Option Pricing

Iker Caballero

2023-05-14

Quantitative Finance Assignment

Exercise 1

Implement a discretization for the Black-Scholes-Merton price dynamics for the parameter set (S_0, T, μ, σ) = (12, 1, 0.04, 0.18) with n = 252 steps. Generate M1 = 10, M2 = 100, M3 = 1 000, M4 = 10 000, and M5 = 100 000 paths.

For M1, M2, and M3, plot these paths in three separate figures. Calculate the Monte Carlo estimator for $\hat{S}_T$ separately for M1, . . . , M5. Provide the 5% confidence intervals for each estimator. Compare your findings with the analytical solution for $E(S_T)$ and explain possible differences and how and why the confidence intervals differ.

Fix M^* = 1 000. Let n1 = 12, n2 = 24, n3 = 250, and n4 = 1 000. Calculate the MC estimator and the 5% CI for $\hat{S}_T$ for these different n_i. Discuss if your results differ and if so, explain possible reasons.

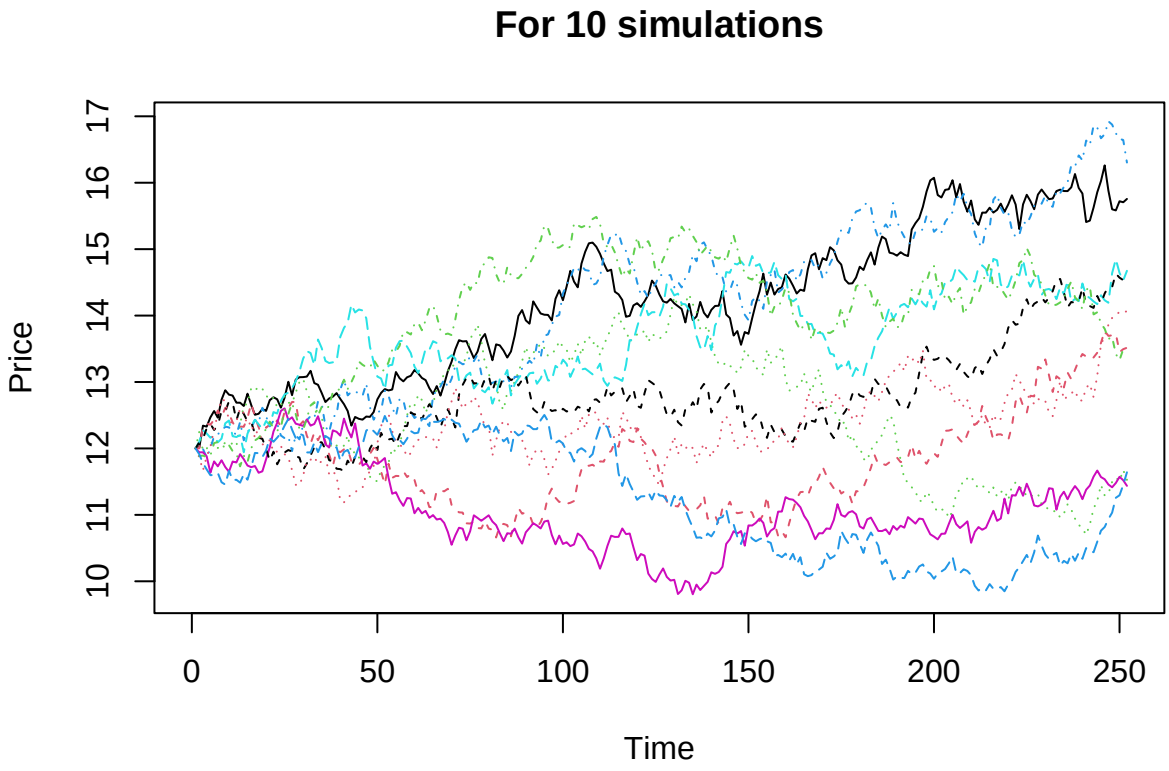
First, we implement the discretization through the Euler-Maruyama algorithm. We create a function called EM_sim for this purpose.

```
### Euler-Maruyama Algorithm ###

# First, we set the random seed

set.seed(02062000)

EM_sim<-function(s0,time,mu,sigma,n,M,graf=F){

  # We then generate a matrix for the whole n points for each
  # of the M realizations

  S=matrix(0,nrow=M,ncol=n)

  h=time/n

  for (i in 1:M){

  # We first generate standard normal random variables

  z=rnorm(n,0,1)
```

