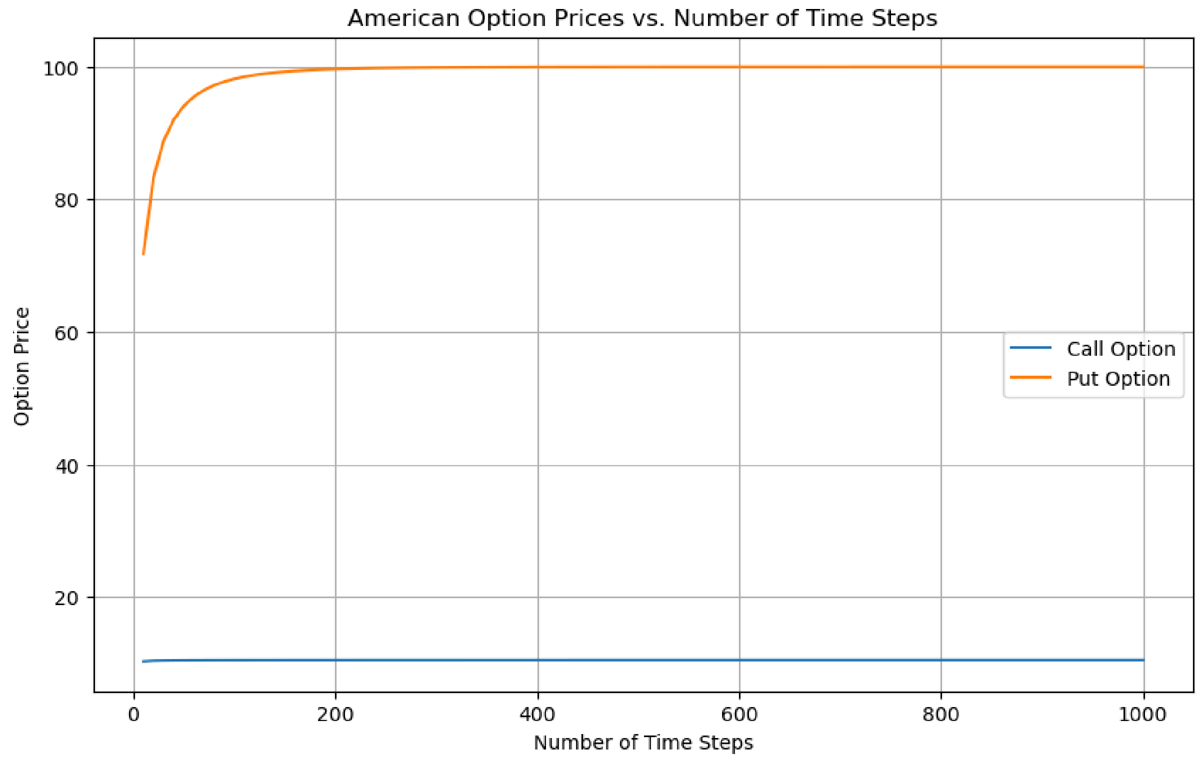
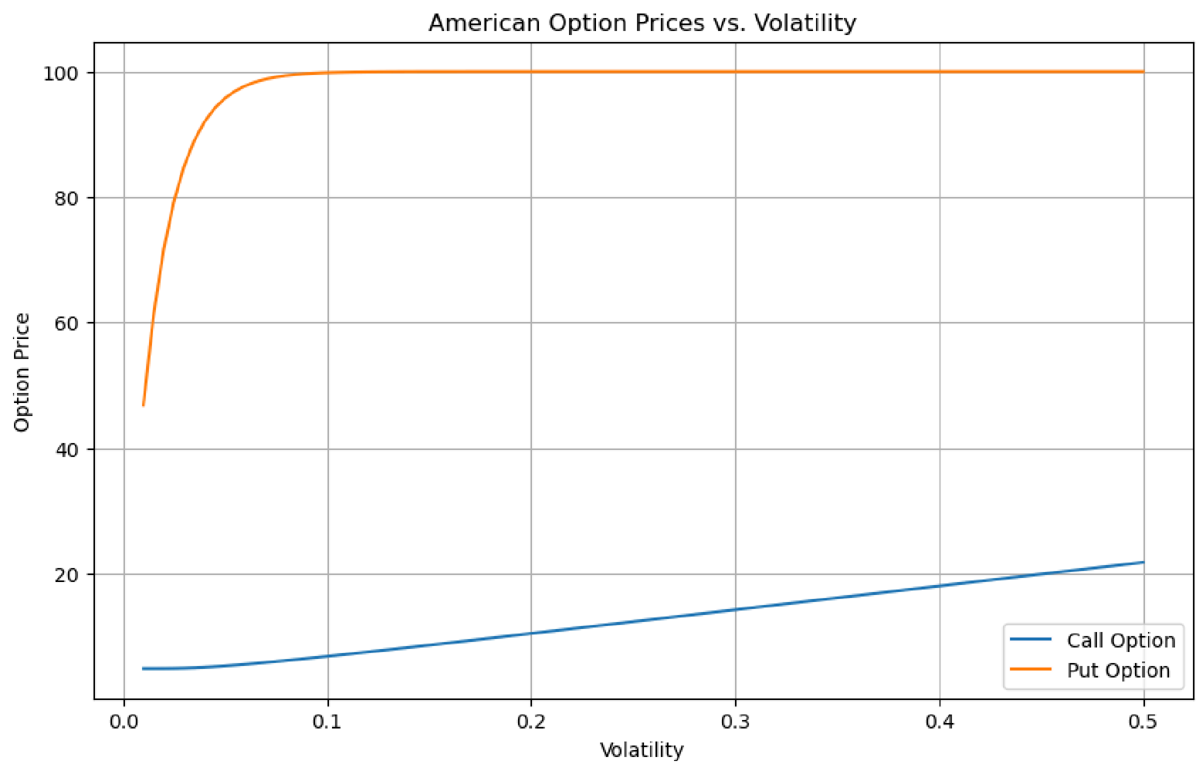
In [8]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
from scipy import stats
```

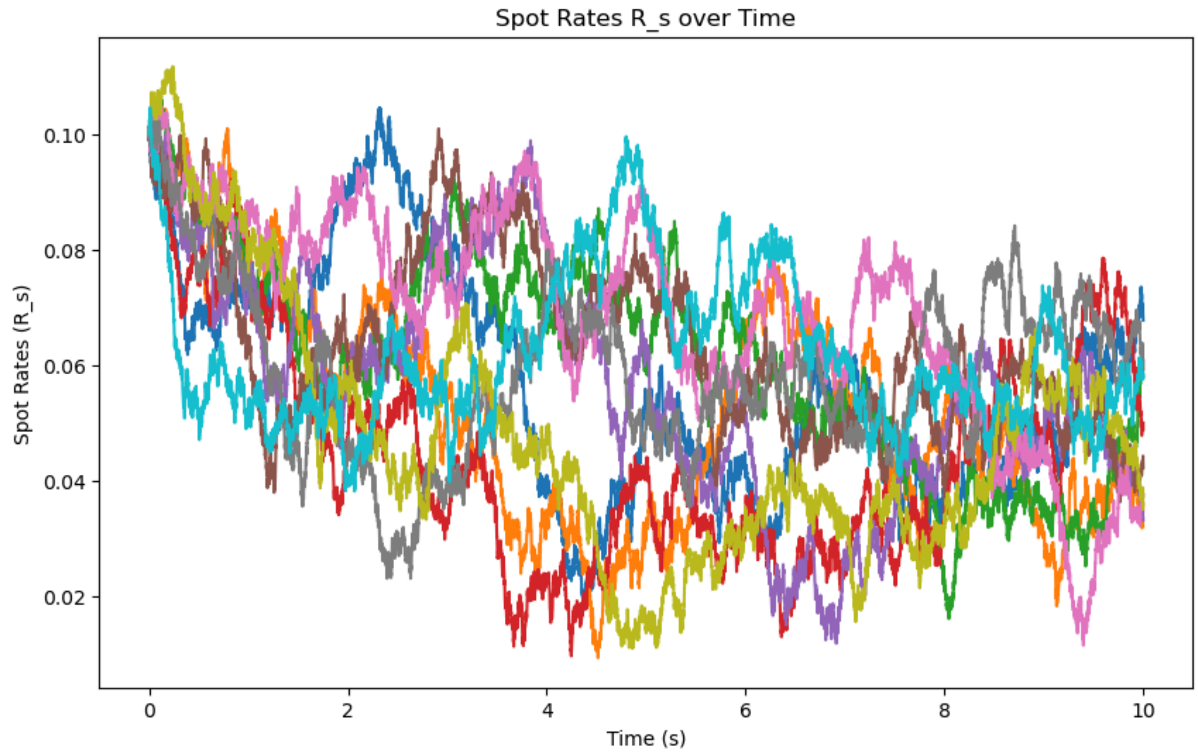
In [3]:
```
# 1. American Option Pricing using Binomial Tree Model
def american_option_price(S, K, T, r, sigma, N, option_type='call'):
    dt = T / N
    u = np.exp(sigma * np.sqrt(dt))
    d = 1 / u
    p = (np.exp(r * dt) - d) / (u - d)

