GWP 3

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as ss
import pandas as pd
import seaborn as sns
```

Step 1

Stochastic Volatility Modeler

Q5: Heston Model -> Pricing ATM European Call and Put option with correlation value -0.30

```
def SDE_vol(v0, kappa, theta, sigma, T, M, Ite, rand, row,
cho_matrix):
    dt = T / M  # T = maturity, M = number of time steps
    v = np.zeros((M + 1, Ite), dtype=float)
    v[0] = v0
    sdt = np.sqrt(dt)  # Sqrt of dt
    for t in range(1, M + 1):
        ran = np.dot(cho_matrix, rand[:, t])
        v[t] = np.maximum(
            0,
            v[t - 1]
            + kappa * (theta - v[t - 1]) * dt
            + np.sqrt(v[t - 1]) * sigma * ran[row] * sdt,
        )
    return v

def Heston_paths(S0, r, v, row, cho_matrix):
    S = np.zeros((M + 1, Ite), dtype=float)
    S[0] = S0
    sdt = np.sqrt(dt)
    for t in range(1, M + 1, 1):
        ran = np.dot(cho_matrix, rand[:, t])
        S[t] = S[t - 1] * np.exp((r - 0.5 * v[t-1]) * dt +
np.sqrt(v[t-1]) * ran[row] * sdt)

    return S

def random_number_gen(M, Ite):
  np.random.seed(1)
```

```python
    rand = np.random.standard_normal((2, M + 1, Ite))
    return rand

v0 = 0.032
kappa_v = 1.85
sigma_v = 0.35
theta_v = 0.045
rho = -0.30

S0 = 80  # Current underlying asset price
r = 0.055  # Risk-free rate
M0 = 100  # Number of time steps in a year
T = 3/12  # Number of years
M = int(M0 * T)  # Total time steps
Ite = 1000 * 2**9  # Number of simulations
dt = T / M  # Length of time step

# Generating random numbers from standard normal
rand = random_number_gen(M, Ite)


# Covariance Matrix
covariance_matrix = np.zeros((2, 2), dtype=float)
covariance_matrix[0] = [1.0, rho]
covariance_matrix[1] = [rho, 1.0]
cho_matrix = np.linalg.cholesky(covariance_matrix)
print(covariance_matrix)
print(cho_matrix)
```

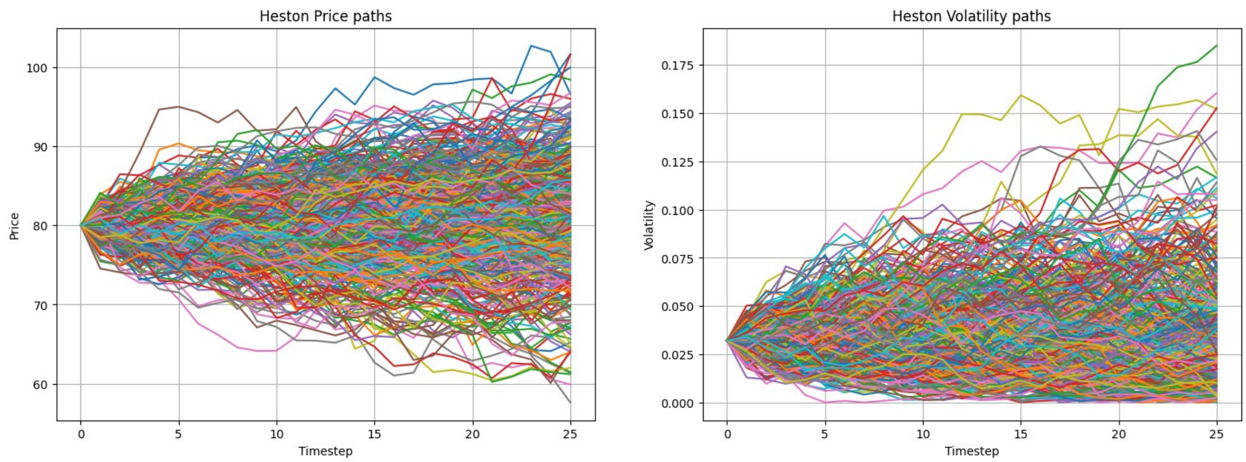
```
[[ 1.  -0.3]
 [-0.3  1. ]]
[[ 1.         0.       ]
 [-0.3        0.9539392]]
```

```python
np.dot(cho_matrix, cho_matrix.T)
```

```
array([[ 1. , -0.3],
       [-0.3,  1. ]])
```

```python
# Volatility process paths
V = SDE_vol(v0, kappa_v, theta_v, sigma_v, T, M, Ite, rand, 1,
cho_matrix)

# Underlying price process paths
S = Heston_paths(S0, r, V, 0, cho_matrix)

def plot_paths(n):
    fig = plt.figure(figsize=(18, 6))
    ax1 = fig.add_subplot(121)
    ax2 = fig.add_subplot(122)
```

```
    ax1.plot(range(len(S)), S[:, :n])
    ax1.grid()
    ax1.set_title("Heston Price paths")
    ax1.set_ylabel("Price")
    ax1.set_xlabel("Timestep")

    ax2.plot(range(len(V)), V[:, :n])
    ax2.grid()
    ax2.set_title("Heston Volatility paths")
    ax2.set_ylabel("Volatility")
    ax2.set_xlabel("Timestep")


plot_paths(500)
```




```
def heston_call_mc(S, K, r, T, t):
    payoff = np.maximum(0, S[-1, :] - K)

    average = np.mean(payoff)

    return np.exp(-r * (T - t)) * average

K = 80
print("European Call Price under Heston: ", np.round(heston_call_mc(S,
K, r, T, 0), 2))

European Call Price under Heston:  3.48

def heston_put_mc(S, K, r, T, t):
    payoff = np.maximum(0, K - S[-1, :])

    average = np.mean(payoff)

    return np.exp(-r * (T - t)) * average
```

```
K = 80
print("European put Price under Heston: ", np.round(heston_put_mc(S,
K, r, T, 0), 2))
```

```
European put Price under Heston:  2.38
```

Delta For European call option with correlation value -0.30

```
epsilon = 0.5

S1 = 80
S2 = 80.5

C1 = np.round(heston_call_mc(S, K, r, T, 0), 2)
print("European Call Price under Heston: ", C1)

C2 = np.round(heston_call_mc(S + epsilon, K, r, T, 0), 2)
print("European Call Price under Heston: ", C2)

print("Delta for Call correlation value -0.30:", np.round((C2 - C1) /
(S2 - S1), 2))
```

```
European Call Price under Heston:  3.48
European Call Price under Heston:  3.76
Delta for Call correlation value -0.30: 0.56
```

```
C1 = np.round(heston_put_mc(S, K, r, T, 0), 2)
print("European put Price under Heston: ", C1)

C2 = np.round(heston_put_mc(S + epsilon, K, r, T, 0), 2)
print("European put Price under Heston: ", C2)

print("Delta for put with correlation value -0.30:", np.round((C2 -
C1) / (S2 - S1), 2))
```

```
European put Price under Heston:  2.38
European put Price under Heston:  2.17
Delta for put with correlation value -0.30: -0.42
```

Gamma For European call option with correlation value -0.30

```
epsilon = 0.5

S1 = 80
S2 = 80.5
S3 = 79.5

C1 = np.round(heston_call_mc(S, K, r, T, 0), 2)
print("European Call Price under Heston: ", C1)

C2 = np.round(heston_call_mc(S + epsilon, K, r, T, 0), 2)
```

```python
print("European Call Price under Heston: ", C2)

C3 = np.round(heston_call_mc(S - epsilon, K, r, T, 0), 2)
print("European Call Price under Heston: ", C3)

print("Gamma for Call correlation value -0.30:", np.round((C2 - 2 * C1
+ C3) / (epsilon**2), 2))
```

