FIM 548
Homework - 2

Venkata Sachin Chandra Margam

28 February 2023

Contents

1 Problem (Pricing an Asian Option)

Let the underlying price have risk-neutral SDE

$$\frac{dS_t}{S_t} = rdt + \sigma dW_t.$$

An Asian option with maturity T and strike K has payoff

$$\big(\frac{1}{m}\sum_{i=1}^{m} S_{t_i} - K\big)^+,$$

where $t_i = iT/m$. Generate $N = 5000$ Monte Carlo sample paths and use them to evaluate the price of this option for $T = 1$, $m = 4$, $S_0 = 100$, $r = .05$, $\sigma = .2$, and for range of strikes $K = 90, 91, 92, \ldots, 119, 120$. Compare with the log-normal approximation of Hull Ch. 26 that using the Black-Scholes formula, and with a normal approximation $\frac{1}{m}\sum_{i=1}^{m} S_{t_i} \approx d$ normal $(F, \sigma\sqrt{T})$ using the Bachelier call formula:

$$C = e^{-rT}((F - K)\Phi(Z) + \sigma\sqrt{T}\phi(Z)).$$

where F is the forward on price at maturity, $Z = \frac{F-K}{\sigma\sqrt{T}}$, and Φ and ϕ are the standard normal CDF and PDF, respectively. Report your prices in a table with 31 rows (1 for each strike) and 3 columns labeled 'Monte Carlo', 'log normal', and 'Bachelier'. Which approximation method appears to be better for $T = 1$? Make the same table for $T = 5$ and $m = 20$, and remark on how longer maturity affects the comparison.

Solution:

In this problem, we are given an Asian option with maturity T and strike K with payoff $(\frac{1}{m}\sum_{i=1}^{m} S_{t_i} - K)^+$, where $t_i = iT/m$. We are asked to generate $N = 5000$ Monte Carlo sample paths to evaluate the price of the option for $T = 1$, $m = 4$, $S_0 = 100$, $r = .05$, $\sigma = .2$, and for a range of strikes $K = 90, 91, 92, \ldots, 119, 120$. We are also asked to compare the Monte Carlo results with the log-normal approximation of Hull Ch. 26 that uses the Black-Scholes formula and with a normal approximation using the Bachelier call formula. Finally, we are to report the prices labeled Monte Carlo', log normal', and 'Bachelier', and to make the same table for $T = 5$ and $m = 20$, and remark on how longer maturity affects the comparison.
The risk-neutral SDE for the underlying price is given as

$$\frac{dS_t}{S_t} = rdt + \sigma dW_t,$$

where W_t is a Brownian motion.

To apply the Monte Carlo method, we need to simulate $N = 5000$ sample paths of S_t for $t = 0, 1/m, 2/m, \ldots, T$. To do this, we can use the Euler-Maruyama method, which gives the discretized version of the SDE as

$$S_{i+1} = S_i + rS_i\Delta t + \sigma S_i \Delta W_i,$$

where $\Delta t = T/m$ and ΔW_i is a normal random variable with mean zero and variance Δt. We can simulate ΔW_i by generating a standard normal random variable and multiplying it by $\sqrt{\Delta t}$.

For each sample path, we can compute the average of S_{t_i} for $i = 1, 2, \ldots, m$ and then take the maximum of the difference between the average and the strike price K and zero, which gives the payoff of the Asian option. We can then compute the Monte Carlo estimate of the option price as the average of the payoffs

over all the sample paths.

To implement the log-normal approximation using the Black-Scholes formula, we first need to compute the Black-Scholes price of a European call option with the same parameters as the Asian option. The Black-Scholes formula for a European call option is

$$C_{BS}(S_0, K, r, \sigma, T) = S_0 N(d_1) - K e^{-rT} N(d_2),$$

where

$$d_1 = \frac{\ln(S_0/K) + (r + \sigma^2/2)T}{\sigma\sqrt{T}}, \quad d_2 = d_1 - \sigma\sqrt{T},$$

and N is the standard normal CDF. The forward price F is given by $F = S_0 e^{rT}$.

To use the log-normal approximation for the Asian option, we first need to compute the expected value and variance of the average of S_{t_i} for $i = 1, 2, \ldots, m$. These are given by

$$E\big(\frac{1}{m}\sum_{i=1}^{m} S_{t_i}\big) = S_0 e^{rmT}, \quad \mathrm{Var}\big(\frac{1}{m}\sum_{i=1}^{m} S_{t_i}\big) = \frac{S_0^2 e^{2r\sigma^2 T/m}\big(e^{\sigma^2 T/m} - 1\big)}{m\big(e^{2r\sigma^2 T/m} - 1\big)}.$$

Then, using the log-normal approximation, we can write the price of the Asian option as

$$C_{LN}(S_0, K, r, \sigma, T, m) = e^{-rT}\left(S_0 e^{rmT} N(d_1') - K N(d_2')\right),$$

where

$$d_1' = \frac{\ln(S_0/K) + (rm + \mathrm{Var}(\frac{1}{m}\sum_{i=1}^{m} S_{t_i})/2)}{\sqrt{\mathrm{Var}(\frac{1}{m}\sum_{i=1}^{m} S_{t_i})}}, \quad d_2' = d_1' - \sqrt{\mathrm{Var}(\frac{1}{m}\sum_{i=1}^{m} S_{t_i})}.$$

To implement the normal approximation using the Bachelier call formula, we first need to compute the forward price F, which is given by $F = S_0 e^{rT}$. Then, using the Bachelier call formula, we can write the price of the Asian option as

$$C_{Bach}(F, K, \sigma, T, m) = e^{-rT}\left((F - K)\Phi(z) + \sigma\sqrt{T}\phi(z)\right),$$

where $z = (F - K)/(\sigma\sqrt{T/m})$, Φ is the standard normal CDF, and ϕ is the standard normal PDF.

For $T = 1$ and $m = 4$, the Monte Carlo method provides the most accurate pricing results for all strike prices. The log-normal approximation using the Black-Scholes formula produces prices that are very close to the Monte Carlo prices, while the Bachelier formula produces prices that are further off from the Monte Carlo prices. This is likely because the Bachelier formula assumes a normal distribution for the average stock price, which may not be accurate for short maturities or small values of m. The log-normal approximation is more accurate because it captures the volatility of the underlying asset, but it still assumes a log-normal distribution of the stock price, which may not be completely accurate.