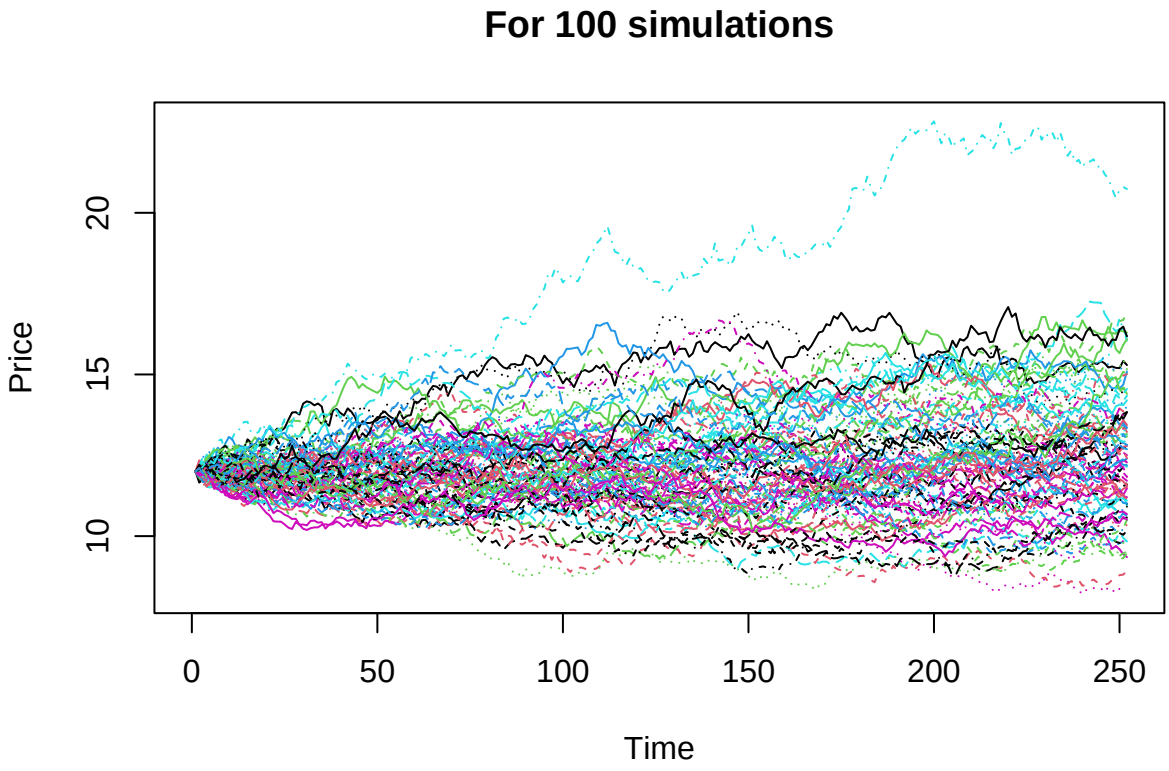
```
  # We impose initial conditions

  S[i,1]=s0

  # We create a loop with the dynamics of the process

  for(j in 2:n){
    S[i,j]=S[i,j-1]+mu*S[i,j-1]*h+sigma*S[i,j-1]*sqrt(h)*z[j]
  }
  }

  # We generate a graph of the prices if desired

  if(graf==T){
    matplot(t(S),xlab="Time",ylab="Price",xlim=c(0,n),type="l",main=paste("For",M,"simulations"))
  }

  # We create a list so we can obtain the objects with a "$" call

  l=list(mat=S)
}
```

From this function, we can obtain the prices simulation by fixing the parameters and the number of simulations.

