667 0.08333333 0.5000000
## [7,] 0.00 0.0000000 0.0000000 0.000 0.0000000 0.42857140 0.5714286
```

```
# Initial
w = c(0,0,0,1,0,0,0)

# 1 iteration
(w <- w %*% P)
```

```
##      [,1] [,2]  [,3]  [,4] [,5] [,6] [,7]
## [1,]    0    0 0.375 0.125  0.5    0    0
```

```
# 2 iteration
(w <- w %*% P)
```

```
##      [,1]  [,2]     [,3]     [,4]   [,5] [,6] [,7]
## [1,]    0 0.125 0.109375 0.403125 0.1125 0.25    0
```

```
# 3 iteration
(w <- w %*% P)
```

```
##         [,1]       [,2]     [,3]      [,4]      [,5]       [,6]  [,7]
## [1,] 0.03125 0.06770833 0.231901 0.1500781 0.3169792 0.07708333 0.125
```

```
# 4 iteration
(w <- w %*% P)
```

```
##            [,1]      [,2]      [,3]     [,4]     [,5]      [,6]      [,7]
## [1,] 0.03255208 0.1098524 0.1287836 0.261502 0.138855 0.2184846 0.1099702
```

```
# 5 iteration
(w <- w %*% P)
```

```
##            [,1]       [,2]      [,3]      [,4]      [,5]      [,6]
## [1,] 0.04373915 0.08666703 0.1744534 0.1526216 0.2356717 0.1347647
##           [,7]
## [1,] 0.1720825
```

```
# 6 iteration
(w <- w %*% P)
```

```
##            [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
## [1,] 0.04353633 0.1016875 0.1296422 0.2005731 0.1560299 0.2028159
##           [,7]
## [1,] 0.1657152
```

```
# Find the eigenvectors of the transpose of the transition matrix.
(eigenvector<- eigen(t(P)))
```

```
## eigen() decomposition
## $values
## [1]  1.00000001  0.88703286 -0.66675124  0.65010696 -0.37404143  0.32891817
## [7] -0.02883674
##
## $vectors
##              [,1]       [,2]        [,3]        [,4]        [,5]        [,6]
## [1,] 0.08451542 -0.2493564  0.02552355  0.29478898  0.09510561  0.26842968
## [2,] 0.16903084 -0.3860365 -0.11911853  0.17699952 -0.33250497 -0.18369376
## [3,] 0.25354627 -0.3637192  0.28932087 -0.22972725  0.47983221 -0.44613485
## [4,] 0.33806170 -0.1839807 -0.48417580 -0.53215778 -0.24852447  0.05189554
## [5,] 0.42257713  0.1041507  0.59671590 -0.41144033 -0.28973025  0.58412467
## [6,] 0.50709254  0.4175048 -0.51706737  0.09538247  0.62785525  0.25864553
## [7,] 0.59160799  0.6614372  0.20880140  0.60615439 -0.33203340 -0.53326683
##             [,7]
## [1,]  0.18961361
## [2,] -0.40109857
## [3,]  0.05110263
## [4,]  0.50815613
## [5,] -0.25931099
## [6,] -0.52960623
## [7,]  0.44114344
```

```
# Show that it is a scalar multiple of the stationary (target) distribution
# specified by the example.
(target <- 1:7 / 28)
```

```
## [1] 0.03571429 0.07142857 0.10714286 0.14285714 0.17857143 0.21428571
## [7] 0.25000000
```

```
pos_eigen <- eigenvector$vectors[,1]
(target/pos_eigen)
```

```
## [1] 0.4225772 0.4225772 0.4225771 0.4225771 0.4225771 0.4225771 0.4225771
```

```
0.4225772 * pos_eigen
```

```
## [1] 0.03571429 0.07142858 0.10714287 0.14285717 0.17857146 0.21428575
## [7] 0.25000005
```

```
as.numeric(target %*% P)
```

```
## [1] 0.03571429 0.07142857 0.10714286 0.14285714 0.17857144 0.21428571
## [7] 0.25000001
```

```
all.equal(0.4225772 * pos_eigen,as.numeric(target %*% P),tolerance = 1e-6)
```

```
## [1] TRUE
```

Multiply w by P 500 times. Show the results after the final iteration. Do NOT show the steps in between. Did the distribution converge to the stationary distribution?

```
w = c(0,0,0,1,0,0,0)

for(i in 1:500){
        w <- w %*% P
}
w
```

```
##           [,1]      [,2]      [,3]      [,4]     [,5]      [,6]      [,7]
## [1,] 0.0357144 0.0714288 0.1071432 0.1428576 0.178572 0.2142864 0.2500008
```

```
target
```

```
## [1] 0.03571429 0.07142857 0.10714286 0.14285714 0.17857143 0.21428571
## [7] 0.25000000
```