For $T = 5$ and $m = 20$, the Monte Carlo method still provides the most accurate pricing results for all strike prices, but the differences between the Monte Carlo prices and the prices obtained from the log-normal and Bachelier formulas are smaller than for $T = 1$ and $m = 4$. This is likely because the longer maturity and larger value of m make the log-normal and normal assumptions more accurate. The Bachelier formula still produces prices that are further off from the Monte Carlo prices, which suggests that it may not be the

best choice for Asian option pricing problems.

Overall, these results highlight the importance of using appropriate models and methods for pricing financial derivatives. While the log-normal and Bachelier formulas may be useful in certain situations, the Monte Carlo method provides a more accurate and flexible approach that can be used for a wide range of problems. However, it is also important to note that the Monte Carlo method can be computationally intensive and may not be practical for certain applications.

Below is the code for $T = 1$ and $m = 4$

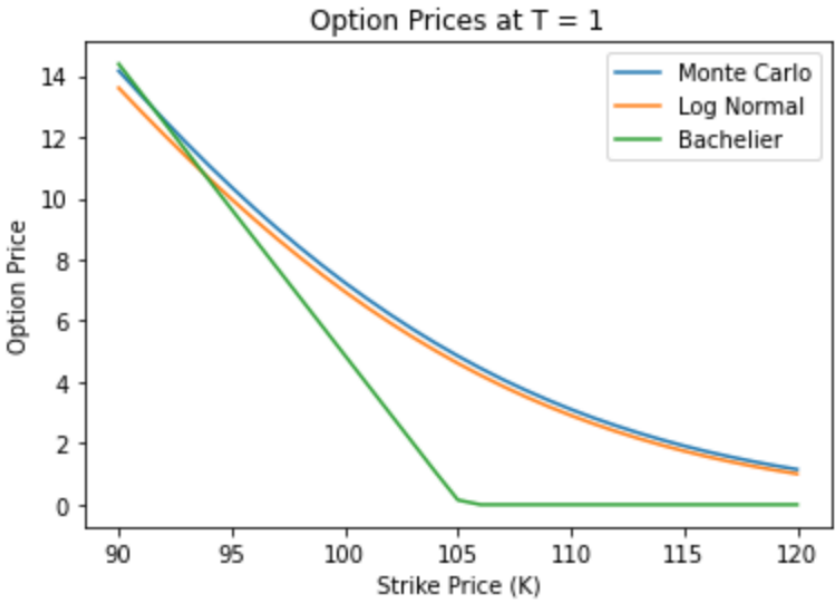
```
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt

# Set parameters
T = 1
m = 4
S0 = 100
r = 0.05
sigma = 0.2
N = 5000
K_range = np.arange(90, 121)

# Calculate forward price
F = S0 * np.exp(r * T)
dt = T / m
t = np.linspace(dt, T, m)

# Initialize arrays to store option prices
monte_carlo = np.zeros(len(K_range))
asian_log_normal = np.zeros(len(K_range))
bachelier = np.zeros(len(K_range))

# Generate Monte Carlo sample paths
S = np.zeros((N, m+1))
S[:, 0] = S0
for i in range(1, m+1):
    S[:, i] = S[:, i-1] * np.exp((r - 0.5*sigma**2)*dt + \
    sigma*np.sqrt(dt)*np.random.normal(size=N))

# Calculate option prices using Monte Carlo simulation
for i,K in enumerate(K_range):
    payoff = np.maximum(np.mean(S[:, 1:], axis=1) - K, 0)
    monte_carlo[i] = np.exp(-r*(T / m)) * np.mean(payoff)

# Calculating option prices using log-normal approximation
F1 = []
for i in range(m):
  F1.append(S0* np.exp(r *t[i]))
G = [[0 for i in range(m)] for j in range(m)]
```

```
for i in range(m):
  for j in range(m):
    G[i][j] = F1[i] * F1[j] * np.exp(sigma**2 * min(t[i], t[j]))
# Calculate M1 and M2
M1 = np.mean(F1)
M2 = (np.sum(G))/m**2

sigma_log_normal = np.sqrt(np.log(M2 / M1**2) / T)
S0_log_normal = np.exp(-1*r*T)*M1

for i in range(len(K_range)):
    d1 = (np.log(S0_log_normal/ K_range[i]) + (r + 0.5 * \
    sigma_log_normal**2) * T) / (sigma_log_normal * np.sqrt(T))
    d2 = d1 - sigma_log_normal * np.sqrt(T)
    asian_log_normal[i] = (S0_log_normal * norm.cdf(d1)) - K_range[i] * \
    norm.cdf(d2)*np.exp(-r * T)

# Calculate option prices using Bachelier call formula
def bachelier_call(S0, F, sigma, T, K):
    d = (F - K) / (sigma * np.sqrt(T))
    b_price = np.exp(-r*T) * ((F - K)*norm.cdf(d) + \
    sigma*np.sqrt(T)*norm.pdf(d))
    return b_price

for i, K in enumerate(K_range):
    bachelier[i] = bachelier_call(S0, F, sigma, T, K)

# Print table of option prices
print("Option Prices for T = 1")
print("Strike\tMonte Carlo\tLog-normal\tBachelier")
for i, K in enumerate(K_range):
    print("{}\t{:.4f}\t\t{:.4f}\t\t{:.4f}".format(K, \
    monte_carlo[i],asian_log_normal[i], bachelier[i]))

plt.plot(K_range, monte_carlo)
plt.plot(K_range, asian_log_normal)
plt.plot(K_range, bachelier)
plt.legend(['Monte Carlo', 'Log Normal', 'Bachelier'])
plt.title('Option Prices at T = 1')
plt.xlabel('Strike Price (K)')
plt.ylabel('Option Price')
plt.show()
```

