Homework1

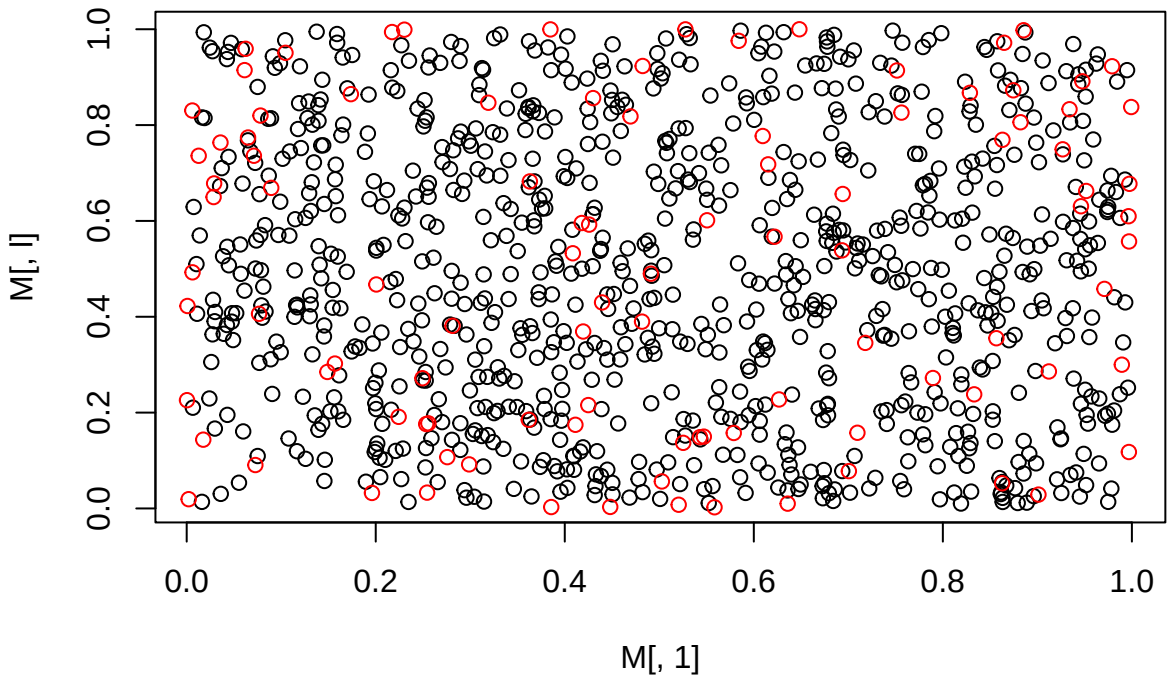
Aidan Lytle

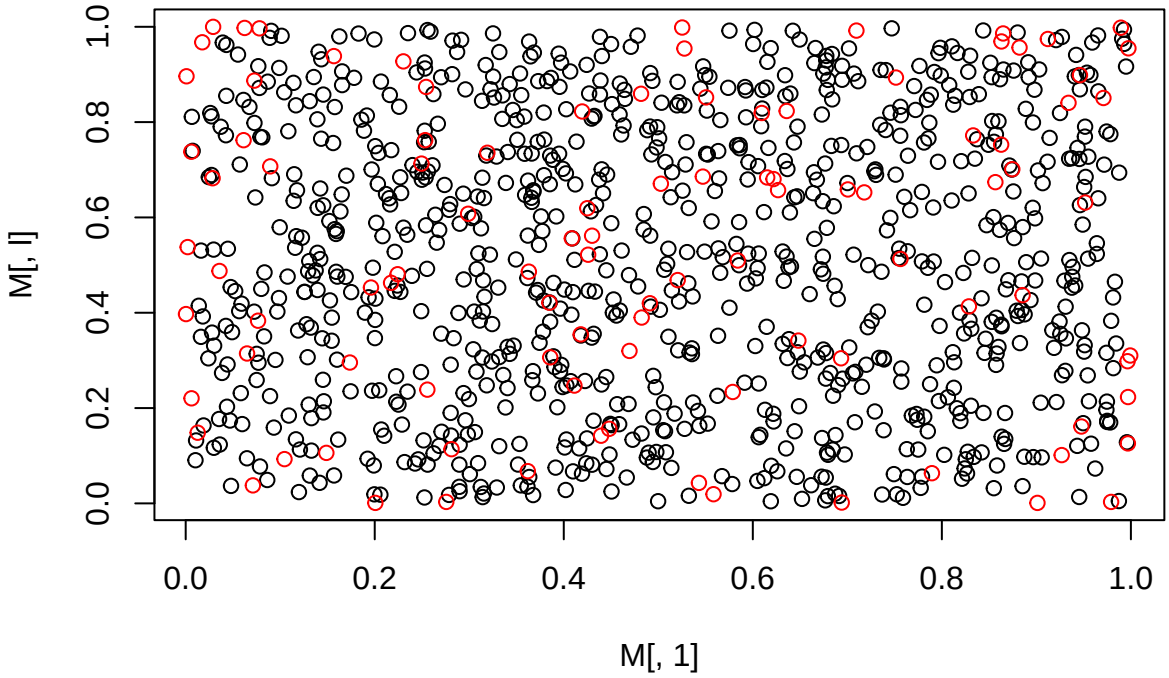
2022-09-04

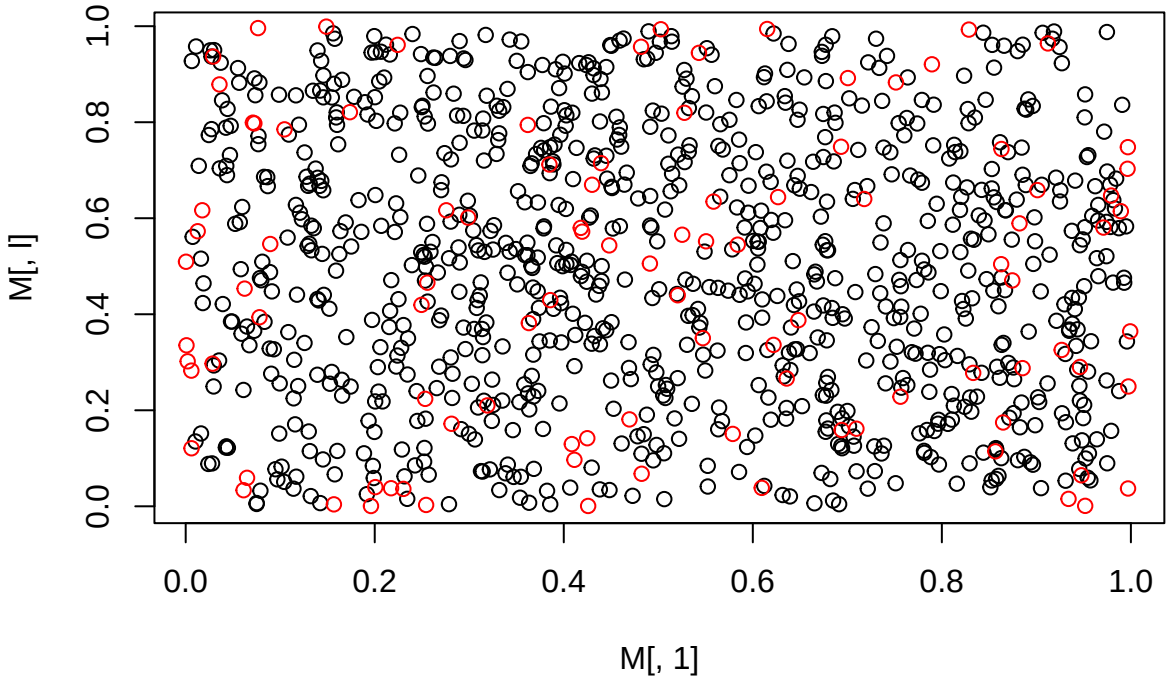
Problem 1

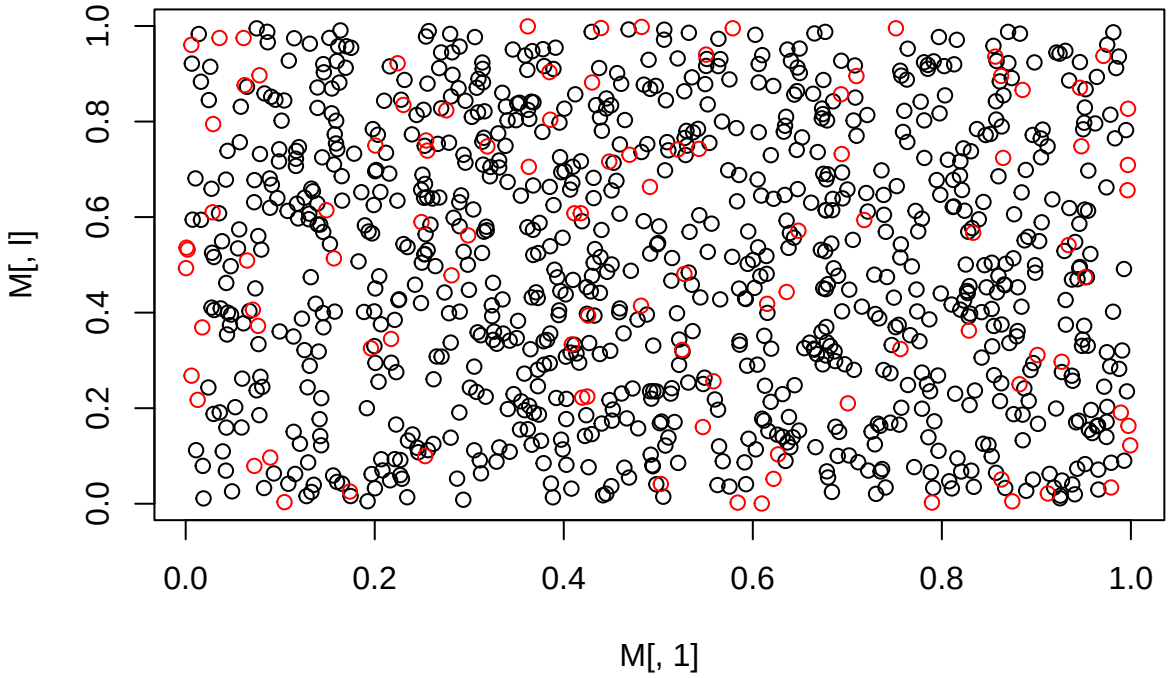
```
# make the variables for the problem

m = 10
n = 1000
k = 100


if (k >= n)
{
  print('error: k must be less than n.')
}

M = matrix(runif(n*m), nrow = n)
newM = matrix(0*1:m, nrow = 1)


for (i in 1:m) # iterate over rows
{

  MCheck = M[,i] # vector we're checking over: ith col of M

  for (j in 1:ceiling(k/(2*m)))
  {
    maxRowInd = which.max(MCheck)[1] #find max row
    newMRow = M[maxRowInd,] #assign row
    newM = rbind(newM, newMRow) #bind row to matrix
    M = M[-c(maxRowInd),] #delete row from data
    MCheck = MCheck[-maxRowInd] #delete index from search vector

  }

  MCheck = M[,i] #have to reassign outside loop in case anything broke

  for (j in 1:ceiling(k/(2*m)))
  {
    minRowInd = which.min(MCheck)[1]
    newMRow = M[minRowInd,]
    newM = rbind(newM, newMRow)
```