```
European Call Price under Heston:  3.48
European Call Price under Heston:  3.76
European Call Price under Heston:  3.2
Gamma for Call correlation value -0.30: 0.0
```

```python
C1 = np.round(heston_put_mc(S, K, r, T, 0), 2)
print("European put Price under Heston: ", C1)

C2 = np.round(heston_put_mc(S + epsilon, K, r, T, 0), 2)
print("European put Price under Heston: ", C2)

C3 = np.round(heston_put_mc(S - epsilon, K, r, T, 0), 2)
print("European put Price under Heston: ", C3)

print("Delta for put with correlation value -0.30:", np.round((C2 - 2
* C1 + C3) / (epsilon**2), 2))
```

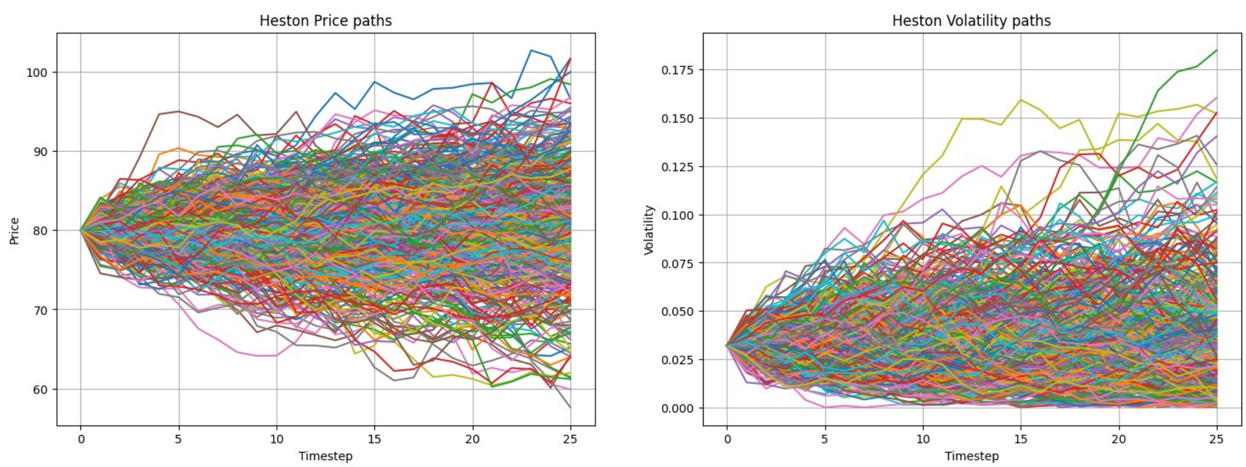
```
European put Price under Heston:  2.38
European put Price under Heston:  2.17
European put Price under Heston:  2.6
Delta for put with correlation value -0.30: 0.04
```

Q6: Heston Model -> Pricing ATM European Call and Put option with correlation value -0.70

```python
def SDE_vol(v0, kappa, theta, sigma, T, M, Ite, rand, row,
cho_matrix):
    dt = T / M  # T = maturity, M = number of time steps
    v = np.zeros((M + 1, Ite), dtype=float)
    v[0] = v0
    sdt = np.sqrt(dt)  # Sqrt of dt
    for t in range(1, M + 1):
        ran = np.dot(cho_matrix, rand[:, t])
        v[t] = np.maximum(
            0,
            v[t - 1]
            + kappa * (theta - v[t - 1]) * dt
            + np.sqrt(v[t - 1]) * sigma * ran[row] * sdt,
        )
    return v
```

```python
def Heston_paths(S0, r, v, row, cho_matrix):
    S = np.zeros((M + 1, Ite), dtype=float)
    S[0] = S0
    sdt = np.sqrt(dt)
    for t in range(1, M + 1, 1):
        ran = np.dot(cho_matrix, rand[:, t])
        S[t] = S[t - 1] * np.exp((r - 0.5 * v[t-1]) * dt +
np.sqrt(v[t-1]) * ran[row] * sdt)

    return S

def random_number_gen(M, Ite):
  np.random.seed(1)
  rand = np.random.standard_normal((2, M + 1, Ite))
  return rand

v0 = 0.032
kappa_v = 1.85
sigma_v = 0.35
theta_v = 0.045
#rho = -0.995
rho = -0.70

S0 = 80  # Current underlying asset price
r = 0.055  # Risk-free rate
M0 = 100  # Number of time steps in a year
T = 3/12  # Number of years
M = int(M0 * T)  # Total time steps
Ite = 1000 * 2**9  # Number of simulations
dt = T / M  # Length of time step

# Generating random numbers from standard normal
rand = random_number_gen(M, Ite)


# Covariance Matrix
covariance_matrix = np.zeros((2, 2), dtype=float)
covariance_matrix[0] = [1.0, rho]
covariance_matrix[1] = [rho, 1.0]
cho_matrix = np.linalg.cholesky(covariance_matrix)
print(covariance_matrix)
print(cho_matrix)

# Volatility process paths
V = SDE_vol(v0, kappa_v, theta_v, sigma_v, T, M, Ite, rand, 1,
cho_matrix)

# Underlying price process paths
S = Heston_paths(S0, r, V, 0, cho_matrix)
```

```
[[ 1.  -0.7]
 [-0.7  1. ]]
[[ 1.          0.        ]
 [-0.7         0.71414284]]

def plot_paths(n):
    fig = plt.figure(figsize=(18, 6))
    ax1 = fig.add_subplot(121)
    ax2 = fig.add_subplot(122)

    ax1.plot(range(len(S)), S[:, :n])
    ax1.grid()
    ax1.set_title("Heston Price paths")
    ax1.set_ylabel("Price")
    ax1.set_xlabel("Timestep")

    ax2.plot(range(len(V)), V[:, :n])
    ax2.grid()
    ax2.set_title("Heston Volatility paths")
    ax2.set_ylabel("Volatility")
    ax2.set_xlabel("Timestep")


plot_paths(500)
```




```
def heston_call_mc_2(S, K, r, T, t):
    payoff = np.maximum(0, S[-1, :] - K)

    average = np.mean(payoff)

    return np.exp(-r * (T - t)) * average

K = 80
call_price = np.round(heston_call_mc_2(S, K, r, T, 0), 2)
print("European Call Price under Heston: ", call_price)
```

```
European Call Price under Heston:  3.48
```

```
def heston_put_mc_2(S, K, r, T, t):
    payoff = np.maximum(0, K - S[-1, :])

    average = np.mean(payoff)

    return np.exp(-r * (T - t)) * average

K = 80
put_price = np.round(heston_put_mc_2(S, K, r, T, 0), 2)
print("European Put Price under Heston: ", put_price)
```

```
European Put Price under Heston:  2.38
```

```
def heston_S_mc_2(S, r):
    payoff = S[-1, :]

    average = np.mean(payoff)

    return np.exp(-r * T) * average

stock = np.round(heston_S_mc_2(S, r))
stock
```

```
80.0
```

Delta For European call option with correlation value -0.70

```
epsilon = 0.5

S1 = 80
S2 = 80.5

C1 = np.round(heston_call_mc_2(S, K, r, T, 0), 2)
print("European Call Price under Heston: ", C1)

C2 = np.round(heston_call_mc_2(S + epsilon, K, r, T, 0), 2)
print("European Call Price under Heston: ", C2)

print("MC Delta for Call correlation value -0.30:", np.round((C2 - C1)
/ (S2 - S1), 2))
```

```
European Call Price under Heston:  3.48
European Call Price under Heston:  3.76
MC Delta for Call correlation value -0.30: 0.56
```

```
C1 = np.round(heston_put_mc_2(S, K, r, T, 0), 2)
print("European put Price under Heston: ", C1)

C2 = np.round(heston_put_mc_2(S + epsilon, K, r, T, 0), 2)
print("European put Price under Heston: ", C2)
```