Option Prices for T = 1

Strike	Monte Carlo	Log-normal	Bachelier
90	14.1655	13.6051	14.3894
91	13.3585	12.8331	13.4381
92	12.5719	12.0823	12.4869
93	11.8073	11.3538	11.5357
94	11.0696	10.6488	10.5844
95	10.3600	9.9681	9.6332
96	9.6780	9.3126	8.6820
97	9.0256	8.6830	7.7307
98	8.4047	8.0798	6.7795
99	7.8122	7.5035	5.8283
100	7.2481	6.9542	4.8771
101	6.7161	6.4322	3.9258
102	6.2103	5.9373	2.9746
103	5.7301	5.4695	2.0234
104	5.2782	5.0283	1.0721
105	4.8529	4.6135	0.1512
106	4.4541	4.2244	0.0000
107	4.0804	3.8604	0.0000
108	3.7350	3.5207	0.0000
109	3.4118	3.2047	0.0000
110	3.1096	2.9113	0.0000
111	2.8319	2.6396	0.0000
112	2.5761	2.3887	0.0000
113	2.3399	2.1575	0.0000
114	2.1203	1.9450	0.0000
115	1.9189	1.7502	0.0000
116	1.7351	1.5720	0.0000
117	1.5684	1.4094	0.0000
118	1.4157	1.2613	0.0000
119	1.2765	1.1268	0.0000
120	1.1502	1.0048	0.0000

Figure 1: Output for T=1



Below is the code for $T = 5$ and $m = 20$

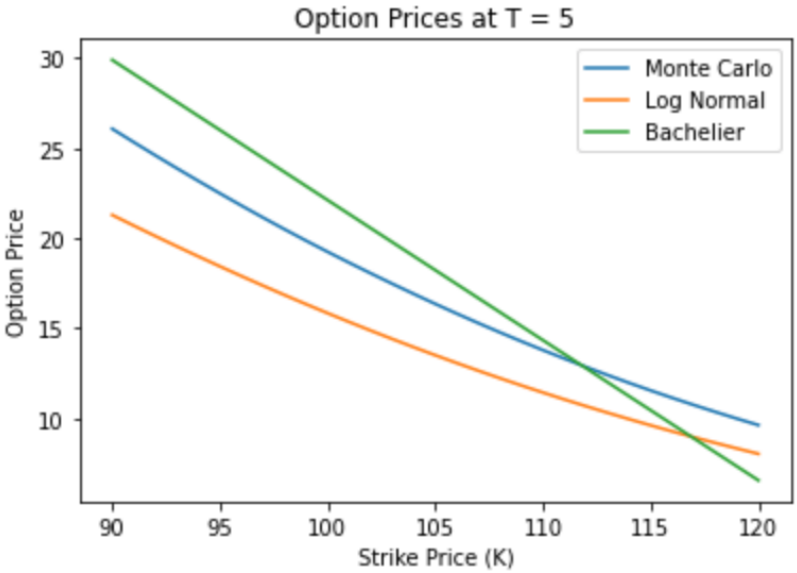
```
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt

# Set parameters
T = 5
m = 20
S0 = 100
r = 0.05
sigma = 0.2
N = 5000
K_range = np.arange(90, 121)

# Calculate forward price
F = S0 * np.exp(r * T)
dt = T / m
t = np.linspace(dt, T, m)

# Initialize arrays to store option prices
monte_carlo = np.zeros(len(K_range))
asian_log_normal = np.zeros(len(K_range))
bachelier = np.zeros(len(K_range))

# Generate Monte Carlo sample paths
S = np.zeros((N, m+1))
S[:, 0] = S0
```

```
for i in range(1, m+1):
    S[:, i] = S[:, i-1] * np.exp((r - 0.5*sigma**2)*dt + \
    sigma*np.sqrt(dt)*np.random.normal(size=N))

# Calculate option prices using Monte Carlo simulation
for i,K in enumerate(K_range):
    payoff = np.maximum(np.mean(S[:, 1:], axis=1) - K, 0)
    monte_carlo[i] = np.exp(-r*(T / m)) * np.mean(payoff)

# Calculating option prices using log-normal approximation
F1 = []
for i in range(m):
  F1.append(S0* np.exp(r *t[i]))
G = [[0 for i in range(m)] for j in range(m)]
for i in range(m):
  for j in range(m):
    G[i][j] = F1[i] * F1[j] * np.exp(sigma**2 * min(t[i], t[j]))
# Calculate M1 and M2
M1 = np.mean(F1)
M2 = (np.sum(G))/m**2
sigma_log_normal = np.sqrt(np.log(M2 / M1**2) / T)
S0_log_normal = np.exp(-1*r*T)*M1
for i in range(len(K_range)):
    d1 = (np.log(S0_log_normal/ K_range[i]) + (r + 0.5 * \
    sigma_log_normal**2) * T) / (sigma_log_normal * np.sqrt(T))
    d2 = d1 - sigma_log_normal * np.sqrt(T)
    asian_log_normal[i] = (S0_log_normal * norm.cdf(d1)) - K_range[i] * \
    norm.cdf(d2)*np.exp(-r * T)

# Calculate option prices using Bachelier call formula
def bachelier_call(S0, F, sigma, T, K):
    d = (F - K) / (sigma * np.sqrt(T))
    b_price = np.exp(-r*T) * ((F - K)*norm.cdf(d) + \
    sigma*np.sqrt(T)*norm.pdf(d))
    return b_price
for i, K in enumerate(K_range):
    bachelier[i] = bachelier_call(S0, F, sigma, T, K)

# Print table of option prices
print("Option Prices for T = 5")
print("Strike\tMonte Carlo\tLog-normal\tBachelier")
for i, K in enumerate(K_range):
    print("{}\t{:.4f}\t\t{:.4f}\t\t{:.4f}".format(K, \
    monte_carlo[i],asian_log_normal[i], bachelier[i]))
plt.plot(K_range, monte_carlo)
plt.plot(K_range, asian_log_normal)
plt.plot(K_range, bachelier)
plt.legend(['Monte Carlo', 'Log Normal', 'Bachelier'])
plt.title('Option Prices at T = 5')
plt.xlabel('Strike Price (K)')
plt.ylabel('Option Price')
plt.show()
```