```r
all.equal(as.numeric(w),target,tolerance = 1e-5)
```

```
## [1] TRUE
```

```r
# we can see the distribution converge to the stationary(target) distribution.
```

Problem 4 - MCMC (Metropolis Algorithm) for a single continuous random variable

We will compare Rejection Sampling to the Metropolis Algorithm for producing a sample from a distribution.

The logisitic distribution is a unimodal and symmetric distribution, where the CDF is a logistic curve. The shape is similar to a normal distribution, but has heavier tails (though not as heavy as a Cauchy distribution).

The PDF is:

$$f(x;\mu,s) = \frac{1}{s}\frac{e^{-(\frac{x-\mu}{s})}}{\left(1+e^{-(\frac{x-\mu}{s})}\right)^2}$$

Luckily, this is implemented for us in R with `dlogis()`, which you are allowed to use to calculate the probability density of a (proposed) value.

We will generate two samples drawn from a logistic distribution with mean = 0 and scale = 1.

Task 4A:

First generate a sample from the logistic distribution using rejection sampling. Propose 10ˆ4 values from a random uniform distribution from -20 to 20. Calculate the necessary constant M, and implement rejection sampling. If you propose 10ˆ4 values, how many values do you end accepting?

After generating your sample, plot the empirical CDF, and plot the theoretic CDF (using plogis).

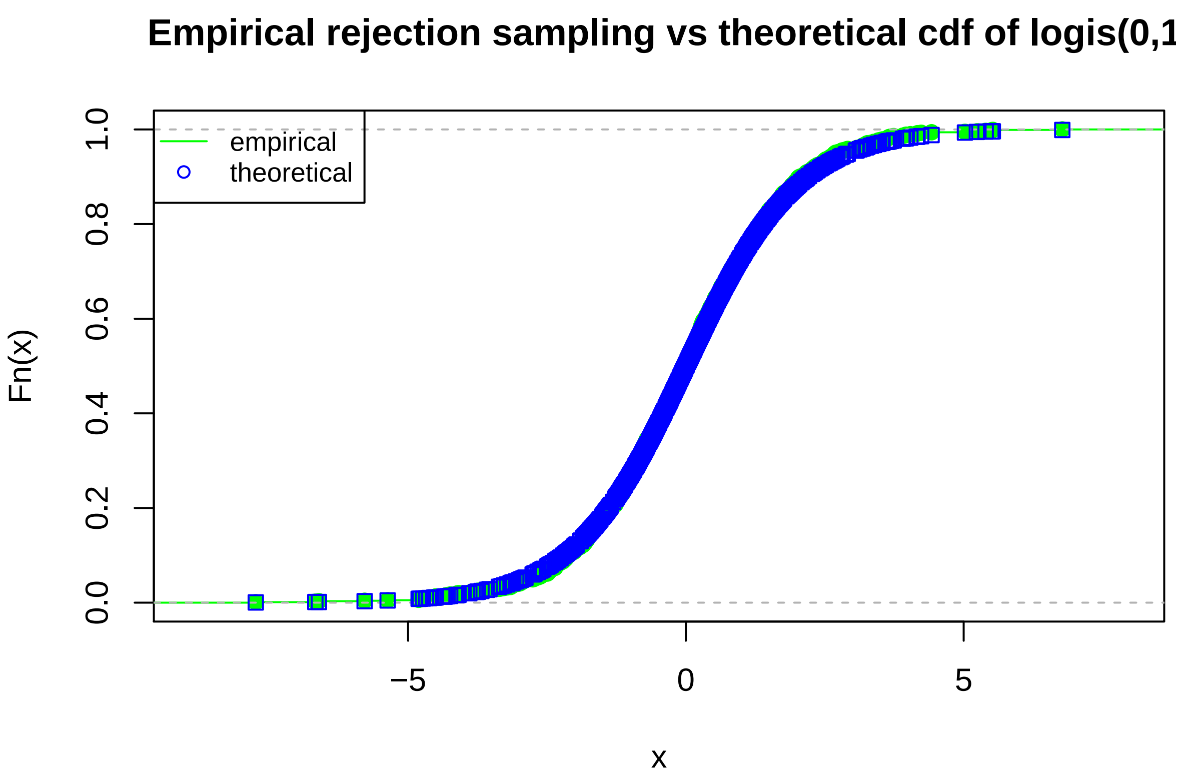
```r
set.seed(1)
n <- 10^4
x <- runif(n,-20,20)
uni <- runif(n,0,1)

f <- function(x) exp(-x)/(1+ exp(-x))^2

# g(x) is uniform dist with pdf 1/40.
# since we know the max value of logistic distribution occurs at mean
(M <- dlogis(0,0,1) * 40)
```

```
## [1] 10
```

```r
#rejection sampling
M_g <- M/40
fx <- dlogis(x,0,1)
accept <- uni <= (fx/M_g)
r_accept <- x[accept]
(n_accept<- length(r_accept))
```

```
## [1] 994
```

```r
plot(ecdf(r_accept), col='green' ,main = ' Empirical rejection sampling vs theoretical cdf of logis(0,1
points(r_accept, plogis(r_accept,0,1), col= 'blue', pch=0.8)
legend('topleft', legend=c('empirical', 'theoretical'), col = c('green','blue'), lty = c(1,NA), pch=c(N
```