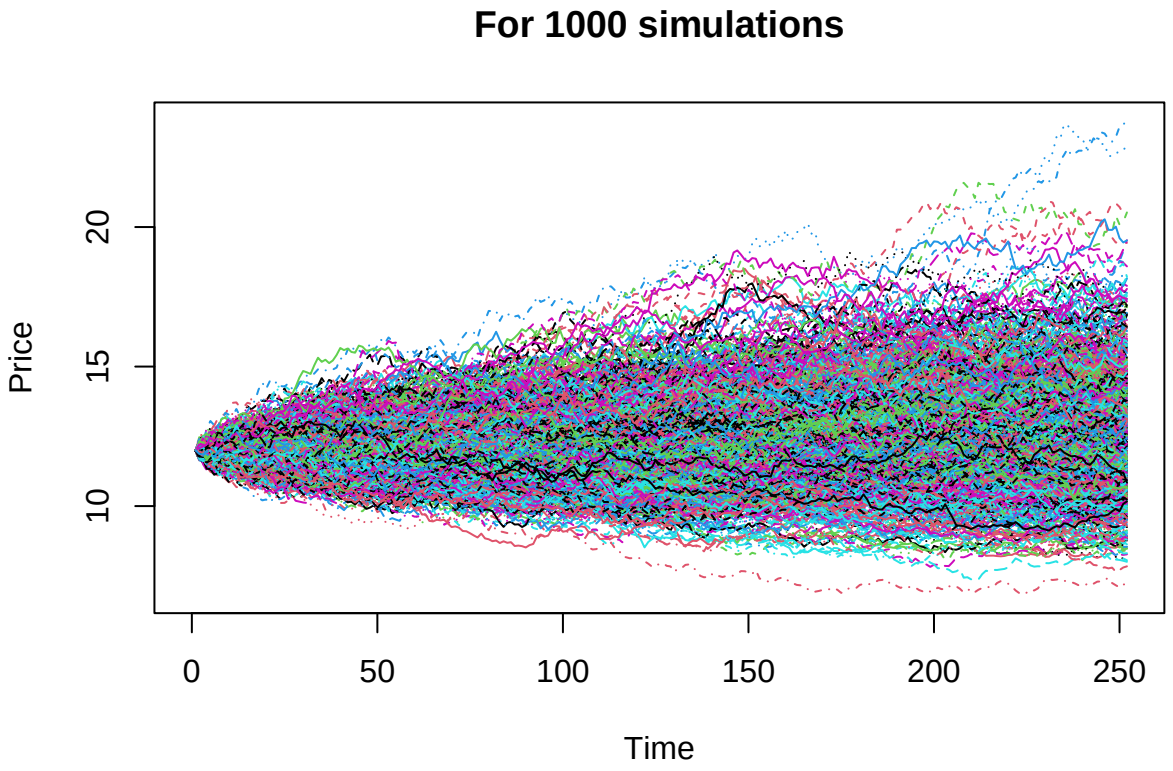
```
### Simulations for each M ###

# These are the number of realizations we will simulate

M1=10;M2=100;M3=1000;M4=10000;M5=100000

# We obtain the matrix of prices for each M and plot the prices for some

mat1=EM_sim(s0=12,time=1,mu=0.04,sigma=0.18,n=252,M1,graf=T)$mat
```




```
mat2=EM_sim(s0=12,time=1,mu=0.04,sigma=0.18,n=252,M2,graf=T)$mat
```




```
mat3=EM_sim(s0=12,time=1,mu=0.04,sigma=0.18,n=252,M3,graf=T)$mat
```




```
mat4=EM_sim(s0=12,time=1,mu=0.04,sigma=0.18,n=252,M4)$mat
mat5=EM_sim(s0=12,time=1,mu=0.04,sigma=0.18,n=252,M5)$mat
```

Using the matrices for the simulated prices, we can obtain the Monte Carlo estimator for S_T and the confidence intervals for a 95% confidence for each one:

```
### Monte Carlo estimator ###

# Because the Monte Carlo estimator is just a mean, we can use the mean function
# with the last column for each matrix, as it contains the final prices

mc_est1 = mean(mat1[,252])
mc_est2 = mean(mat2[,252])
mc_est3 = mean(mat3[,252])

mc_est_vec=c(mc_est1,mc_est2,mc_est3)
names(mc_est_vec)=c("M1","M2","M3")
mc_est_vec
```

```
##       M1       M2       M3
## 13.69018 12.49854 12.57803
```

```
### Monte Carlo intervals ###

# We do the same as before but computing the sd, so we can get the intervals
```

```r
mc_sd1 = sd(mat1[,252])
mc_sd2 = sd(mat2[,252])
mc_sd3 = sd(mat3[,252])

interval<-function(mean,sd,z,M){
  up=round(mean+z*(sd/sqrt(M)),3)
  low=round(mean-z*(sd/sqrt(M)),3)
  int = paste("[",low,", ",up,"]", sep="")
  return(int)
}

int1=interval(mc_est1,mc_sd1,1.96,M1)
int2=interval(mc_est2,mc_sd2,1.96,M2)
int3=interval(mc_est3,mc_sd3,1.96,M3)

mc_int_vec=c(int1,int2,int3)
names(mc_int_vec)=c("M1","M2","M3")
mc_int_vec
```

```
##                 M1                 M2                 M3
## "[12.618, 14.763]" "[12.096, 12.901]" "[12.437, 12.719]"
```

We can compare the findings with the analytical solution. In this case, we can see that the analytical solution would yield:

$$E(S_T) = S_0 e^{\mu T} = 12e^{0.04} = 12.48973$$

We can see that, in this case, the estimator for M1 is quite different from $E(S_T)$, while the estimators for M2 and M3 simulations seem to be quite near. We can also see that the 12.48973 does not pertain to the confidence interval of M1, while it does for the other three. Hence, as the number of realizations increase, the more similar are the estimations with the analytical expected value.

The differences are mostly due to convergence issues. Given that the Monte Carlo methods are supported by the Law of Large Numbers, we need a high number of realizations for the estimators to converge to their theoretical value. For M1 simulations, the law of large numbers does not hold and we can see a notable difference. However, when we tend to ∞, we see that the estimators are nearer the expected value, indicating convergence.

For different values of n, we can get the following results:

```r
### Different n numbers ###

n1=12;n2=24;n3=250;n4=1000;M=10000

matn1=EM_sim(s0=12,time=1,mu=0.04,sigma=0.18,n=n1,M)$mat
matn2=EM_sim(s0=12,time=1,mu=0.04,sigma=0.18,n=n2,M)$mat
matn3=EM_sim(s0=12,time=1,mu=0.04,sigma=0.18,n=n3,M)$mat
matn4=EM_sim(s0=12,time=1,mu=0.04,sigma=0.18,n=n4,M)$mat
```

```r
### Monte Carlo estimator ###

mc_estn1 = mean(matn1[,n1])
```

```
mc_estn2 = mean(matn2[,n2])
mc_estn3 = mean(matn3[,n3])
mc_estn4 = mean(matn4[,n4])

mc_est_vec1=c(mc_estn1,mc_estn2,mc_estn3,mc_estn4)
names(mc_est_vec1)=c("N1","N2","N3","N4")
mc_est_vec1
```

```
##       N1       N2       N3       N4
## 12.44720 12.45755 12.49111 12.46097
```

```
### Monte Carlo intervals ###

mc_sdn1 = sd(matn1[,n1])
mc_sdn2 = sd(matn2[,n2])
mc_sdn3 = sd(matn3[,n3])
mc_sdn4 = sd(matn4[,n4])

int1=interval(mc_estn1,mc_sdn1,1.96,M)
int2=interval(mc_estn2,mc_sdn2,1.96,M)
int3=interval(mc_estn3,mc_sdn3,1.96,M)
int4=interval(mc_estn4,mc_sdn4,1.96,M)

mc_int_vec1=c(int1,int2,int3,int4)
names(mc_int_vec1)=c("N1","N2","N3","N4")
mc_int_vec1
```

```
##                 N1                 N2                 N3                 N4
## "[12.405, 12.489]" "[12.414, 12.501]" "[12.447, 12.536]" "[12.416, 12.505]"
```

In this case, we can see that the estimations are near of the analytical solution, without any clear pattern for the difference in prices (all are near but not exactly). The one that is nearer is the estimation for N4. Moreover, the higher the n, the more accurate is the path (seems more continuous). This is due to the fact that, the shorter the time steps, the more the paths will resemble the analytical results.

