QF 435 - Project 1

Siddharth Iyer, Zheng Li, Leonid Maksymenko

3/29/2021

Data

Our universe of stocks consists of the following 27 US equities:

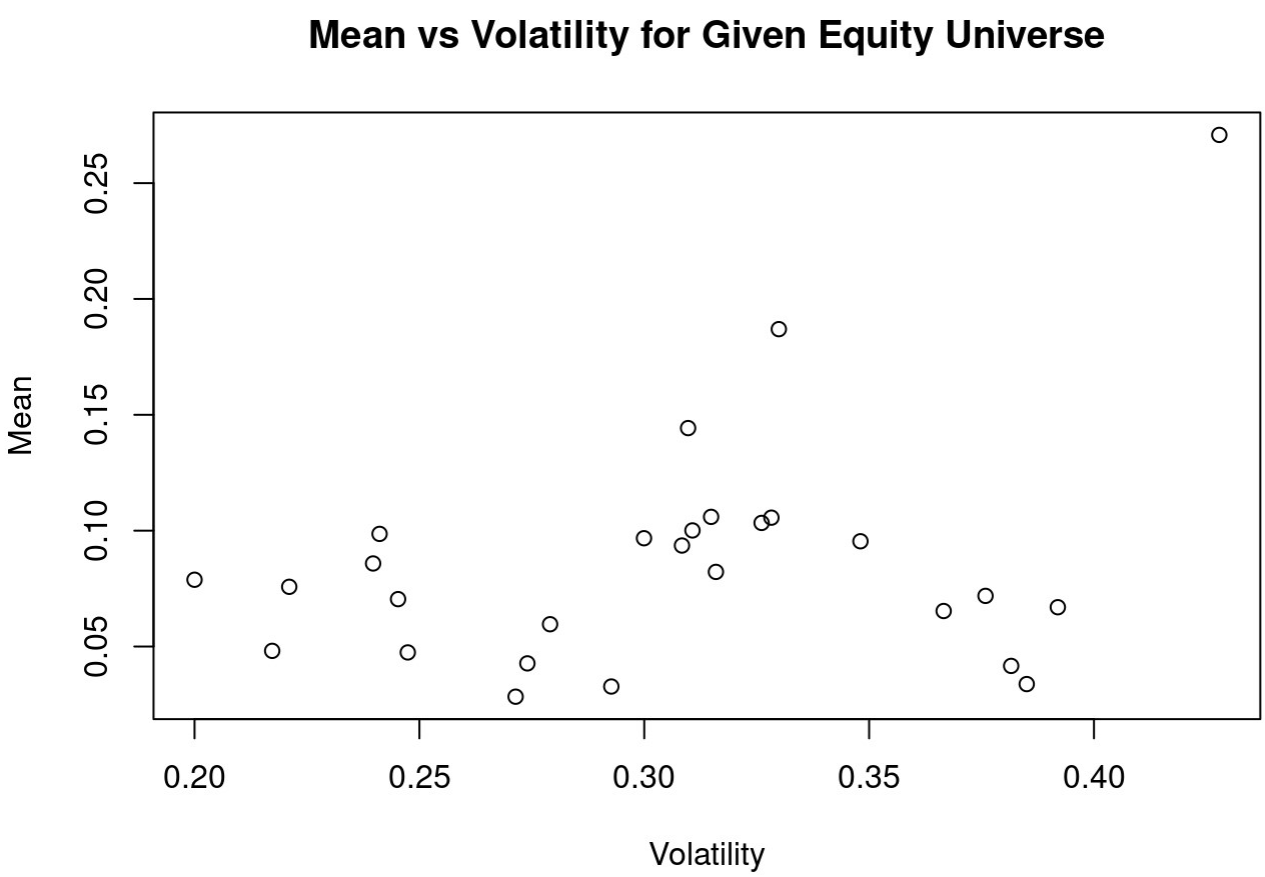
```
##  [1] "SPY"  "AAPL" "CSCO" "HON"  "KO"   "NKE"  "WBA"  "AMGN" "CVX"  "IBM"
## [11] "MCD"  "PG"   "WMT"  "AXP"  "DIS"  "INTC" "MMM"  "TRV"  "BA"   "GS"
## [21] "JNJ"  "MRK"  "UNH"  "CAT"  "HD"   "JPM"  "MSFT" "VZ"
```

Performance Summary

1. Among our 27 equities, the Min, Mean, and Max are listed for a few key metrics:

```
##                      Min       Mean       Max
## Log Returns  0.02834260 0.08640489 0.2708011
## Volatility   0.19999728 0.30555174 0.4278728
## Sharpe Ratio 0.08774196 0.28332973 0.6329009
```

2.



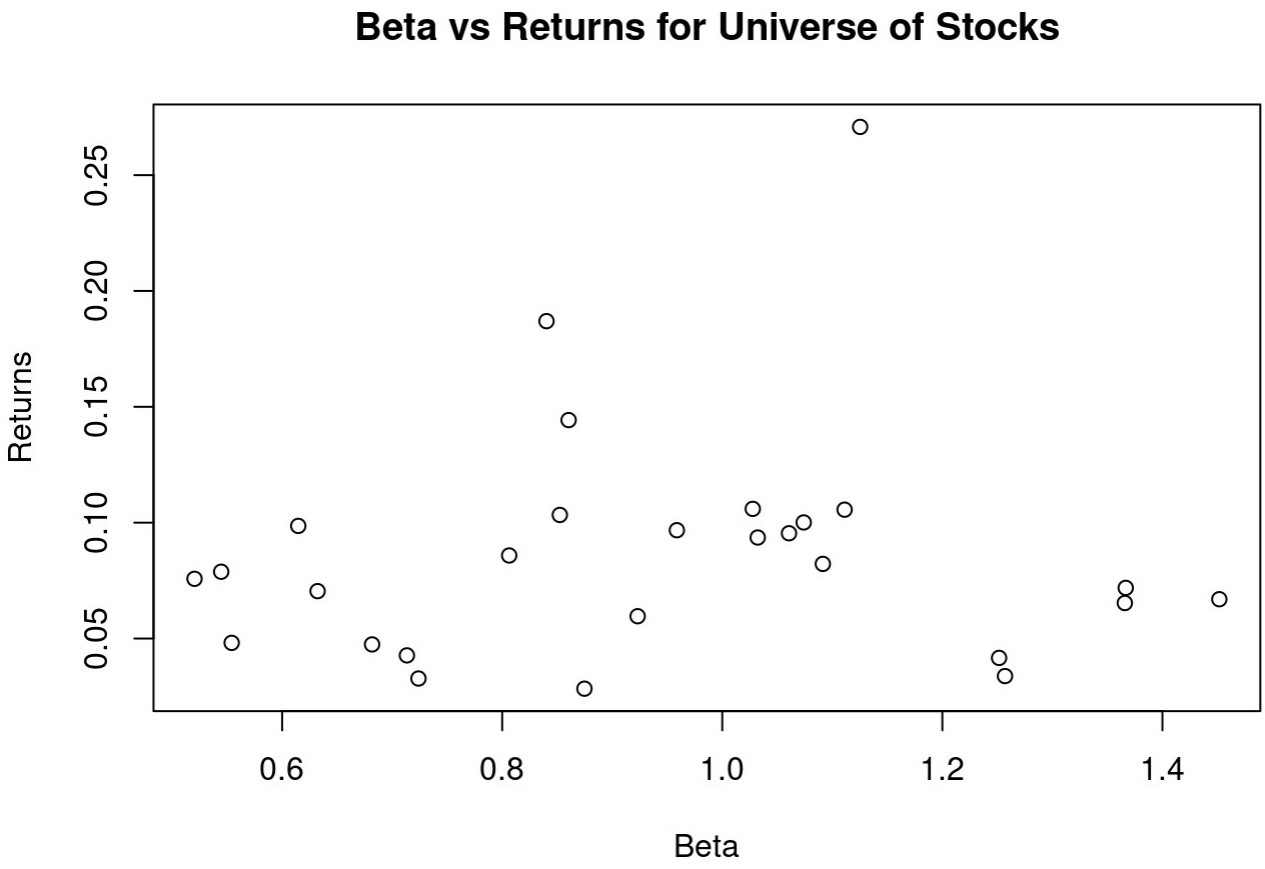
It seems that most of these companies have poor Sharpe Ratios. For several companies, a higher risk doesn't bring higher rewards as the cruve dips. There are 5 stock at the top of the curve that makes sense for each risk tolerance level.