```
# we propsed 10000 values and ended up accepting 994 values.
# We can notice that empirical and theoretic CDF are identical.
```

Task 4B:

Use the metropolis algorithm to generate values from the logisitc distribution.

For your proposal distribution, use a random uniform distribution that ranges from your current value - 1 to your current value + 1.

As a reminder, the steps of the algorithm are as follows:

- Propose a single value from the proposal distribution.
- Calculate the probability of moving = min(1, P(proposed)/P(current))
- Draw a random value to decide if you will move or not. If you move, update the current position. If you do not move, keep the current position for another iteration.
- Repeat.

Start at the terrible location x = -19.

Run the Markov Chain for 10,000 iterations. Plot the first 1000 values of the chain and eyeball where you think the chain starts has finished 'burning-in' and is now drawing values from the target distribution. Throw away those initial values.

Plot a histogram of the remaining values.

Plot the empirical CDF of the remaining values, and plot the theoretic CDF (using plogis).

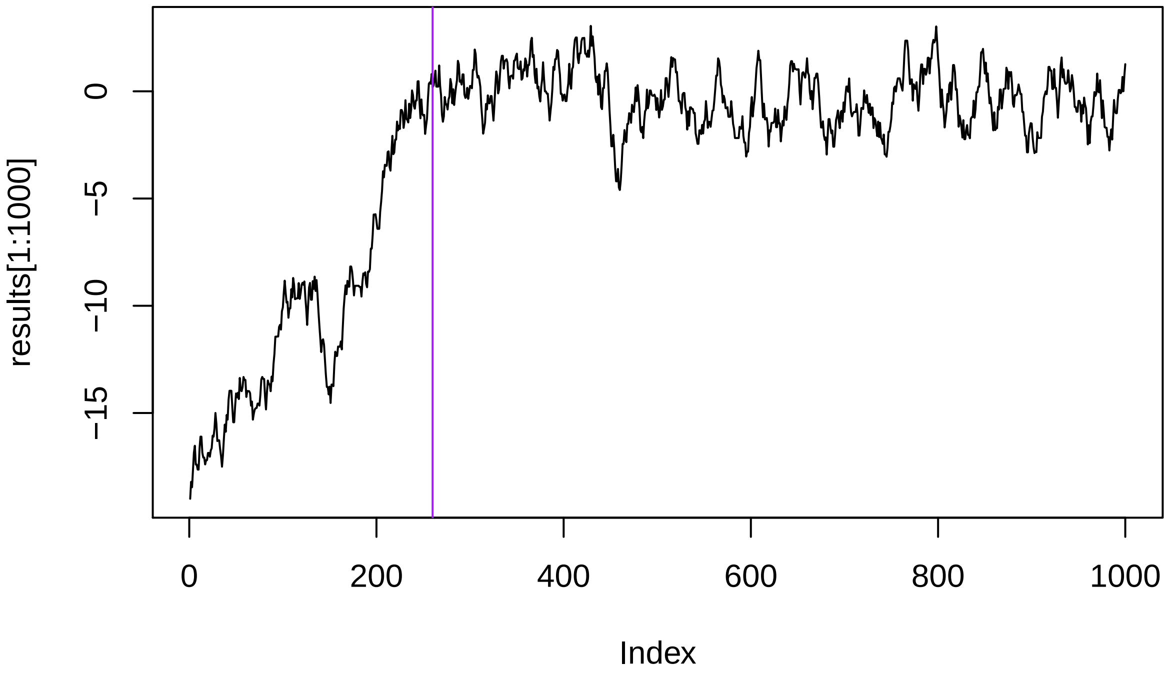
```
set.seed(0)
c_loc <- -19
results<-rep(NA,10000)
for(i in 1:10000){
        proposed <- runif(1,c_loc - 1,c_loc + 1)
        ratio <- dlogis(proposed)/dlogis(c_loc)
        p_move <- min(1,ratio)
        u <- runif(1)
    if(u < p_move){
            c_loc <- proposed
        }
        results[i]<- c_loc
}
results <- append(results,-19,0)
head(results)
```