```
print("MC Delta for put with correlation value -0.30:", np.round((C2 -
C1) / (S2 - S1), 2))
```

```
European put Price under Heston:  2.38
European put Price under Heston:  2.17
MC Delta for put with correlation value -0.30: -0.42
```

Gamma For European call option with correlation value -0.30

```
epsilon = 0.5

S1 = 80
S2 = 80.5
S3 = 79.5

C1 = np.round(heston_call_mc_2(S, K, r, T, 0), 2)
print("European Call Price under Heston: ", C1)

C2 = np.round(heston_call_mc_2(S + epsilon, K, r, T, 0), 2)
print("European Call Price under Heston: ", C2)

C3 = np.round(heston_call_mc_2(S - epsilon, K, r, T, 0), 2)
print("European Call Price under Heston: ", C3)

print("Gamma for Call correlation value -0.30:", np.round((C2 - 2 * C1
+ C3) / (epsilon**2), 2))
```

```
European Call Price under Heston:  3.48
European Call Price under Heston:  3.76
European Call Price under Heston:  3.2
Gamma for Call correlation value -0.30: 0.0
```

```
C1 = np.round(heston_put_mc_2(S, K, r, T, 0), 2)
print("European put Price under Heston: ", C1)

C2 = np.round(heston_put_mc_2(S + epsilon, K, r, T, 0), 2)
print("European put Price under Heston: ", C2)

C3 = np.round(heston_put_mc_2(S - epsilon, K, r, T, 0), 2)
print("European put Price under Heston: ", C3)

print("Delta for put with correlation value -0.30:", np.round((C2 - 2
* C1 + C3) / (epsilon**2), 2))
```

```
European put Price under Heston:  2.38
European put Price under Heston:  2.17
European put Price under Heston:  2.6
Delta for put with correlation value -0.30: 0.04
```

Jump Modeler

Q8: Merton Model -> Pricing ATM European call and put with jump intensity parameter equal to 0.75.

The model has the following SDE:

$$dS_t = (r - r_j) S_t dt + \sigma S_t dZ_t + J_t S_t dN_t$$

with the following discretized form:

$$S_t = S_{t-1}\left(e^{\left(r - r_j - \frac{\sigma^2}{2}\right)dt + \sigma\sqrt{dt}\, z_t^1} + \left(e^{\mu_j + \delta z_t^2} - 1\right) y_t\right)$$

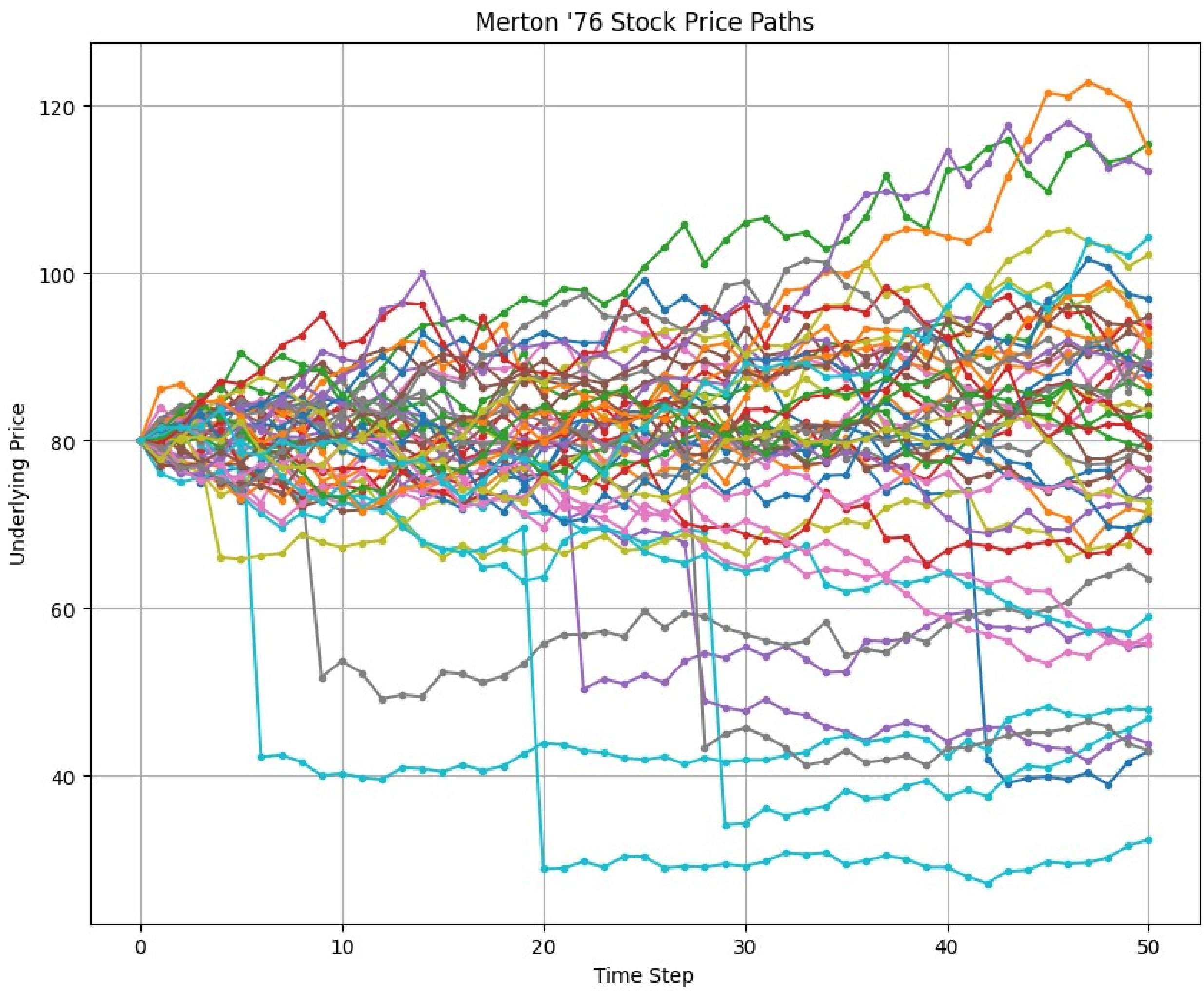
where z_t^1 and z_t^2 follow a standard normal and y_t follows a Poisson process. Finally, r_j equals to:

$$r_j = \lambda\left(e^{\mu_j + \frac{\delta^2}{2}}\right) - 1$$

- S0 = 80
- r = 5.5%
- sigma = 35%
- Time to maturity = 3 months
- $\mu = -0.5$
- $\delta = 0.22$

```
lamb = 0.75  # Lambda of the model

def Merton_paths(S0, M, Ite, lamb, mu, delta, r, sigma, T):
  dt = T/M
  SM = np.zeros((M + 1, Ite))
  SM[0] = S0

  # rj
  rj = lamb * (np.exp(mu + 0.5 * delta**2) - 1)

  # Random numbers
  z1 = np.random.standard_normal((M + 1, Ite))
  z2 = np.random.standard_normal((M + 1, Ite))
  y = np.random.poisson(lamb * dt, (M + 1, Ite))

  for t in range(1, M + 1):
      SM[t] = SM[t - 1] * (
          np.exp((r - rj - 0.5 * sigma**2) * dt + sigma * np.sqrt(dt)
* z1[t])
          + (np.exp(mu + delta * z2[t]) - 1) * y[t]
      )
      SM[t] = np.maximum(SM[t], 0.00001)  # To ensure that the price
never goes below zero!
```