3. Below is a table of metrics and their min, mean, and max among stocks in our Universe.

```
##                           Min       Mean       Max
## Jensen's Alpha    -0.04813652 0.02525941 0.1974594
## Beta               0.52038617 0.93763204 1.4517005
## Treynor Ratio      0.02687896 0.09755617 0.2406511
## Tracking Error     0.18275056 0.24672593 0.3660073
## Information Ratio -0.17772683 0.07908202 0.6664504
```

4.

```
plot(beta, rets_mean, main = "Beta vs Returns for Universe of Stocks", xlab ="Beta",
ylab="Returns")
```



The data shows a weak relationship between beta and returns. At least for this small Universe of stocks, the returns seems to have no correlation with beta except for a few stocks (outliers) that outperform the rest.

Back-Testing

1. Split data into “In Sample” and “Out Sample” for back testing. Ranges listed below

```
## [1] "2017-01-03" "2018-12-28"
```

```
## [1] "2019-01-03" "2020-12-29"
```

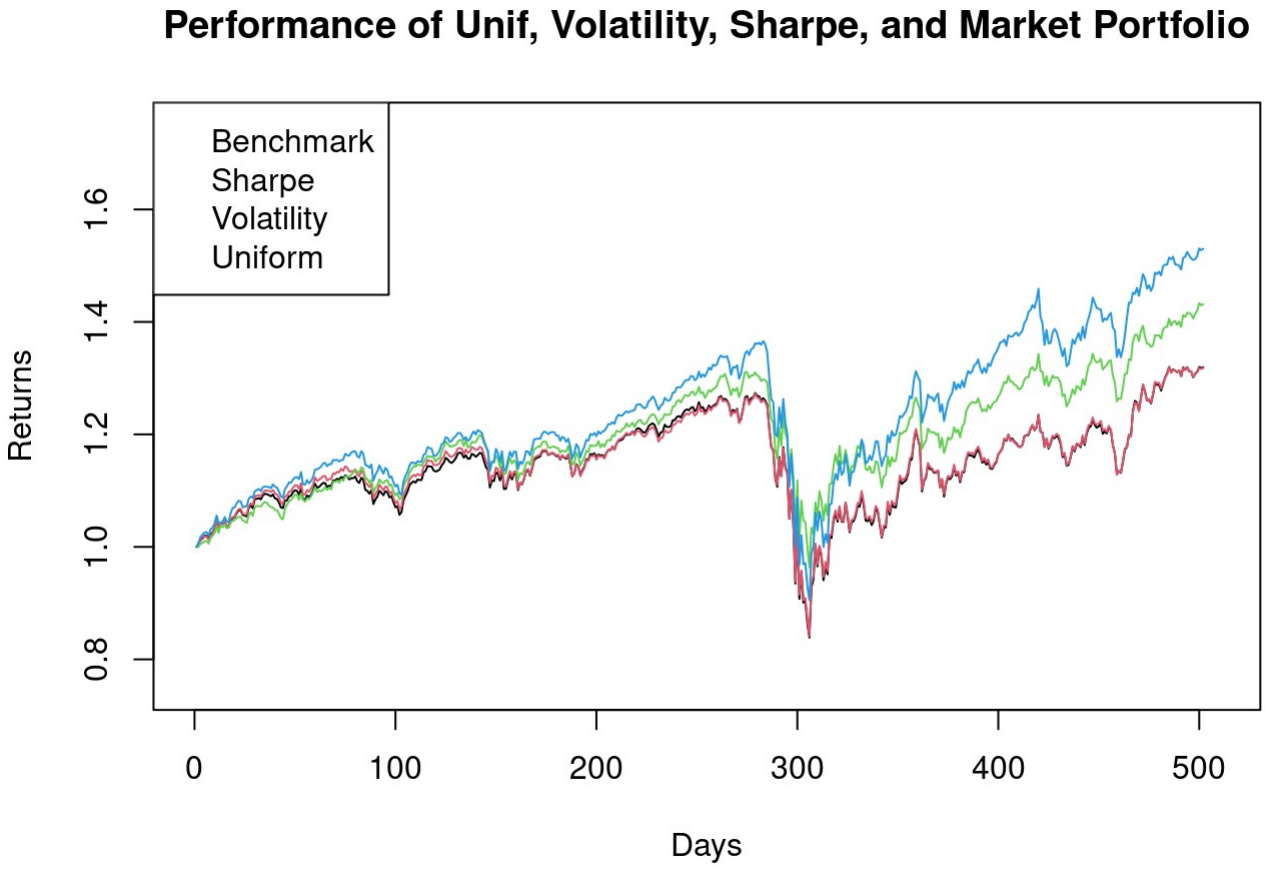
2. Questions: Are we given with share numbers?

unif_weight <dbl>	vol_weight <dbl>	sharpe_weight <dbl>
3.703704	2.924837	6.53011473
3.703704	5.784493	4.68644181
3.703704	9.039243	5.30574489
3.703704	2.113860	5.07931531
3.703704	2.483018	-2.18493631

unif_weight <dbl>	vol_weight <dbl>	sharpe_weight <dbl>
3.703704	3.178937	5.12610712
3.703704	3.526622	-0.05488801
3.703704	3.234802	-5.02448768
3.703704	4.666275	8.85691645
3.703704	5.694351	3.36380959

1-10 of 27 rows Previous **1** 2 3 Next

3.



3 Random Numbers and Monte Carlo Simulation

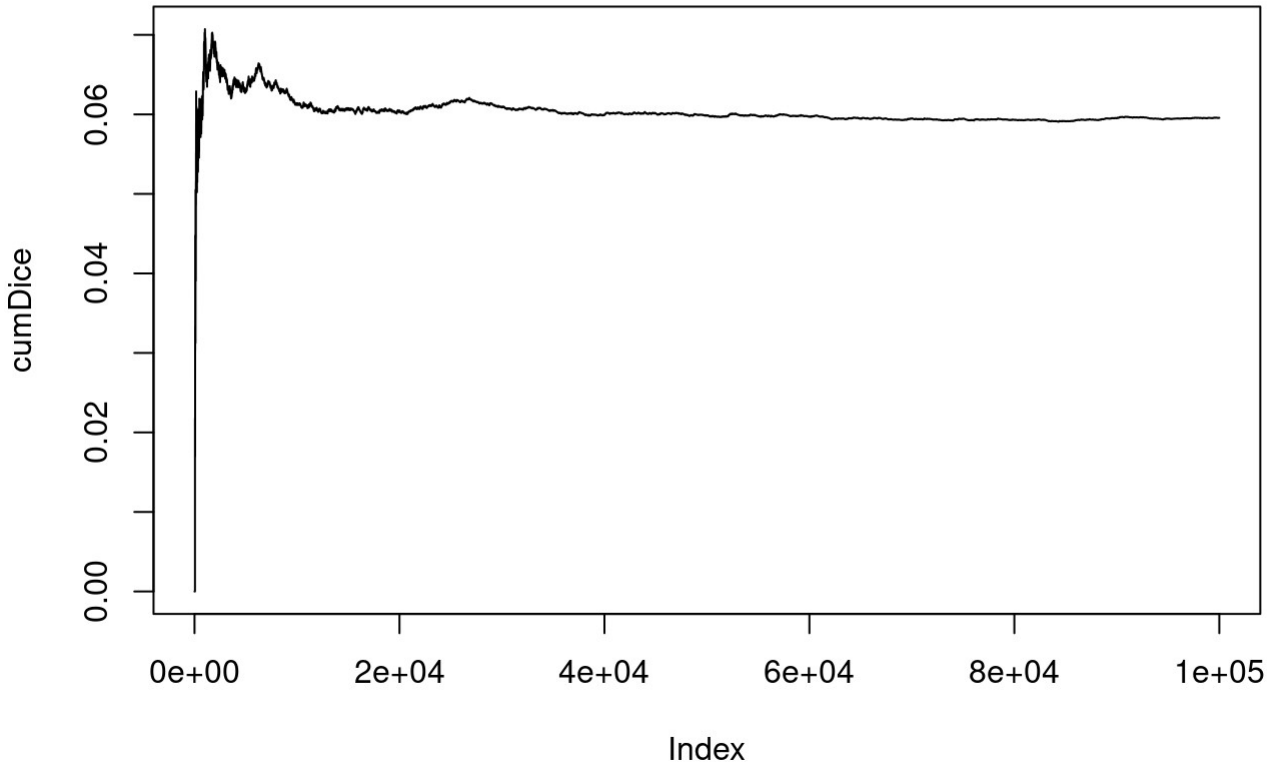
Core Questions (30 Points)

1. Monte Carlo

```
numSims = 100000
game <- function(){
  x = sample.int(6,6, replace = TRUE)
  return (as.integer((max(x) - min(x)) < 3))
}
diceGames <- replicate(numSims, game())
result <- mean(diceGames)
result #0.058444 (10_000_000 simulations)
```

```
## [1] 0.05956
```

```
####Graphing
cumDice <- cumsum(diceGames)/seq_along(diceGames)
plot(cumDice, type ='l')
```



Theoretical We have 6 dice, each with 6 options giving us 46,656 possible combinations. Our difference must be less than 3.