```
## [1] -19.00000 -18.20661 -18.46236 -17.64594 -16.84916 -16.52757
```

```
length(results)
```

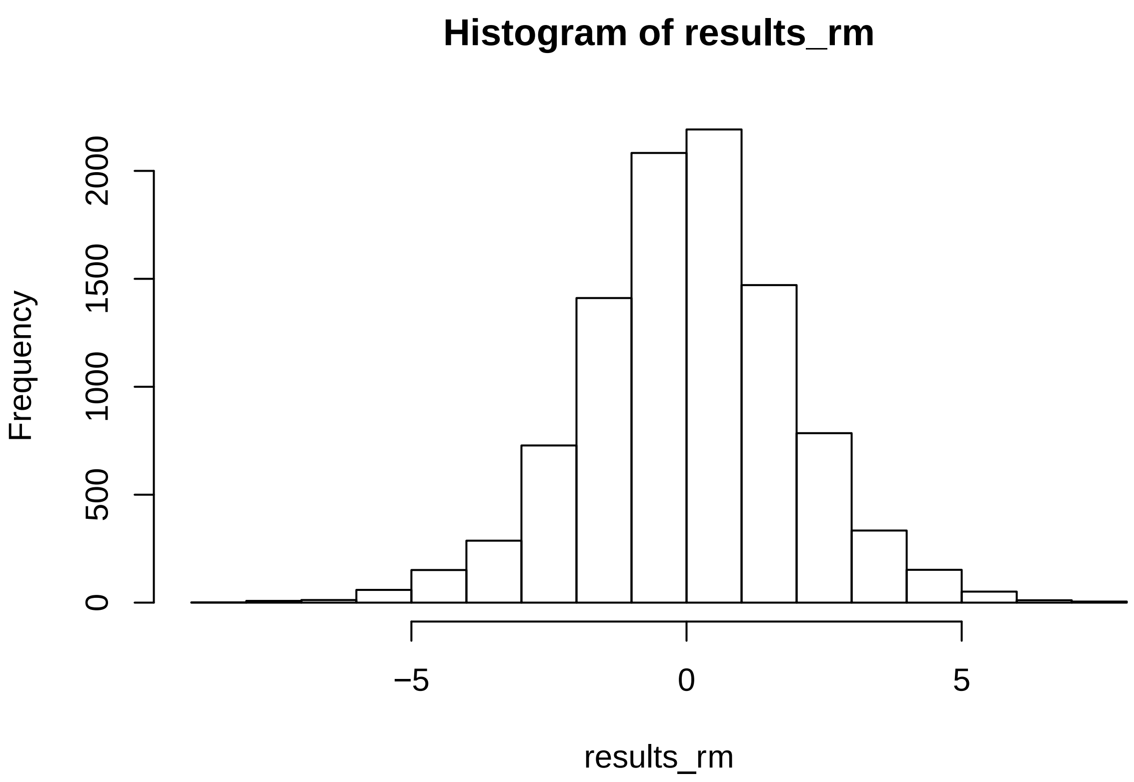
```
## [1] 10001
```

```
plot(results[1:1000], type = 'l')
abline(v = 260,col = 'purple')
```

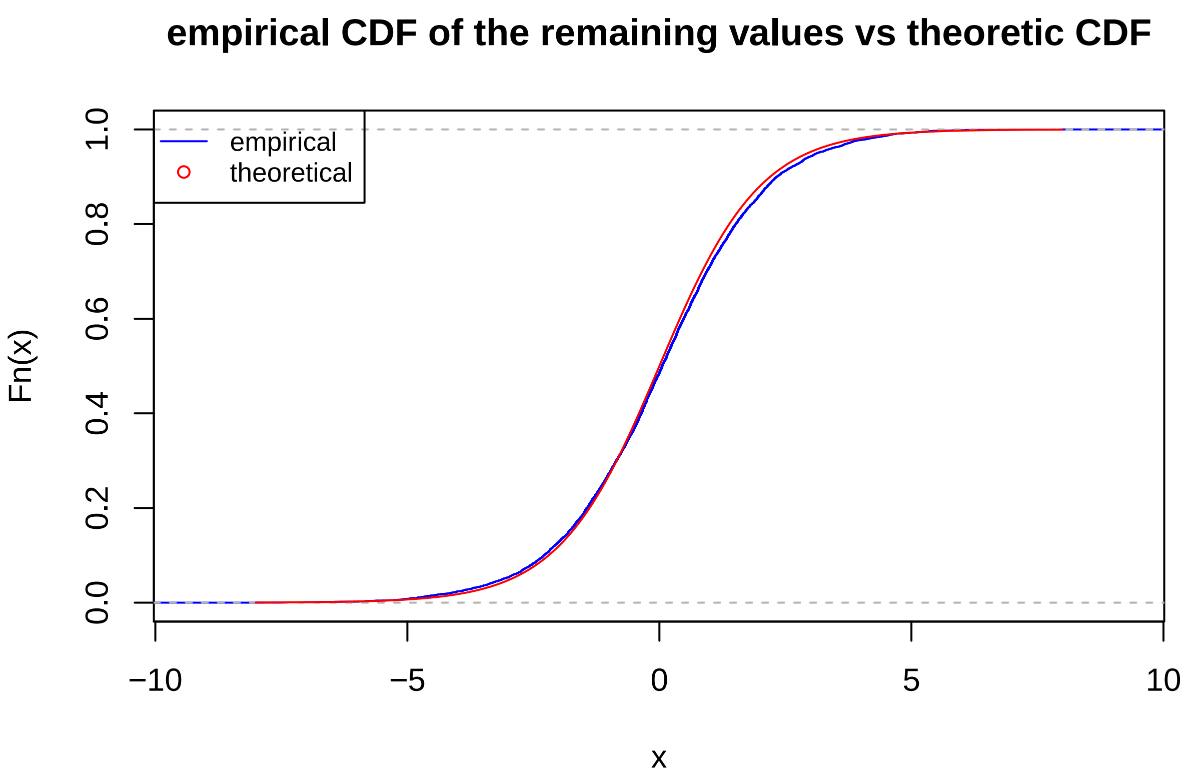


```
# After 260 iterations, chain starts finishing 'burning-in'
```

```
# Plot a histogram of the remaining values.
results_rm <- results[-c(1:260)]
hist(results_rm)
```