```
  return SM

SM = Merton_paths(S0=80, M=50, Ite=1000, lamb=0.75, mu=-0.5,
delta=0.22, r=0.055, sigma=0.35, T=3/12)

plt.figure(figsize=(10, 8))
plt.plot(SM[:, 100:150], marker='o', markersize=3)
plt.title("Merton '76 Stock Price Paths")
plt.xlabel("Time Step")
plt.ylabel("Underlying Price")
plt.grid()
plt.show()
```




```
def merton_opt_mc(S, K, r, T, t, opttype):
    if opttype == 'C':
      payoff = np.maximum(0, S[-1, :] - K)
    elif opttype == 'P':
```

```python
        payoff = np.maximum(0, K - S[-1, :])
    else:
        payoff = 0
        raise Exception("Wrong option type!")
    average = np.mean(payoff)

    return np.exp(-r * (T - t)) * average

K = 80
r = 0.055
T = 3/12
t = 0
params = [K, r, T, t]
C_eu_merton = np.round(merton_opt_mc(SM, *params, 'C'), 2)
P_eu_merton = np.round(merton_opt_mc(SM, *params, 'P'), 2)
print('Results with small number of simulations')
print("European Call Price: ", C_eu_merton)
print("European Put Price: ", P_eu_merton)
```

```
Results with small number of simulations
European Call Price:  8.12
European Put Price:  7.52
```

```python
255 * 3/12
```

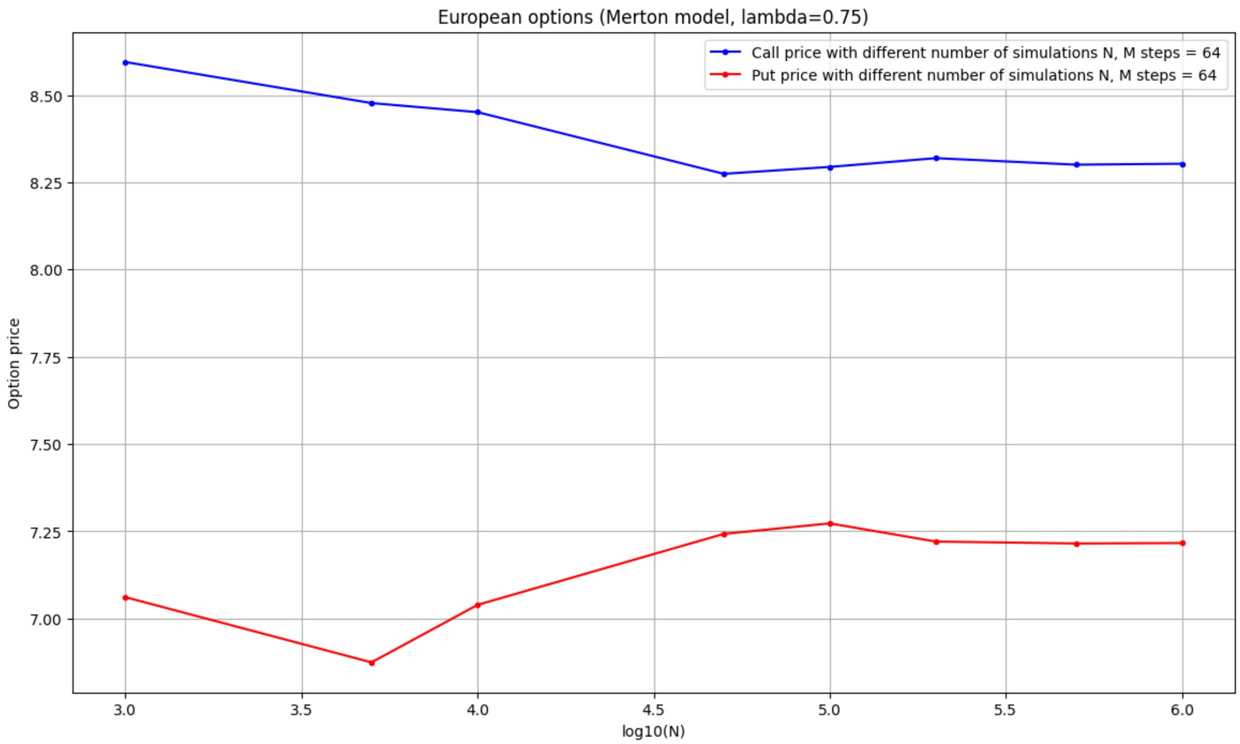
```
63.75
```

```python
M_steps = [50, 64, 130]
N_steps = [1000, 5000, 10000, 50000, 100000]
np.zeros(len(N_steps))[0]
```

```
0.0
```

```python
M_steps = 64 # daily basis
N_steps = [1000, 5000, 10000, 50000, 100000, 200000, 500000, 1000000]

C_eu_merton_prices = np.zeros(len(N_steps))
P_eu_merton_prices = np.zeros(len(N_steps))

for i in range(len(N_steps)):
    np.random.seed(1)
    SM_N = Merton_paths(S0=80, M=M_steps, Ite=N_steps[i], lamb=0.75,
mu=-0.5, delta=0.22, r=0.055, sigma=0.35, T=3/12)
    C_eu_merton_prices[i] = merton_opt_mc(SM_N, *params, 'C')
    P_eu_merton_prices[i] = merton_opt_mc(SM_N, *params, 'P')

plt.figure(figsize=(14,8))

plt.plot(np.log10(N_steps), C_eu_merton_prices, marker='o',
color='blue', markersize=3, label=f'Call price with different number
of simulations N, M steps = {M_steps}')
```

```
plt.plot(np.log10(N_steps), P_eu_merton_prices, marker='o',
color='red', markersize=3, label=f'Put price with different number of
simulations N, M steps = {M_steps}')

plt.title("European options (Merton model, lambda=0.75)")
plt.xlabel('log10(N)')
plt.ylabel('Option price')
plt.legend()
plt.grid()
plt.show()
```

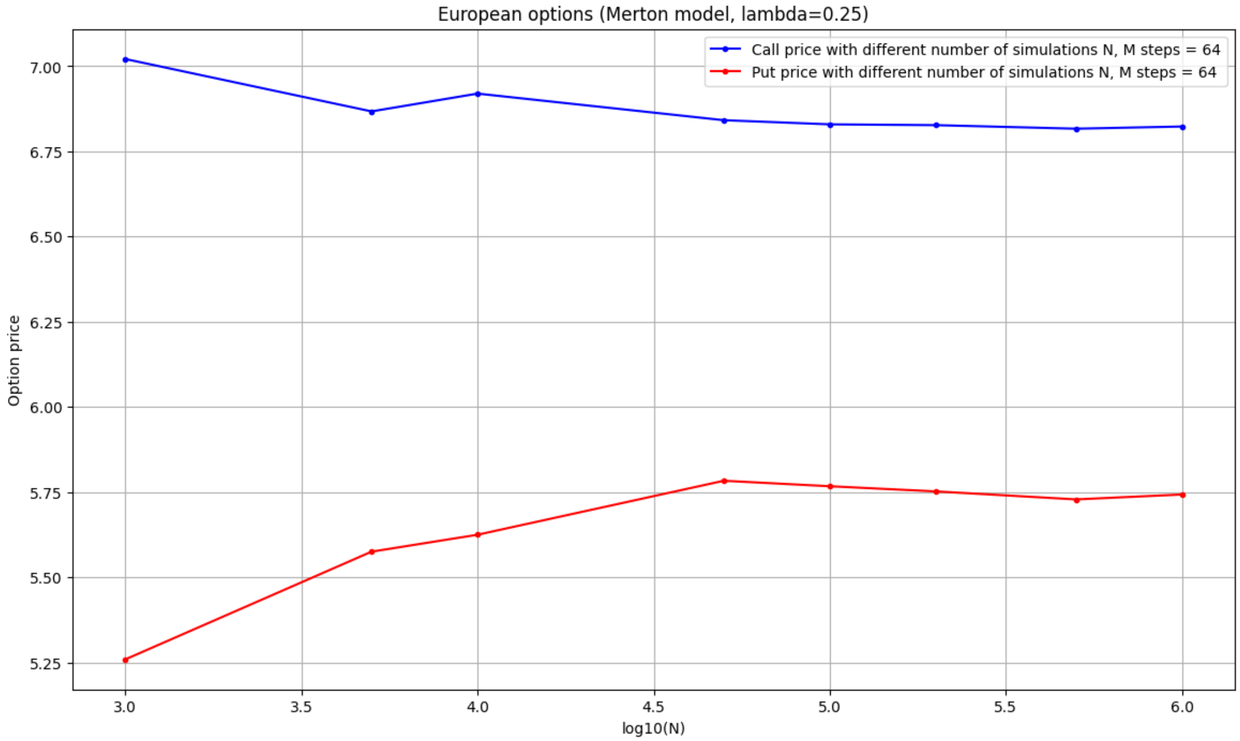