Analytical Solution

There are 6^6 total possible outcomes from 6 rolls. We want the probability that the max and min are 0, 1, or 2 apart. 0 apart means they are all the same number (6 ways).

1 apart means [1,2], [2,3], [3,4], [4,5], [5,6]…5 * (2^6 - 2) = 310.

2 apart means [1,3], [2,4], [3,5], [4,6]…4 * (3^6 - 2 * 2^6) = 2404.

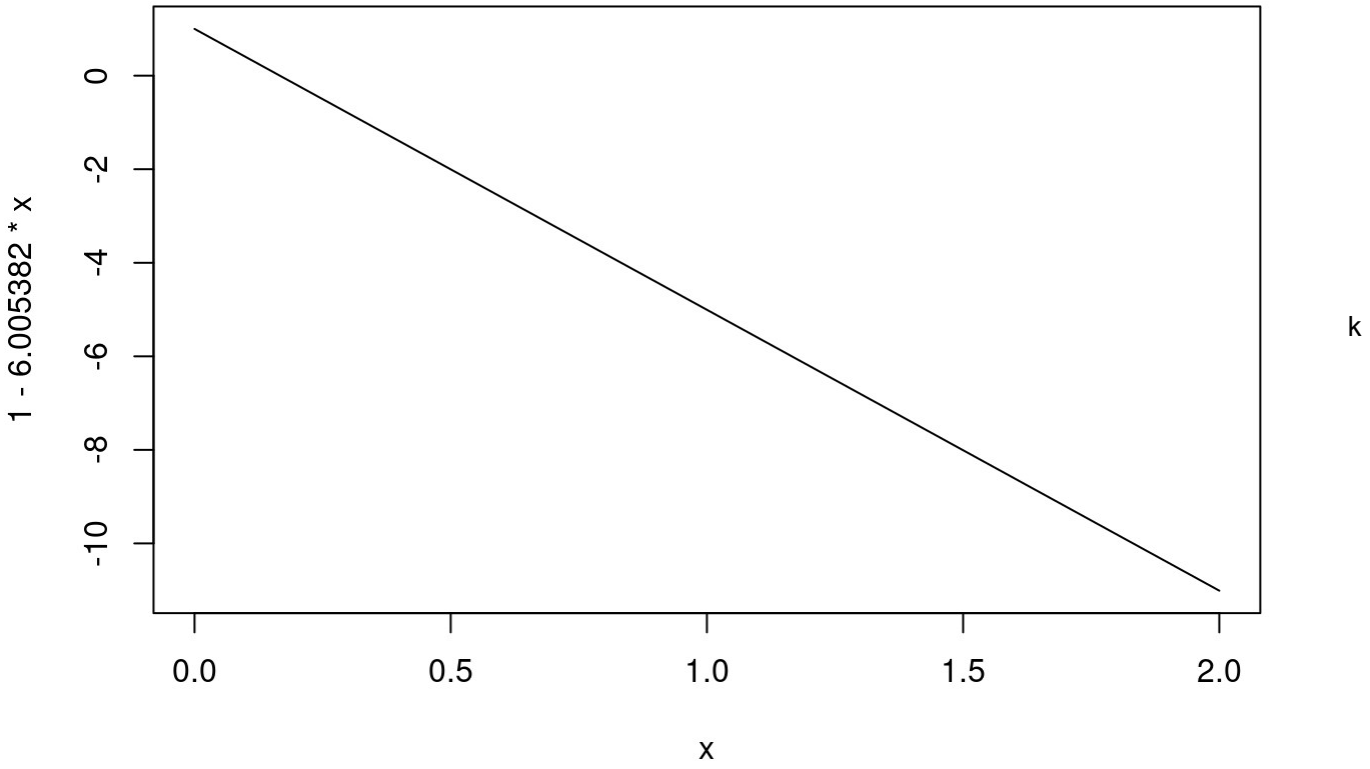
(1 + 310 + 2404)/6^6 = 0.05819187.

#2 To calculate k we must calculate X. To do that we need to do trials until we can get 2 wins in a row

```
#install.packages(Rcpp)
cppFunction("int flipCoins(){
  int prev = 0;
  int next = 0;
  int fails = 0;
  while ((prev != 1) || (next != 1)){
    prev = next;
    next = rand()%2+1;
    fails++;
  }
  return fails;
            }")
numSims = 1000000
flips <- replicate(numSims, flipCoins())
mean(flips) #6.005382
```

```
## [1] 6.010984
```

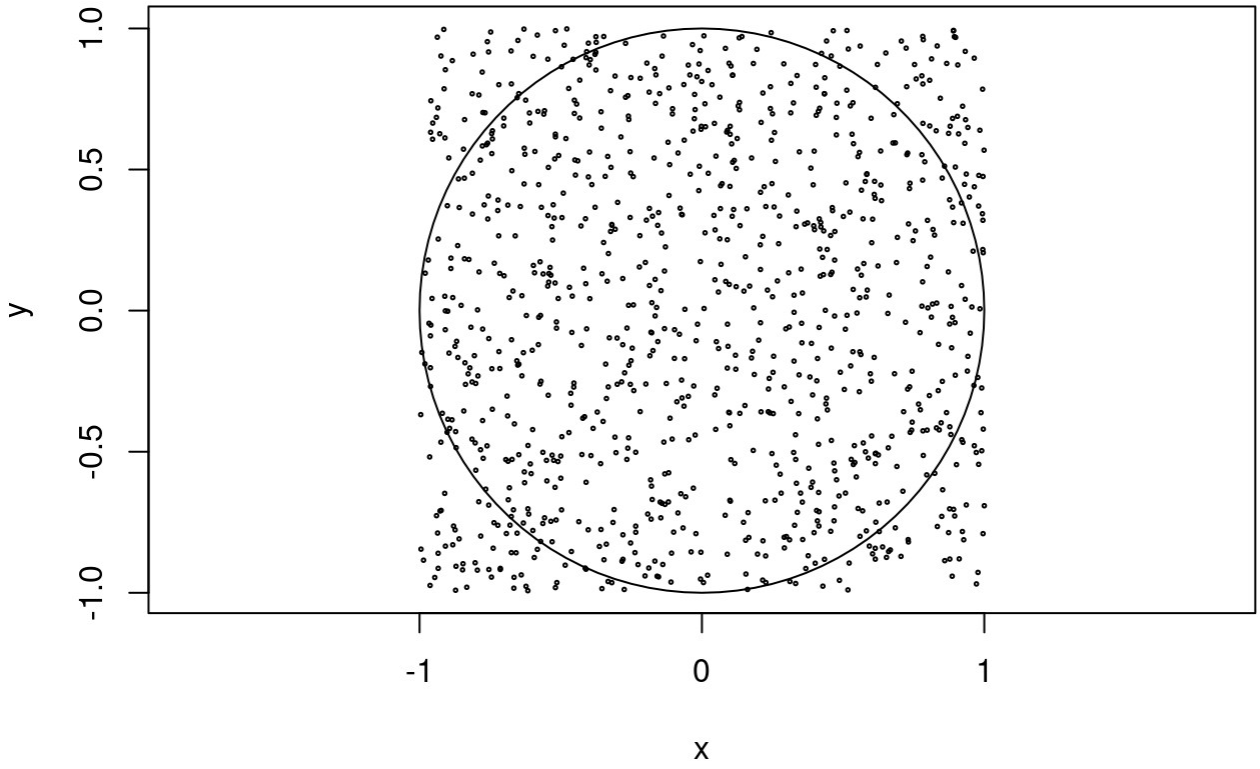
```
x<- seq(0,2,by=0.01)
plot(x, 1-6.005382*x, type = 'l')
```