```
# Plot the empirical CDF of the remaining values, and plot the theoretic CDF (using plogis).
plot(ecdf(results_rm), col ='blue', main = 'empirical CDF of the remaining values vs theoretic CDF')

s <- seq(min(results_rm),max(results_rm), by = 0.01)
lines(s , plogis(s ,0,1), col= 'red')

legend('topleft', legend=c('empirical', 'theoretical'), col = c('blue','red'), lty = c(1,NA), pch=c(NA,
```



Problem 5 - MCMC - the effect of sigma in the proposal distribution

Write code to perform 50,000 iterations of the metropolis algorithm for a single continuous random variable.

Let the pdf of the target distribution be:

$$f(x) = c \cdot (sin(x) + 2)$$

for $0 \leq x \leq 3 * \pi$, where c is some constant so that $\int_0^{3\pi} f(x)dx = 1$

For your proposal distribution, use a normal distribution, centered at the current value, with a standard deviation of σ, which we will adjust in this problem.

Begin your Markov Chain at the location x = 2.

Keep in mind that the probability of a value greater than 3 * pi or less than 0 is 0.

Gather 50,000 samples using MCMC three different times.

The first time, use a sigma of 0.5 for the proposal distribution.

The second time, use a sigma of 3 for the proposal distribution.

The third time, use a sigma = 20.

Keep track of whether your proposed values are accepted or rejected, and print out the acceptance ratio.

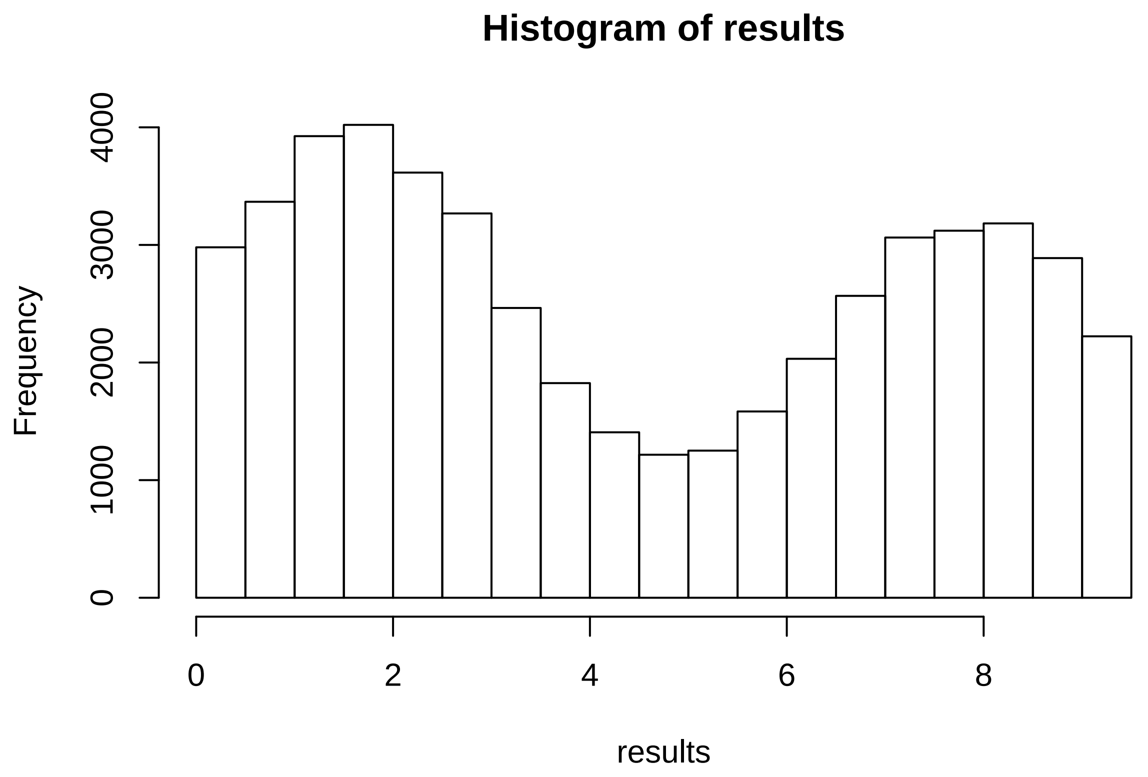
For each MCMC run, print out the acceptance ratio, create a histogram of the sampled values, and plot the first 500 values of the chain `plot(x[1:500], type = "l")`.

```r
# For the first time, use a sigma of 0.5 for the proposal distribution.
s <- function(x) {
        if(x < 0 || x > 3 * pi){
        return(0)
```

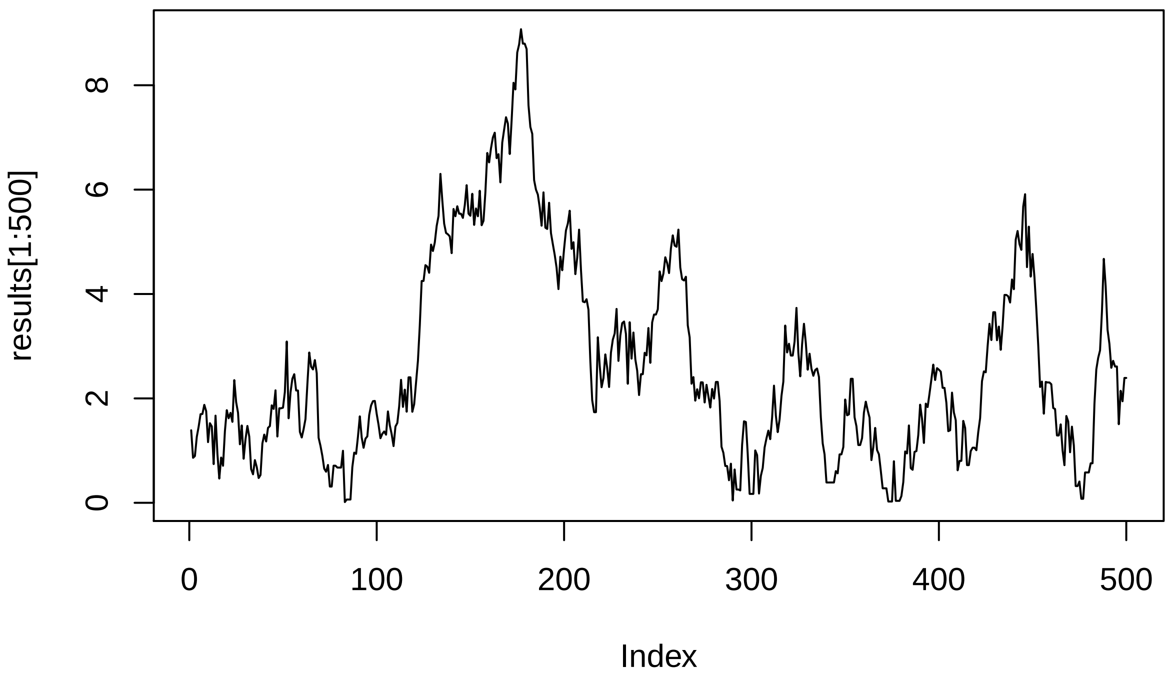
```r
       }else{
        sin(x) + 2
        }
}

c_loc <- 2
results<-rep(NA,50000)
accepted1 <- 1
for(i in 1:50000){
        proposed <- rnorm(1,c_loc,0.5)
        ratio <- s(proposed)/s(c_loc)
        p_move <- min(1,ratio)
        u <- runif(1)
    if(u < p_move){
            c_loc <- proposed
            accepted1 <- accepted1 + 1
        }
        results[i]<- c_loc
}
# Acceptence ratio.
(accepted1 / 50000)
```