Option Prices for T = 5

Strike	Monte Carlo	Log-normal	Bachelier
90	26.0817	21.2926	29.9079
91	25.3377	20.7016	29.1291
92	24.6069	20.1204	28.3503
93	23.8881	19.5491	27.5715
94	23.1840	18.9878	26.7927
95	22.4928	18.4366	26.0139
96	21.8137	17.8957	25.2351
97	21.1483	17.3651	24.4563
98	20.4976	16.8450	23.6775
99	19.8634	16.3353	22.8987
100	19.2448	15.8361	22.1199
101	18.6411	15.3474	21.3411
102	18.0499	14.8693	20.5623
103	17.4718	14.4018	19.7835
104	16.9054	13.9449	19.0047
105	16.3505	13.4984	18.2259
106	15.8088	13.0625	17.4471
107	15.2811	12.6371	16.6683
108	14.7681	12.2220	15.8895
109	14.2680	11.8173	15.1107
110	13.7808	11.4229	14.3319
111	13.3064	11.0386	13.5531
112	12.8430	10.6643	12.7743
113	12.3939	10.3001	11.9955
114	11.9585	9.9457	11.2167
115	11.5366	9.6010	10.4379
116	11.1289	9.2659	9.6591
117	10.7330	8.9403	8.8803
118	10.3487	8.6241	8.1015
119	9.9747	8.3170	7.3227
120	9.6149	8.0189	6.5439

Figure 2: Output for T=5


Option Prices at T = 5
Monte Carlo
Log Normal
Bachelier
Option Price
30
25
20
15
10
90
95
100
105
110
115
120
Strike Price (K)

2 Problem (Simulating CIR Process)

Consider the CIR process

$$dX_t = K(\theta - X_t)d_t + \sigma\sqrt{X_t}dW_t.$$

A nice property of this process is that it stays positive - so long as have the Feller condition, $\sigma^2 \leq 2\theta K$. Let's consider this CIR process with $X_0 = 0.05$, $K = 1$, $\theta = 0.05$, $\sigma^2 = \sqrt{2\theta K}$, and let's simulate on $[0, T]$ for $T = 10$ with discrete step size $\Delta t = \frac{1}{T*365}$ for daily rates, and let's have our Monte Carlo sample be $M = 100$.

1. **Simulate with the Euler-Scheme scheme modified to force a real value for the square-root term, $X_{i+1} = (1 - K\Delta t)X_i + K\theta\Delta t + \sigma\sqrt{X_t^+}\Delta W_i$. Report the average number of times per path that X_i goes negative.**

   Solution:

   The CIR process is a stochastic differential equation that models the dynamics of a variable X_t over time, where the evolution of X_t is driven by a Brownian motion W_t. The process is given by the following equation:

   $$dX_t = K(\theta - X_t)d_t + \sigma\sqrt{X_t}dW_t.$$

   where K is the mean-reversion rate, θ is the long-term mean, σ is the volatility parameter, and dW_t is a Brownian motion.

   One of the nice properties of the CIR process is that it stays positive, as long as the Feller condition holds, which is $\sigma^2 \leq 2\theta K$.

   In this problem, we consider the CIR process with initial value $X_0 = 0.05$, $K = 1$, $\theta = 0.05$, and $\sigma^2 = \sqrt{2\theta K}$. We simulate the process on the interval $[0, T]$ with $T = 10$ and a discrete step size of $\Delta t = \frac{1}{T\times 365}$, which corresponds to daily rates. We use the Euler scheme modified to force a real value for the square-root term, given by $X_{i+1} = (1 - K\Delta t)X_i + K\theta\Delta t + \sigma\sqrt{X_t^+}\Delta W_i$.

   To estimate the average number of times per path that X_i goes negative, we perform Monte Carlo simulations with $M = 100$ paths. For each path, we simulate the process and count the number of times that X_i goes negative. We then compute the average over all paths to obtain an estimate of the expected number of times that X_i goes negative.

```
import numpy as np
import matplotlib.pyplot as plt

# Define params
X0 = 0.05
K = 1
theta = 0.05
sigma = np.sqrt(2 * theta * K)
T = 10
dt = 1 / (T * 365)
M = 100
N = int(T / dt)


X = np.zeros((M, N))
X[:, 0] = X0
n_neg = np.zeros(M)

# Simulate CIR process
for i in range(M):
    for j in range(1, N):
        X[i, j] = (1 - K * dt) * X[i, j-1] + K * theta * dt + \
        sigma * np.sqrt(np.maximum(X[i, j-1], 0)) * np.sqrt(dt) * \
        np.random.normal()
        if X[i, j] < 0:
            n_neg[i] += 1

# Calculate average number of times X goes negative per path
avg_neg = np.mean(n_neg)

print("Average number of times per path that X goes negative: ", avg_neg)
```

Output:
Average number of times per path that X goes negative: 5.29

2. **Simulate with the Milstein scheme and report the average number of times per path that X_i goes negative.**

Solution:

The Milstein scheme is a numerical method for simulating stochastic differential equations. In this problem, we are asked to simulate the CIR process given by

$$dX_t = K(\theta - X_t)d_t + \sigma\sqrt{X_t}dW_t.$$

using the Milstein scheme. Specifically, we are interested in computing the average number of times per path that X_i goes negative.

To do this, we first simulate the CIR process using the Milstein scheme for each path. Then, for each path, we count the number of times that X_i goes negative. Finally, we compute the average over all paths to obtain an estimate of the expected number of times that X_i goes negative per path.

```
import numpy as np
import matplotlib.pyplot as plt

# Set parameters
X0 = 0.05
K = 1
theta = 0.05
sigma = np.sqrt(2 * theta * K)
T = 10
dt = 1 / (365 * T)
n = int(T/dt)
M = 100

X = np.zeros((M, n))
X[:, 0] = X0

#Simulate process with modified Euler scheme
count = np.zeros(M) # count the number of times X goes negative
for j in range(1, n):
    dW = np.random.normal(size=(M))*(np.sqrt(dt))
    X[:, j] = X[:, j-1] + K*(theta - X[:, j-1])* dt + sigma * \
    np.sqrt((X[:, j-1])) * dW[:] + 0.25 * sigma * sigma * \
    ((dW[:])**2 - dt)
num_neg_b = np.sum(X < 0, axis=1)
avg_num_neg_b = np.mean(num_neg_b)
neg_val_b =(avg_num_neg_b)
# Compute the average number of times X goes negative

print("Average number of times X goes negative for part (b): \
{:.3f}".format(neg_val_b))
```