```
C_eu_merton_prices

array([8.59515556, 8.47745196, 8.45144713, 8.27466159, 8.29441276,
       8.31952098, 8.3008759 , 8.30356026])

P_eu_merton_prices

array([7.0609993 , 6.87406327, 7.03885803, 7.24232572, 7.27252841,
       7.22039686, 7.21490858, 7.21612862])

print("European Call Price under Merton: ",
np.round(C_eu_merton_prices[-1], 2))
print("European Put Price under Merton: ",
np.round(P_eu_merton_prices[-1], 2))
```

```
European Call Price under Merton:  8.3
European Put Price under Merton:  7.22
```

Q9: Merton Model -> Pricing ATM European call and put with jump intensity parameter equal to 0.25.

```
M_steps = 64 # daily basis
N_steps = [1000, 5000, 10000, 50000, 100000, 200000, 500000, 1000000]

C_eu_merton_prices = np.zeros(len(N_steps))
P_eu_merton_prices = np.zeros(len(N_steps))

for i in range(len(N_steps)):
    np.random.seed(1)
    SM_N = Merton_paths(S0=80, M=M_steps, Ite=N_steps[i], lamb=0.25,
mu=-0.5, delta=0.22, r=0.055, sigma=0.35, T=3/12)
    C_eu_merton_prices[i] = merton_opt_mc(SM_N, *params, 'C')
    P_eu_merton_prices[i] = merton_opt_mc(SM_N, *params, 'P')

plt.figure(figsize=(14,8))

plt.plot(np.log10(N_steps), C_eu_merton_prices, marker='o',
color='blue', markersize=3, label=f'Call price with different number
of simulations N, M steps = {M_steps}')
plt.plot(np.log10(N_steps), P_eu_merton_prices, marker='o',
color='red', markersize=3, label=f'Put price with different number of
simulations N, M steps = {M_steps}')

plt.title("European options (Merton model, lambda=0.25)")
plt.xlabel('log10(N)')
plt.ylabel('Option price')
plt.legend()
plt.grid()
plt.show()
```




```
C_eu_merton_prices
```

```
array([7.02099548, 6.86730836, 6.91953859, 6.84155206, 6.82935296,
       6.82697003, 6.81658355, 6.82297368])
```

```
P_eu_merton_prices
```

```
array([5.25970923, 5.57568716, 5.62535237, 5.78365611, 5.76759606,
       5.75248999, 5.72898829, 5.74332277])
```

```
print("European Call Price under Merton: ",
np.round(C_eu_merton_prices[-1], 2))
print("European Put Price under Merton: ",
np.round(P_eu_merton_prices[-1], 2))
```

```
European Call Price under Merton:  6.82
European Put Price under Merton:  5.74
```

Q10: Calculate delta and gamma for each of the options in Questions 8 and 9.

Lambda = 0.75:

Deltas and gammas:

```
N = 500000
```

```
eps = 0.8

S1 = 80
S2 = 80 + eps
S3 = 80 - eps

lamb = 0.75
np.random.seed(1)
SM1 = Merton_paths(S0=S1, M=M_steps, Ite=N, lamb=0.75, mu=-0.5,
delta=0.22, r=0.055, sigma=0.35, T=3/12)
C_eu_merton1 = merton_opt_mc(SM1, *params, 'C')
P_eu_merton1 = merton_opt_mc(SM1, *params, 'P')

np.random.seed(1)
SM2 = Merton_paths(S0=S2, M=M_steps, Ite=N, lamb=0.75, mu=-0.5,
delta=0.22, r=0.055, sigma=0.35, T=3/12)
C_eu_merton2 = merton_opt_mc(SM2, *params, 'C')
P_eu_merton2 = merton_opt_mc(SM2, *params, 'P')

np.random.seed(1)
SM3 = Merton_paths(S0=S3, M=M_steps, Ite=N, lamb=0.75, mu=-0.5,
delta=0.22, r=0.055, sigma=0.35, T=3/12)
C_eu_merton3 = merton_opt_mc(SM3, *params, 'C')
P_eu_merton3 = merton_opt_mc(SM3, *params, 'P')

C_delta = (C_eu_merton2 - C_eu_merton1) / (S2 - S1)
P_delta = (P_eu_merton2 - P_eu_merton1) / (S2 - S1)

C_gamma = (C_eu_merton2 - 2 * C_eu_merton1 + C_eu_merton3) / (eps**2)
P_gamma = (P_eu_merton2 - 2 * P_eu_merton1 + P_eu_merton3) / (eps**2)

print('Call delta:', np.round(C_delta, 3))
print('Put delta:', np.round(P_delta, 3))
```

```
Call delta: 0.658
Put delta: -0.342
```

```
print('Call gamma:', np.round(C_gamma, 3))
print('Put gamma:', np.round(P_gamma, 3))
```

```
Call gamma: 0.023
Put gamma: 0.023
```

Lambda = 0.25:

Deltas and gammas:

```
N = 500000

eps = 0.8
```

```
S1 = 80
S2 = 80 + eps
S3 = 80 - eps

lamb = 0.75
np.random.seed(1)
SM1 = Merton_paths(S0=S1, M=M_steps, Ite=N, lamb=0.25, mu=-0.5,
delta=0.22, r=0.055, sigma=0.35, T=3/12)
C_eu_merton1 = merton_opt_mc(SM1, *params, 'C')
P_eu_merton1 = merton_opt_mc(SM1, *params, 'P')

np.random.seed(1)
SM2 = Merton_paths(S0=S2, M=M_steps, Ite=N, lamb=0.25, mu=-0.5,
delta=0.22, r=0.055, sigma=0.35, T=3/12)
C_eu_merton2 = merton_opt_mc(SM2, *params, 'C')
P_eu_merton2 = merton_opt_mc(SM2, *params, 'P')

np.random.seed(1)
SM3 = Merton_paths(S0=S3, M=M_steps, Ite=N, lamb=0.25, mu=-0.5,
delta=0.22, r=0.055, sigma=0.35, T=3/12)
C_eu_merton3 = merton_opt_mc(SM3, *params, 'C')
P_eu_merton3 = merton_opt_mc(SM3, *params, 'P')

C_delta = (C_eu_merton2 - C_eu_merton1) / (S2 - S1)
P_delta = (P_eu_merton2 - P_eu_merton1) / (S2 - S1)

C_gamma = (C_eu_merton2 - 2 * C_eu_merton1 + C_eu_merton3) / (eps**2)
P_gamma = (P_eu_merton2 - 2 * P_eu_merton1 + P_eu_merton3) / (eps**2)

print('Call delta:', np.round(C_delta, 3))
print('Put delta:', np.round(P_delta, 3))
```

```
Call delta: 0.608
Put delta: -0.392
```

```
print('Call gamma:', np.round(C_gamma, 3))
print('Put gamma:', np.round(P_gamma, 3))
```

```
Call gamma: 0.027
Put gamma: 0.027
```

Gammas:

Q12

Heston Model for 7 different strike prices.