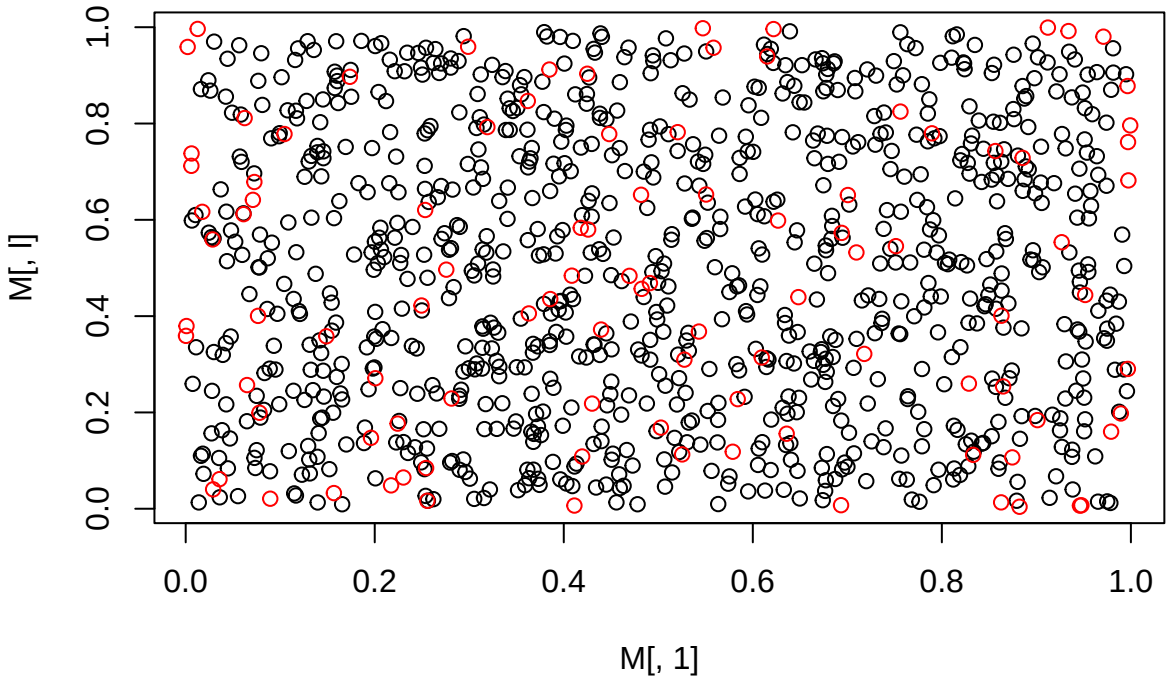
```
    M = M[-c(minRowInd),]
    MCheck = MCheck[-minRowInd]
  }

}
newM = newM[-1,] #delete initial row of zeroes




for (l in 2:m) #plots for comparison
{
  plot(M[,1],M[,l] )
  points(newM[,1],newM[,l],col = 'red')

}
```

