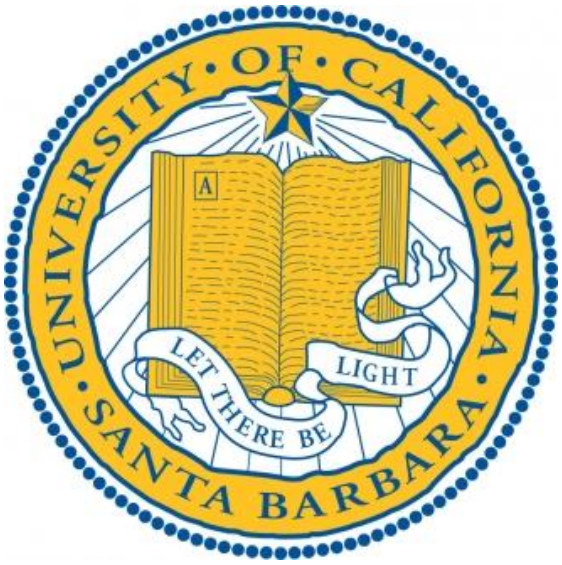

PSTAT 176/276 Final Project

Spring 2020

Monte Carlo Methods & Estimation:
Application with Microsoft stock prices

He, Rebecca

Khan, Samran

Kinel, Ron

Li, Wenjing

Zhou, Lingyu

Supervisor:

Tao Chen

Problem 1

Make a function StockVol to calibrate the stock volatility under geometric Brownian motion model. Input histoPrice is an array of 1-year historical prices. You can choose a stock that does not pay dividends for simplicity. The function should return a number that is the historical volatility of the stock.

```
StockVol <- function(histoPrice){
  R <- array(0 , length(histoPrice)-1)
  for (i in 1:(length(R))){
    R[i] <- log(histoPrice[i+1]/histoPrice[i])
  }
  Daily_vol <- sqrt(var(R))
  Annual_Vol <- Daily_vol*sqrt(252)
  return(Annual_Vol)
}
```

Historical volatility is based on historical prices and represents the degree of variability in the return of an asset. For this problem, we are analyzing daily returns for an indivdual stock so once we calculated the volatilty of our stock price data, we scaled it in order to obtain the annual volatility from the daily volatility.

Problem 2

Make a function StockPath to generate n stock paths where n is one of the inputs, as well as sigma which is the volatility of the stock. Other inputs needed (might not be limited to) are S0: current stock price; T: terminal time in yearly unit; np: number of time periods; r: interest rate; delta: continuous dividend yield of the stock.

```
#n: number of paths
#sigma: volatilty of stock
#S0: current stock price
#t: terminal time in yearly unit
#np: number of periods
#r: interest rate
#delta: continuous dividend yield of the stock
StockPath <- function(n, sigma, S0, t=1, np=52, r, delta=0){
  S <- array(0, dim=c(n,np+1))
  S[,1] <- S0
  Z <- matrix( rnorm(n*(np)), n, (np))
  h <- t/np
  #h: length of each period = t/np

  for(i in 1:n){
    for (j in 1:np){
      S[i,j+1] <- S[i,j]*exp((r-delta-((sigma^2)/2))*h + sqrt(h*(sigma^2))*Z[i,j])
    }
  }

  mypath <- S
  return(mypath)
}
```

Once we are able to calcualte the historical volatility of our stock, we can simulate possible stock paths based on stock and time parameters. This is done through Monte Carlo simulation under the Black-Scholes model assumptions.

Problem 3

Make a function EurOptPrice that takes the stock paths to generate the European put option price through Monte Carlo method. One input is n stock paths. The function should return the discounted payoff vector, price, and variance.

```
EurOptPrice <- function(mypath, t, r, k){
  n <- nrow(mypath)
  np <- ncol(mypath) - 1
  disc_payoff <- array(0, dim=c(n,1)) #discounted payoff vector

  for (i in 1:n){
    disc_payoff[i,1] <- exp(-r*t)*pmax(k-mypath[i,(np+1)] ,0)
  }

  option_price <- mean(disc_payoff)
  option_var <- var(disc_payoff)

  return(list(option_price,disc_payoff, option_var))
}
```

Using the stock path simulation we can obtain a payoff for each scenarios for a European option with with a given strike and expiration expiration. We can discount the payoff for each path using a given interest rate and estimate a price as well as the risk for the specified option by taking the average and standard deviation of the discounted payoffs.