```
strike_prices = {
    'Moneyness = 0.85: K1': 68,
```

```
    'Moneyness = 0.90: K2': 72,
    'Moneyness = 0.95: K3': 76,
    'Moneyness = 1:    K4': 80,
    'Moneyness = 1.05: K5': 84,
    'Moneyness = 1.10: K6': 88,
    'Moneyness = 1.15: K7': 92

}

for moneyness, strike in strike_prices.items():
    S = Heston_paths(strike, r, V, 0, cho_matrix)
    call_price = np.round(heston_call_mc(S, K, r, T, 0), 2)
    print(moneyness, ", Call Price is: ", call_price)
```

```
Moneyness = 0.85: K1 , Call Price is:  0.12
Moneyness = 0.90: K2 , Call Price is:  0.5
Moneyness = 0.95: K3 , Call Price is:  1.52
Moneyness = 1:    K4 , Call Price is:  3.48
Moneyness = 1.05: K5 , Call Price is:  6.28
Moneyness = 1.10: K6 , Call Price is:  9.66
Moneyness = 1.15: K7 , Call Price is:  13.36
```

Merton Model for 7 different strike prices.

```
strike_prices = {
    'Moneyness = 0.85: K1': 68,
    'Moneyness = 0.90: K2': 72,
    'Moneyness = 0.95: K3': 76,
    'Moneyness = 1:    K4': 80,
    'Moneyness = 1.05: K5': 84,
    'Moneyness = 1.10: K6': 88,
    'Moneyness = 1.15: K7': 92

}

K = 80
r = 0.055
T = 3/12
t = 0


for moneyness, strike in strike_prices.items():
    params = [strike, r, T, t]
    SM = Merton_paths(S0=80, M=50, Ite=1000, lamb=0.75, mu=-0.5,
delta=0.22, r=0.055, sigma=0.35, T=3/12)
    C_eu_merton = np.round(merton_opt_mc(SM, *params, 'C'), 2)
    print(moneyness, ", Call Price is: ", C_eu_merton)
```

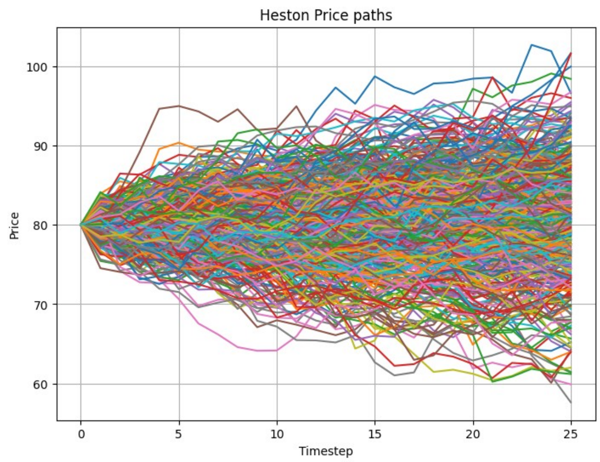
```
Moneyness = 0.85: K1 , Call Price is:  15.7
Moneyness = 0.90: K2 , Call Price is:  13.0
Moneyness = 0.95: K3 , Call Price is:  11.0
Moneyness = 1:    K4 , Call Price is:  7.97
Moneyness = 1.05: K5 , Call Price is:  6.28
Moneyness = 1.10: K6 , Call Price is:  5.29
Moneyness = 1.15: K7 , Call Price is:  3.12
```

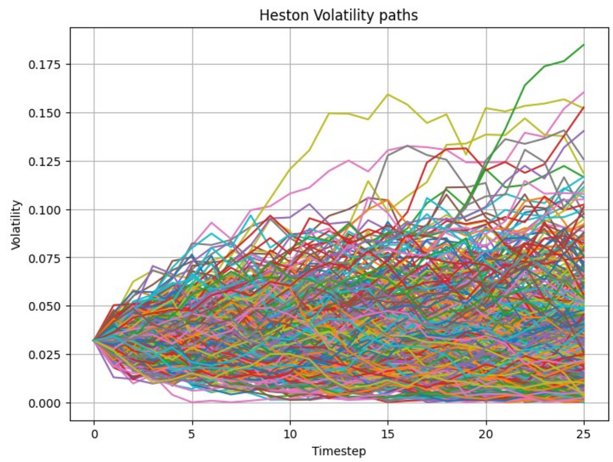
Step 2

Heston Model -> Pricing ATM American Call option with correlation value -0.30

```
def SDE_vol(v0, kappa, theta, sigma, T, M, Ite, rand, row,
cho_matrix):
    dt = T / M  # T = maturity, M = number of time steps
    v = np.zeros((M + 1, Ite), dtype=float)
    v[0] = v0
    sdt = np.sqrt(dt)  # Sqrt of dt
    for t in range(1, M + 1):
        ran = np.dot(cho_matrix, rand[:, t])
        v[t] = np.maximum(
            0,
            v[t - 1]
            + kappa * (theta - v[t - 1]) * dt
            + np.sqrt(v[t - 1]) * sigma * ran[row] * sdt,
        )
    return v

def Heston_paths(S0, r, v, row, cho_matrix):
    S = np.zeros((M + 1, Ite), dtype=float)
    S[0] = S0
    sdt = np.sqrt(dt)
    for t in range(1, M + 1, 1):
        ran = np.dot(cho_matrix, rand[:, t])
        S[t] = S[t - 1] * np.exp((r - 0.5 * v[t-1]) * dt +
np.sqrt(v[t-1]) * ran[row] * sdt)

    return S

def random_number_gen(M, Ite):
  np.random.seed(1)
  rand = np.random.standard_normal((2, M + 1, Ite))
  return rand

v0 = 0.032
kappa_v = 1.85
```

```python
sigma_v = 0.35
theta_v = 0.045
rho = -0.30

S0 = 80  # Current underlying asset price
r = 0.055  # Risk-free rate
M0 = 100  # Number of time steps in a year
T = 3/12  # Number of years
M = int(M0 * T)  # Total time steps
Ite = 1000 * 2**9  # Number of simulations
dt = T / M  # Length of time step

# Generating random numbers from standard normal
rand = random_number_gen(M, Ite)


# Covariance Matrix
covariance_matrix = np.zeros((2, 2), dtype=float)
covariance_matrix[0] = [1.0, rho]
covariance_matrix[1] = [rho, 1.0]
cho_matrix = np.linalg.cholesky(covariance_matrix)

# Volatility process paths
V = SDE_vol(v0, kappa_v, theta_v, sigma_v, T, M, Ite, rand, 1,
cho_matrix)

# Underlying price process paths
S = Heston_paths(S0, r, V, 0, cho_matrix)

def plot_paths(n):
    fig = plt.figure(figsize=(18, 6))
    ax1 = fig.add_subplot(121)
    ax2 = fig.add_subplot(122)

    ax1.plot(range(len(S)), S[:, :n])
    ax1.grid()
    ax1.set_title("Heston Price paths")
    ax1.set_ylabel("Price")
    ax1.set_xlabel("Timestep")

    ax2.plot(range(len(V)), V[:, :n])
    ax2.grid()
    ax2.set_title("Heston Volatility paths")
    ax2.set_ylabel("Volatility")
    ax2.set_xlabel("Timestep")


plot_paths(500)
```






```
def heston_american_call(S, K, r, T, t):

  discount_factor = np.exp(-r * dt)

  payoff = np.maximum(0, S - K)


  option_values = payoff[:, -1]

  for t in range(M - 1, 0, -1):
    # Discounted expected continuation value
    continuation_values = discount_factor * option_values
    # Check early exercise condition
    option_values = np.maximum(payoff[:, t], continuation_values)

  # Discount payoffs back to present value

  return np.mean(option_values) * np.exp(-r * T)