    # Initialize asset prices at maturity
    ST = np.array([S * u**j * d**(N-j) for j in range(N+1)])

    # Initialize option values at maturity
    if option_type == 'call':
        C = np.maximum(ST - K, 0)
    else:  # put
        C = np.maximum(K - ST, 0)

    # Iterate backwards
    for i in range(N-1, -1, -1):
        ST = ST[:-1] / u  # Asset prices one step earlier
        C_hold = np.exp(-r * dt) * (p * C[1:] + (1-p) * C[:-1])
        if option_type == 'call':
            C = np.maximum(C_hold, ST - K)
        else:  # put
            C = np.maximum(C_hold, K - ST)

    return C[0]
```

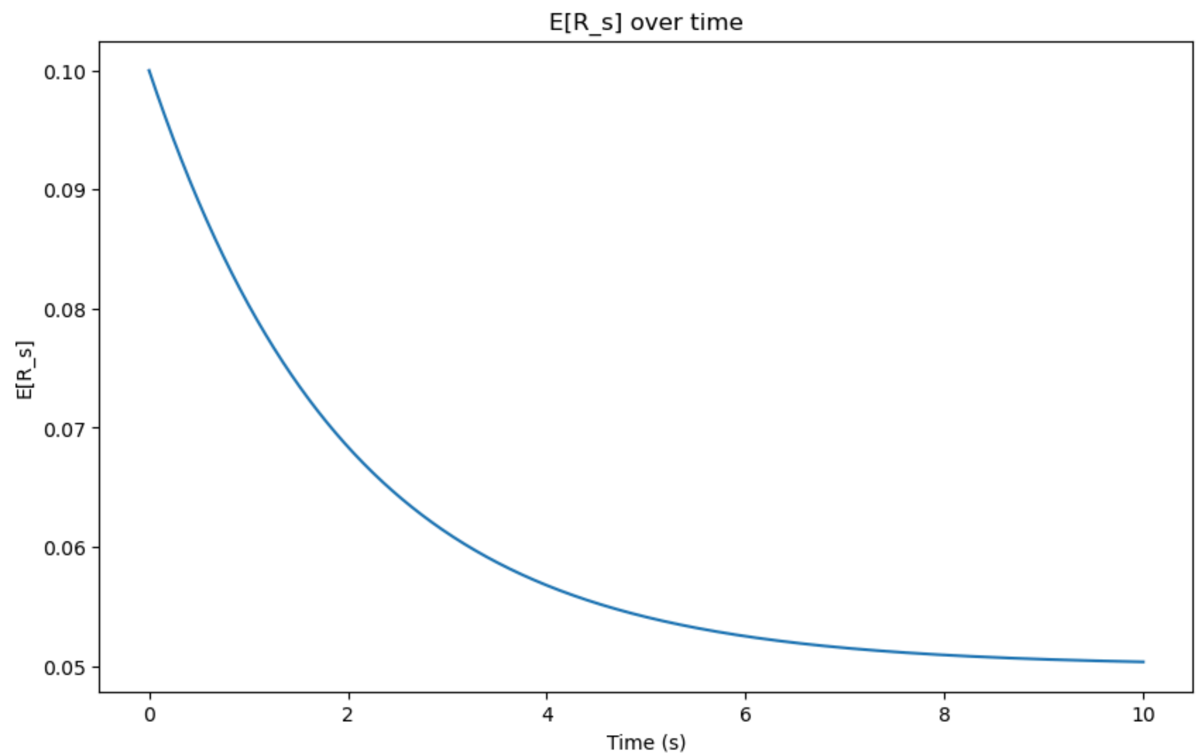
In [4]:
```
# 2. Analysis of American Option Prices
S0 = 100  # Current stock price
K = 100   # Strike price
T = 1     # Time to expiration (in years)
r = 0.05  # Risk-free rate
sigma = 0.2  # Volatility