```
## [1] 0.90362
```

```r
hist(results)
```



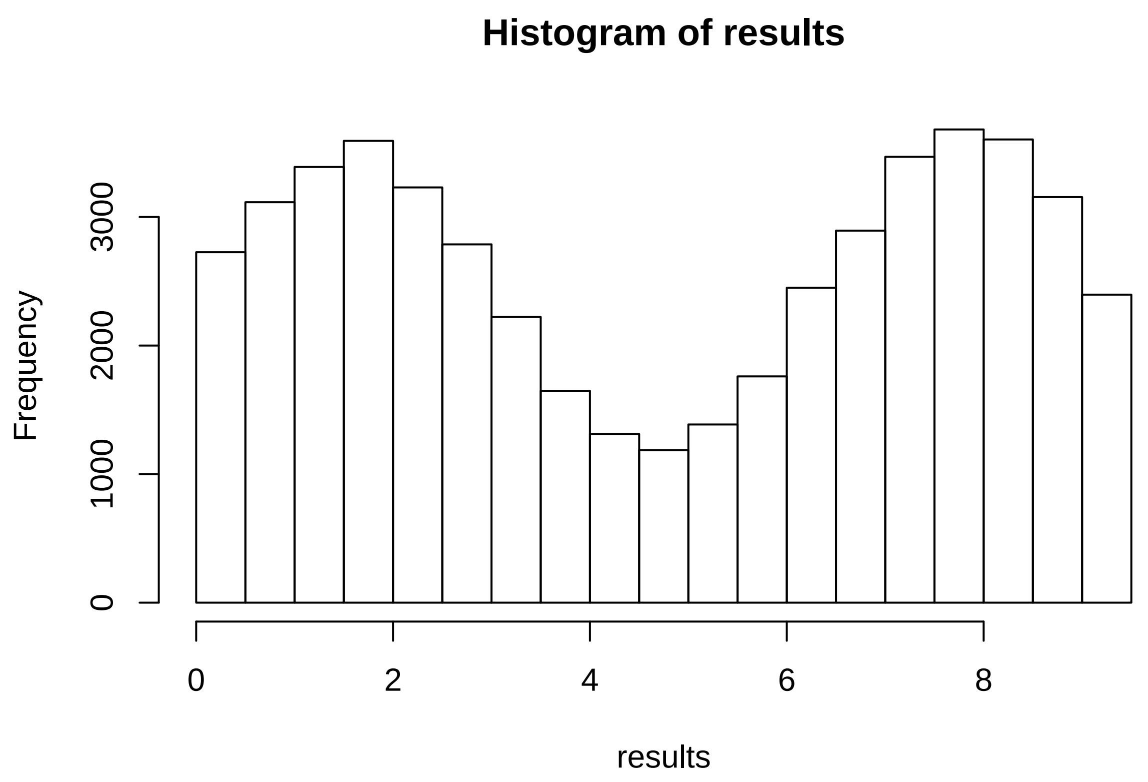
```r
plot(results[1:500], type = "l")
```



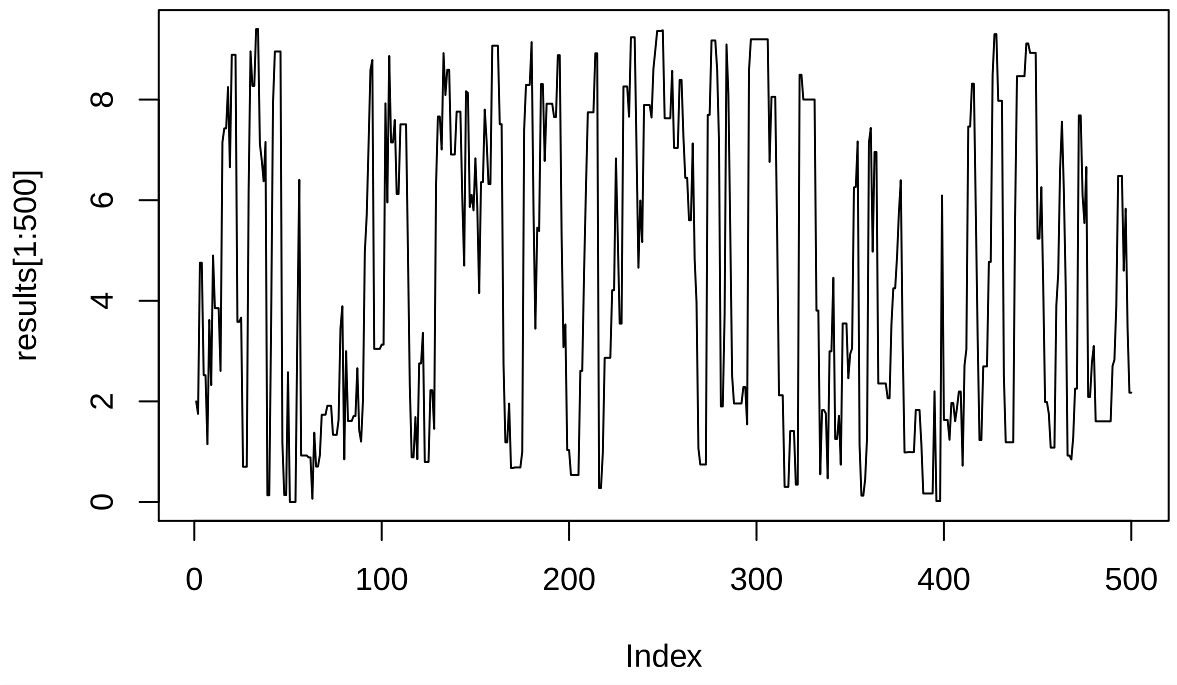
```r
# For the second time, use a sigma of 3 for the proposal distribution.
c_loc <- 2
results <- rep(NA,50000)
accepted2 <- 1
for(i in 1:50000){
        proposed <- rnorm(1,c_loc,3)
        ratio <- s(proposed)/s(c_loc)
        p_move <- min(1,ratio)
        u <- runif(1)
    if(u < p_move){
            c_loc <- proposed
            accepted2 <- accepted2 + 1
        }
        results[i]<- c_loc
}
# Acceptence ratio.
(accepted2 / 50000)
```