K = 80
option_price = np.round(heston_american_call(S, K, r, T, 0), 2)
print("American Call Price under Heston: ", option_price)

American Call Price under Heston:  9.03
```

Delta For American call option with correlation value -0.30

```
epsilon = 0.5

S1 = 80
S2 = 80.5

C1 = np.round(heston_american_call(S, K, r, T, 0), 2)
print("European Call Price under Heston: ", C1)

C2 = np.round(heston_american_call(S + epsilon, K, r, T, 0), 2)
print("European Call Price under Heston: ", C2)
```

```
print("Delta for Call correlation value -0.30:", np.round((C2 - C1) /
(S2 - S1), 2))
```

```
European Call Price under Heston:  9.03
European Call Price under Heston:  9.52
Delta for Call correlation value -0.30: 0.98
```

Gamma For American call option with correlation value -0.30

```
epsilon = 0.5

S1 = 80
S2 = 80.5
S3 = 79.5

C1 = np.round(heston_american_call(S, K, r, T, 0), 2)
print("European Call Price under Heston: ", C1)

C2 = np.round(heston_american_call(S + epsilon, K, r, T, 0), 2)
print("European Call Price under Heston: ", C2)

C3 = np.round(heston_american_call(S - epsilon, K, r, T, 0), 2)
print("European Call Price under Heston: ", C3)

print("Gamma for Call correlation value -0.30:", np.round((C2 - 2 * C1
+ C3) / (epsilon**2), 2))
```

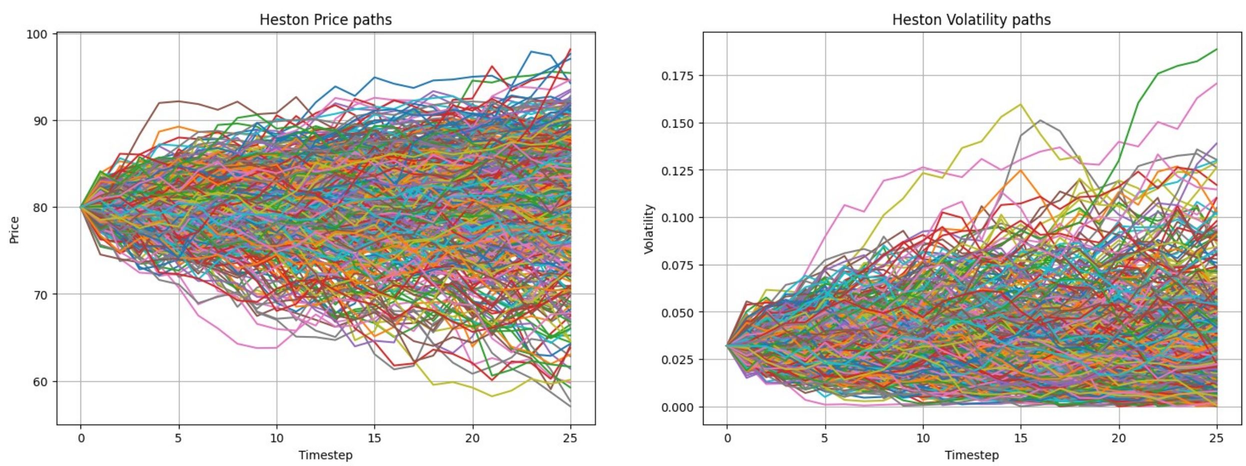
```
European Call Price under Heston:  9.03
European Call Price under Heston:  9.52
European Call Price under Heston:  8.56
Gamma for Call correlation value -0.30: 0.08
```

Heston Model -> Pricing ATM American Call option with correlation value -0.70

```
def SDE_vol(v0, kappa, theta, sigma, T, M, Ite, rand, row,
cho_matrix):
    dt = T / M  # T = maturity, M = number of time steps
    v = np.zeros((M + 1, Ite), dtype=float)
    v[0] = v0
    sdt = np.sqrt(dt)  # Sqrt of dt
    for t in range(1, M + 1):
        ran = np.dot(cho_matrix, rand[:, t])
        v[t] = np.maximum(
            0,
            v[t - 1]
            + kappa * (theta - v[t - 1]) * dt
            + np.sqrt(v[t - 1]) * sigma * ran[row] * sdt,
        )
```

```
    return v

def Heston_paths(S0, r, v, row, cho_matrix):
    S = np.zeros((M + 1, Ite), dtype=float)
    S[0] = S0
    sdt = np.sqrt(dt)
    for t in range(1, M + 1, 1):
        ran = np.dot(cho_matrix, rand[:, t])
        S[t] = S[t - 1] * np.exp((r - 0.5 * v[t-1]) * dt +
np.sqrt(v[t-1]) * ran[row] * sdt)

    return S

def random_number_gen(M, Ite):
  np.random.seed(1)
  rand = np.random.standard_normal((2, M + 1, Ite))
  return rand

v0 = 0.032
kappa_v = 1.85
sigma_v = 0.35
theta_v = 0.045
rho = -0.70

S0 = 80  # Current underlying asset price
r = 0.055  # Risk-free rate
M0 = 100  # Number of time steps in a year
T = 3/12  # Number of years
M = int(M0 * T)  # Total time steps
Ite = 1000 * 2**9  # Number of simulations
dt = T / M  # Length of time step

# Generating random numbers from standard normal
rand = random_number_gen(M, Ite)


# Covariance Matrix
covariance_matrix = np.zeros((2, 2), dtype=float)
covariance_matrix[0] = [1.0, rho]
covariance_matrix[1] = [rho, 1.0]
cho_matrix = np.linalg.cholesky(covariance_matrix)

# Volatility process paths
V = SDE_vol(v0, kappa_v, theta_v, sigma_v, T, M, Ite, rand, 1,
cho_matrix)

# Underlying price process paths
S = Heston_paths(S0, r, V, 0, cho_matrix)

def plot_paths(n):
    fig = plt.figure(figsize=(18, 6))
```

```
    ax1 = fig.add_subplot(121)
    ax2 = fig.add_subplot(122)

    ax1.plot(range(len(S)), S[:, :n])
    ax1.grid()
    ax1.set_title("Heston Price paths")
    ax1.set_ylabel("Price")
    ax1.set_xlabel("Timestep")

    ax2.plot(range(len(V)), V[:, :n])
    ax2.grid()
    ax2.set_title("Heston Volatility paths")
    ax2.set_ylabel("Volatility")
    ax2.set_xlabel("Timestep")


plot_paths(500)
```




```
def heston_american_call_2(S, K, r, T, t):

  discount_factor = np.exp(-r * dt)

  payoff = np.maximum(0, S - K)


  option_values = payoff[:, -1]

  for t in range(M - 1, 0, -1):
    # Discounted expected continuation value
    continuation_values = discount_factor * option_values
    # Check early exercise condition
    option_values = np.maximum(payoff[:, t], continuation_values)

  return np.mean(option_values) * np.exp(-r * T)
```

```
K = 80
option_price_2 = np.round(heston_american_call_2(S, K, r, T, 0), 2)
print("American Call Price under Heston: ", option_price_2)
```

```
American Call Price under Heston:  8.43
```

Delta For call option with correlation value -0.70

```
epsilon = 0.5

S1 = 80
S2 = 80.5

C1 = np.round(heston_american_call_2(S, K, r, T, 0), 2)
print("European Call Price under Heston: ", C1)

C2 = np.round(heston_american_call_2(S + epsilon, K, r, T, 0), 2)
print("European Call Price under Heston: ", C2)

print("Delta for Call correlation value -0.30:", np.round((C2 - C1) /
(S2 - S1), 2))
```

```
European Call Price under Heston:  8.43
European Call Price under Heston:  8.92
Delta for Call correlation value -0.30: 0.98
```

Gamma For call and Put option with correlation value -0.70