# Analyze price with respect to number of time steps
N_values = range(10, 1001, 10)
call_prices = [american_option_price(S0, K, T, r, sigma, N) for N in N_v
put_prices = [american_option_price(S0, K, T, r, sigma, N, 'put') for N 

plt.figure(figsize=(10, 6))
plt.plot(N_values, call_prices, label='Call Option')
plt.plot(N_values, put_prices, label='Put Option')
plt.xlabel('Number of Time Steps')
plt.ylabel('Option Price')
plt.title('American Option Prices vs. Number of Time Steps')
plt.legend()
plt.grid(True)
plt.show()

# Analyze price with respect to volatility
sigma_values = np.linspace(0.01, 0.5, 100)
call_prices_sigma = [american_option_price(S0, K, T, r, sigma, 1000) for
put_prices_sigma = [american_option_price(S0, K, T, r, sigma, 1000, 'put

plt.figure(figsize=(10, 6))
plt.plot(sigma_values, call_prices_sigma, label='Call Option')
plt.plot(sigma_values, put_prices_sigma, label='Put Option')
plt.xlabel('Volatility')
plt.ylabel('Option Price')
plt.title('American Option Prices vs. Volatility')
plt.legend()
plt.grid(True)
plt.show()
```

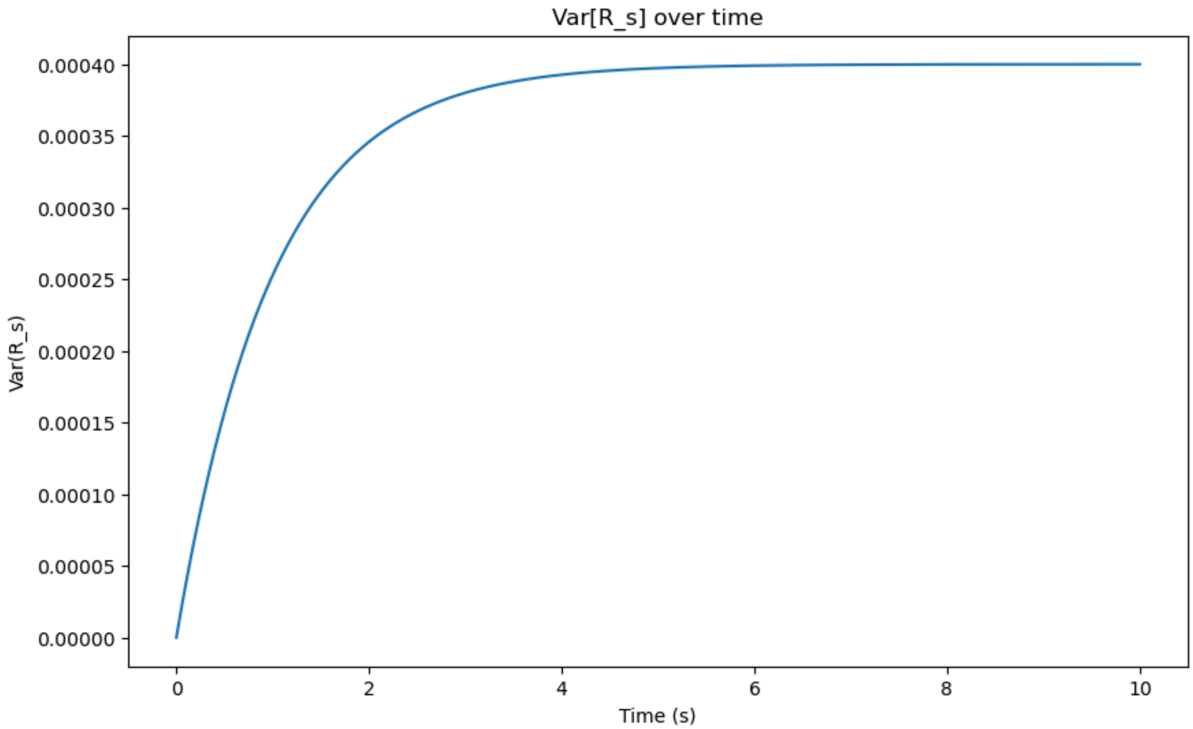