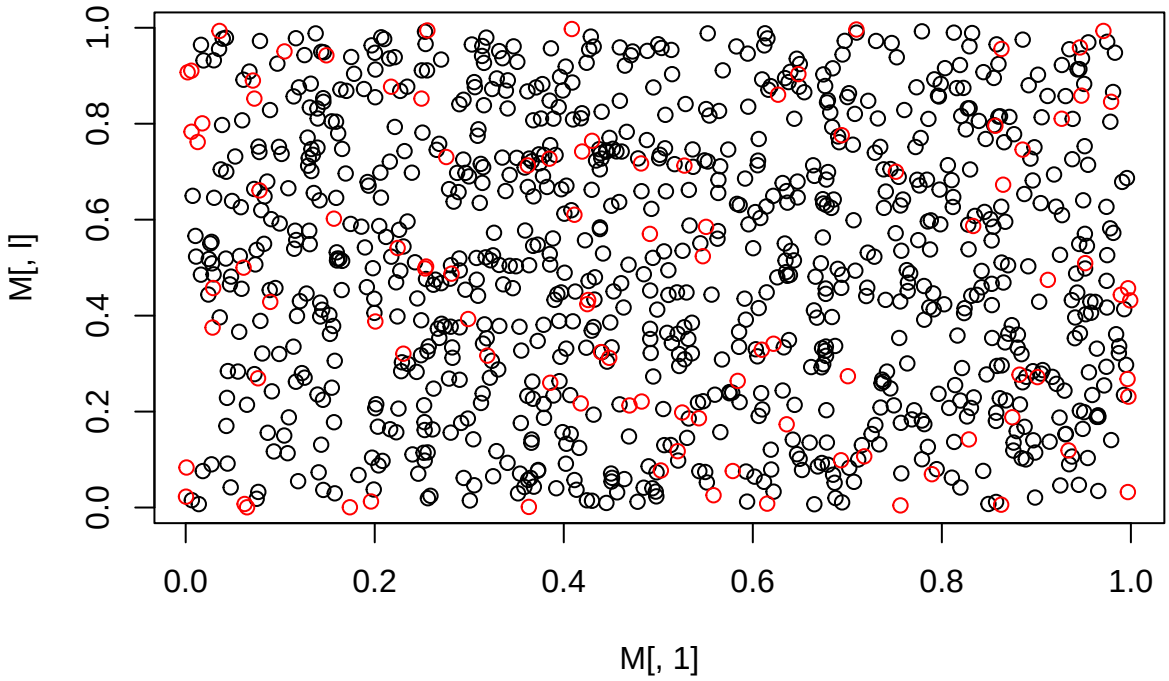


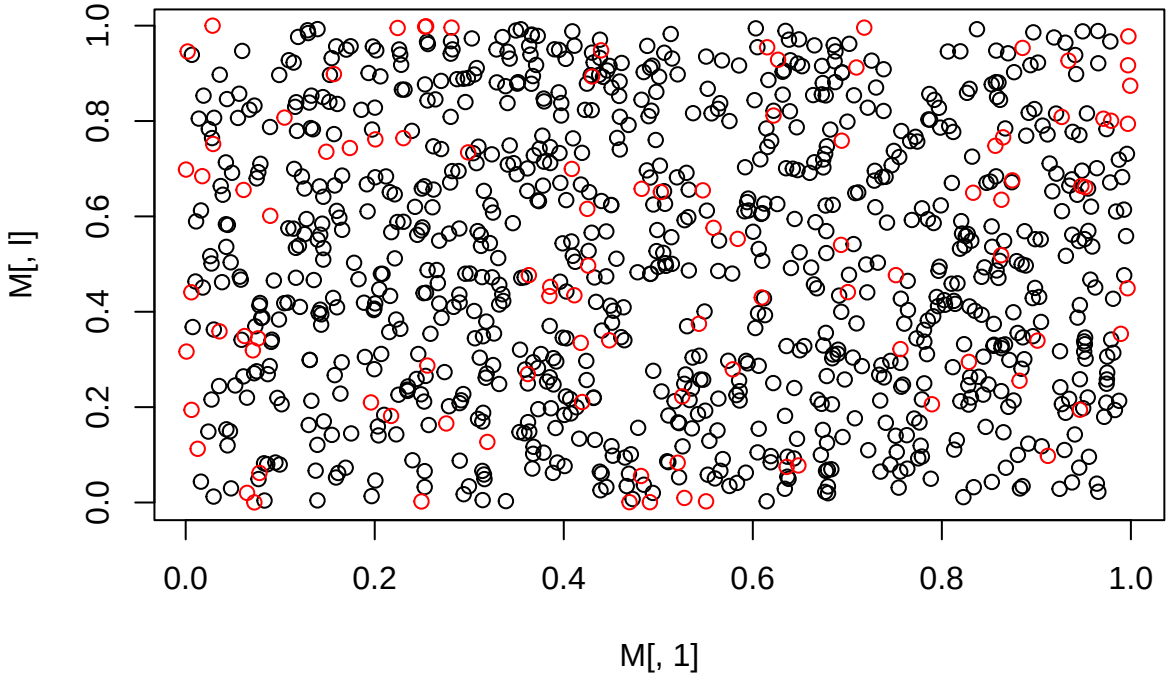

M[, 1]
M[, l]
0.0
0.2
0.4
0.6
0.8
1.0

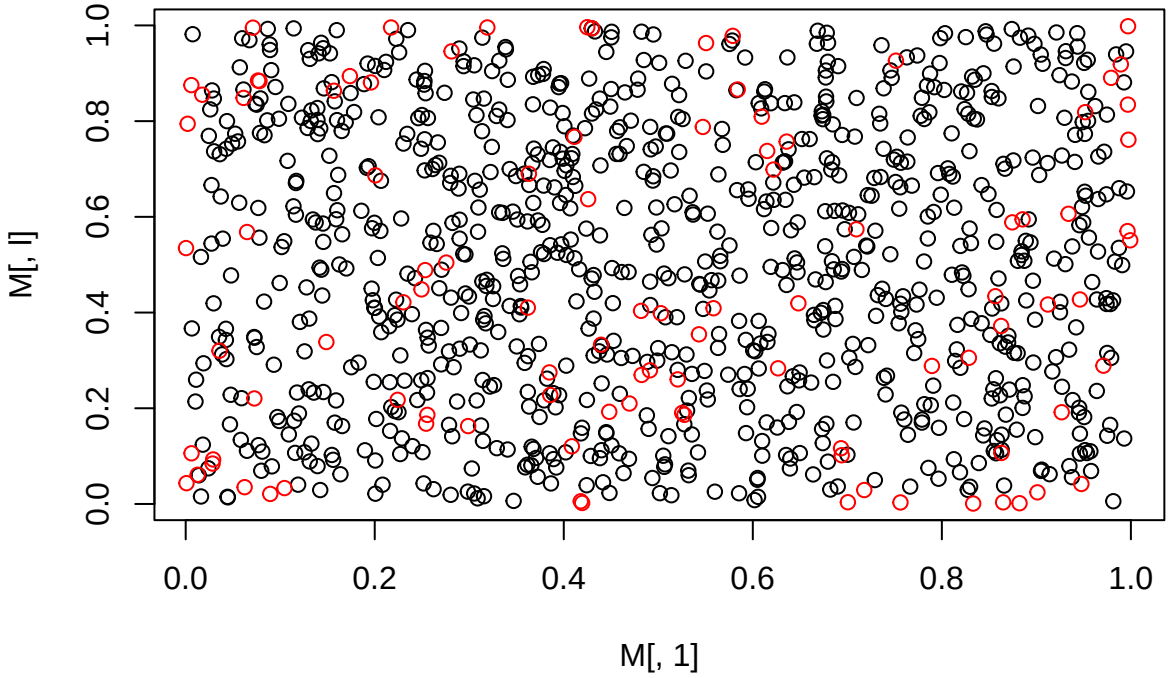

M[, l]
0.0
0.2
0.4
0.6
0.8
1.0
M[, 1]