```
## [1] 0.59802
```

```r
hist(results)
```



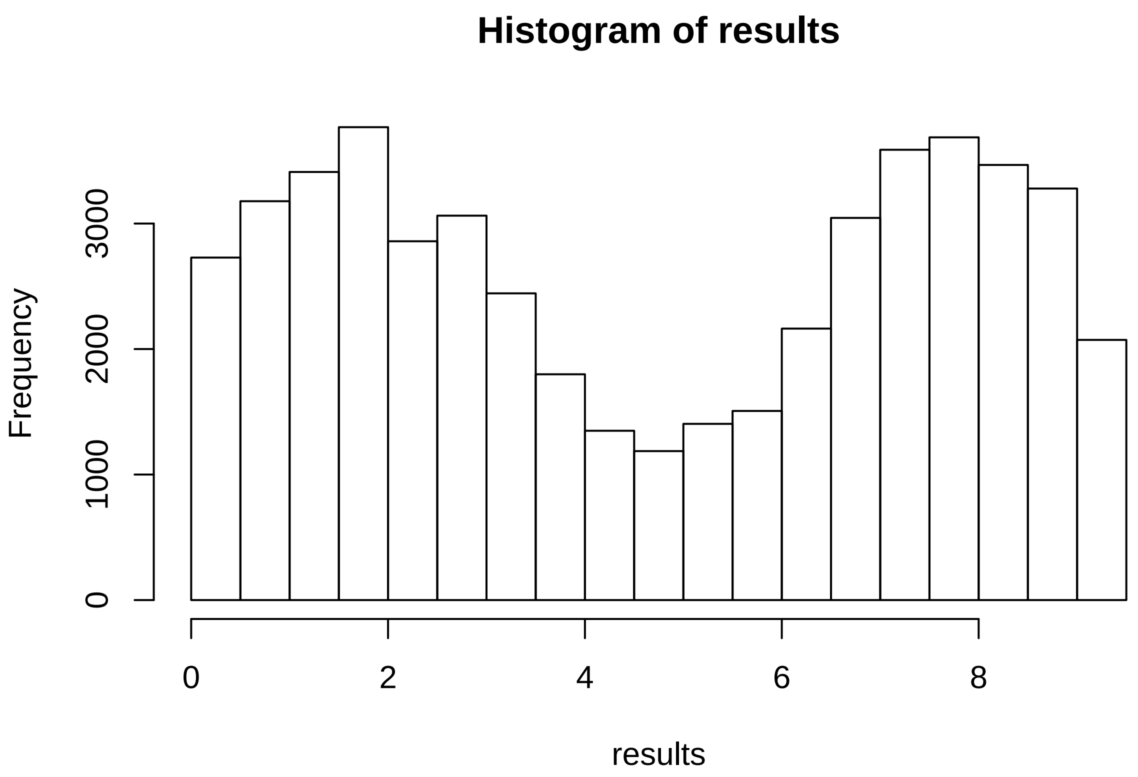
```
plot(results[1:500], type = "l")
```



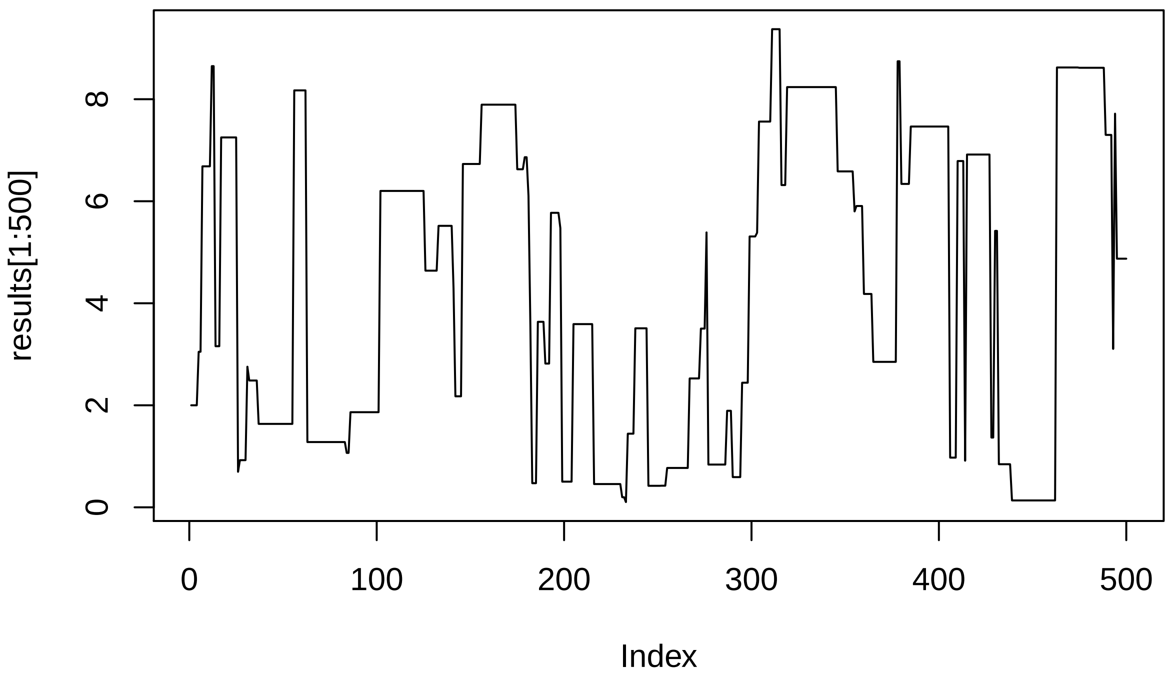
```
# For the third time, use a sigma of 20 for the proposal distribution.
c_loc <- 2
results <- rep(NA,50000)
accepted3 <- 1
for(i in 1:50000){
        proposed <- rnorm(1,c_loc,20)
        ratio <- s(proposed)/s(c_loc)
        p_move <- min(1,ratio)
        u <- runif(1)
    if(u < p_move){
            c_loc <- proposed
            accepted3 <- accepted3 + 1
        }
        results[i]<- c_loc
}
# Acceptence ratio.
(accepted3 / 50000)
```

```
## [1] 0.15334
```

```
hist(results)
```



```
plot(results[1:500], type = "l")
```


results[1:500]
0
2
4
6
8
0
100
200
300
400
500
Index