American Option Prices vs. Number of Time Steps
Option Price
Number of Time Steps
Call Option
Put Option
0
200
400
600
800
1000
20
40
60
80
100


American Option Prices vs. Volatility
Option Price
Volatility
Call Option
Put Option
0.0
0.1
0.2
0.3
0.4
0.5
20
40
60
80
100

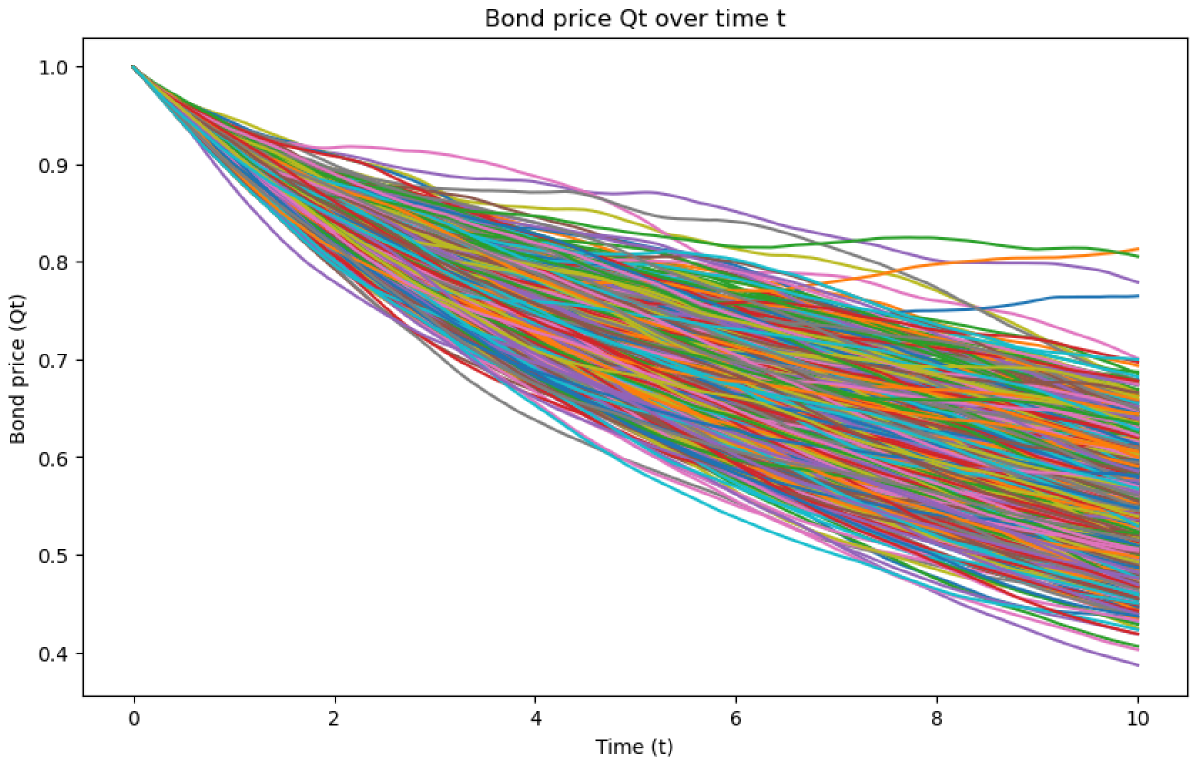
In [5]:
```
# 3. Ornstein-Uhlenbeck Process
def simulate_ou_process(n, N, Delta, theta, sigma, R0, mu):
    times = np.arange(n+1) * Delta
    X = np.zeros((N, n+1))
    for i in range(1, n+1):
        x = X[:, i-1]
        m = x * np.exp(-theta * Delta)
        v = (sigma**2 / (2*theta)) * (1 - np.exp(-2*theta * Delta))
        X[:, i] = np.random.normal(m, np.sqrt(v), N)

    Rt = np.exp(-theta * times) * R0 + (1 - np.exp(-theta * times)) * mu
    return times, Rt

# Simulate OU process
n = 10000
N = 10
Delta = 10 / n
theta = 0.5
sigma = 0.02
R0 = 0.1
mu = 0.05

times, Rt = simulate_ou_process(n, N, Delta, theta, sigma, R0, mu)

plt.figure(figsize=(10, 6))
for i in range(N):
    plt.plot(times, Rt[i, :])
plt.xlabel('Time (s)')
plt.ylabel('Spot Rates (R_s)')
plt.title('Spot Rates R_s over Time')
plt.show()