```
epsilon = 0.5

S1 = 80
S2 = 80.5
S3 = 79.5

params = [S, K, r, T, 0]

C1 = np.round(heston_american_call_2(*params), 2)
print("European Call Price under Heston: ", C1)

C2 = np.round(heston_american_call_2(S + epsilon, K, r, T, 0), 2)
print("European Call Price under Heston: ", C2)

C3 = np.round(heston_american_call_2(S - epsilon, K, r, T, 0), 2)
print("European Call Price under Heston: ", C3)

print("Gamma for Call correlation value -0.30:", np.round((C2 - 2 * C1
+ C3) / (epsilon**2), 2))
```

```
European Call Price under Heston:  8.43
European Call Price under Heston:  8.92
```

```
European Call Price under Heston:  7.96
Gamma for Call correlation value -0.30: 0.08
```

Merton Model -> Pricing ATM American Call option with lambda = 0.75

```
M_steps = 64 # daily basis
N = 200000
K = 80
T = 3/12
r = 0.055
dt = T/M

np.random.seed(1)
SM = Merton_paths(S0=80, M=M_steps, Ite=N, lamb=0.75, mu=-0.5,
delta=0.22, r=r, sigma=0.35, T=T)

# Initialize the payoff matrix
payoff = np.maximum(SM - K, 0)

# Initialize the cashflow matrix
cashflow = payoff[-1]


for t in range(M - 1, 0, -1):
    # Discount future cashflows
    discounted_cashflow = cashflow * np.exp(-r * dt)

    # Compare with intrinsic value and decide whether to exercise
    cashflow = np.maximum(payoff[t], discounted_cashflow)

# Discount the final cashflow back to present value
option_price = np.mean(cashflow * np.exp(-r * dt))

print(f"American call option price: {option_price:.2f}")
```

```
American call option price: 13.91
```

Q14: Price a European up-and-in call option (UAI) with a barrier level of $95 and a strike price of $95 as well. This UAI option becomes alive only if the stock price reaches (at some point before maturity) the barrier level (even if it ends below it). Compare the price obtained to the one from the simple European call.

```
def heston_call_mc_2(S, K, r, T, t):
    payoff = np.maximum(0, S[-1, :] - K)

    average = np.mean(payoff)

    return np.exp(-r * (T - t)) * average
```

```python
def heston_barier_call_mc_2(S, K, B, r, T, t):
    L = S.shape[0]
    # Up-and-In (UAI) barrier call option
    price_paths_in = S[np.max(S, axis=1) >= B, :]

    # Like call option:
    option_values = np.maximum(price_paths_in[:, -1] - K, 0)

    average = np.sum(option_values) / L

    return np.exp(-r * (T - t)) * average

v0 = 0.032
kappa_v = 1.85
sigma_v = 0.35
theta_v = 0.045
rho = -0.70

S0 = 80  # Current underlying asset price
r = 0.055  # Risk-free rate
M0 = 255  # Number of time steps (days)
T = 3/12  # Number of years
M = int(M0 * T)  # Total time steps
Ite = 1000 * 2**8  # Number of simulations
dt = T / M  # Length of time step
t = 0

# Generating random numbers from standard normal
# rand = random_number_gen(M, Ite)


# Covariance Matrix
covariance_matrix = np.zeros((2, 2), dtype=float)
covariance_matrix[0] = [1.0, rho]
covariance_matrix[1] = [rho, 1.0]
cho_matrix = np.linalg.cholesky(covariance_matrix)

# Volatility process paths
V = SDE_vol(v0, kappa_v, theta_v, sigma_v, T, M, Ite, rand, 1,
cho_matrix)

# Underlying price process paths
SH = Heston_paths(S0, r, V, 0, cho_matrix)

K = 95
B = 95

payoff = np.maximum(0, SH[-1, :] - K)

average = np.mean(payoff)
```

```
C_price = np.exp(-r * (T - t)) * average



# Up-and-In (UAI) barrier call option
price_paths_in = SH[np.max(SH, axis=1) >= B, :]

# Like call option:
option_values = np.maximum(price_paths_in[:, -1] - K, 0)

avg = np.sum(option_values) / M

UAI_call = max(np.exp(-r * (T - t)) * avg, 0.01)


print("European Call Price under Heston: ", np.round(C_price, 3))
print("European UAI Call Price under Heston: ", UAI_call)
```

```
European Call Price under Heston:  0.031
European UAI Call Price under Heston:  0.01
```

```
SH.shape
```

```
(64, 256000)
```

```
B = 90
SH[np.max(SH, axis=1) >= B, :][:, -1]
```

```
(57,)
```

```
from scipy.stats import norm
def black_scholes_call(S, K, T, r, sigma):
    # Calculate d1 and d2
    d1 = (np.log(S / K) + (r + 0.5 * sigma**2) * T) / (sigma *
np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)

    # Calculate put price using Black-Scholes formula
    call_price = S * norm.cdf(d1) -K * np.exp(r * T) * norm.cdf(d2)
    return call_price

# Given parameters
S0 = 80
K = 95
T = 3/12
r = 0.055
sigma = 0.35

# Calculate the European put option price
call_price = black_scholes_call(S0, K, T, r, sigma)
print(f"The price of the European call option is: {call_price:.2f}")
```

```
The price of the European call option is: 1.09
```

Q15: Price a European down-and-in put option (DAI) with a barrier level of $65 and a strike price of $65 as well. This UAO option becomes alive only if the stock price reaches (at some point before maturity) the barrier level (even if it ends above it).

```
def simulate_jump_diffusion(S0, r, sigma, T, mu, delta, lamb,
num_steps, num_paths, B, K):
    dt = T / num_steps
    discount_factor = np.exp(-r * T)

    # Initialize asset price paths
    paths = np.zeros((num_paths, num_steps + 1))
    paths[:, 0] = S0

    for t in range(1, num_steps + 1):
        # Generate random numbers
        z = np.random.standard_normal(num_paths)
        jump = np.random.poisson(lamb * dt, num_paths)
        paths[:, t] = paths[:, t-1] * np.exp((r - 0.5 * sigma**2) * dt
+ sigma * np.sqrt(dt) * z)
        paths[:, t] += (np.exp(mu + delta *
np.random.standard_normal(num_paths)) - 1) * jump

    # Check if the barrier is breached
    barrier_breached = (paths.min(axis=1) <= B)

    # Payoff calculation for down-and-in put option
    payoff = np.maximum(K - paths[:, -1], 0)
    payoff *= barrier_breached

    # Price calculation
    option_price = discount_factor * np.mean(payoff)
    return option_price

# Parameters
S0 = 80
r = 0.055
sigma = 0.35
T = 3/12
mu = -0.5
delta = 0.22
lamb = 0.75
B = 65
K = 65
num_steps = 1000
num_paths = 100000
```

```
# Price the option
dai_put_price = simulate_jump_diffusion(S0, r, sigma, T, mu, delta,
lamb, num_steps, num_paths, B, K)
print(f"The price of the European down-and-in put option is:
{dai_put_price:.2f}")
```

```
The price of the European down-and-in put option is: 0.61
```

Simple European Put Price

```
import numpy as np
from scipy.stats import norm

def black_scholes_put(S, K, T, r, sigma):
    # Calculate d1 and d2
    d1 = (np.log(S / K) + (r + 0.5 * sigma**2) * T) / (sigma *
np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)

    # Calculate put price using Black-Scholes formula
    put_price = K * np.exp(-r * T) * norm.cdf(-d2) - S * norm.cdf(-d1)
    return put_price

# Given parameters
S0 = 80
K = 65
T = 0.25
r = 0.055
sigma = 0.35

# Calculate the European put option price
put_price = black_scholes_put(S0, K, T, r, sigma)
print(f"The price of the European put option is: {put_price:.2f}")
```

```
The price of the European put option is: 0.61
```