Jackpot Model, a Study in Monte Carlo Simulation

The Monte Carlo method is a powerful tool that allows for predictive statistical analysis on the price of assets that have a random component. We will focus our predictive analysis on European call and put options. A put option is a contract that allows, for a premium, the buyer the right to sell a certain number of shares at the strike price at expiration. A long holder can exercise the option and sell the shares at a fixed price, at a price higher than the market, and thus pocket the difference between the fixed and market price at expiration. The call option, by contrast, is an agreement that allows the buyer to purchase shares at the strike price at expiration. A long holder can exercise the option and buy the shares at a fixed price, at a price lower than the market, and thus profit from the difference between the market and fixed price at expiration. We will be focusing on the European Call option, with a random interest rate component.

The model we wanted to use for this project was the Jackpot model. This model combined the Vasicek and the Variance-Gamma models together. We needed to determine a sample size suitable for an error tolerance of 5%. With an initial sample size of 10^4 when d=12, we obtained a sample size of 87,281 and a sample size of 92,609 when d = 52. These numbers
will vary slightly each time we run the code as we are generating random instances.

Once our sample size variable was found, we then needed to generate sample paths for the interest rate in the Jackpot model using a recursive function. After we generated random sample paths for the interest rates, we used these interest rates to determine the sample paths for the asset prices using a stochastic SDE recursive model. Those asset prices were then used to calculate the payoff, which is the stock price minus the strike price, discounted back to the present. Once we found the discounted payoff for each of the sample paths, we then found the estimated fair price of our option by finding the mean of the *n* sample paths' discounted payoffs. We also found the estimated error (the standard error times the critical value at 99% confidence), so we could have a confidence interval for our estimate.

By running the code in R, we were able to come up with a fair price estimate of approximately 4.57 USD, with an estimated error of ± 0.04. This value was calculated with a value of d=12, or a time step of one month for one year. The next step was to calculate the options fair price with d=52, or a time step of one week for one year. Running the code with the new time step, we obtained a fair price estimate of 4.58 USD. While this value is slightly higher than the estimated price using a time step of 12, it is not significantly higher.

If we were to change and use a different model for the sample paths of our options prices, such as the Geometric Brownian Motion Model, we can obtain the exact value for the option price. This model assumes a constant interest rate of 7% and constant volatility of 13%, and when we run the code, we obtain a result of 2.31 USD for the exact price according to the
Black-Scholes formula. This suggests that assuming constant interest rates and volatility will depress the options price as varying interest rates and volatility can increase the uncertainty and thus price of the call option.

The Jackpot model converges back over time to an interest rate with a mean of 8.6%. The computational time of these can vary as well. When our d variable (the number of times we monitor the asset price) is 12, the computational time is 0.624 seconds. When the d variable was changed to 52, we experienced a computational time of roughly 2.458