# Plot expectation and variance
expRt = np.exp(-theta * times) * R0 + (1 - np.exp(-theta * times)) * mu
varRt = (sigma**2 / (2*theta)) * (1 - np.exp(-2*theta * times))

plt.figure(figsize=(10, 6))
plt.plot(times, expRt)
plt.xlabel('Time (s)')
plt.ylabel('E[R_s]')
plt.title('E[R_s] over time')
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(times, varRt)
plt.xlabel('Time (s)')
plt.ylabel('Var(R_s)')
plt.title('Var[R_s] over time')
plt.show()
```

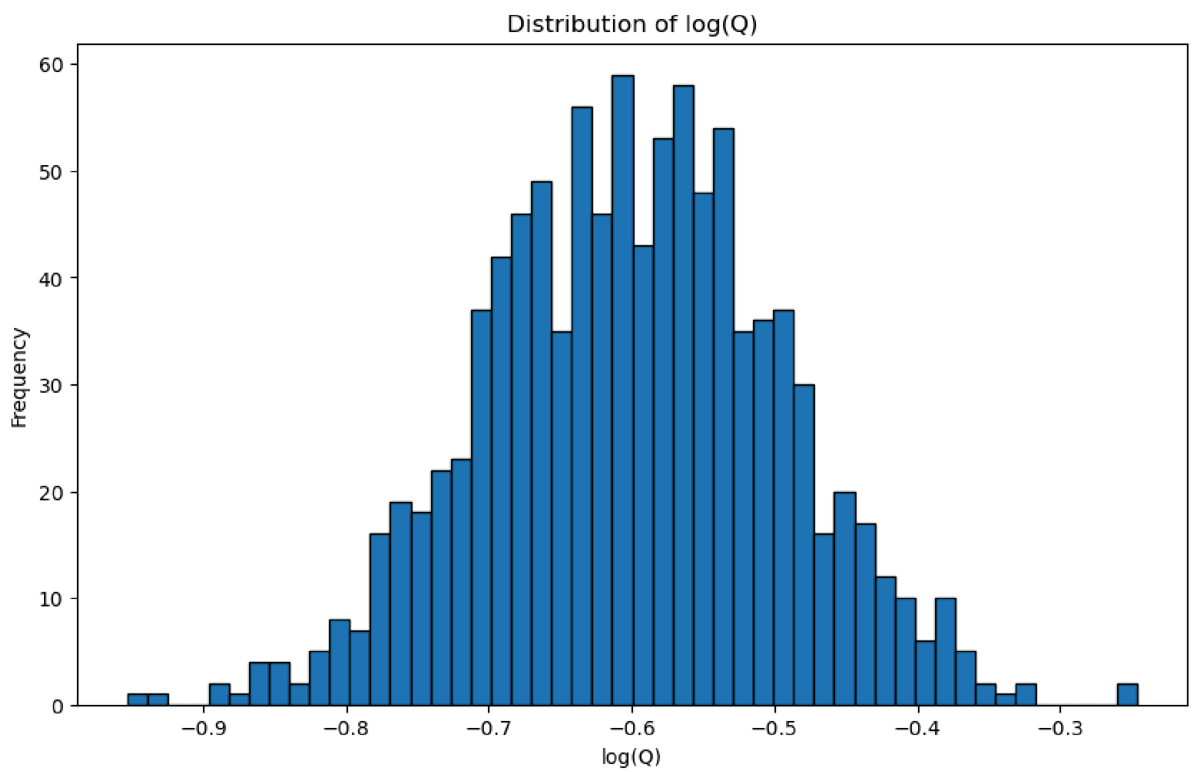

Spot Rates R_s over Time
Spot Rates (R_s)
Time (s)
0.10
0.08
0.06
0.04
0.02
0
2
4
6
8
10

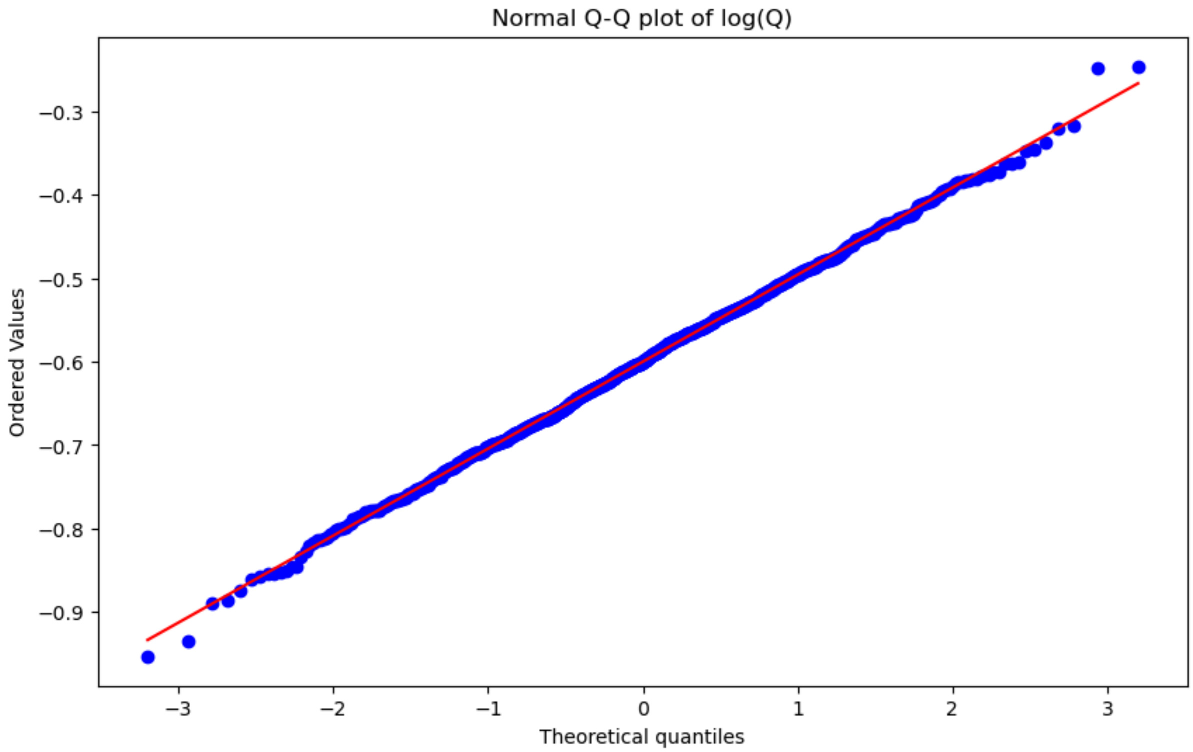

E[R_s] over time
E[R_s]
Time (s)
0.10
0.09
0.08
0.07
0.06
0.05
0
2
4
6
8
10


Var[R_s] over time
0.00040
0.00035
0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
Var(R_s)
0
2
4
6
8
10
Time (s)

In [10]:
```
# 4. Vasicek Model
def simulate_vasicek(n, N, Delta, theta, sigma, R0, mu):
    times, Rt = simulate_ou_process(n, N, Delta, theta, sigma, R0, mu)
    Q = np.exp(-Delta * np.cumsum(Rt, axis=1))
    return times, Q

