Group Work Project 3

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as ss
```

STEP 1 Team member A: Stohastic Volatility Modeler

5.

Pricing an ATM European call and put using the Heston Model and Monte-Carlo simulation (for ρ = -0.3)**

The Heston model is a stochastic volatility model that assumes the underlying asset follows a geometric Brownian motion, while the volatility follows a square-root process with mean-reversion. The Heston model is defined by the following stochastic differential equations:

$$dS_t = \mu S_t dt + \sqrt{v_t} S_t dW_t{}^1$$
$$dv_t = \kappa(\theta - v_t)dt + \sigma\sqrt{v_t} dW_t{}^2$$

where S_t is the price of the underlying asset at time t, v_t is the instantaneous variance at time t, μ is the drift rate of the asset, κ is the mean-reversion rate of the variance, θ is the long-run mean of the variance, σ is the volatility of the volatility, and $W_t{}^1$ and $W_t{}^2$ are independent Brownian motions.

To price a European call or put option using the Heston model, we can use a numerical method such as the Monte Carlo simulation. Here's some Python code to price a European call and put option using the Heston model with Monte Carlo simulation:

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

#Model Parameters
S0 = 80.0  # stock price
r = 0.055  # Risk-free IR
T = 1/4  # Time to maturity
K = 80.0  # Strike
V0 = 0.032
rho = -0.3
kappa = 1.85
theta = 0.045
sigma = 0.35
N = 100000  # Number of simulations
np.random.seed(10)
#Heston model
def heston(S0, r, T, K, V0, rho, kappa, theta, sigma, N):
    dt = T / 252.0
    ST = np.zeros(N)
    VT = np.zeros(N)
    ST[...] = S0
    VT[...] = V0
    for t in range(1, 252):
        Z1 = np.random.randn(N)
        Z2 = rho * Z1 + np.sqrt(1 - rho**2) * np.random.randn(N)
        VT = np.maximum(VT + kappa * (theta - VT) * dt + sigma * np.sqrt(VT * dt) * Z1, 0)
        ST = ST * np.exp((r - 0.5 * VT) * dt + np.sqrt(VT * dt) * Z2)
    return np.exp(-r * T) * np.maximum(ST - K, 0).mean(), np.exp(-r * T) * np.maximum(K - ST, 0).mean()

# Calculation of Call and put option
call_price, put_price = heston(S0, r, T, K, V0, rho, kappa, theta, sigma, N)

# Output the results
print("Heston Model and Monte-Carlo simulation, price of an ATM European call and put, using a correlation value of -0.
print(f"ATM European Call Price: {call_price:.4f}")
print(f"ATM European Put Price: {put_price:.4f}")
```

```
Heston Model and Monte-Carlo simulation, price of an ATM European call and put, using a correlation value of -0.3
ATM European Call Price: 2.8770
ATM European Put Price: 2.8328
```

Answer 5.

The price of the ATM european call and put option using the Heston model and monte-carlo simualtion with -0.3 correlation is 2.88 and 2.83 respectively.

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

#Model Parameters
S0 = 80.0  # stock price
r = 0.055  # Risk-free IR
T = 1/4  # Time to maturity
K = 80.0  # Strike
V0 = 0.032
rho = -0.7
kappa = 1.85
theta = 0.045
sigma = 0.35
N = 100000  # Number of simulations
np.random.seed(10)
#Heston model
def heston(S0, r, T, K, V0, rho, kappa, theta, sigma, N):
    dt = T / 252.0
    ST = np.zeros(N)
    VT = np.zeros(N)
    ST[...] = S0
    VT[...] = V0
    for t in range(1, 252):
        Z1 = np.random.randn(N)
        Z2 = rho * Z1 + np.sqrt(1 - rho**2) * np.random.randn(N)
        VT = np.maximum(VT + kappa * (theta - VT) * dt + sigma * np.sqrt(VT * dt) * Z1, 0)
        ST = ST * np.exp((r - 0.5 * VT) * dt + np.sqrt(VT * dt) * Z2)
    return np.exp(-r * T) * np.maximum(ST - K, 0).mean(), np.exp(-r * T) * np.maximum(K - ST, 0).mean()

# Calculation of Call and put option
call_price, put_price = heston(S0, r, T, K, V0, rho, kappa, theta, sigma, N)

# Output the results
print("Heston Model and Monte-Carlo simulation, price of an ATM European call and put, using a correlation value of -0.
print(f"ATM European Call Price: {call_price:.4f}")
print(f"ATM European Put Price: {put_price:.4f}")
```

```
Heston Model and Monte-Carlo simulation, price of an ATM European call and put, using a correlation value of -0.7(
ATM European Call Price: 2.1237
ATM European Put Price: 3.4647
```

Answer 6.

The price of the ATM european call and put option using the Heston model and monte-carlo simualtion with -0.7 correlation is 2.12 and 3.46 respectively.

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

#Model Parameters
S0 = 81.0  # stock price
r = 0.055  # Risk-free IR
T = 1/4  # Time to maturity
K = 80.0  # Strike
V0 = 0.032
rho = -0.3
kappa = 1.85
theta = 0.045
sigma = 0.35
N = 100000  # Number of simulations
np.random.seed(10)
#Heston model
def heston(S0, r, T, K, V0, rho, kappa, theta, sigma, N):
    dt = T / 252.0
    ST = np.zeros(N)
    VT = np.zeros(N)
    ST[...] = S0
    VT[...] = V0
    for t in range(1, 252):
        Z1 = np.random.randn(N)
        Z2 = rho * Z1 + np.sqrt(1 - rho**2) * np.random.randn(N)
        VT = np.maximum(VT + kappa * (theta - VT) * dt + sigma * np.sqrt(VT * dt) * Z1, 0)
        ST = ST * np.exp((r - 0.5 * VT) * dt + np.sqrt(VT * dt) * Z2)
    return np.exp(-r * T) * np.maximum(ST - K, 0).mean(), np.exp(-r * T) * np.maximum(K - ST, 0).mean()

# Calculation of Call and put option
call_price, put_price = heston(S0, r, T, K, V0, rho, kappa, theta, sigma, N)

# Output the results
print("Heston Model and Monte-Carlo simulation, price of an European call and put, using a correlation value of -0.30.:
print(f"European Call Price: {call_price:.4f}")
print(f"European Put Price: {put_price:.4f}")
```

```
Heston Model and Monte-Carlo simulation, price of an European call and put, using a correlation value of -0.30.:
European Call Price: 3.4419
European Put Price: 2.4109
```

Answer 7.

Firstly we are using the -0.3 correlation , to calculate the delta we have changed the price of the initail stock from 80 to 81.

The change in the option price due to the change in underlying price will tell us the delta of the option.

In this case the call delta1 = (3.4419-2.8770) = 0.56 (rounding 2 decimal)

put delta1 = (2.4109-2.8328) = -0.42

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

#Model Parameters
S0 = 82.0  # stock price
r = 0.055  # Risk-free IR
T = 1/4  # Time to maturity
K = 80.0  # Strike
V0 = 0.032
rho = -0.3
kappa = 1.85
theta = 0.045
sigma = 0.35
N = 100000  # Number of simulations
np.random.seed(10)
#Heston model
def heston(S0, r, T, K, V0, rho, kappa, theta, sigma, N):
    dt = T / 252.0
    ST = np.zeros(N)
    VT = np.zeros(N)
    ST[...] = S0
    VT[...] = V0
    for t in range(1, 252):
        Z1 = np.random.randn(N)
        Z2 = rho * Z1 + np.sqrt(1 - rho**2) * np.random.randn(N)
        VT = np.maximum(VT + kappa * (theta - VT) * dt + sigma * np.sqrt(VT * dt) * Z1, 0)
        ST = ST * np.exp((r - 0.5 * VT) * dt + np.sqrt(VT * dt) * Z2)
    return np.exp(-r * T) * np.maximum(ST - K, 0).mean(), np.exp(-r * T) * np.maximum(K - ST, 0).mean()

# Calculation of Call and put option
call_price, put_price = heston(S0, r, T, K, V0, rho, kappa, theta, sigma, N)

# Output the results
print("Heston Model and Monte-Carlo simulation, price of an European call and put, using a correlation value of -0.30.:
print(f"European Call Price: {call_price:.4f}")
print(f"European Put Price: {put_price:.4f}")
```

```
Heston Model and Monte-Carlo simulation, price of an European call and put, using a correlation value of -0.30.:
European Call Price: 4.0595
European Put Price: 2.0415
```

Answer 7. (contnd.)

For the calculation of the gamma we have calculated 2 delta one at 81 at initial stock price and another at initial stock price at 82.

The difference between these 2 deltas will give us the gamma.

call delta2 = 4.0595-3.4419 =0.62 put delta2 = 2.0415-2.4109 = -0.37

gamma = delta1- delta2 = 0.56-0.62 = -0.6 gamma = delta1- delta2 = -0.42-0.37= -0.5

The gamma should be same for a call and put option, the observed difference is due to the rounding error. Also, here we are deducting the deltas i.e. negative gamma imply the selling of the options.

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

#Model Parameters
S0 = 81.0  # stock price
r = 0.055  # Risk-free IR
T = 1/4  # Time to maturity
K = 80.0  # Strike
V0 = 0.032
rho = -0.7
kappa = 1.85
theta = 0.045
sigma = 0.35
N = 100000  # Number of simulations
np.random.seed(10)
#Heston model
def heston(S0, r, T, K, V0, rho, kappa, theta, sigma, N):
    dt = T / 252.0
    ST = np.zeros(N)
    VT = np.zeros(N)
    ST[...] = S0
    VT[...] = V0
    for t in range(1, 252):
        Z1 = np.random.randn(N)
        Z2 = rho * Z1 + np.sqrt(1 - rho**2) * np.random.randn(N)
        VT = np.maximum(VT + kappa * (theta - VT) * dt + sigma * np.sqrt(VT * dt) * Z1, 0)
        ST = ST * np.exp((r - 0.5 * VT) * dt + np.sqrt(VT * dt) * Z2)
    return np.exp(-r * T) * np.maximum(ST - K, 0).mean(), np.exp(-r * T) * np.maximum(K - ST, 0).mean()

# Calculation of Call and put option
call_price, put_price = heston(S0, r, T, K, V0, rho, kappa, theta, sigma, N)

# Output the results
print("Heston Model and Monte-Carlo simulation, price of an European call and put, using a correlation value of -0.70.:
print(f"European Call Price: {call_price:.4f}")
print(f"European Put Price: {put_price:.4f}")
```

```
Heston Model and Monte-Carlo simulation, price of an European call and put, using a correlation value of -0.70.:
European Call Price: 2.6341
European Put Price: 3.0056
```

Answer 7.(contnd.)

Now we are using the -0.7 correlation , to calculate the delta we have changed the price of the initail stock from 80 to 81.

The change in the option price due to the chane in underlying price will tell us the delta of the option.

In this case the call delta = (2.6341-2.1237) = 0.51 (rounding 2 decimal)

put delta = (3.0056-3.4647) = -0.46

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

#Model Parameters
S0 = 82.0  # stock price
r = 0.055  # Risk-free IR
T = 1/4  # Time to maturity
K = 80.0  # Strike
V0 = 0.032
rho = -0.7
kappa = 1.85
theta = 0.045
sigma = 0.35
N = 100000  # Number of simulations
np.random.seed(10)
#Heston model
def heston(S0, r, T, K, V0, rho, kappa, theta, sigma, N):
    dt = T / 252.0
    ST = np.zeros(N)
    VT = np.zeros(N)
    ST[...] = S0
    VT[...] = V0
    for t in range(1, 252):
        Z1 = np.random.randn(N)
        Z2 = rho * Z1 + np.sqrt(1 - rho**2) * np.random.randn(N)
        VT = np.maximum(VT + kappa * (theta - VT) * dt + sigma * np.sqrt(VT * dt) * Z1, 0)
        ST = ST * np.exp((r - 0.5 * VT) * dt + np.sqrt(VT * dt) * Z2)
    return np.exp(-r * T) * np.maximum(ST - K, 0).mean(), np.exp(-r * T) * np.maximum(K - ST, 0).mean()

# Calculation of Call and put option
call_price, put_price = heston(S0, r, T, K, V0, rho, kappa, theta, sigma, N)

# Output the results
print("Heston Model and Monte-Carlo simulation, price of an European call and put, using a correlation value of -0.70.:
print(f"European Call Price: {call_price:.4f}")
print(f"European Put Price: {put_price:.4f}")
```

```
Heston Model and Monte-Carlo simulation, price of an European call and put, using a correlation value of -0.70.:
European Call Price: 3.1979
European Put Price: 2.5998
```

Answer 7.(contnd.)

Now we are using the -0.7 correlation , to calculate the delta we have changed the price of the initail stock from 81 to 82.

the change in the option price due to the chane in underlying price will tell us the delta of the option.

In this case the call delta2 = (3.1979-2.6341) = 0.56 (rounding 2 decimal)

put delta2 = (2.5998-3.0056) = -0.41

call gamma of the option with -0.7 correlation = (delta - delta2) = 0.51-0.56 = -0.05

put gamma of the option with -0.7 correlation = (delta - delta2) = -0.46 +0.41 =-0.05

Gamma is same here , as mentioned the negative gamma ~ selling of the options.

Answer 8.

Calculating ATM European call and put prices using the Merton Model (for μ = -0.5 and δ = 0.22 and λ = 0.75))**

The Merton Jump Diffusion Model is a mathematical model that combines both continuous and discontinuous processes to describe the dynamics of asset prices. The model was introduced by Robert Merton in 1976, and it is widely used in finance to analyze the behavior of stock prices, interest rates, and other financial instruments.

In this model, the price of an asset is modeled as a continuous process with a stochastic volatility, combined with random jumps in the price at discrete time intervals. The model assumes that the size and timing of the jumps are random, but follow a certain distribution, usually the Poisson distribution.

The Merton Jump Diffusion Model can be expressed mathematically as follows:

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

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:

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

The Merton Jump Diffusion Model is widely used in options pricing and risk management, as it allows for a more accurate representation of the distribution of asset prices and the possibility of sudden, unexpected changes in price due to jumps.

This code defines three functions for simulating Merton jump-diffusion processes and pricing European call and put options based on the simulated stock prices.

The create_merton_jumps() function takes in the following parameters:

S: the current stock price

K: the strike price of the option

r: the risk-free interest rate

T: the time to expiration of the option

λ :the jump intensity parameter μ: the mean of the jump size distribution

δ: the standard deviation of the jump size distribution

σ: the annualized volatility of the stock price

Ite: the number of simulations to run

M: the number of time steps to use in the simulation

The function creates an array SM of shape (M+1, Ite) to store the simulated stock prices over time for each simulation. The initial stock price is set to S.

The function then calculates the jump component of the Merton jump-diffusion process by computing the expected jump size.

It also generates arrays of random standard normal variates z1 and z2 and a Poisson-distributed random variable y to use in the simulation.

```
import numpy as np
def create_merton_jumps(S, K, r, T, lamb, mu, delta, sigma, Ite, M):

    dt = T/M
    SM = np.zeros((M + 1, Ite))
    SM[0] = S

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

    # Random numbers
    np.random.seed(10)
    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 #delta_call, delta_put

def merton_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

def merton_put_mc(S, K, r, T, t):
    payoff = np.maximum(0, -S[-1, :] + K)
    average = np.mean(payoff)
    return np.exp(-r * (T - t)) * average
```

This code snippet is related to the valuation of European call and put options using the Merton jump-diffusion model (for μ = -0.5 and δ = 0.22 and λ = 0.75)

```
SM = create_merton_jumps(80, 80, 0.055, 1/4, 0.75, -0.5, 0.22, 0.35, 10000, 50)
call_price75 = np.round(merton_call_mc(SM, K, r, T, 0),4)
put_price75 = np.round(merton_put_mc(SM, K, r, T, 0),4)
print("Case lambda = 0.75")
print(f"European Call Price under Merton with lambda: 0.75 = {call_price75} USD")
print(f"European Put Price under Merton with lambda: 0.75 = {put_price75} USD")
```

```
Case lambda = 0.75
European Call Price under Merton with lambda: 0.75 = 8.3633 USD
European Put Price under Merton with lambda: 0.75 = 7.2509 USD
```

Answer 9.

Calculating ATM European call and put prices using the Merton Model (for μ = -0.5 and δ = 0.22 and λ = 0.25)**

Using the function above, below code snippet is related to the valuation of European call and put options using the Merton jump-diffusion model (for μ = -0.5 and δ = 0.22 and λ = 0.25)

```
# ATM call and put prices need to calculated after calculating the Merton jump paths
# SM stands for varying price with Brownian motion modelled by Merton jump diffusion model

SM = create_merton_jumps(80, 80, 0.055, 1/4, 0.25, -0.5, 0.22, 0.35, 10000, 50)
call_price25 = np.round(merton_call_mc(SM, K, r, T, 0),4)
put_price25 = np.round(merton_put_mc(SM, K, r, T, 0),4)
print("Case lambda = 0.25")
print(f"European Call Price under Merton with lambda: 0.25 = {call_price25} USD")
print(f"European Put Price under Merton with lambda: 0.25 = {put_price25} USD")
```

```
Case lambda = 0.25
European Call Price under Merton with lambda: 0.25 = 6.8219 USD
European Put Price under Merton with lambda: 0.25 = 5.7319 USD
```

Answer 10.

Calculating Δ and γ using the Merton Model (for μ = -0.5 and δ = 0.22 and λ = 0.75 and 0.25)**

Using the function above, below code is related to the valuation of δ of European call and put options using the Merton jump-diffusion model (for μ = -0.5 and δ = 0.22) with different cases of λ.

δ is a measure of the sensitivity of an option's price to changes in the underlying asset's price.

It is the ratio of the change in the option price to the change in the underlying asset price. Here as we change the price of the option one USD we can observe how much this change effected the change of the option which our first δ. Thus we change the value of S to 81 USD and try to observe the change.

Find δ in case 1: λ = 0.75 deltas

```
# Now let us increase the price of the asset 1 USD so that we can calculate call and put delta

SM = create_merton_jumps(81, 80, 0.055, 1/4, 0.75, -0.5, 0.22, 0.35, 10000, 50)
call_price75_2 = np.round(merton_call_mc(SM, K, r, T, 0),4)
put_price75_2 = np.round(merton_put_mc(SM, K, r, T, 0),4)

# print("European Call Price under Merton: ", call_price2)
# print("European Put Price under Merton: ", put_price2)
print("Case lambda = 0.75")
delta1_call = np.round(call_price75_2 - call_price75, 4)
print("Call Delta under Merton: ", delta1_call )

delta1_put = np.round(put_price75_2 - put_price75,4)
print("Put Delta under Merton: ", delta1_put )
```

```
Case lambda = 0.75
Call Delta under Merton:  0.6632
Put Delta under Merton:  -0.3371
```

Find δ in case 2: λ = 0.25

```
# Now let us do the same for lambda = 0.25

SM = create_merton_jumps(81, 80, 0.055, 1/4, 0.25, -0.5, 0.22, 0.35, 10000, 50)
call_price25_2 = np.round(merton_call_mc(SM, K, r, T, 0),4)
put_price25_2 = np.round(merton_put_mc(SM, K, r, T, 0),4)

# print("European Call Price under Merton: ", call_price25_2)
# print("European Put Price under Merton: ", put_price25_2)
print("Case lambda = 0.25")
delta2_call = np.round(call_price25_2 - call_price25,4)
print("Call Delta under Merton: ", delta2_call )
```

```
delta2_put = np.round(put_price25_2 - put_price25,4)
print("Put Delta under Merton: ", delta2_put )
```

```
    Case lambda = 0.25
    Call Delta under Merton:  0.6124
    Put Delta under Merton:  -0.3875
```

γ is a measure of the rate of change of an option's delta in response to changes in the underlying asset's price. It is the second derivative of the option price with respect to the underlying asset price. γ represents the convexity of an option's price curve and measures how much the delta changes when the underlying asset price changes.

A high gamma means that the delta of the option is highly sensitive to changes in the underlying asset price, while a low gamma means that the delta is less sensitive to changes in the underlying asset price. γ is particularly important for traders and investors who use δ hedging strategies, as it helps them to adjust their hedge positions more accurately and effectively as the underlying asset price changes.

Now here, we needed to calculate 2 seperate δs, for when S changes from 80 to 81 and 81 to 82, to calculate one δ for each case and then we take the difference of these two δs to finally calculate γ.

Find γ in case λ = 0.75

```
# Now let us increase the price of the asset 1 USD more so that we can calculate gamma

SM = create_merton_jumps(82, 80, 0.055, 1/4, 0.75, -0.5, 0.22, 0.35, 10000, 50)
call_price75_3 = np.round(merton_call_mc(SM, K, r, T, 0),4)
put_price75_3 = np.round(merton_put_mc(SM, K, r, T, 0),4)

# print("European Call Price under Merton: ", call_price3)
# print("European Put Price under Merton: ", put_price3)

print("Case lambda = 0.75")
delta2_call = np.round(call_price75_3 - call_price75_2, 4)
# print("Call Delta under Merton: ", delta2_call )

delta2_put = np.round(put_price75_3 - put_price75_2,4)
# print("Put Delta under Merton: ", delta2_put )

call_gamma = np.round(delta2_call - delta1_call,4)
put_gamma = np.round(delta2_put - delta1_put,4)

print("Call Gamma under Merton: ", call_gamma)
print("Put Gamma under Merton: ", put_gamma)
```

```
    Case lambda = 0.75
    Call Gamma under Merton:  0.0192
    Put Gamma under Merton:  0.0192
```

Find γ in case λ = 0.25

```
# Now let us do the same with lambda = 0.25

SM = create_merton_jumps(82, 80, 0.055, 1/4, 0.25, -0.5, 0.22, 0.35, 10000, 50)
call_price25_3 = np.round(merton_call_mc(SM, K, r, T, 0),4)
put_price25_3 = np.round(merton_put_mc(SM, K, r, T, 0),4)

# print("European Call Price under Merton: ", call_price3)
# print("European Put Price under Merton: ", put_price3)

print("Case lambda = 0.25")
delta2_call = np.round(call_price25_3 - call_price25_2, 4)
# print("Call Delta under Merton: ", delta2_call )

delta2_put = np.round(put_price25_3 - put_price25_2,4)
# print("Put Delta under Merton: ", delta2_put )

call_gamma = np.round(delta2_call - delta1_call,4)
put_gamma = np.round(delta2_put - delta1_put,4)

print("Call Gamma under Merton: ", call_gamma)
print("Put Gamma under Merton: ", put_gamma)
```

```
Case lambda = 0.25
Call Gamma under Merton:  -0.0245
Put Gamma under Merton:  -0.0242
```

Double-click (or enter) to edit

Answer 13. (With Q5 parameters)

```
import numpy as np

# Heston model parameters
S0 = 80     # initial stock price
K = 80      # strike
r = 0.055     # risk-free IR
T = 1/4        # time to maturity
sigma = 0.35
kappa = 1.85
theta = 0.045
rho = -0.3
v0 = 0.032

# sim params
N = 100
M = 100000

# simulate stock price and variance paths using Cholesky decomposition
dt = T / N
np.random.seed(10)
dw = np.random.normal(size=(N, 2, M))
cov = np.array([[1, rho], [rho, 1]])
L = np.linalg.cholesky(cov)
v = np.zeros((N + 1, M))
v[0] = v0
S = np.zeros((N + 1, M))
S[0] = S0
for n in range(N):
    dW1 = dw[n, 0]
    dW2 = dw[n, 1]
    dZ1 = L[0, 0] * dW1 + L[0, 1] * dW2
    dZ2 = L[1, 0] * dW1 + L[1, 1] * dW2
    v[n + 1] = np.maximum(v[n] + kappa * (theta - v[n]) * dt + sigma * np.sqrt(v[n] * dt) * dZ1, 0)
    S[n + 1] = S[n] * np.exp((r - 0.5 * v[n]) * dt + np.sqrt(v[n] * dt) * dZ2)

# Longstaff-Schwartz used for the pricing of the American option
CF = np.maximum(S[-1] - K, 0)
for n in range(N - 1, 0, -1):
    df = np.exp(-r * dt)
    reg = np.polyfit(S[n], CF * df, 5)
    C = np.polyval(reg, S[n])
    CF = np.where((S[n] > (C)), CF, (CF * df))
    CF = np.maximum(CF, np.maximum(S[n] - K, 0))
call_price = np.mean(CF * df)

CF = np.maximum(K - S[-1], 0)
for n in range(N - 1, 0, -1):
    df = np.exp(-r * dt)
    reg = np.polyfit(S[n], CF * df, 5)
    C = np.polyval(reg, S[n])
    CF = np.where((S[n] < (C)), CF, (CF * df))
    CF = np.maximum(CF, np.maximum(K - S[n], 0))
put_price = np.mean(CF * df)

print("American call price:", call_price)
print("American put price:", put_price)
```

```
/usr/local/lib/python3.8/dist-packages/IPython/core/interactiveshell.py:3326: RankWarning: Polyfit may be poorly c
  exec(code_obj, self.user_global_ns, self.user_ns)
American call price: 5.921395910848832
American put price: 4.876547719653745
/usr/local/lib/python3.8/dist-packages/IPython/core/interactiveshell.py:3326: RankWarning: Polyfit may be poorly c
  exec(code_obj, self.user_global_ns, self.user_ns)
```

We have tried to solve this problem in standard way, so we did some research and used the longstadd-Schwartz method which is based on regression to solve this question the price of the options we got are 5.92 for the American call option and 4.88 for the American put option.

Reference - "Valuing American Options by Simulation: A Simple Least-Squares Approach" Francis A. Longstaff UCLA; Eduardo S. Schwartz UCLA

Answer 14. (considering param from Q6)

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

# model parameters
S0 = 80.0   # initial stock price
r = 0.055     # risk-free IR
V0 = 0.032
theta = 0.045
kappa = 1.85
sigma = 0.35
rho = -0.7
T = 1/4      # time to maturity
K = 95.0     # strike
B = 95.0     # barrier
M = 100000
dt = 1/252   # time step
N = int(T/dt)

# Simulation
np.random.seed(10)
S = np.zeros((M, N+1))
V = np.zeros((M, N+1))
S[:,0] = S0
V[:,0] = V0
for i in range(N):
    dW1 = np.random.normal(0, np.sqrt(dt), M)
    dW2 = rho*dW1 + np.sqrt(1-rho**2)*np.random.normal(0, np.sqrt(dt), M)
    S[:,i+1] = S[:,i] * np.exp((r - 0.5*V[:,i])*dt + np.sqrt(V[:,i]*dt)*dW1)
    V[:,i+1] = V[:,i] + kappa*(theta - V[:,i])*dt + sigma*np.sqrt(V[:,i]*dt)*dW2

# Payoff
payoff = np.maximum(S[:,-1] - K, 0)
barrier = np.max(S[:,1:N+1], axis=1) >= B
payoff[~barrier] = 0
price = np.exp(-r*T) * np.mean(payoff)

print(price)
```

```
    0.0
```

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

def BS_call(S, K, r, sigma, T):
    d1 = (np.log(S/K) + (r + sigma**2/2)*T) / (sigma*np.sqrt(T))
    d2 = d1 - sigma*np.sqrt(T)
    return S*norm.cdf(d1) - K*np.exp(-r*T)*norm.cdf(d2)


BS_call(80,95,0.055,0.35,1/4)
```

```
    1.508603424223038
```

As we can see the price of the Up and in european call option is showing of a price 0 with spot at 80 with strike and barrier at 95 using heston model.

The key understanding is that, the simulated price is not touching the 95 level in the short life i.e. 3 months/ 63 steps.

However when we use the simple black scholes option pricing model we can see the price of the option is 1.51 with the constant volatility of 35%.