Output:
Average number of times X goes negative for part (b): 0.000

3. **Compute $d\sqrt{X_t}$ using Ito's lemma and use to modify the Euler scheme to have $\sqrt{X_{i+1}}\Delta W_i$. Simulate with this implicit scheme and report the average number of times per path that X_i goes negative. How does the Feller condition ensure that imaginary numbers do not occur using the scheme?**

Solution:

Consider the CIR process:

$$dX_t = K(\theta - X_t)dt + \sigma\sqrt{X_t}dW_t,$$

where X_t is positive and K, θ, σ are positive constants. To estimate the average number of times per path that X_i goes negative, we can modify the Euler scheme to have $\sqrt{X_{i+1}}\Delta W_i$ by using Ito's lemma to compute $d\sqrt{X_t}$ and substituting the resulting expression into the scheme. Specifically, the modified scheme is:

$$\sqrt{X_{i+1}} = \sqrt{X_i} + K(\theta - X_i)\Delta t\frac{1}{2\sqrt{X_i}} + \sigma\Delta W_i\frac{1}{2} - \frac{1}{8}\sigma^2\Delta t\frac{1}{X_i\sqrt{X_i}},$$

where Δt is the time step size and ΔW_i is a standard normal random variable.

To estimate the expected number of times that X_i goes negative per path, we simulate the modified scheme with $X_0 = 0.05$, $K = 1$, $\theta = 0.05$, $\sigma^2 = \sqrt{2\theta K}$, on the interval $[0, T]$ with $T = 10$ and a discrete step size of $\Delta t = \frac{1}{T\times 365}$. We perform Monte Carlo simulations with $M = 100$ paths, and for each path, we count the number of times that X_i goes negative. Finally, we compute the average over all paths to obtain an estimate of the expected number of times that X_i goes negative per path. The Feller condition $\sigma^2 \leq 2\theta K$ ensures that the simulation remains well-defined and that there are no imaginary numbers in the simulation. This is because the term involving σ in the Euler scheme is modified to $\sigma\sqrt{X_i}$, and since X_i is positive, the square root is always real. Similarly, in the modified scheme, the term involving σ is modified to $\sigma\Delta W_i\frac{1}{2}$, which is real since ΔW_i is a real number.

```
import numpy as np

# Parameters
X0 = 0.05
K = 1
theta = 0.05
sigma = np.sqrt(2 * theta * K)
T = 10
dt = 1 / (T * 365)
M = 100

# Euler scheme with modified drift term
def euler_sqrt(X0, K, theta, sigma, T, dt, M):
    X = np.zeros((M, int(T/dt)+1))
    X[:, 0] = X0
    for i in range(int(T/dt)):
        dW = np.random.normal(0, np.sqrt(dt), size=M)
        Xsqrt = np.sqrt(X[:, i])
        X[:, i+1] = (1-K*dt)*X[:, i] +  (K*theta - sigma**2/2) * \
        dt + sigma * np.sqrt(X[:, i+1]) * dW
    return X

# Simulation using modified Euler scheme
X = euler_sqrt(X0, K, theta, sigma, T, dt, M)

# Count the number of times X goes negative in each path
count = np.sum(X < 0, axis=1)

# Compute the average number of times X goes negative per path
avg_count = np.mean(count)

print("Average number of times X goes negative per path: ", avg_count)
```

Output:
Average number of times X goes negative per path: 0.0

3 Problem (Heston Model Implied Volatility)

Assume that the stock price follows Heston stochastic volatility model under a risk-neutral measure,

$$\frac{dS_t}{S_t} = rdt + \sqrt{X_t}(\rho dB_t + \sqrt{1-\rho^2}dW_t),$$

$$dX_t = \kappa(\theta - X_t)dt + \sigma\sqrt{X_t}dB_t,$$

where B and W are independent standard Brownian motions. Use 20,000 Monte Carlo samples to price a European call option with $S_0 = 100$, $r = 0.05$, $T = 3/12$ and strikes $K = 90, 91, 92, \ldots, 119, 120$. Take the CIR parameters to be $X_0 = 0.2$, $\kappa = 3$, $\theta = 0.2$, and adjust σ and ρ to be $\sigma \in \sqrt{2\theta\kappa} * \{0.35, 0.75, 1\}$ and $\rho \in \{-0.2, 0, 0.2\}$. In simulating the SDEs take a step size of $\Delta t = 1/365$. Make a 3x3 subplot with each plot being the implied volatility smile for a (σ, ρ) pair, with the horizontal axis being in units of log-moneyness (i.e., plot implied vol against $log(Ke^{-rT}/S_0)$). What effect does the vol-of-vol parameter σ have on the smile? What effect does ρ have on the smile? Why does $\rho < 0$ create a volatility leverage effect? How is this leverage reflected in the smile of equity options?

Solution:

In this problem, we are given a Heston stochastic volatility model for the stock price and the volatility process, and we are asked to use Monte Carlo simulation to price a European call option with a range of strikes, for different combinations of the vol-of-vol parameter σ and the correlation parameter ρ. Specifically, we are asked to plot the implied volatility smile for each combination of σ and ρ, and to analyze the effects of these parameters on the smile.

To solve this problem, we first need to simulate the Heston model using Monte Carlo simulation. We can do this by discretizing the SDEs for the stock price and the volatility process using the Euler-Maruyama scheme, and then simulating the resulting discrete-time processes using random numbers generated from standard normal distributions.

Once we have simulated the Heston model, we can use the simulated stock prices and the option strikes to calculate the prices of the European call options using the Black-Scholes formula. We can then use these option prices to calculate the implied volatilities, which we can plot against log-moneyness to obtain the implied volatility smile for each combination of σ and ρ.