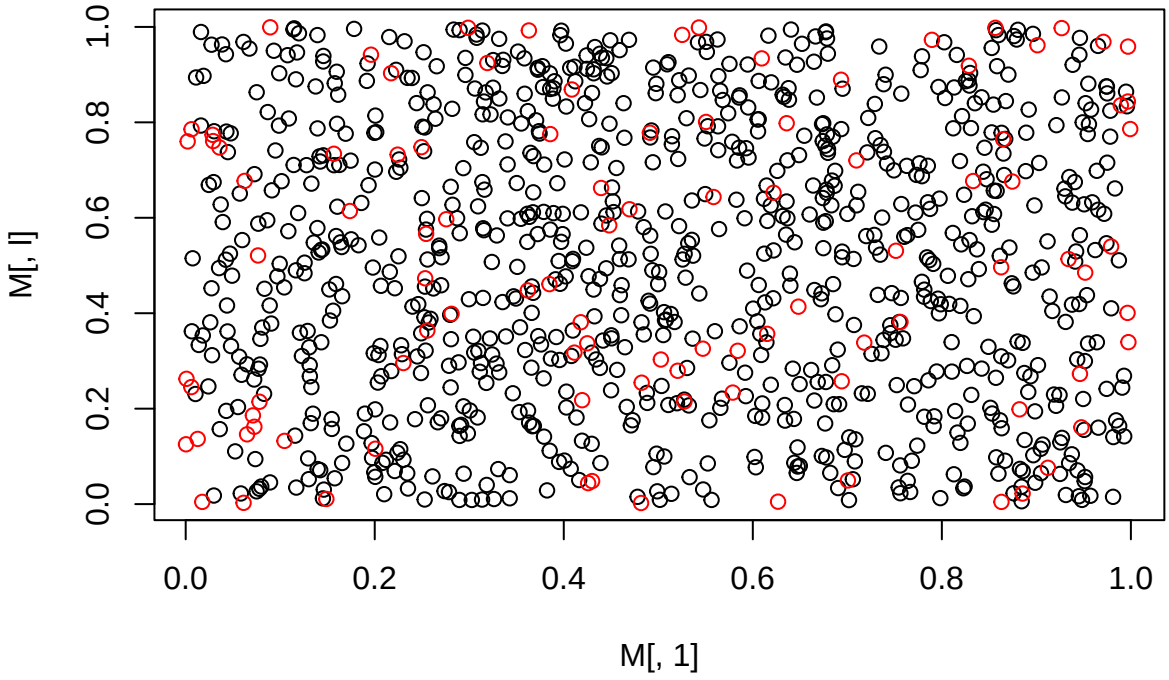

M[, l]
0.0
0.2
0.4
0.6
0.8
1.0
M[, 1]


M[, l]
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
M[, 1]


M[, l]
0.0
0.2
0.4
0.6
0.8
1.0
M[, 1]


M[, l]
0.0
0.2
0.4
0.6
0.8
1.0
M[, 1]