should be 1/6.005382

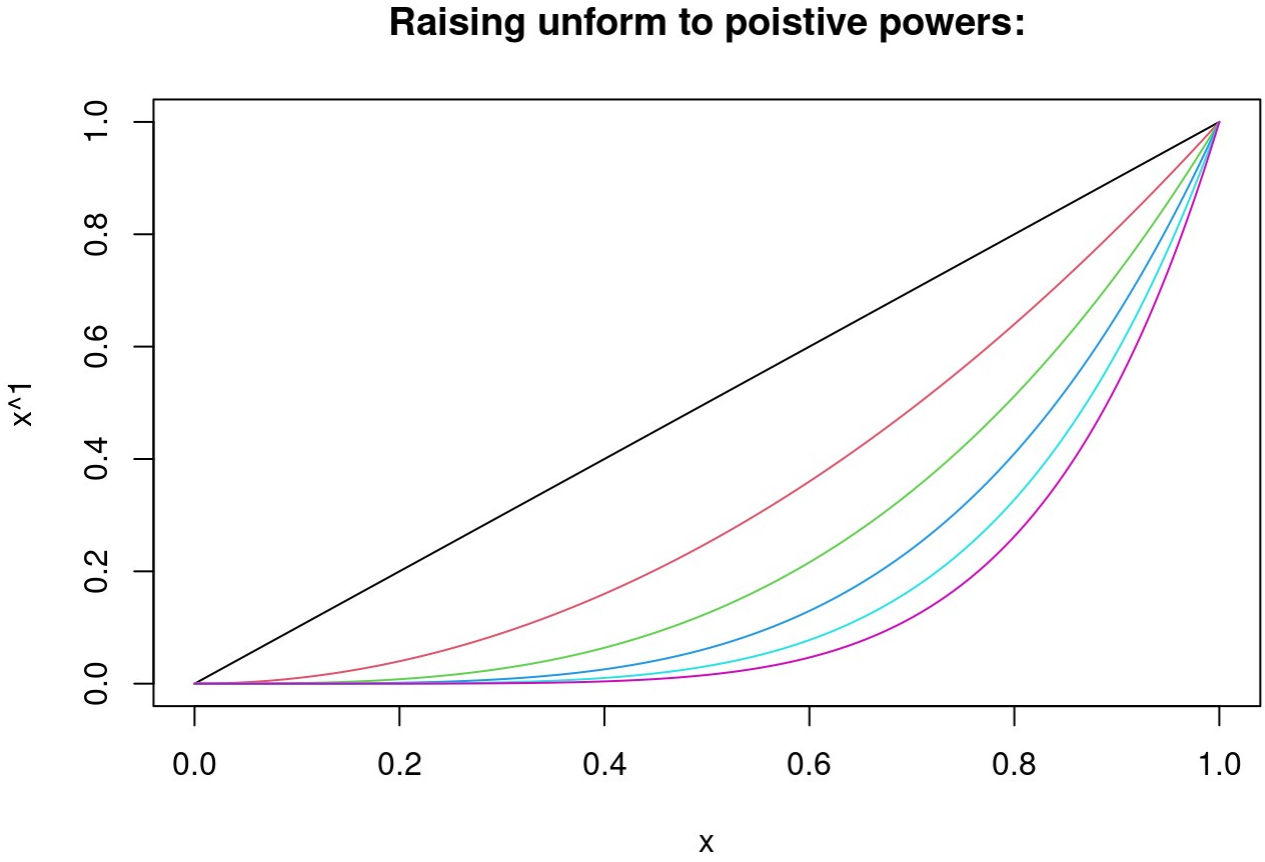
#3 calculate pi using a circle simulation

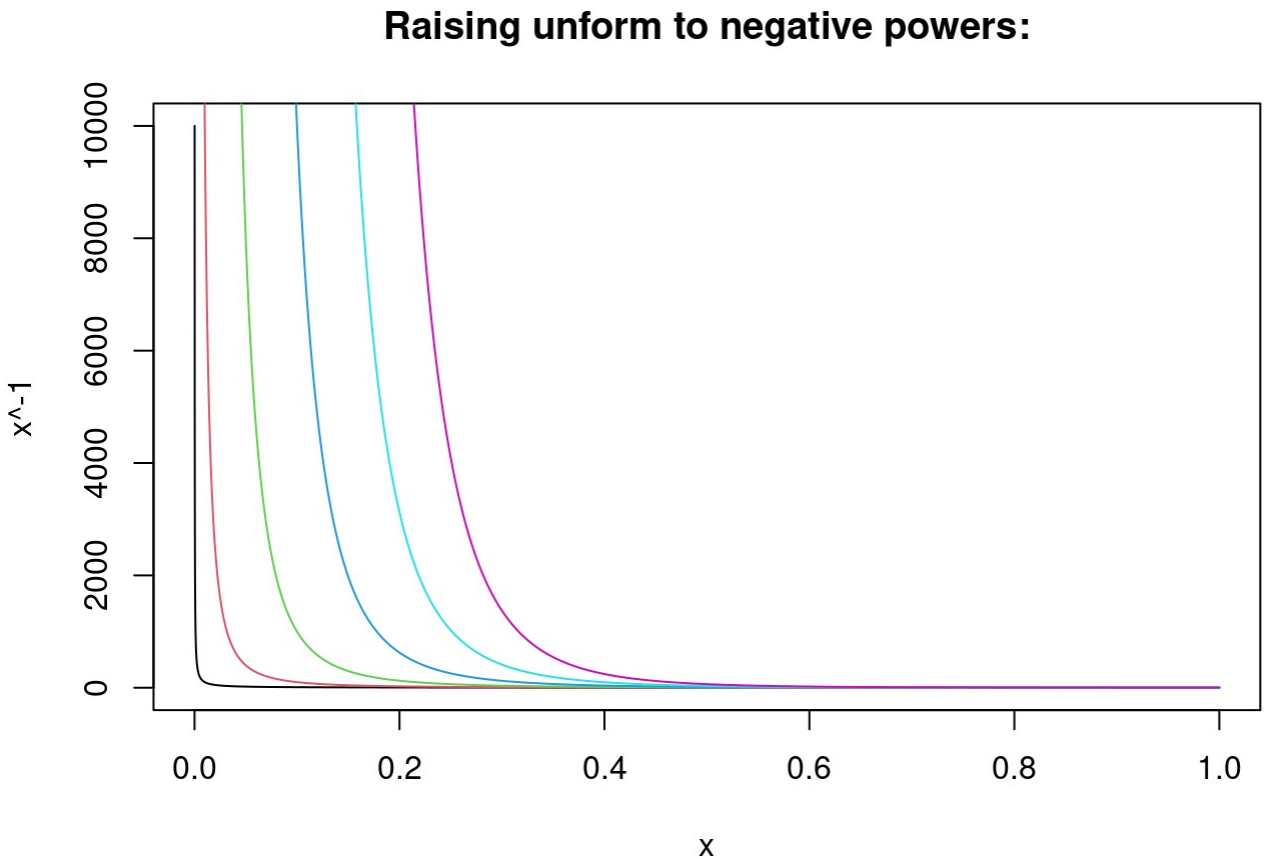
```
# install.packages('plotrix')
library(plotrix)
pi = function(n) {
  x = runif(n, -1, 1)
  y = runif(n, -1, 1)
  plot(x, y, asp = 1, xlim = c(-1, 1), cex = 0.25)
  draw.circle(0, 0, 1)
  pin = sum(ifelse(sqrt(x^2 + y^2 <= 1), 1, 0))
  print(4 * pin/n)
}
pi(1000)
```




```
## [1] 3.144
```