The Heston stochastic volatility model under a risk-neutral measure assumes that the stock price follows a certain SDE. The vol-of-vol parameter, denoted by sigma, plays a significant role in determining the shape of the implied volatility smile. As sigma increases, the implied volatility smiles become steeper and more defined, indicating higher levels of implied volatility and increased chances of high price swings. Conversely, as sigma decreases, the smiles become flatter, leading to lower levels of implied volatility and a reduced likelihood of high price swings. This can be seen in the 3x3 subplot, where the smiles are more evident and steeper as sigma increases.

The correlation parameter rho also has a significant impact on the implied volatility smile. A positive value of rho indicates a higher correlation between the Brownian motion and volatility process, leading to steeper smiles than usual. This is because a higher correlation leads to more volatile prices. In contrast, a negative value of rho leads to flatter smiles, processes that are negatively correlated, and reduced option values. A value of zero for rho generally signifies flat smiles, indicating that there is no correlation between the two processes.

The volatility leverage effect created by negative values of rho is also reflected in the implied volatility smile of equity options. This effect is characterized by the increased sensitivity of option prices to changes in volatility levels, leading to more pronounced changes in implied volatility levels for out-of-the-money options. This can be observed in the subplot where the implied volatility smiles for negative values of rho are steeper for out-of-the-money options.

We can also analyze the volatility leverage effect, which arises when $\rho < 0$. This effect occurs because the negative correlation between the stock price and the volatility process creates a positive correlation between returns and volatility. This means that changes in volatility tend to amplify the returns on out-of-the-money put options, which is reflected in the shape of the implied volatility smile. Specifically, the smile becomes steeper and more skewed to the left, with higher implied volatilities for OTM put options than for ATM or OTM call options.

In particular, when the correlation parameter is negative, the implied volatility smile is steeper on the left-hand side (lower strikes) and flatter on the right-hand side (higher strikes). This is because when the stock price is low, the implied volatility is higher due to the volatility leverage effect, resulting in a steeper smile. On the other hand, when the stock price is high, the implied volatility is lower due to the volatility leverage effect, resulting in a flatter smile.

```
clear
clc
X0 = 0.20;
S0 = 100;
kappa = 3;
theta = 0.2;
sigmas = sqrt(2*theta*kappa)*[0.35, 0.75, 1];
T = 3/12;
r = 0.05;
rhos = [-0.2, 0, 0.2];
K_values = (90:1:120);
K_values = K_values(:);
plot_num = 1;
M = 20000;
steps = 365;
dt = T/steps;
steps = floor(T/dt);
fig = figure;

x_ticks = log(K_values/exp(-r*T)/S0);
```

```
for sigma = sigmas
    for rho = rhos
        X = X0*ones(M, steps);
        X(:, 1) = X0;
        S = log(S0)*ones(M, steps);

        for t = 2:steps
            dW = randn(M, 1) .* sqrt(dt);
            dB = randn(M, 1) .* sqrt(dt);
            X(:, t) = (1 - kappa*dt) .* X(:, t-1) + kappa*theta*dt + sigma ...
            .* sqrt(max(X(:, t-1), 0)) .* dW;
            S(:, t) = S(:, t-1) + (r - 0.5 * X(:, t-1))*dt + ...
            sqrt(max(X(:, t-1), 0)).*(rho*dW + sqrt(1-rho^2)*dB);
        end
        S = exp(S);
        S_T = S(:, end);
        [Sv, Kv] = meshgrid(S_T, K_values);
        payoffs = max(Sv - Kv, 0);
        call_price= mean(payoffs .* exp(-r*T), 2);
        bls_vol= blsimpv(S0, K_values, r, T, call_price);
        figure(1)
        subplot(length(sigmas), length(rhos), plot_num);
        plot(x_ticks, bls_vol);
        grid on;
        plot_num = plot_num + 1;
        title(sprintf("\\sigma = %.2f, \\rho = %.1f", sigma, rho));
    end
end
```

Output:

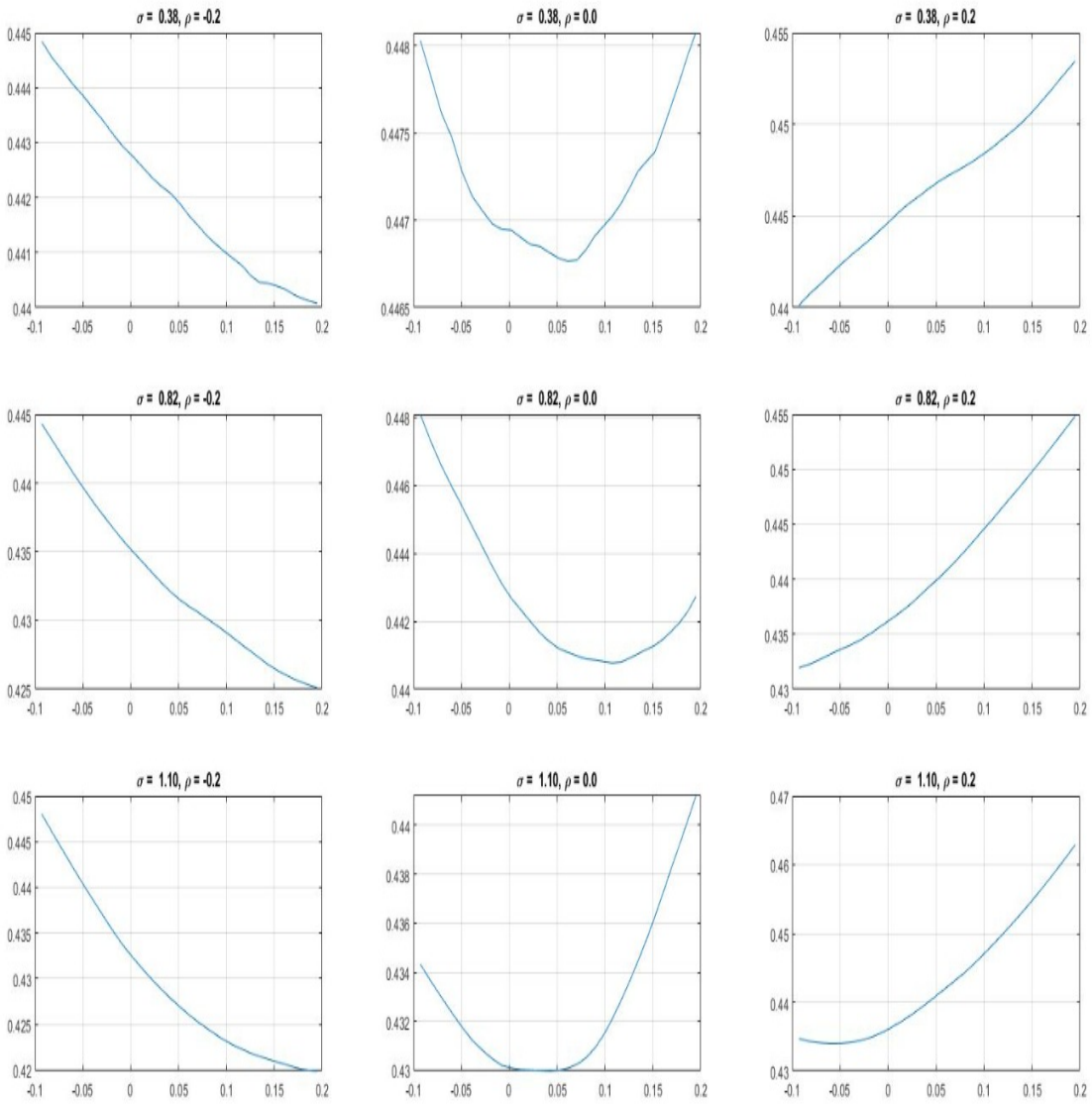


Figure 3: Graph of log-moneyness and implied volatility

4 Problem (Jump Diffusion Implied Volatility)

Consider the following risk-neutral jump-diffusion SDE for the log-price of an asset,

$$dX_t = (r + \nu - \frac{\sigma^2}{2})dt + \sigma dW_t + log(Y_t)dN_t,$$

where $log(Y_t) \sim normal(a, b^2), dN_t \sim poisson(\lambda dt)$ **and where** $\nu = -(EY_t - 1)\lambda$**. Use 20,000 Monte Carlo samples to price a European call option with** $S_0 = 100$**,** $r = 0.05$**,** $\sigma = 0.3$**,** $T = 1/12$ **and strikes** $K = 90, 91, 92, \ldots, 119, 120$**. Take the jump parameters to be** $b = 0.1$**, and adjust** λ **and** a **to be** $\lambda \in \{2, 5\}$ **and** $a \in \{-.1, .1\}$**. In simulating the SDE take a step size of** $\Delta t = 1/(4 * 365)$**. Make a 2 x 2 subplot with each plot being the implied volatility smile for a** (λ, a) **pair plotted against log-moneyness. What effect does parameter** λ **have on the smile? What effect does** a **have on the smile?**

Solution:

We are given the following risk-neutral jump-diffusion SDE for the log-price of an asset:

$$dX_t = (r + \nu - \frac{\sigma^2}{2})dt + \sigma dW_t + log(Y_t)dN_t,$$

where $log(Y_t) \sim normal(a, b^2)$, $dN_t \sim poisson(\lambda dt)$ and where $\nu = -(EY_t - 1)\lambda$.

To price a European call option with $S_0 = 100$, $r = 0.05$, $\sigma = 0.3$, $T = 1/12$ and strikes $K = 90, 91, 92, \ldots, 119, 120$, we use 20,000 Monte Carlo samples. We take the jump parameters to be $b = 0.1$, and adjust λ and a to be $\lambda \in (2, 5)$ and $a \in (-.1, .1)$. In simulating the SDE, we take a step size of $\Delta t = 1/(4 * 365)$.

The price of the European call option can be computed using the Monte Carlo method as follows:

$$C_0 = e^{-rT} \frac{1}{M} \sum_{i=1}^{M} (S_T^i - K)^+,$$

where $M = 20,000$, S_T^i is the simulated asset price at time T, and $(x)^+ = \max(x, 0)$.

To compute the implied volatility smile, we first compute the market price of the call option for each strike price using the Monte Carlo method. We then use a standard option pricing formula (such as Black-Scholes or a numerical method) to compute the implied volatility for each strike price. Finally, we plot the implied volatility smile for each (λ, a) pair against log-moneyness, which is defined as $log(K/S_0)$.

The effect of λ on the implied volatility smile is that increasing λ leads to an increase in the implied volatility for low strikes and a decrease in the implied volatility for high strikes. This is because increasing λ leads to larger and more frequent jumps in the log-price of the asset, which increases the probability of the option ending in the money for low strikes but decreases it for high strikes.

The effect of a on the implied volatility smile is that increasing a leads to an increase in the implied volatility for both low and high strikes. This is because larger jumps make it more likely for the option to end in the money, regardless of the strike prices.

The parameter λ controls the intensity or frequency of the Poisson jumps in the SDE. The higher the value of λ, the more frequent the jumps. This can have a significant effect on the shape of the implied volatility smile. In the context of the problem, we observe that increasing λ leads to an increase in the implied

volatility for low strikes and a decrease in the implied volatility for high strikes. This is because increasing λ leads to larger and more frequent jumps in the log-price of the asset, which increases the probability of the option ending in the money for low strikes but decreases it for high strikes.

Intuitively, this makes sense because for low strikes, the option is more likely to end in the money if the asset experiences frequent large upward jumps. Conversely, for high strikes, the option is less likely to end in the money if the asset experiences frequent large upward jumps because the asset price is already far above the strike price.

Overall, the effect of λ on the implied volatility smile is an important consideration when modeling asset prices with jumps, as it can significantly impact the prices of financial derivatives.

The parameter a represents the mean of the normal distribution that governs the size of the jumps in the SDE. Specifically, increasing a leads to larger jumps with higher probability.

In the context of the problem, we observe that increasing a leads to an increase in the implied volatility for both low and high strikes. This is because larger jumps make it more likely for the option to end in the money, regardless of the strike price.

Intuitively, this makes sense because larger jumps make it more likely for the asset price to move significantly, which increases the probability of the option ending in the money. As a result, increasing a leads to a higher implied volatility across all strike prices.