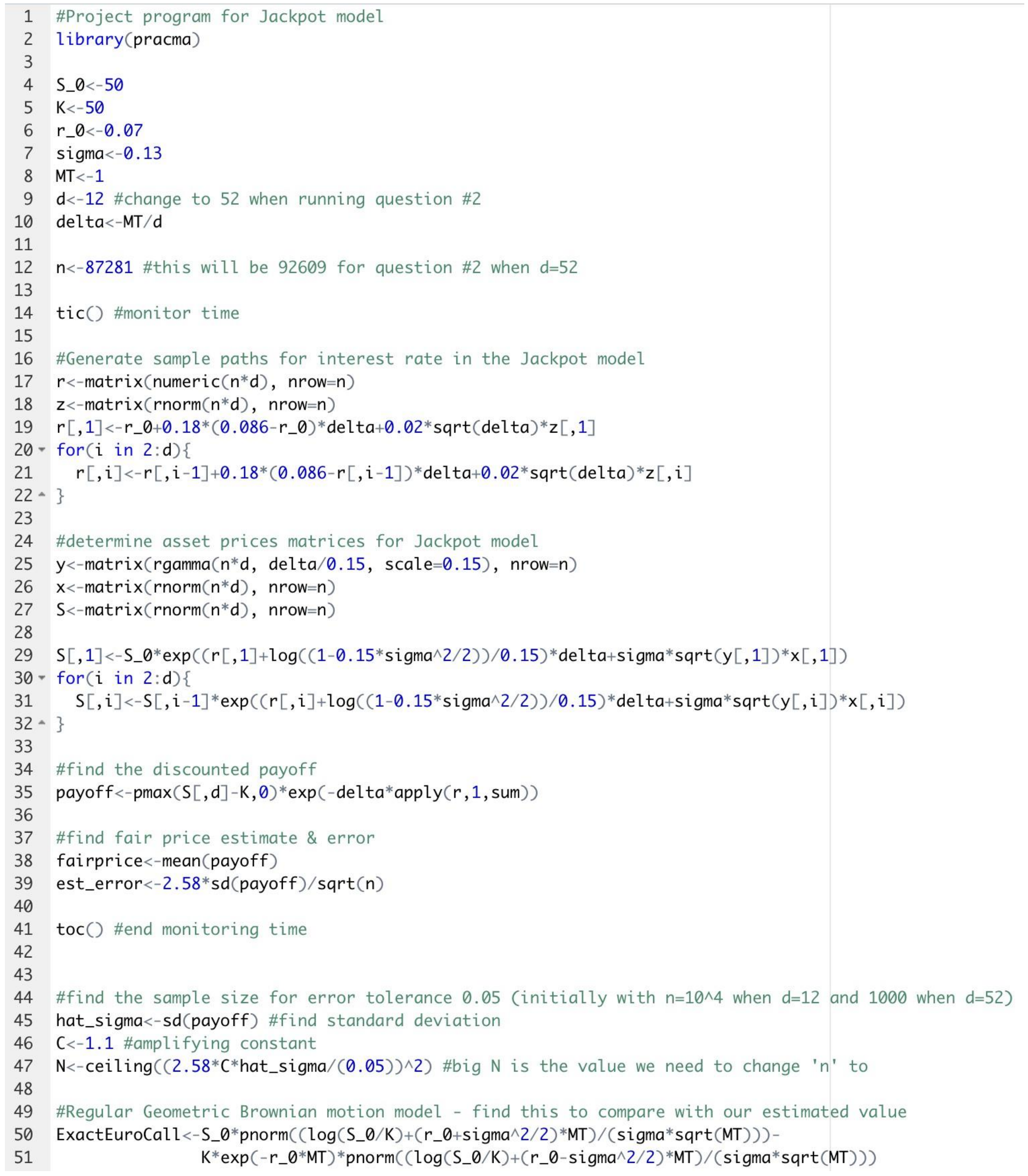
seconds. This is nearly 4 times the amount of time it took when d=12. The main reason for this notable difference is when the code is being run, it accounts for 52 instances instead of only 12. The value 52 is also roughly 4 times greater than 12, which further supports our assumption.

While working through this project, our team found a lot of interesting insights that were worth noting. The first point is the randomness of the Jackpot model. It was higher than the regular Geometric Brownian motion models in this respect: randomness was generated from varying interest rates and the volatility. The Jackpot model's mean interest rate could be around 8.6% which is higher than the GBM constant interest rate of 7%.

Lastly, we found out that the number of time nodes plays a key part in the price of our option. As the Jackpot model does not have an analytical formula to determine each asset price, increasing the number of nodes increases the information from one state (time node) to another. So, the number of time nodes in a recursive model will use more information across the asset sample path to arrive at a better approximation to the "true analytic" asset price used in the payoff function. While the Jackpot model like many others does not have an exact solution, running Monte Carlo simulation will give us an approximation to the "true", theoretical fair price of a European option governed by changing interest rates and volatility.

<u>Appendix A</u>

R Code:

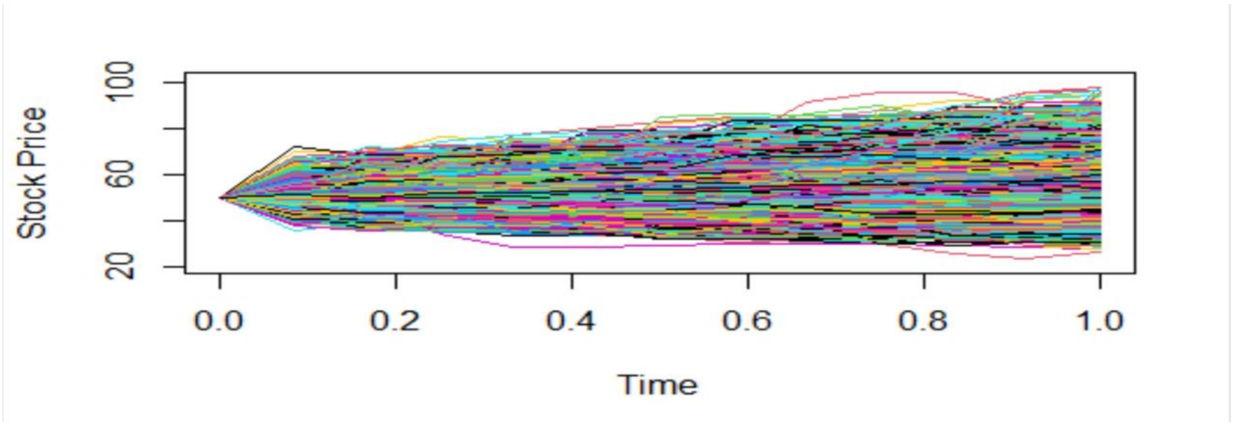


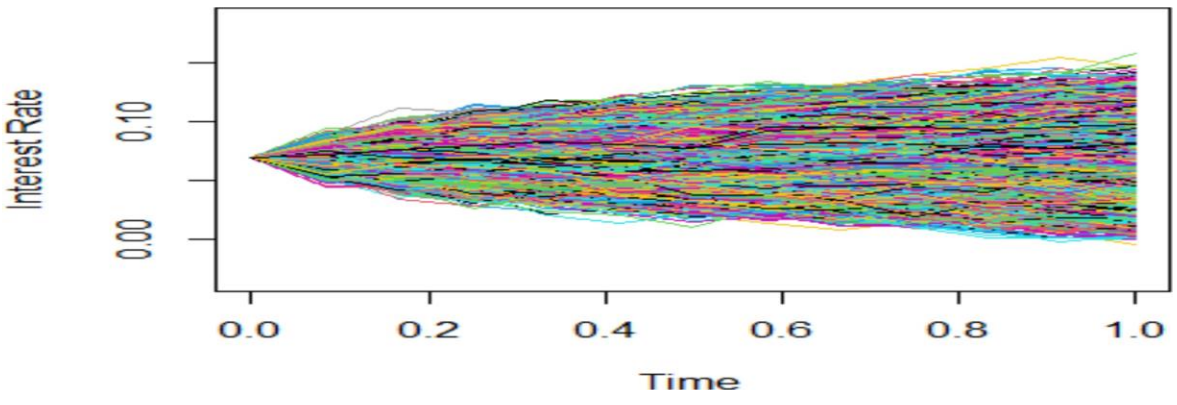
```r
#Project program for Jackpot model
library(pracma)

S_0<-50
K<-50
r_0<-0.07
sigma<-0.13
MT<-1
d<-12 #change to 52 when running question #2
delta<-MT/d

n<-87281 #this will be 92609 for question #2 when d=52

tic() #monitor time

#Generate sample paths for interest rate in the Jackpot model
r<-matrix(numeric(n*d), nrow=n)
z<-matrix(rnorm(n*d), nrow=n)
r[,1]<-r_0+0.18*(0.086-r_0)*delta+0.02*sqrt(delta)*z[,1]
for(i in 2:d){
  r[,i]<-r[,i-1]+0.18*(0.086-r[,i-1])*delta+0.02*sqrt(delta)*z[,i]
}

#determine asset prices matrices for Jackpot model
y<-matrix(rgamma(n*d, delta/0.15, scale=0.15), nrow=n)
x<-matrix(rnorm(n*d), nrow=n)
S<-matrix(rnorm(n*d), nrow=n)

S[,1]<-S_0*exp((r[,1]+log((1-0.15*sigma^2/2))/0.15)*delta+sigma*sqrt(y[,1])*x[,1])
for(i in 2:d){
  S[,i]<-S[,i-1]*exp((r[,i]+log((1-0.15*sigma^2/2))/0.15)*delta+sigma*sqrt(y[,i])*x[,i])
}

#find the discounted payoff
payoff<-pmax(S[,d]-K,0)*exp(-delta*apply(r,1,sum))

#find fair price estimate & error
fairprice<-mean(payoff)
est_error<-2.58*sd(payoff)/sqrt(n)

toc() #end monitoring time


#find the sample size for error tolerance 0.05 (initially with n=10^4 when d=12 and 1000 when d=52)
hat_sigma<-sd(payoff) #find standard deviation
C<-1.1 #amplifying constant
N<-ceiling((2.58*C*hat_sigma/(0.05))^2) #big N is the value we need to change 'n' to

#Regular Geometric Brownian motion model - find this to compare with our estimated value
ExactEuroCall<-S_0*pnorm((log(S_0/K)+(r_0+sigma^2/2)*MT)/(sigma*sqrt(MT)))-
               K*exp(-r_0*MT)*pnorm((log(S_0/K)+(r_0-sigma^2/2)*MT)/(sigma*sqrt(MT)))
```

Appendix B

Code for plotting the sample stock paths (very similar for interest rates):

```
S<-cbind(numeric(n)+S_0, S)
plot((0:d)*delta, S[1,],ylim=c(min(S)-3,max(S)+3), col=1, ylab='Stock Price',
xlab='Time',type='l')

for(i in 2:n){
  points((0:d)*delta,S[i,],type='l',col=i)
}
n=87,281
```


Stock Price
Time


Interest Rate
Time