#4 Consider the case of a continuously distributed uniform random variable $X \sim U(0,1)$. At the same time, consider $Y_k = X^k$, where $k \in Z$ is an integer. Your task is the following:


Raising unform to poistive powers:
x^1
0.0
0.2
0.4
0.6
0.8
1.0
x


Raising unform to negative powers:
x^-1
0
2000
4000
6000
8000
10000
0.0
0.2
0.4
0.6
0.8
1.0
x

(a) Find a closed-form expression for the expectation and variance of Y_k, i.e.$E[Y_k]$and $V[Y_k]$.Show the steps needed to derive each expression. (4 Points)

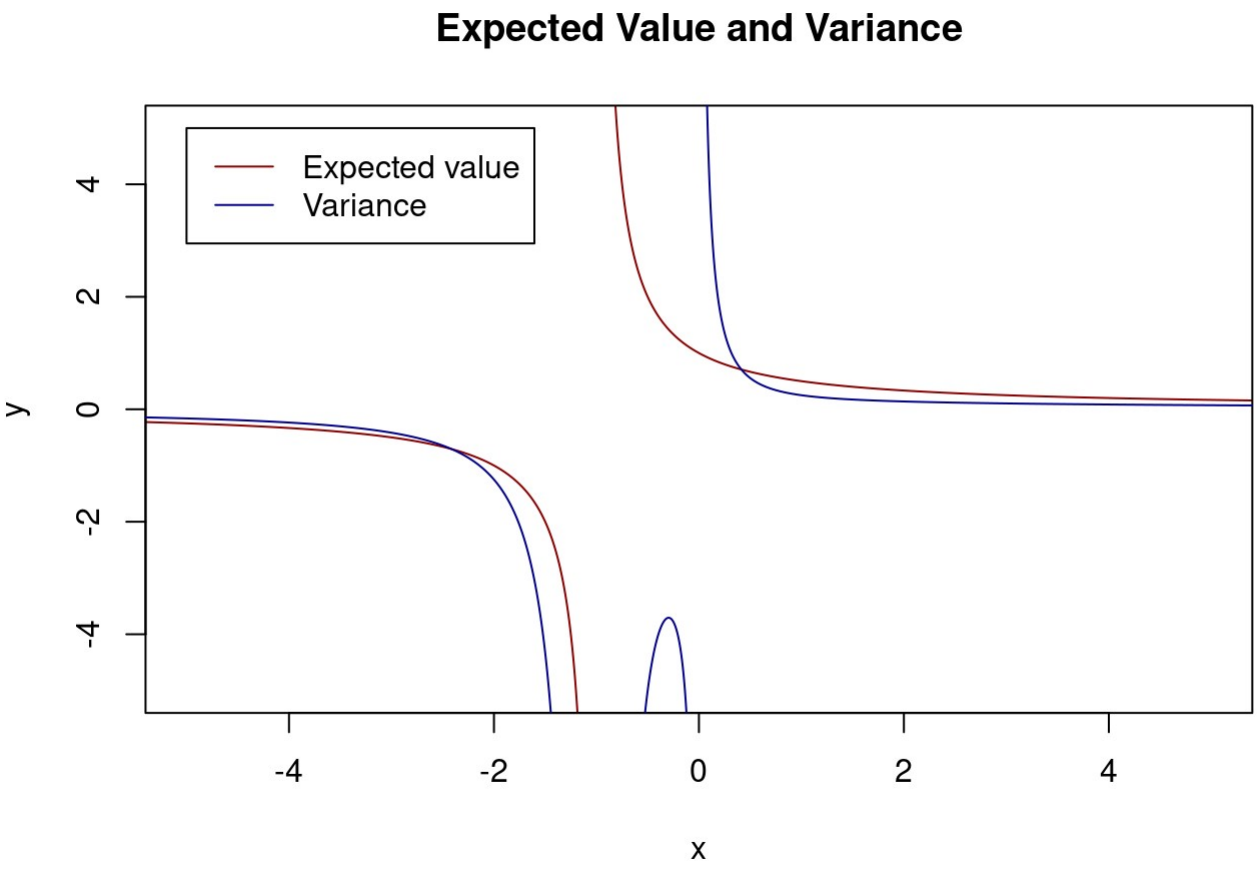
Let $Y_k = X^k$. Then Y_k is also a random variable. The cumulative distribution function of Y_k is given by:

$F_Y(y) = \mathbb{P}(X^k \leq y) = \mathbb{P}(X \leq y^{\frac{1}{k}}) = \int_0^{y^{\frac{1}{k}}} dx = y^{\frac{1}{k}}$
and the density function of Y is given by
$\frac{d}{dy} F_Y(y) = f_Y(y) = \frac{1}{k} y^{\frac{1-k}{k}}$ for $y \in [0,1]$.

The expected value $E[Y_k]$ is given by
$E[Y_k] = \frac{1}{k} \int_0^1 y \cdot y^{\frac{1-k}{k}} dy = \frac{1}{k+1}$.

Variance $V[Y_k]$ is given by
$V[Y_k] = E[Y_k^2] - E[Y_k]^2 = \frac{1}{2k} - \frac{1}{(k+1)^2} = \frac{k^2+1}{2k(k+1)^2}$

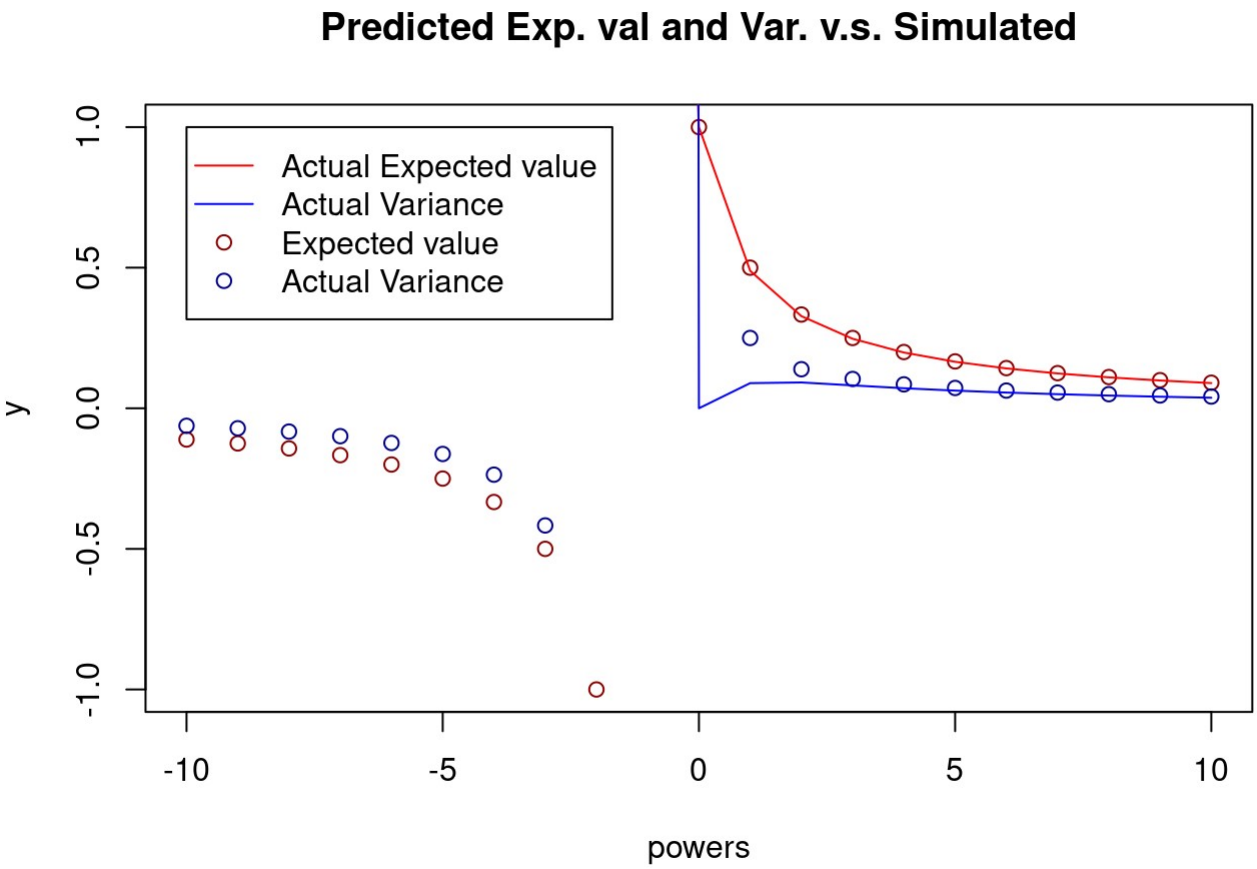
(b) In terms of k, find the condition for which $E[Y^k]$and $V[Y^k]$are finite while at the same time satisfying the condition $V[Y_k] > 0$. (2 Points)



The condition is satisfied for $k \in (0, \infty)$

(c) Using MC simulation, create a function that estimates $E[Y^k]$and $V[Y^k]$for a given k based on N= 105 samples. Plot each estimate versus the true value from step 1, for $k = -10, \ldots, -1, 0, 1, \ldots, 10$.What do the conditions from the previous step tell us? (3 Points)

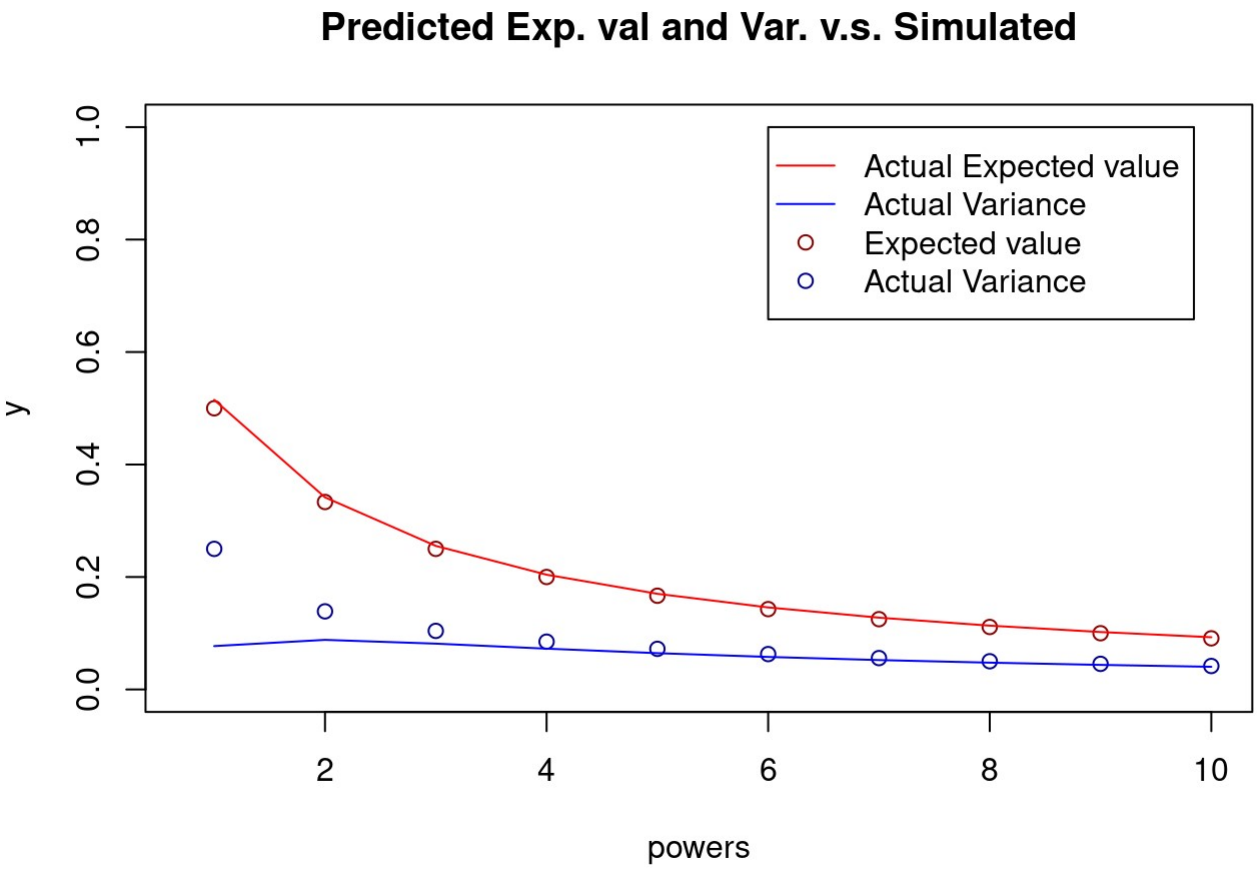
```
numSim = 105
powers = -10:10
#create a matrix of uniform values
unif_rand = runif(numSim)

#calculate the mean, var vectors for each power
means <-c()
vars <- c()
for(i in 1:length(powers)){
  unif_pow <- unif_rand^powers[i]
  means <- c(means, mean(unif_pow))
  vars <- c(vars, var(unif_pow))
}
```



The expected value and variance for values less than 0 is very large and off the graph.

(d) Finally, impose the conditions for k from step (b) and repeat the same plot from step (c). Elaborate. (1 Points)

```
numSim = 105
powers = 1:10
#create a matrix of uniform values
unif_rand = runif(numSim)

#calculate the mean, var vectors for each power
means <-c()
vars <- c()
for(i in 1:length(powers)){
  unif_pow <- unif_rand^powers[i]
  means <- c(means, mean(unif_pow))
  vars <- c(vars, var(unif_pow))
}
```



With the restrictions imposed we now see that our theoretical and predicted values are rather close, aside from the variance diverging towards the end.

Bonus Question

We can define the probabilities as a recursive function. 1/2 - head 1/2 - tail

Let x be the expected number of flips to get 1 head

$$x = \frac{1}{2} * 1 + \frac{1}{2}(x+1) \; \$x = \frac{1}{2} + \frac{x+1}{2} \; 2x = 2 + x \; x = 2$$

Observe above: we have a $\frac{1}{2}$ to get a head. In that case only 1 toss is required. In the case that a tail is thrown first we need to throw again. Hence, we mulitply the probability to get heads $\frac{1}{2}$ by $x+1$ because we have wasted 1 toss on a diifferent outcome. Effectivley, we repeat the same experiment but from a new point (one flip ahead) and add that to the result.

It is expected that 2 tirals are required to get a head. We extend the same logic recursively to find the probability to find the number of flips for 2 heads in a row.

$$x = \frac{1}{2} * (x+1) + \frac{1}{4} * (x+2) + \frac{1}{4} * 2$$
$$x = \frac{x+1}{2} + \frac{x+2}{4} + \frac{1}{2}$$
$$4x = 2x + 2 + x + 2 + 2$$
$$4x = 3x + 6$$
$$x = 6$$

Where $\frac{1}{2} * (x+1)$ is getting a tail on the next throw. $\frac{1}{4} * (x+2)$ is throwing a head and a tail next, *not a tail and a head*.
$\frac{1}{4} * 2$ is throwing 2 heads in a row, we do not add to x because no throws are wasted.
We can conclude that we expect to need 6 throws to get 2 heads in a row.

4. Value at Risk and Stress Testing

Task 1

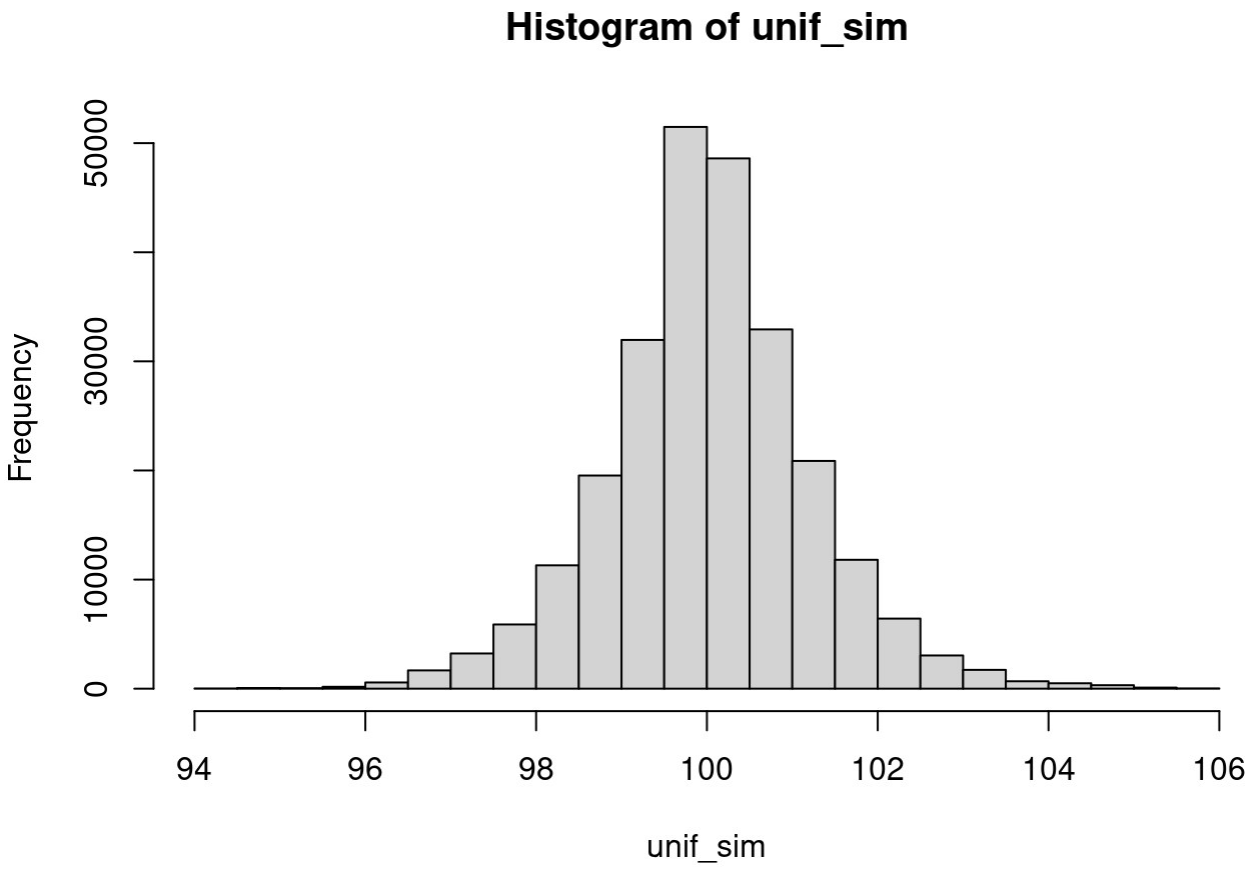
#1

```
unif_d = c()
vol_d = c()
sharpe_d = c()


for(i in 1:nrow(out_sample)){
  unif_d = c(unif_d, sum(as.numeric(out_sample[i, 2:28] * unif_weight), na.rm = TRU
E))
  vol_d = c(vol_d, sum(as.numeric(out_sample[i, 2:28] * vol_weight), na.rm = TRUE))
  sharpe_d = c(sharpe_d, sum(as.numeric(out_sample[i, 2:28] * sharpe_weight), na.rm =
TRUE))
}

unif_m = mean(unif_d)
unif_s = sd(unif_d)
vol_m = mean(vol_d)
vol_s = sd(vol_d)
sharpe_m = mean(sharpe_d)
sharpe_s = sd(sharpe_d)

table = matrix(c(unif_m, vol_m, sharpe_m, unif_s, vol_s, sharpe_s), ncol = 3, byrow =
TRUE)
colnames(table) = c("Portfolio 1", "Portfolio 2", "Portfolio 3")
rownames(table) = c("Mean", "STD")
table
```

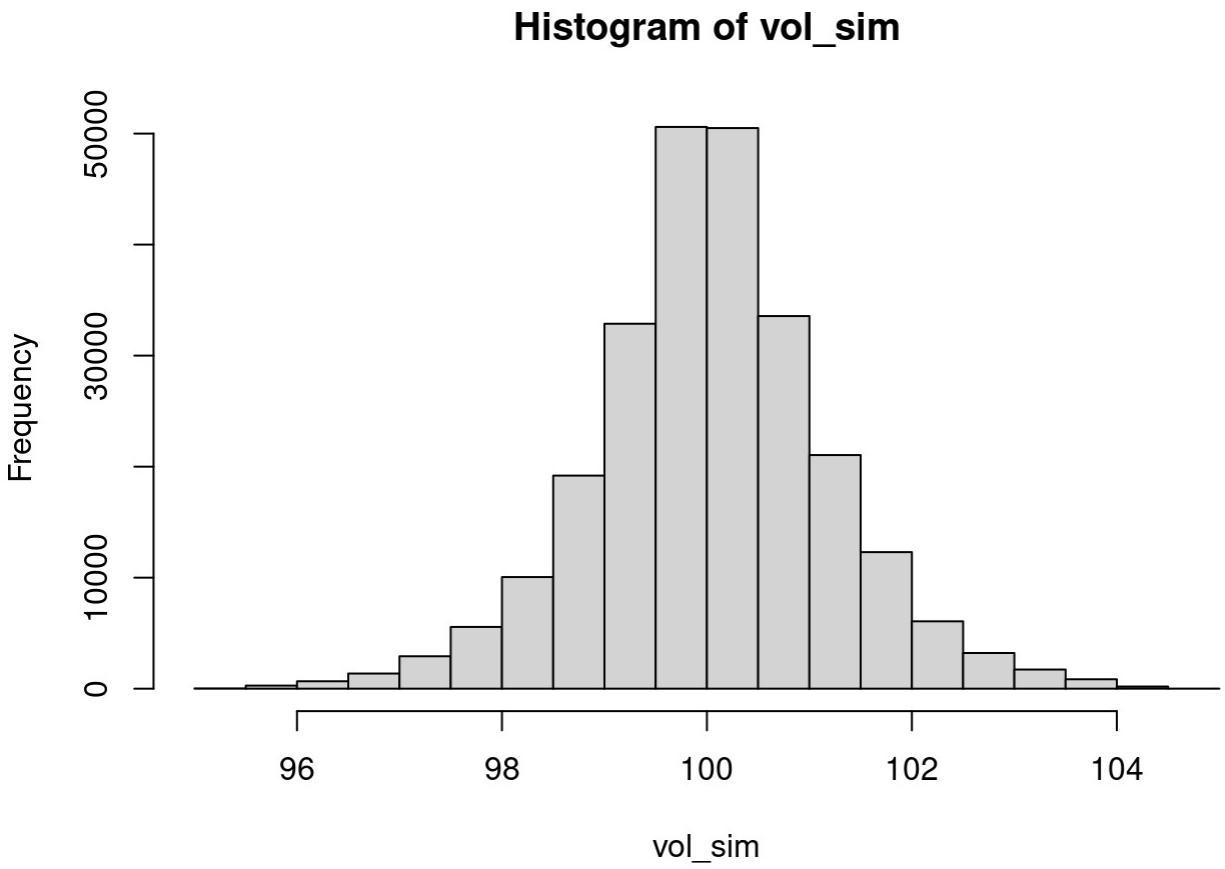
```
##       Portfolio 1 Portfolio 2  Portfolio 3
## Mean 0.0006148328 0.000615357 0.0007726735
## STD  0.0167541929 0.016606361 0.0146933748
```

#2

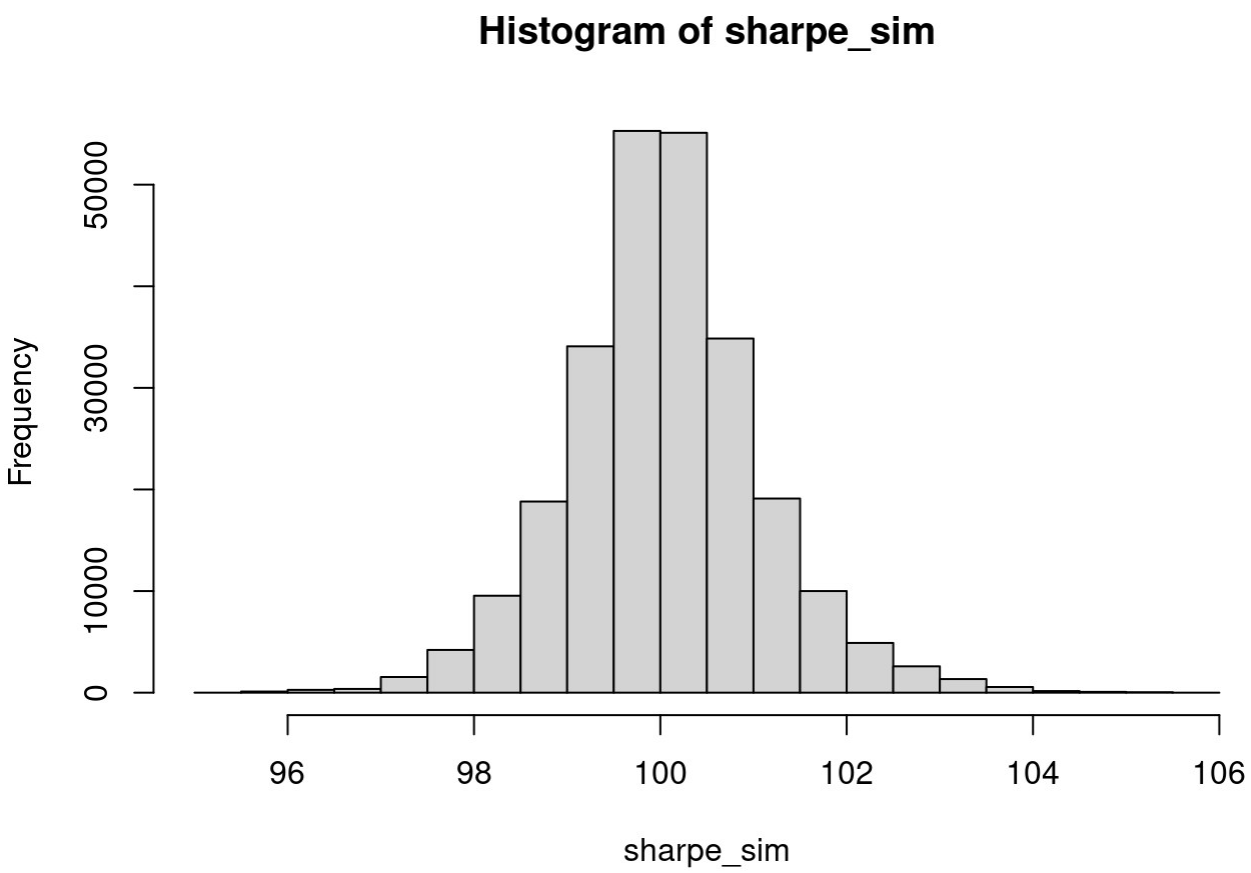
```
simGBM <- function(S = 100, DTintervals = 252, mu = 0.1, sig = 0.2, numSims = 1){

   dt <- 1/DTintervals
   gbmVals <- matrix( rnorm(DTintervals*numSims, dt*(mu - 0.5*sig^2), sig*sqrt(dt)),
nrow = DTintervals, ncol = numSims)
   gbmVals <- apply(gbmVals, 2, cumsum)
   sims <- (matrix(S, nrow = DTintervals, ncol = numSims)) *exp(gbmVals)
   sims <- rbind(c(rep(S, numSims)),sims )
   return(sims)
}
unif_sim = simGBM(100, 252, unif_m, unif_s, 1000)
vol_sim = simGBM(100, 252, vol_m, vol_s, 1000)
sharpe_sim = simGBM(100, 252, sharpe_m, sharpe_s, 1000)

hist(unif_sim)
```


Histogram of unif_sim
Frequency
0
10000
30000
50000
94
96
98
100
102
104
106
unif_sim

```
hist(vol_sim)
```




```
hist(sharpe_sim)
```



All three portfolio follow normal distribution. The second portfolio tends to have a larger tail.

3

```
mean(unif_sim[253,])
```

```
## [1] 100.0788
```

```
mean(vol_sim[253,])
```

```
## [1] 100.0629
```

```
mean(sharpe_sim[253,])
```

```
## [1] 100.1091
```

#4

```
unif_log = diff(log(unif_sim[,5]))[-1]
vol_log = diff(log(vol_sim[,5]))[-1]
sharpe_log = diff(log(sharpe_sim[-5]))[-1]

unif_var = (sd(unif_log) * 1.65) -  mean(unif_log)
vol_var = (sd(vol_log) * 1.65) - mean(vol_log)
sharpe_var = (sd(sharpe_log) * 1.65) - mean(sharpe_log)

var_report = matrix(c(unif_var,vol_var,sharpe_var), ncol = 3, byrow = TRUE)
colnames(var_report) = c("Portfolio 1", "Portfolio 2", "Portfolio 3")
rownames(var_report) = "Var(0.05)"
var_report
```

```
##           Portfolio 1 Portfolio 2 Portfolio 3
## Var(0.05) 0.001822517 0.001751383 0.002148048
```

#Task 2

```
SPY_OUT = getSymbols("SPY", from="2019-01-01", to="2020-12-31", env = NULL)[, 6]
SPY_log = log(lag(SPY_OUT, -1) / SPY_OUT)[-1]
SPY_log = SPY_log[-length(SPY_log)]
SPY_s = sd(SPY_log)

beta_unif = as.numeric(cor(unif_d[-503], SPY_log)) * sd(unif_d)/SPY_s
beta_vol = as.numeric(cor(sharpe_d[-503], SPY_log)) * sd(unif_d)/SPY_s
beta_sharpe = as.numeric(cor(sharpe_d[-503], SPY_log)) * sd(unif_d)/SPY_s


a = 0.1
unif_new_s = a * SPY_s * beta_unif
vol_new_s = a * SPY_s * beta_vol
sharpe_new_s = a * SPY_s * beta_sharpe


unif_new_sim = simGBM(100, 252, unif_m, unif_new_s, 1000)
vol_new_sim = simGBM(100, 252, vol_m, vol_new_s, 1000)
sharpe_new_sim = simGBM(100, 252, sharpe_m, sharpe_new_s, 1000)

unif_new_log = diff(log(unif_new_sim[,5]))[-1]
vol_new_log = diff(log(vol_new_sim[,5]))[-1]
sharpe_new_log = diff(log(sharpe_new_sim[-5]))[-1]

unif_new_var = (sd(unif_new_log) * 1.65) -  mean(unif_new_log)
vol_new_var = (sd(vol_new_log) * 1.65) - mean(vol_new_log)
sharpe_new_var = (sd(sharpe_new_log) * 1.65) - mean(sharpe_new_log)

var_new_report = matrix(c(unif_new_var,vol_new_var,sharpe_new_var), ncol = 3, byrow =
TRUE)
colnames(var_new_report) = c("Portfolio 1", "Portfolio 2", "Portfolio 3")
rownames(var_new_report) = "New Var(0.05)"
var_new_report
```

```
##                Portfolio 1  Portfolio 2  Portfolio 3
## New Var(0.05) 0.0001664211 0.0001569171 0.0002478129
```

5. Mean-Variance Efficient Frontier

```
new_returns_data = returns_data[-1]
mu = c()
for (i in (1:28)){
  if (i == 20){
    mu = c(mu, mean(new_returns_data[,i]))
  }
  else{
    mu = c(mu, mean(new_returns_data[,i]))
  }
}

cov = cov(new_returns_data)
inv_cov = solve(cov)

one_col = matrix(c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1))
one_row = t(one_col)

w0 = as.vector(inv_cov %*% one_col) / (one_row %*% inv_cov %*% one_col)
```

```
## Warning in as.vector(inv_cov %*% one_col)/(one_row %*% inv_cov %*% one_col): Recyc
ling array of length 1 in vector-array arithmetic is deprecated.
##   Use c() or as.vector() instead.
```

```
I = diag(28)

B = inv_cov %*% (I - (one_col %*% t(w0)))

w1 = B %*% mu
```

Last Bonus Question

$$\lambda\omega_0 + (1-\lambda)\omega_{SR} = \omega_0 + \frac{1}{A_m}B\mu$$

$$(\lambda - 1)(\omega_0 - \omega_{SR}) = \frac{1}{A_m}B\mu$$

$$\lambda = \frac{B\mu}{A_m(\omega_0 - \omega_{SR})} + 1$$

This should work because B, A, w_o, w_{SR} are all in terms of the covarience matrix.