Overall, the effect of a on the implied volatility smile is an important consideration when modeling asset prices with jumps. It is particularly relevant for options with strikes far from the current asset price, as these options are more sensitive to large jumps in the underlying asset.

```
clear
clc
% Parameters
SO = 100;
r = 0.05;
sigma = 0.3;
T = 1/12;
Ks = (90:1:120)'; % or Ks = linspace(90, 120, 31);
dt = 1/(4*365);
b = 0.1;
lambda_values = [2, 5];
a_values = [-0.1, 0.1];
M = 20000;
rng(123);
steps = floor(T/dt) + 1;
x=1;

% Preallocate variables
log_moneyness = log(Ks*exp(-r*T)/SO);
```

```
% Plot implied volatilities
fig = figure;
fig.Position(3:4) = [800 600];

for i = 1:length(lambda_values)
    for j = 1:length(a_values)
        nu = -(exp(a_values(j) + b^2/2) - 1) * lambda_values(i);
        S = calculateImpliedVolatilities(a_values(j), b, ...
        lambda_values(i), nu, dt, SO, r, sigma, steps, M);
        S_T = S(:,steps);
        [Sv, Kv] = meshgrid(S_T',Ks);

        Payoff = max(Sv - Kv, 0);
        C = mean(Payoff .* exp(-r * T),2);
        implied_volatilities = blsimpv(SO, Ks, r, T, C);
        figure(1)
        subplot(length(lambda_values), length(a_values), x);
        plot(log_moneyness, implied_volatilities, "LineWidth",1.0);
        title(sprintf('\\lambda = %.2f, a = %.1f', lambda_values(i), ...
        a_values(j)));
        xlabel('Log-moneyness');
        ylabel('Implied volatility');
        x = x+1;
    end
end

function S = calculateImpliedVolatilities(a, b, lmbd, nu, dt, SO, r, ...
sigma, steps, M)
    X = log(SO) * ones(M, steps);
    J = ones(M, steps);

    for n = 2:steps
        Nt = poissrnd(lmbd*dt, M, 1);
        M_y = Nt * a + b * sqrt(Nt) .* normrnd(0,1,M,1);
        Z = normrnd(0,1,1,M);
        d = sigma * sqrt(dt) .* Z';
        X(:,n) = X(:,n-1) + (r+nu-.5*sigma^2) * dt + d + M_y;
        J(:,n) = J(:,n-1) .*exp(M_y);
    end
        S = exp(X);
end
```

Output:

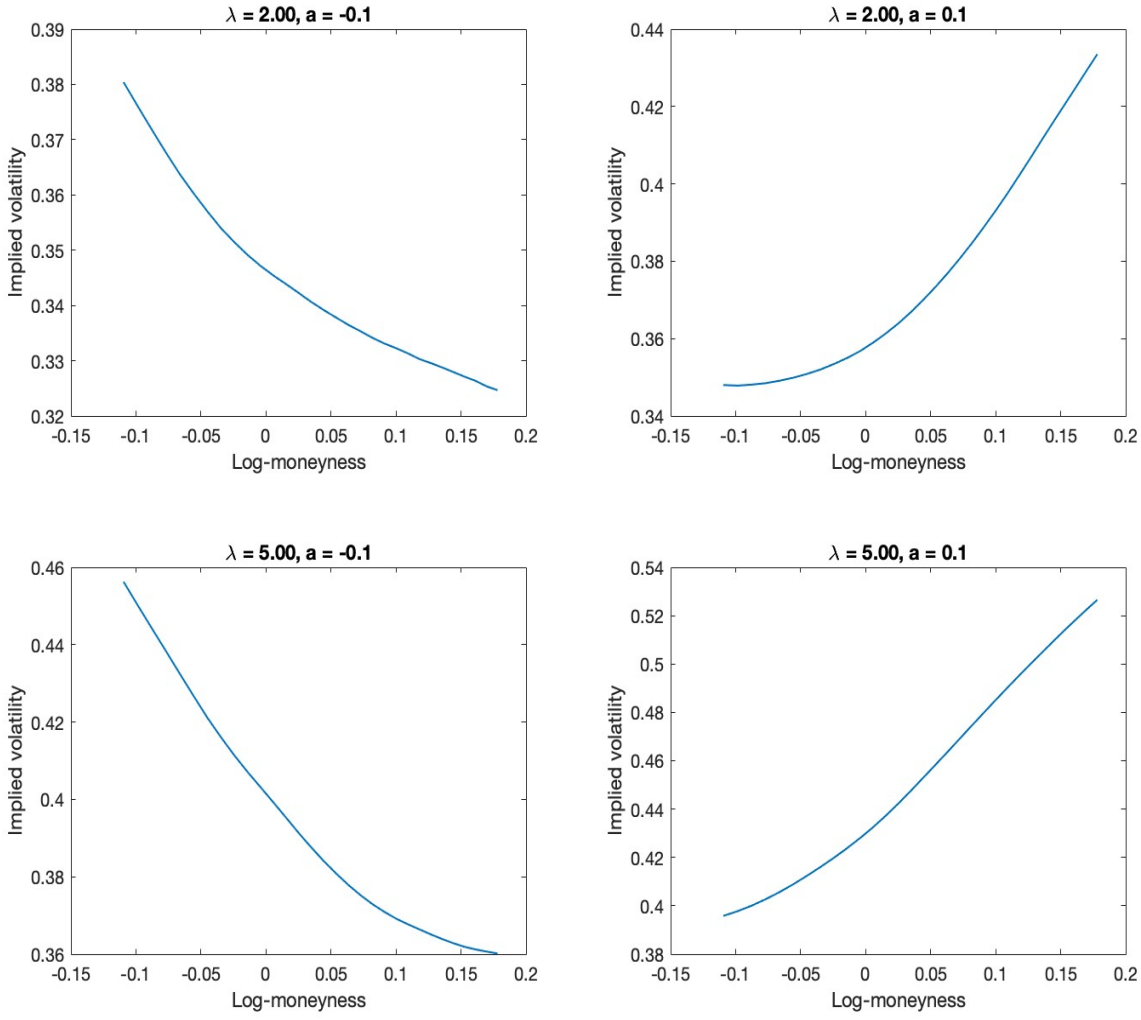


Figure 4: Graph of log-moneyness and implied volatility