Exercise 2

Calculate the price of a Call option with the Black-Scholes-Merton formula for (S_0, T, r, σ, K) = (25, 1.5, 0.04, 0.23, 24.50).

Implement a Monte Carlo estimator for this option price by simulating paths with the Euler-Maruyama method for steps n∈{10,100,1000} and paths M∈{10, 100, 1 000, 10 000, 100 000}. This yields 15 results.

For each combination (n_i,M_j) with i = 1, . . . , 3 and j = 1, . . . , 5, calculate the relative error

$$\varepsilon_{ij}^{rel} = \frac{C^{BSM} - C_{ij}^{MC}}{C^{BSM}}$$

and the absolute error

$$\varepsilon_{ij}^{abs} = |C^{BSM} - C_{ij}^{MC}|$$

where C^{BSM} refers to the price obtained with the closed-form solution and C_{ij}^{MC} denotes the estimated price for the corresponding (n_i, M_j) pair. Tabulate your results. Interpret these errors in view of n and M.

We first compute the prices of a call option using the following function:

```
### BS functions and prices ###

# First, we set the random seed

set.seed(02062000)

# We then construct the function of the call price formula of the BS model

callbs <- function(s0,time,r,sigma,K){

  d1=(log(s0/K)+(r+0.5*sigma**2)*(time))/(sigma*sqrt(time))

  d2=d1-sigma*sqrt(time)

  call = s0*pnorm(d1,0,1)-K*exp(-1*r*(time))*pnorm(d2,0,1)

  return(call)
}

# We finally obtain the desired price analitically

(call_price=callbs(25,1.5,0.04,0.23,24.50))
```

```
## [1] 3.763288
```

Now that we have the theoretical price through the Black-Scholes formula, we can implement a Monte Carlo simulation that allows to simulate with different combinations of n and M:

```
### Monte Carlo Simulation ###

# We create the desired vectors

M=c(10,100,1000,10000,100000)
N=c(10,100,1000)

# We create a matrix to storage prices and dummy variables needed

mc_estim=matrix(0,nrow=3,ncol=5)
a=0;b=1;K=24.5;r=0.04;time=1.5

# We construct a loop for the different prices

for (i in M){
  for(j in N){
     mat=EM_sim(s0=25,time=1.5,mu=0.04,sigma=0.23,j,i)$mat
     a=a+1
```

```
    mc_estim[a,b]=mean(ifelse(mat[,j]>K,mat[,j]-K,0))*exp(-r*time)
  }
  b=b+1
  a=0
}

# We present the results in a matrix

colnames(mc_estim)<- c("M=10","M=100","M=1000","M=10000","M=100000")
rownames(mc_estim)<- c("N=10","N=100","N=1000")
mc_estim
```

```
##            M=10    M=100   M=1000  M=10000 M=100000
## N=10   5.270852 2.248629 3.744521 3.509109 3.522257
## N=100  6.381371 3.576251 3.546606 3.708914 3.723615
## N=1000 5.886572 3.906369 3.634260 3.680409 3.731144
```

Now, for each combination, we can compute the relative error and the absolute error between the Black-Scholes price and the one obtained through Monte Carlo simulation.

```
### Error metrics ###

bs_mat = matrix(call_price,nrow=3,ncol=5)

print("Relative Errors")
```

```
## [1] "Relative Errors"
```

```
(bs_mat-mc_estim)/bs_mat
```

```
##              M=10       M=100      M=1000    M=10000   M=100000
## N=10   -0.4005977  0.40248270 0.004986865 0.06754161 0.06404777
## N=100  -0.6956904  0.04970031 0.057577726 0.01444836 0.01054200
## N=1000 -0.5642101 -0.03802043 0.034285962 0.02202300 0.00854148
```

```
print("Absolute Errors")
```

```
## [1] "Absolute Errors"
```

```
abs(bs_mat-mc_estim)
```

```
##            M=10     M=100     M=1000    M=10000   M=100000
## N=10   1.507564 1.5146582 0.01876701 0.25417852 0.24103018
## N=100  2.618083 0.1870365 0.21668154 0.05437332 0.03967257
## N=1000 2.123285 0.1430818 0.12902793 0.08287887 0.03214405
```

As we can see here, the errors illustrate that, as we increase the number of simulations and we shorten the time steps, the results tend to be more similar to the theoretical Black-Scholes price. With every combination, there is randomness arising from the simulation itself, so there can be combinations which have a higher deviance than normal or, instead, they are nearer than expected, but this can be attributed to the simulation.

To see this, one can choose another seed and understand that the same combination that yields a nearer price can now be very different from the theoretical.

Exercise 3

Implement a Monte Carlo estimator for an Asian Call Option that averages prices every Friday. Use the arithmetic average. Set n = 252. Assume n = 1 is Monday, hence average prices for $t_5, t_10, t_15, ...$ Set M = 100000. Use the parameter set (S_0, T, r, σ, K) = (20, 1, 0.02, 0.24, 20).

Implement a Monte Carlo estimator for a Lookback option with payoff profile $(S_max - K)$ where Smax refers to the maximum price during the time to maturity of the option with K = 21. Use the parameter set (S_0, T, r, σ, K) = (22, 2, 0.02, 0.29) with n = 1000. Calculate confidence intervals of this option price for M1 = 1 000, M2 = 10 000, and M3 = 100,000 paths. Interpret your confidence intervals. Explain possible differences in your obtained prices.

To implement a Monte Carlo estimator for an Asian Call option, we can use the following function:

```
### Asian Call MC Estimator ###

# First we set the random seed

set.seed(02062000)

# We program the asian call option function

asian_mc<-function(s0i,timei,r,sigmai,K,ni,Mi,freq){

  mat=EM_sim(s0=s0i,time=timei,mu=r,sigma=sigmai,n=ni,M=Mi)$mat

  mean_s=rep(0,Mi)
  payoff=rep(0,Mi)
  x=rep(0,length(seq(freq,ni,by=freq)))

  for(i in 1:Mi){
    a=1
    for(k in seq(freq,ni,by=freq)){
      x[a]=mat[i,k]
      a=a+1
    }
    mean_s[i]=mean(x)
    payoff[i]=ifelse(mean_s[i]>K,mean_s[i]-K,0)*exp(-r*timei)
  }

  asian=mean(payoff)
  return(asian)
}


asian_mc(20,1,0.02,0.24,20,252,100000,5)
```

```
## [1] 1.192292
```

Now, for the lookback option, we can use a similar algorithm:

```
### Lookback Call MC Estimator ###
```

```
lb_mc<-function(s0i,timei,r,sigmai,k,ni,Mi){

  mat=EM_sim(s0=s0i,time=timei,mu=r,sigma=sigmai,n=ni,M=Mi)$mat

  mean_s=rep(0,Mi)
  payoff=rep(0,Mi)
  x=rep(0,Mi)

  for(i in 1:Mi){
    x[i]=max(mat[i,])
    payoff[i]=ifelse(x[i]>k,x[i]-k,0)*exp(-r*timei)
  }

  lb=mean(payoff)
  sd=sd(payoff)

  return(c(lb,sd))
}

Mi1=1000;Mi2=10000;Mi3=100000
M_vec=c(Mi1,Mi2,Mi3)

prices=rep(0,3)
sds=rep(0,3)
int=rep(0,3)

for(i in 1:3){
  simul=lb_mc(22,2,0.02,0.29,21,1000,M_vec[i])
  prices[i]=simul[1]
  sds[i]=simul[2]
  int[i]=interval(prices[i],sds[i],1.96,M[i])
}

names(prices)<-c("M1","M2","M3")
prices
```

```
##       M1       M2       M3
## 9.064012 9.192747 9.149191
```

For the different simulation numbers, we have the different confidence intervals:

```
names(int)<-c("M1","M2","M3")
int
```

```
##                M1                M2                M3
## "[4.237, 13.891]" "[7.633, 10.752]"  "[8.652, 9.646]"
```

The results show how the confidence intervals are longer the less simulations we use. This has a very simple explanation: given that the option takes into account just the maximum price in the path of a realization, when using very few paths, the standard deviation is larger because of the random height of each of these. However, when we use a large number of paths, then the variability is lower because, even though there will be higher and lower paths than the mean path, the others will be close to the latter, and hence the variability decreases.