Problem 4

Make a function AmeOptPrice that takes the stock paths to generate the American put option price without control variates. The function should return the discounted payoff vector, price, and variance. In this part, you will need to implement some regression method and you are required to do it by using machine learning or deep learning. Make sure to explain what you did in the analysis file.

```
library(neuralnet)
```

```
## Warning: package 'neuralnet' was built under R version 3.5.3
```

```
AmeOptPrice <- function(C,t, sigma, r, k ,delta){
  S0 <- C[1,1]
  steps <- ncol(C)-1
  dt <- t/steps #time interval for each step
  early_price <- array(0,dim=dim(C))  #create an array for payoffs
  early_price <- pmax(k-C,0) #generate a matrix for payoffs
  holding_value <- array(0,dim=dim(C)) #create an array for holding values
  option_value <- array(0,dim=dim(C)) #creat an array for option values
  holding_value[,ncol(holding_value)] <- early_price[,ncol(early_price)]
  option_value[,ncol(option_value)] <- early_price[,ncol(early_price)]
    #assign last column values of holding value and option value as last column values of payoffs
  Second_to_last <- C[,ncol(C)-1]
    #start iterating from second to last column
  for (i in 1:nrow(C)){
    #iterate each row of second to last column and get all holding values
    Sim_St <- StockPath(nrow(C), sigma, Second_to_last[i], dt, np=1, r, delta)
    current_payoff <- pmax(k-Sim_St[,ncol(Sim_St)],0)*exp(-r*dt)
    holding_value[i,ncol(holding_value)-1] <- mean(current_payoff)
  }
```

```
  option_value[,ncol(option_value)-1]<-
    pmax(holding_value[,ncol(holding_value)-1],
         early_price[,ncol(early_price)-1]) #compare to get option value at second to last column

  #Using neural networks to perform our analysis
  for (j in seq(from=ncol(holding_value)-1, to=2, by=-1)){ #iterate from second to last column all the
    Predictor <- C[,j]
    Response <- option_value[,j]
    newC <- data.frame(Predictor,Response)
    regress_model <- neuralnet(Response~Predictor, data=newC,
                               linear.output = FALSE) #neural network model
    current_col <- C[,(j-1)] #get every column
    for(k in 1:nrow(C)){ #iterate each row
      New_Sim_St <- StockPath(nrow(C), sigma,
                              current_col[k], dt, np=1, r, delta)
        #use each current stock price to perform simulations
      data <- data.frame(New_Sim_St)
      New_data <- data.frame(data$X2) #simulated stock prices
      prob <- predict(regress_model, newdata=New_data)
        #predicted probability based on neural network method
      predicted_value <- prob*exp(-r*dt)*New_Sim_St[k]
        #predicted stock prices
      holding_value[k,j-1] <- mean(predicted_value)
        #assign holding values based on predicted value
    }
    option_value[,j-1] <- pmax(early_price[,j-1],holding_value[,j-1])
      #compare payoffs and holding values
  }
  Option_Price <- array(c(rep(0,nrow(early_price))))
    #an empty array for inserting values
  for (z in 1:nrow(early_price)){
    exercise_indexes <- which(early_price[z,]>holding_value[z,])
      #compare current payoffs and holding values, and determine exercise time
    if(length(exercise_indexes)!=0){
       Option_Price[z] <- option_value[z,exercise_indexes[1]]*exp(-r*exercise_indexes[1]*dt)
        #if early exercise, choose earliest exercise time and discount
    }
    else{
      Option_Price[z] <- option_value[z,ncol(option_value)]*exp(-r*t)
        #if no early exercise, use current option value and discount to beginning
    }
  }
  Price <- mean(Option_Price)
  Variance <- var(Option_Price)
  return(list(Price,Option_Price,Variance))
}
```

For this funtion, the machine learning method we are implementing is the neural network method. Because the stock market was very volatile for the past few months, we decided to use a more complex regression model to obtain better prediction power. This method will create a network, prepare data for it, then train the network and evaluate its performance on the test set.

We first find the option value by comparing the exercise price and holding price at T-1 to initialize our regression model.

Next, we work backwards from the second last column to beginning. We build another for-loop to make iteration. In this part, we build a neural network model with changing stock prices as predictor and changing option values as response. We also generate new simulations based on new current stock prices. Then, we make predictions based on neural network model and get probabilites which determine our prediction scores. By multiplying probabilites and current simulated prices, we can get estimated prices which can be used to find holding values by taking average. And we can get all values for option values by comparing holding values and payoffs.

The last part of codes is used to determine the optimal exercise times. By comparing exercise price and holding values at each time node, we can get indexes for optimal exercising times. If there is no such optimal exercise time, the terminal time is optimal exercising time, we simply discount current option value to beginning. If there are optimal exercise times, we choose the smallest index and discount corresponding periods.

Lastly, we can get estimated option price and variance from the simulation and predction of option prices.

Problem 5

Make a function ContVariate to implement the control variates method. Note that this part is independent of the prices you computed. You should be able to apply this function to any vectors and estimations.

```
ContVariate <- function(Unknown_Vec, Known_Vec, Known_Value){
  B <- cov(Unknown_Vec, Known_Vec)/var(Known_Vec)
  Ca <- as.matrix(Unknown_Vec) + as.numeric(B)*(Known_Vec - Known_Value)
  return(mean(Ca))
}
```

The control variates method is a variance reduction technique used in Monte Carlo methods. It exploits information about the errors in estimates of known quantities to reduce the error of an estimate of an unknown quantity.

Problem 6

Choose your favorite underlying stock and a corresponding put option that expires in 1 year. Use 1 year libor yield as the interest rate. You also need to compute the continuous dividend yield. Apply the ContVariate function to the American put option price you computed. Do the analysis as if you want to convince your manager that this is the right way to estimate the price of American style option:

We chose Microsoft(MSFT) to do our analysis on. We first import the last year of data for Microsoft. There are 252 trading days between 06/13/19 and 06/11/20 which are the start and end dates of our data.

According to nasdaq.com, MSFT Dividend History in this period is as follows:
05/20/2020: $0.51
02/19/2020: $0.51
11/20/2019: $0.51
08/14/2019: $0.46

According to macrotrends.net, the current 1 year LIBOR rate as of June 04, 2020 is 0.63%. Although it is better to use advanced intrest rate modeling, the 1 year LIBOR is a global “benchmark” for short-term interest and so we will use it instead.

Below we input MSFT last year stock information and extract the closing prices. We will use the function from problem 1 to compute MSFT’s annual volatility. We also assign the 1-year LIBOR rate and compute the continuous dividend yield using the Dividend History above. All stock data was extracted from Yahoo! Finance.

```
#Input Prices, Spreadsheet included in pproject submission
MSFT <- read.csv(file = 'C:/Users/ron12/OneDrive/Desktop/PSTAT 276/MSFT.csv',
                 header = T, sep = ",")
MSFT_Price <- MSFT$Close
LIBOR <- 0.0063
MSFT_Vol <- StockVol(MSFT_Price)
MSFT_Div <- log((1 + (0.51/MSFT$Close[MSFT$Date == "5/20/2020"]))*(1 + (0.51/MSFT$Close[MSFT$Date == "2/
  (1 + (0.51/MSFT$Close[MSFT$Date == "11/20/2019"]))*(1 + (0.46/MSFT$Close[MSFT$Date == "8/14/2019"])))
cat("The annual volatility for MSFT is:", MSFT_Vol, "\n\n")
```

```
## The annual volatility for MSFT is: 0.4031884
```

```
cat("The continuous dividend yield is:", MSFT_Div)
```

```
## The continuous dividend yield is: 0.01229299
```

Next, we will simulate the next 1-year MSFT prices by weeks using our StockPath function from Problem 2. This simulation is done using standard Monte Carlo method. For more information, see the description below Problem 2 .

We use weeks because it splits a year into a large number of time periods but its not as computationally heavy as using days. Additionally, it is common to see 1-year stock analysis done in weeks. We will use 200 simulations. This is not a very large amount but should still be sufficient for our analysis while remaining computationally quick.

```
set.seed(50)
n = 200
T = 1
np = 52
r <- LIBOR
MSFT_Path <- StockPath(n, MSFT_Vol , MSFT_Price[length(MSFT_Price)] , T, np, r, MSFT_Div)
```

For our analysis, we will use the 1-year Put: MSFT|20210618|190.00P with Last = 27.25, Bid = 24.1, and Ask = 28.15.
This put expires in almost exactly a year and should be a good comparison point for our analysis.

First, we will use the American Put function from problem 4 to find the Monte-Carlo Machine-Learning Adjusted price. This function employs a neural net and recursive simulation at each time node to estimate the best price for the American Option. For more information on how the function works, see the explanation following Problem 4 above.

```
K <- 170
Simulated_MSFT_Prices <- as.matrix(MSFT_Path)
MSFT_put <- AmeOptPrice(Simulated_MSFT_Prices , T, MSFT_Div,r, K, MSFT_Div)
cat("The estimated price of the put using neural net with Monte Carlo simulation is: $" , MSFT_put[[1]]
```

```
## The estimated price of the put using neural net with Monte Carlo simulation is: $21.2182
```

The result above performs okay - about $4 above the current equivalent option price. We would hope to get a price that is a little closer to the current option price. We are unsure with the reason of the deviation in prices but we think this price may be more 'randomly' computed, and therfore have more variation.

Next, we will use the Black-Scholes models and our European Put Option Function from Problem 4 to incorporate in the control variates function from Problem 5 with our result from the previous part. The Control Variates method is a variance reduction technique applied on Monte Carlo simulations. For more information about the Control Variate technique read the the discription following the chunk in Problem 5.

We obtain the Black-Scholes price using the bsput function from the 'derivmkts' library and the payoffs from our simulated payoffs using the European Option function from Problem 3. For more information on how

this funtion works, see the explanation following Problem 3.

```
library(derivmkts)
```

```
## Warning: package 'derivmkts' was built under R version 3.5.3
```

```
MSFT_Euro_True <- bsput(MSFT_Price[length(MSFT_Price)], K, MSFT_Vol, r, T, MSFT_Div)
cat("The true European BS Put price is:" , MSFT_Euro_True, "\n\n")
```

```
## The true European BS Put price is: 21.28311
```

```
MSFT_Euro_Sim <- EurOptPrice(MSFT_Path, T, r, K)
cat("The Simulated Put price is:", MSFT_Euro_Sim[[1]])
```

```
## The Simulated Put price is: 21.51736
```

Next, we use the results above with control variates function.

```
Cont_Variate_put <- ContVariate(MSFT_put[[2]], MSFT_Euro_Sim[[2]], MSFT_Euro_True)
Cont_Variate_put
```

```
## [1] 21.44257
```

This result did not effect our option price too significantly. It did shift our result in the opposite direction of the real option price. This further supports our American put option price estimation and that the current price for this option is too low. Both our machine-learning MC price and the control variate estimation estimated higher prices than the market which leads us to belive the price should be increased to avoid creating arbitrage opportunities.

Conclusion

Through this project, our group members optimized the financial knowledge we learned throughout this course along with our machine learning skills. In the past few weeks, we explored different machine learning methods and read multiple papers in the field of financial mathematics. We first tried linear regression. However, this model is too simple for our volatile stock. Next, we tried a random forest and multinomial logistic regression. Finally, based on the many prominent papers we read, we decided to use a neural network as our model. Although this project only constructs a simple pricing model, it is the first financial model we created-and the process turned out to be very complex, yet rewarding. Overall, we are proud to have worked on this project. It provided us with an opportunity to be introduced into the real world applications and the field of research related to financial mathematics.