times, Q = simulate_vasicek(n, N, Delta, theta, sigma, R0, mu)

plt.figure(figsize=(10, 6))
for i in range(N):
    plt.plot(times, Q[i, :])
plt.xlabel('Time (t)')
plt.ylabel('Bond price (Qt)')
plt.title('Bond price Qt over time t')
plt.show()

# Distribution of Qt
n = 1000
N = 1000
Delta = 10 / n
times, Q = simulate_vasicek(n, N, Delta, theta, sigma, R0, mu)

logQ = np.log(Q[:, -1])

plt.figure(figsize=(10, 6))
plt.hist(logQ, bins=50, edgecolor='black')
plt.xlabel('log(Q)')
plt.ylabel('Frequency')
plt.title('Distribution of log(Q)')
plt.show()

plt.figure(figsize=(10, 6))
stats.probplot(logQ, dist="norm", plot=plt)
plt.title('Normal Q-Q plot of log(Q)')
plt.show()
plt.close()

print(f"Mean of log(Q): {np.mean(logQ)}")
print(f"Variance of log(Q): {np.var(logQ)}")
```


Bond price Qt over time t
Bond price (Qt)
Time (t)
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
2
4
6
8
10


Distribution of log(Q)
Frequency
log(Q)
60
50
40
30
20
10
0
−0.9
−0.8
−0.7
−0.6
−0.5
−0.4
−0.3




```
Mean of log(Q): -0.6000891301263563
Variance of log(Q): 0.010807831141708977
```

In [ ]: