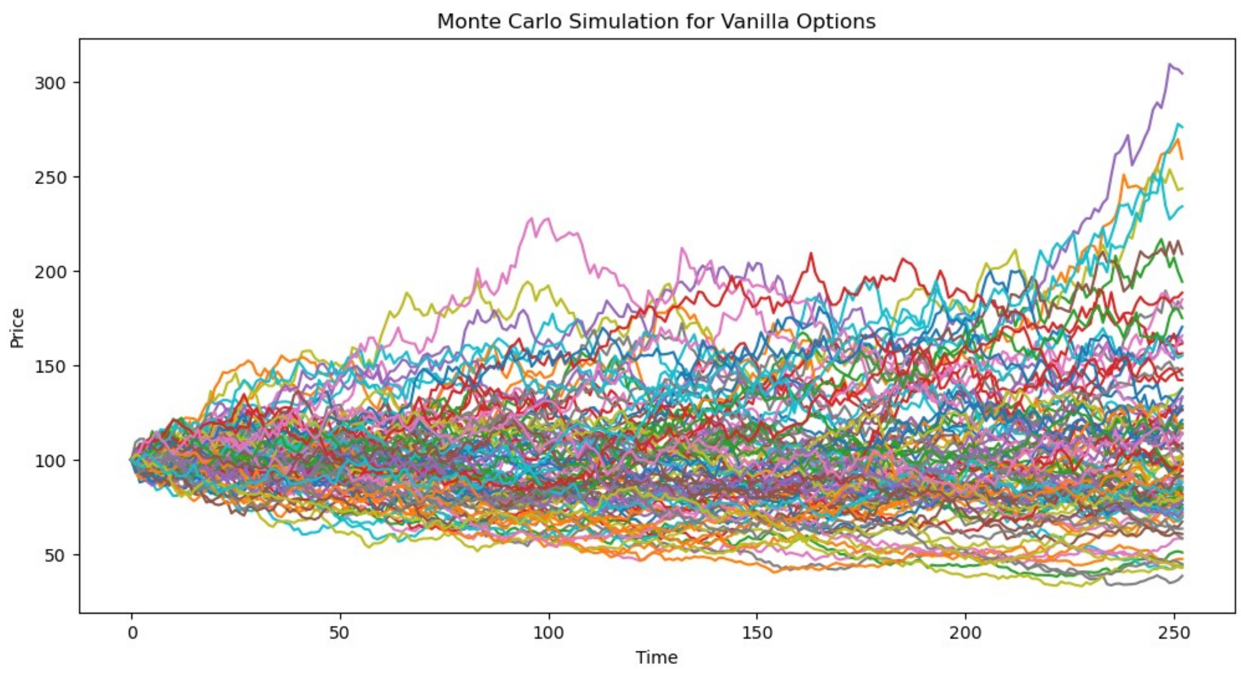
```python
import random
import numpy as np
import matplotlib.pyplot as plt

def boxmuller():
    u1=random.random()
    u2=random.random()
    Z=np.sqrt(-2*np.log(u1))*np.cos(2*np.pi*u2)
    return Z
```

Vanilla Option

```python
def MC():
    S0=100
    r=0.05
    K=120
    T=2
    sigma=0.3
    M=100000
    S=[0 for i in range(M)]
    S[0]=S0
    Payoff=[0 for i in range(M)]
    for i in range(1, M):
        St=S[0]*np.exp((r-
0.5*sigma**2)*T+sigma*np.sqrt(T)*boxmuller())
        S[i]=St
        Payoff[i]=np.exp(-r*T)*max(St-K, 0)
    Call=np.mean(Payoff)
    Put=Call-S0+K*np.exp(-r*T)
    print(f"Call: {Call} | Put: {Put}")

MC()
```

```
Call: 13.548946277141418 | Put: 22.129436441456562
```

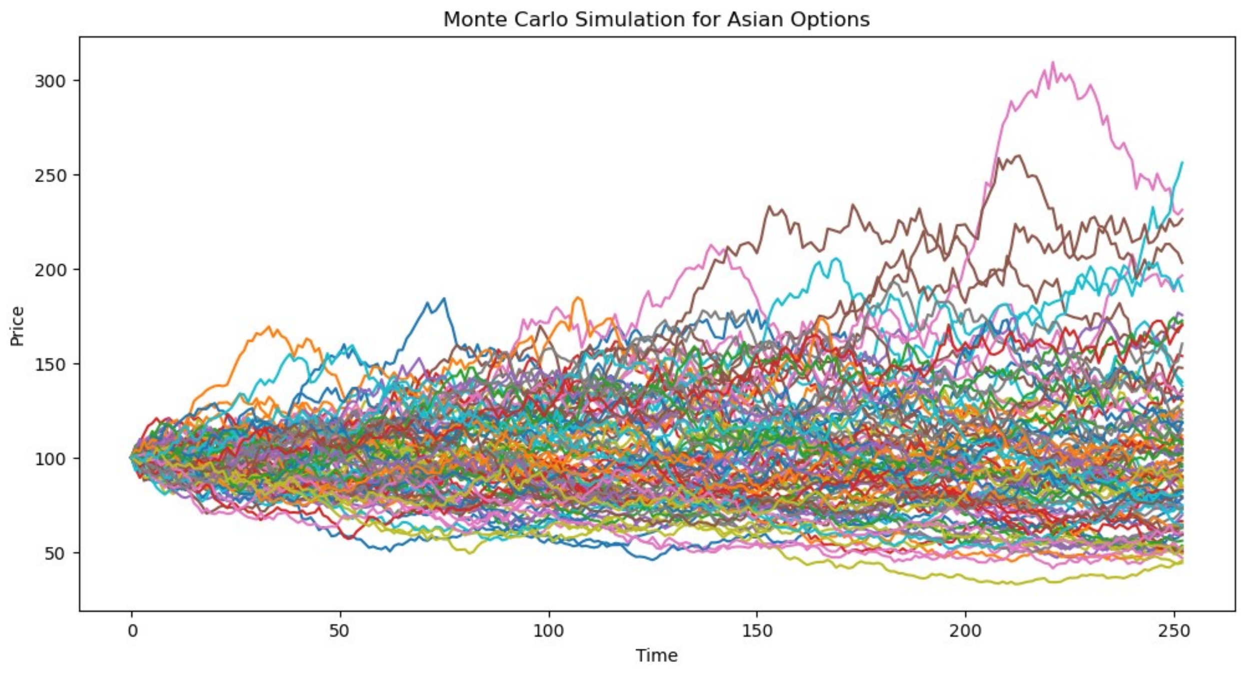
```python
def MC_2():
    S0=100
    r=0.05
    K=120
    T=2
    n=252
    dt=T/n
    sigma=0.3
    M=10000
    S=np.zeros((M, n+1))
    S[:, 0]=S0

    for i in range(1, n+1):
        Z=np.array([boxmuller() for j in range(M)])
        S[:, i]=S[:, i-1]+S[:, i-1]*(r*dt+sigma*np.sqrt(dt)*Z)
    Payoff=np.maximum(S[:, -1]-K, 0)
    Call=np.mean(Payoff)*np.exp(-r*T)
    Put=Call-S0+K*np.exp(-r*T)

    plt.figure(figsize=(12, 6))
    for i in range(100):
        plt.plot(S[i, :])
    plt.title("Monte Carlo Simulation for Vanilla Options")
    plt.xlabel("Time")
    plt.ylabel("Price")
    plt.show()

    print(f"Call: {Call} | Put: {Put}")

MC_2()
```



```
Call: 13.420579414108763 | Put: 22.001069578423895
```

Asian Option

```
def MC_Asian():
    S0 = 100
    r = 0.05
    K = 120
    T = 2
    n = 252
    dt = T / n
    sigma = 0.3
    M = 30000
    S = np.zeros((M, n + 1))
    S[:, 0] = S0

    for i in range(1, n + 1):
        Z = np.array([boxmuller() for j in range(M)])
        S[:, i] = S[:, i - 1] * (1 + r * dt + sigma * np.sqrt(dt) * Z)

    mean_price = np.mean(S[:, 1:], axis=1)
    Payoff = np.maximum(mean_price - K, 0)
    Call = np.mean(Payoff) * np.exp(-r * T)
    Put = np.mean(np.maximum(K - mean_price, 0)) * np.exp(-r * T)

    plt.figure(figsize=(12, 6))
    for i in range(100):
        plt.plot(S[i, :])
    plt.title("Monte Carlo Simulation for Asian Options")
    plt.xlabel("Time")
    plt.ylabel("Price")
    plt.show()

    print(f"Call: {Call} | Put: {Put}")

MC_Asian()
```

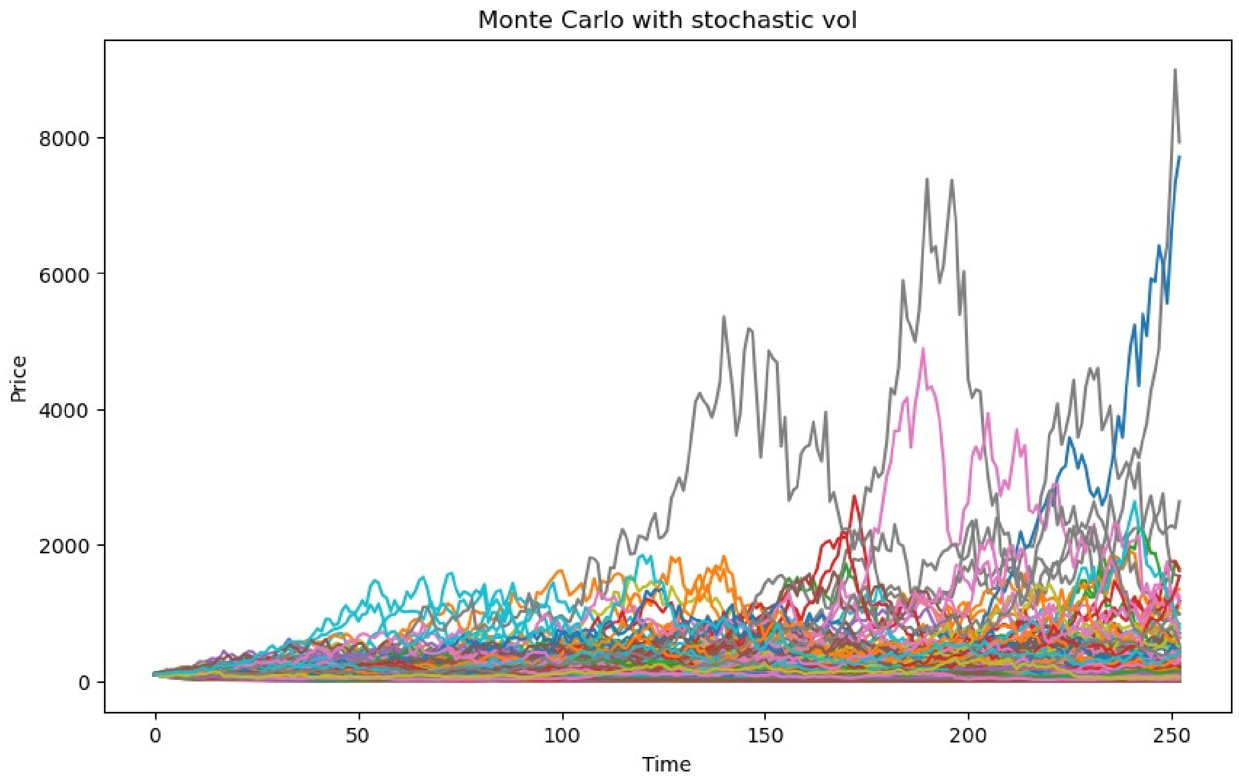


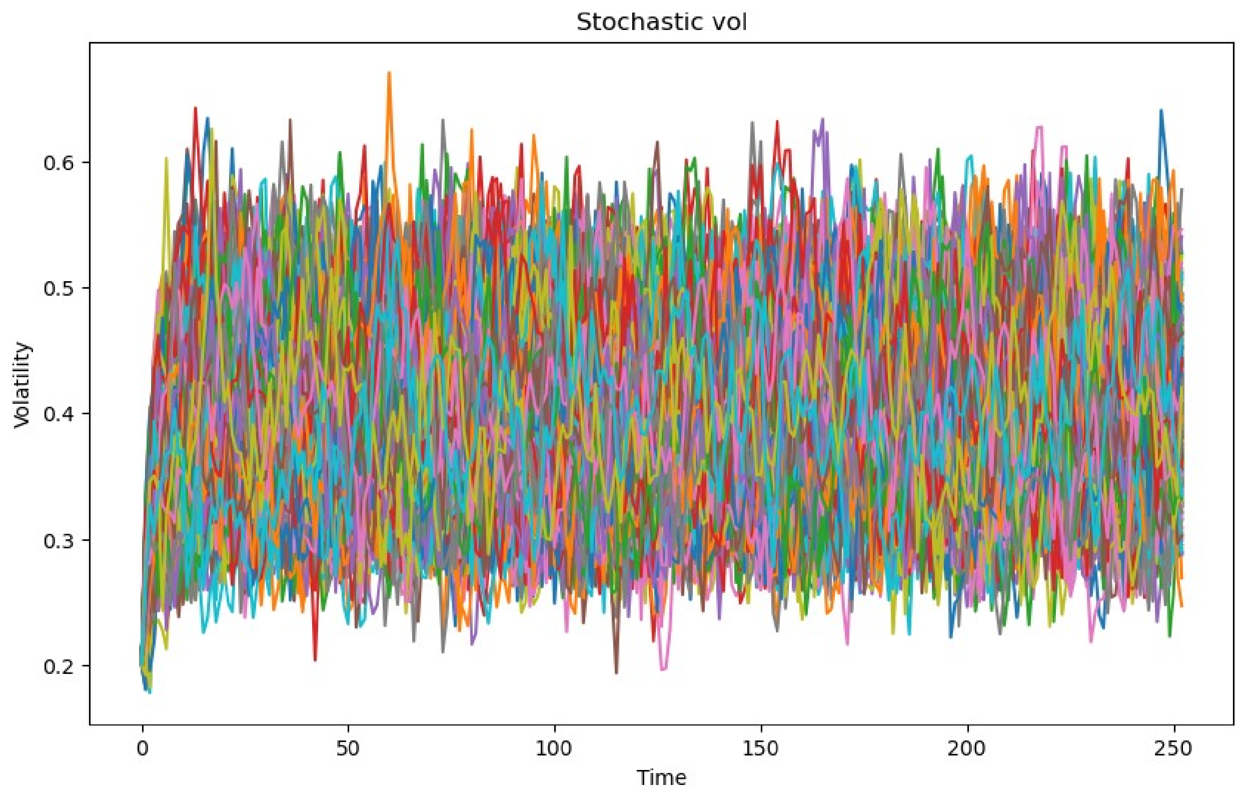
```
Call: 4.819853872505143 | Put: 18.272997181009718
```

Monte Carlo with stochastic volatility

Russian Option

```
def MC_heston_russian():
    S0=100
    r=0.05
    K=1000
    T=10
    n=252
    dt=T/n
    M=30000
    S=np.zeros((M, n+1))
    S[:, 0]=S0
    Payoff=[0]*M

    #Heston parameters
    Kappa=7
    v0=0.2
    Theta=0.4
    sigma=0.3
    vt=np.zeros((M, n+1))
    rho=-0.6
    vt[:, 0]=v0

    for i in range(1, n+1):
        Z=np.array([boxmuller() for i in range(M)])
        Y=np.array([boxmuller() for i in range(M)])
        Z2=rho*Z+np.sqrt(1-rho**2)*Y
        S[:, i]=S[:, i-1]+S[:, i-1]*(r*dt+np.sqrt(vt[:, i-
1])*np.sqrt(dt)*Z)
        vt[:, i]=vt[:, i-1]+(Kappa*(Theta-vt[:, i-
1])*dt+sigma*np.sqrt(vt[:, i-1])*np.sqrt(dt)*Z2)

    maxi=np.max(S, axis=1)
    mini=np.min(S, axis=1)
    PayoffCall=np.maximum(maxi-K, 0)
    PayoffPut=np.maximum(K-mini, 0)
    Call=np.mean(PayoffCall)*np.exp(-r*T)
    Put=np.mean(PayoffPut)*np.exp(-r*T)

    plt.figure(figsize=(10, 6))
    for i in range(500):
        plt.plot(S[i])
    plt.title("Monte Carlo with stochastic vol")
    plt.xlabel("Time")
    plt.ylabel("Price")
    plt.show()

    plt.figure(figsize=(10, 6))
    for i in range(500):
        plt.plot(vt[i])
    plt.title("Stochastic vol")
    plt.xlabel("Time")
    plt.ylabel("Volatility")
    plt.show()

    print(f"Call Price: {Call} | Put Price: {Put}")

MC_heston_russian()
```

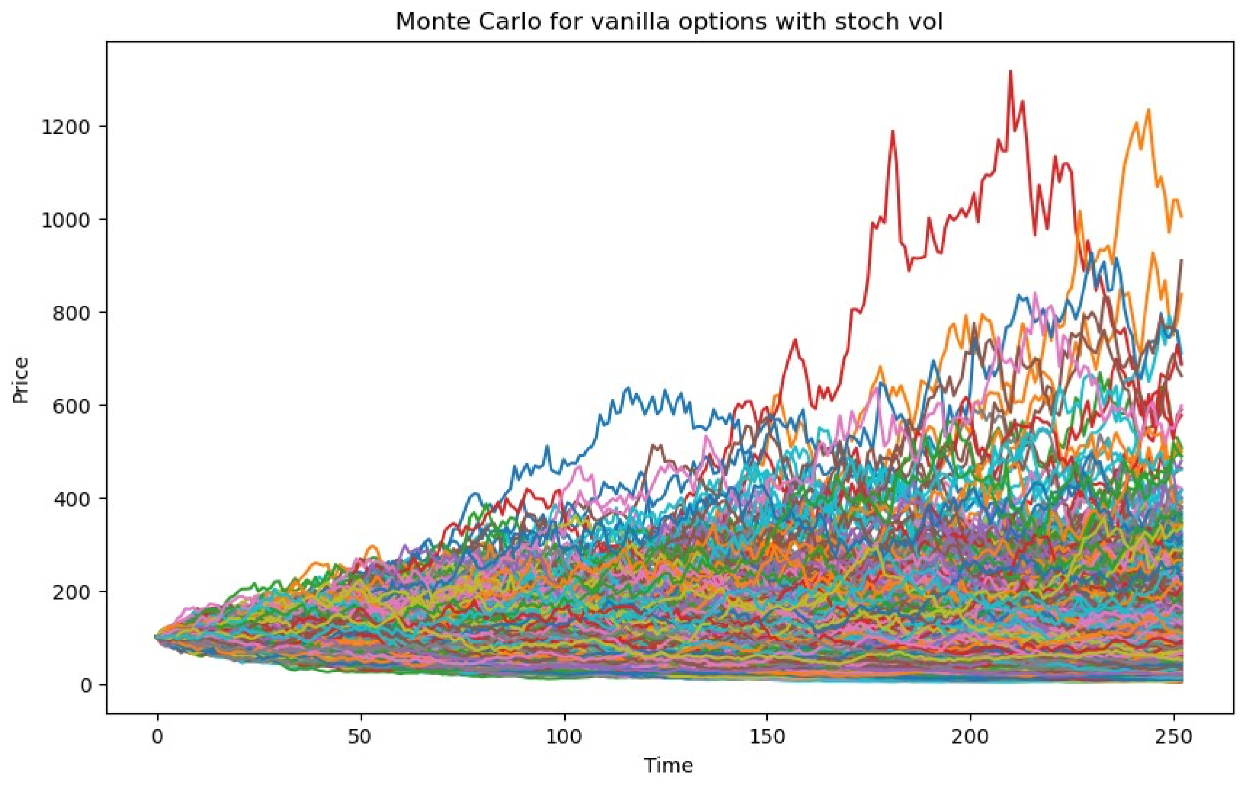


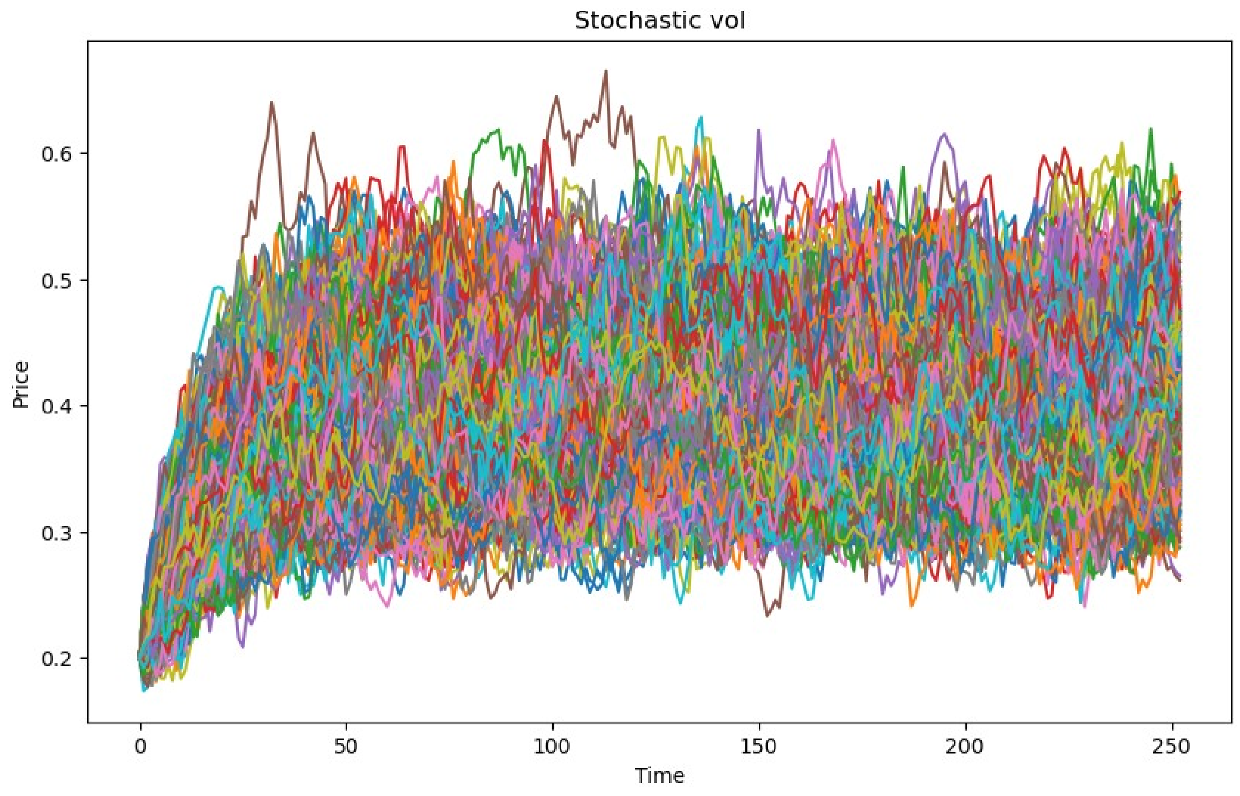


```
Call Price: 91.2085222770063 | Put Price: 594.156794570693
```

Vanilla option with stochastic vol

```
def MC_heston_vanilla():
    S0=100
    r=0.05
    K=120
    T=2
    n=252
    dt=T/n
    M=10000
    S=np.zeros((M, n+1))
    S[:, 0]=S0
    Payoff=[0]*M

    #Heston parameters
    Kappa=8
    v0=0.2
    Theta=0.4
    sigma=0.3
    vt=np.zeros((M, n+1))
    rho=-0.6
    vt[:, 0]=v0
    for i in range(1, n+1):
        Z=np.array([boxmuller() for i in range(M)])
        Y=np.array([boxmuller() for i in range(M)])
        Z2=rho*Z+np.sqrt(1-rho**2)*Y
        S[:, i]=S[:, i-1]+S[:, i-1]*(r*dt+np.sqrt(vt[:, i-
1])*np.sqrt(dt)*Z)
        vt[:, i]=vt[:, i-1]+(Kappa*(Theta-vt[:, i-
1])*dt+sigma*np.sqrt(vt[:, i-1])*np.sqrt(dt)*Z2)

    Payoff=np.maximum(S[:, -1]-K, 0)
    Call=np.mean(Payoff)*np.exp(-r*T)
    Put=Call-S0+K*np.exp(-r*T)

    plt.figure(figsize=(10, 6))
    for i in range(1000):
        plt.plot(S[i])
    plt.title("Monte Carlo for vanilla options with stoch vol")
    plt.xlabel("Time")
    plt.ylabel("Price")
    plt.show()

    plt.figure(figsize=(10, 6))
    for i in range(1000):
        plt.plot(vt[i])
    plt.title("Stochastic vol")
    plt.xlabel("Time")
    plt.ylabel("Price")
    plt.show()

    print(f"Call Price: {Call} | Put Price: {Put}")

MC_heston_vanilla()
```





```
Call Price: 31.596092629413196 | Put Price: 40.17658279372833
```