M[, 1]
M[, l]
0.0
0.2
0.4
0.6
0.8
1.0

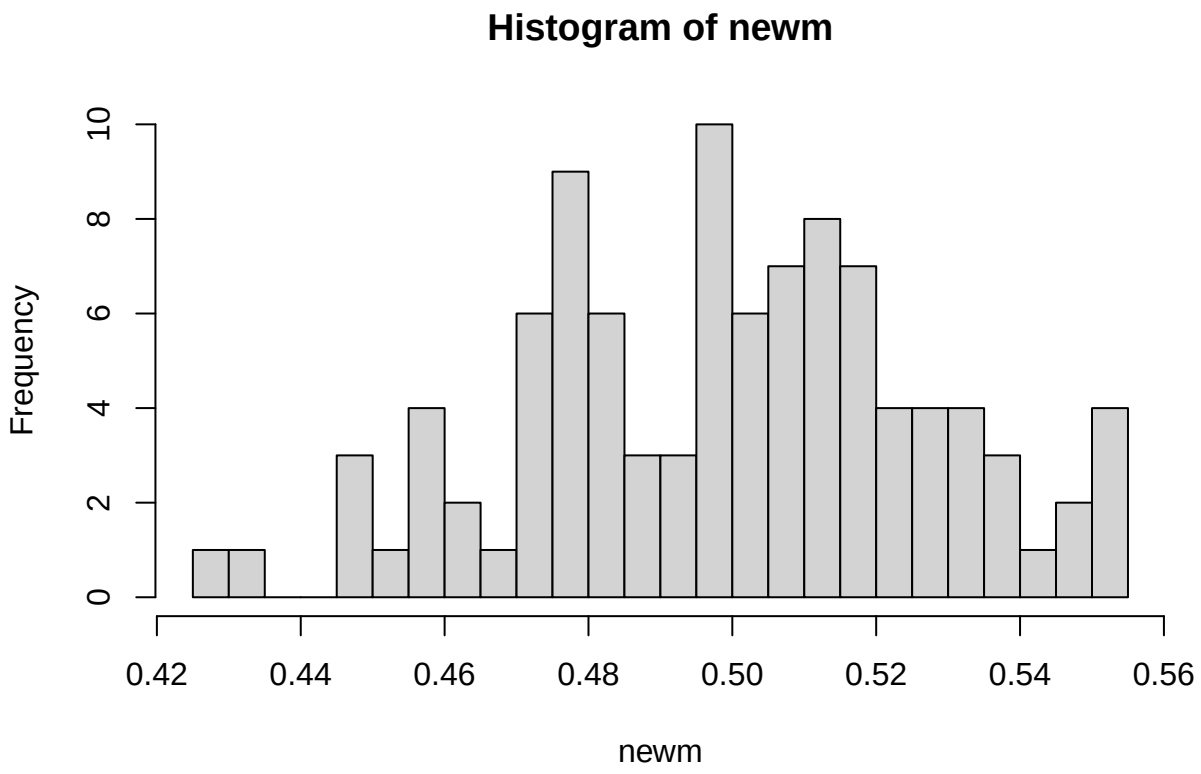


It should be clear from these plots, and from the structure of the problem, that this algorithm is a way to construct a representative sample from a larger dataset. This is because the largest differences along any axis are what is represented, which (when sampled over all) produces a set which maintains the variation in the initial set, while lowering the dimensionality.

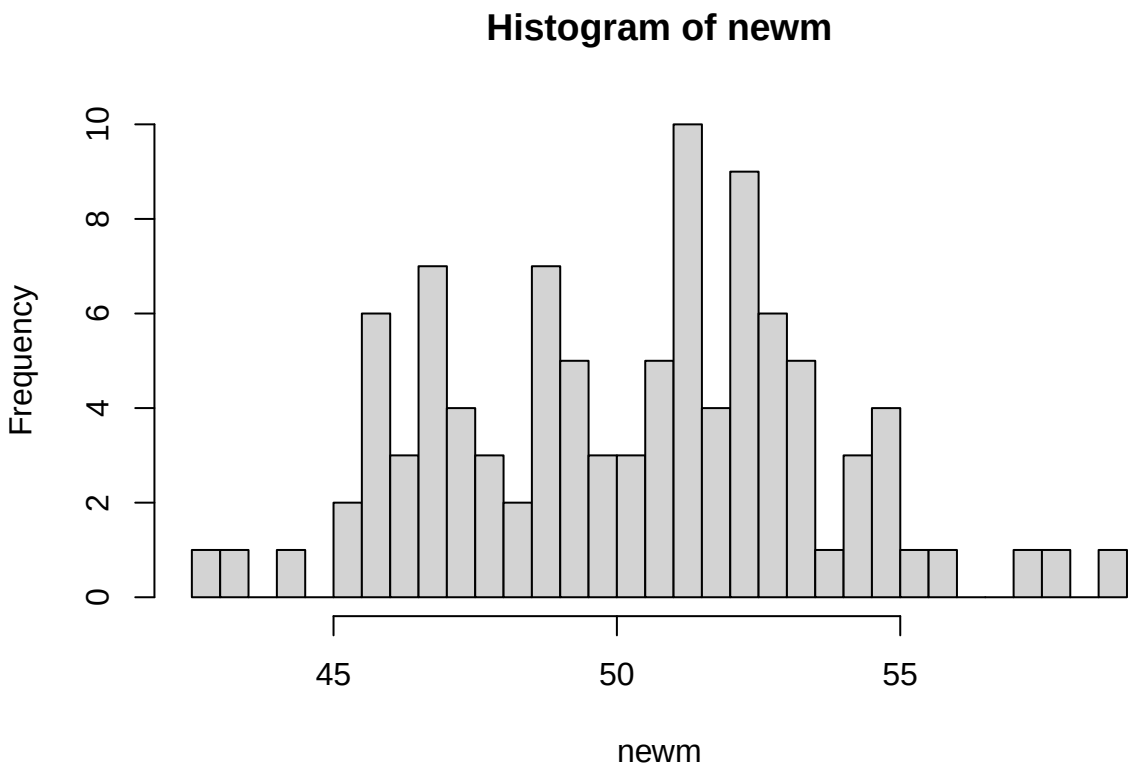
Problem 3

```
#library(ggplot2)

n = 100


M1 = matrix(runif(n*n), ncol = n) #n^2 random entries 0,1
M2 = matrix(runif(n*n, 0, 100), ncol = n) #range from 0,100
M3 = matrix(runif(n*n, 10, 2000), ncol = n) #10,2000
M4 = matrix(runif(n*n, -500, 0), ncol = n) #-500,0
MN1 = matrix(runif(n*n*n*n), ncol = n) #n^4


cltTest <- function(m)
{
  newm = 0*(1:ncol(m))
```

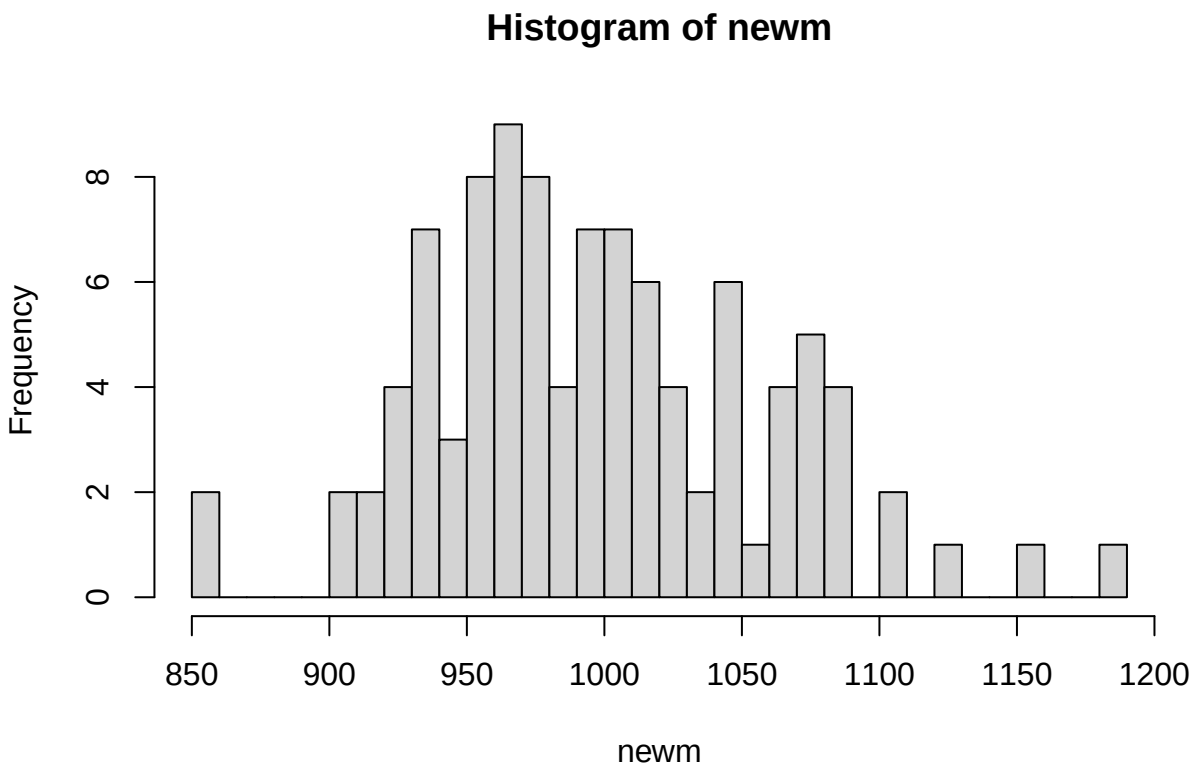
```
  for (i in 1:ncol(m))
  {
    newm[i] = mean(m[,i])
  }
  hist(newm, breaks = 25)
  #return(newm)
}

# calling function to produce the results of the CLT test
cltTest(M1)
```



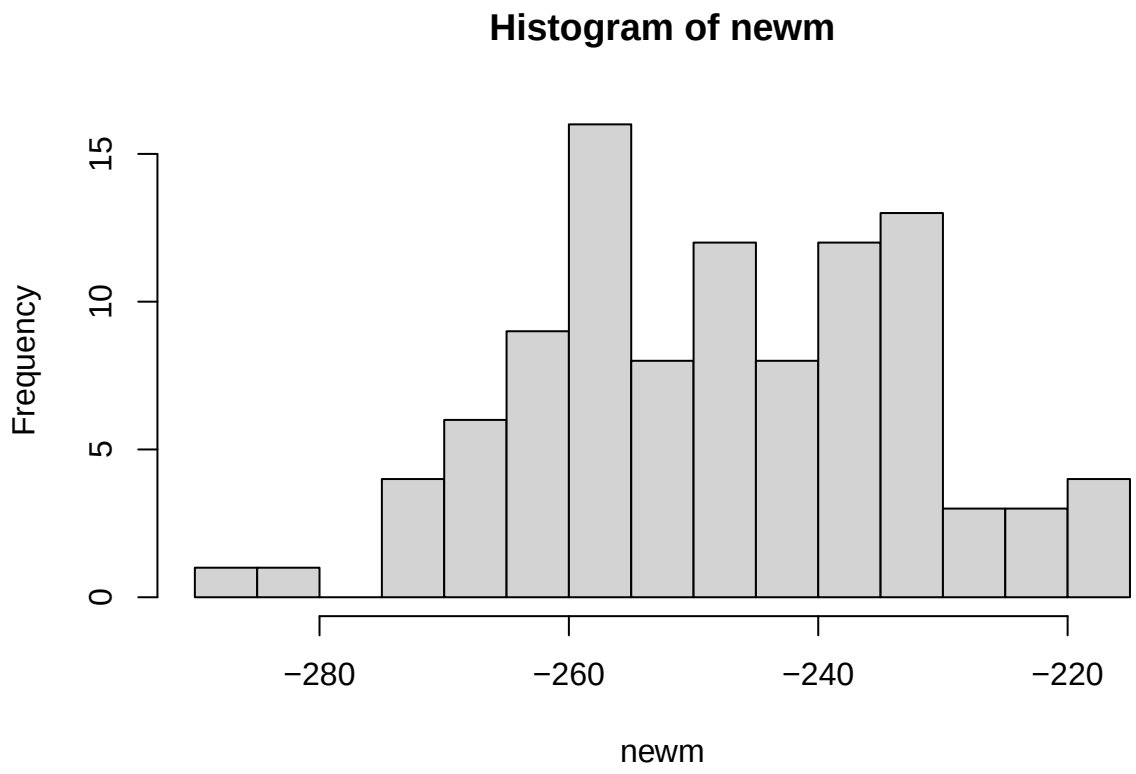

```
cltTest(M2)
```



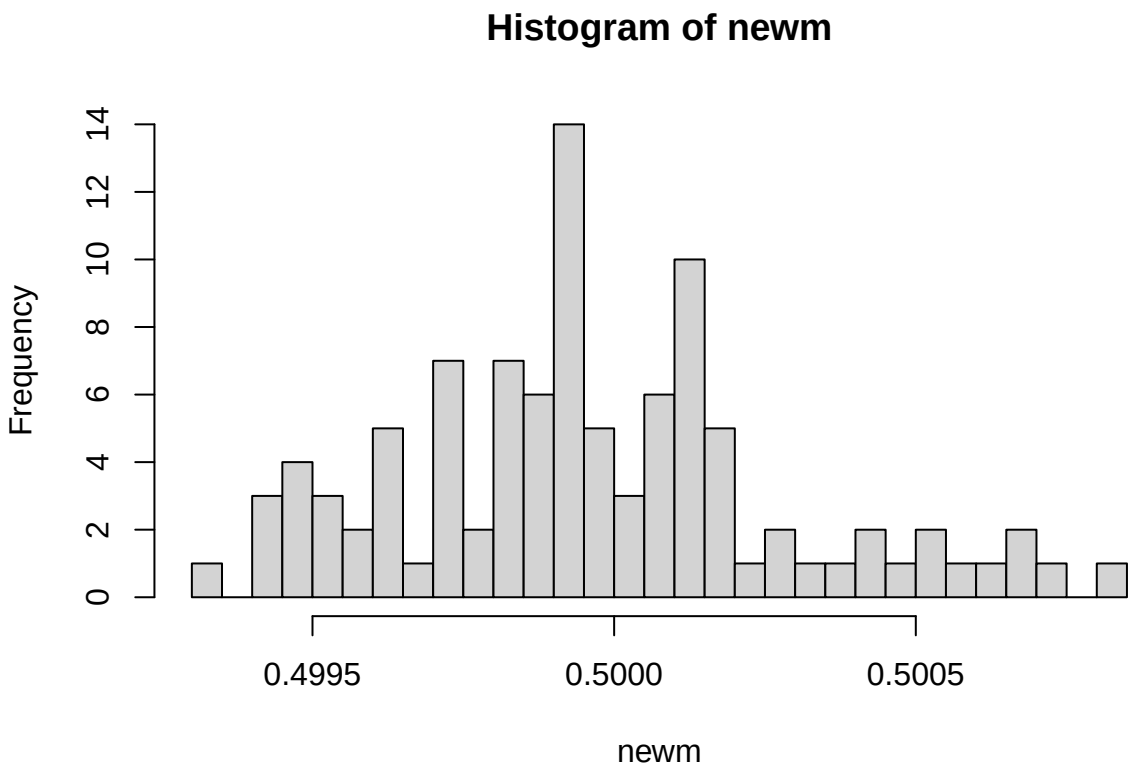

```
cltTest(M3)
```


Histogram of newm
Frequency
0
2
4
6
8
850
900
950
1000
1050
1100
1150
1200
newm

```
cltTest(M4)
```




```
cltTest(MN1)
```



***

Problem 4

Part 1

```
#huber loss function part 1: "original" function

huber <- function(x)
{

  if (abs(x) < 1)
  {

    return( x^2 )

  } else {

    return( 2*abs(x) - 1 )

  }

}
```

```
#tests
print(huber(1))
```

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

```
print(huber(4))
```

```
## [1] 7
```

This code should be self explanatory.

Part 2

```
huber <- function(x,a)
{

  if (abs(x) <= a)
  {

    return( x^2 )

  } else {

    return( 2*a*abs(x) - a^2 )

  }

}

print(huber(3,2))
```

```
## [1] 8
```

```
print(huber(3,4))
```

```
## [1] 9
```

Also self explanatory.

Part 3

```
huber <- function(x,a=1)
{

  if (abs(x) <= a)
  {
```

```
    return( x^2 )

  } else {

    return( 2*a*abs(x) - a^2 )

  }

}

print(huber(3))
```

```
## [1] 5
```

Part 4

```
print(huber(a = 1,x = 3))
```

```
## [1] 5
```

```
print(huber(1,3))
```

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

These return two different values because the variables are named a and x, so if we define a and x in the function call, we merely change the print order.

Part 5

```
huber <- function(x,a=1)
{
  n = length(x)
  out = ifelse(abs(x) <= rep(a,n), x*x, 2*a*abs(x) - a^2 )
  return(out)

}

print(huber(x = 1:6, a = 3))
```

```
## [1]  1  4  9 15 21 27
```

Vectorized results as expected.

Problem 5

```
#define variables
A = matrix(0*1:6, nrow = 2)
b = rep(1,2)
c = rep(2,3)

Ab = matrix(c(A[1,]*b[1], A[2,]*b[2]), nrow = 2)
Ac = matrix(c(A[,1]*c[1], A[,2]*c[2],A[,3]*c[3]), nrow = 3)
```

```
#a times c
AstarC = A*c
```

What follows is the function for standard user inputs.

```
#this kind of product is often called the hadamard product
RowHadamard <- function(n,m,A,b)
{
  M = matrix(to_vec(for(i in 1:n) for(j in 1:m) A[i,j]*b[i]), nrow = n)
  return(M)
}
```

```
DotHadamard <- function(n,m,A,c)
{
  M = matrix(to_vec(for(i in 1:n) for(j in 1:m) A[i,j]*c[j]), nrow = n)
  return(M)
}
```

R treats matrices as a (row) vector of vectors, where each element of the vector is a column.