Black-Scholes and Monte Carlo Simulation Project

Angela Lin, Bolun Tian, Nurul Srianda Putri

2025-11-29

Contents

1 Black-Scholes Method

1.1 Formula for European Call Option:

$$C = S_0\,\Phi\left(\frac{\ln(S_0/c) + (\rho + \sigma^2/2)t_0}{\sigma\sqrt{t_0}}\right) \;-\; c\,e^{-\rho t_0}\,\Phi\left(\frac{\ln(S_0/c) + (\rho - \sigma^2/2)t_0}{\sigma\sqrt{t_0}}\right)$$

where C is the price of the call option,
S_0 is the **spot price**,
c is the **strike price**,
σ is **volatility**,
ρ is the **risk-free rate (annual)**,
t_0 is the **expiry time**, $\Phi(x)$ is the **standard normal CDF**, where $\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{d}\exp\left(-\frac{t^2}{2}\right)\,dt$.

To simplify the model, we use the below equation:

$$C = S_0\,\Phi(d_1) \;-\; ce^{-pt_0}\,\Phi(d_2)$$

where

$$d_1 = \frac{1}{\sigma\sqrt{t_0}}\left[\ln\left(\frac{S_0}{c}\right) + \left(\rho + \frac{\sigma^2}{2}\right)t_0\right],$$

$$d_2 = \frac{1}{\sigma\sqrt{t_0}}\left[\ln\left(\frac{S_0}{c}\right) + \left(\rho - \frac{\sigma^2}{2}\right)t_0\right]$$

In our report, we have chosen to replicate the call option price of the European stock, ASML Holding N.V. (NASDAQ: ASML).

The last traded date of this call option is 24/09/2025 and will expire in 17/4/2026. The last price is 577.70.

[ ]:
```
# import libraries
import matplotlib.pyplot as plt
import numpy as np
from datetime import datetime
from scipy.stats import norm
```

[ ]:
```
# find contract expiry time
start = datetime(2025, 9, 24)
end = datetime(2026, 4, 17)

# difference in days
delta_days = (end - start).days

# convert to years (using 365-day convention)
t = delta_days / 365
```

[ ]:
```
# variables in black-scholes

S = 946.94 # spot price (current stock price as of 2025-9-24)
c = 380 # strike price
sigma = 1E-10 # implied volatility of stock
p = 0.04 # annual risk-free rate (Bank of England as of 2025-09-24)
t = t # expiry time in years

d1 = (np.log(S/c) + (p + sigma**2/2)*t)/(sigma*np.sqrt(t))
d2 = (np.log(S/c) + (p - sigma**2/2)*t)/(sigma*np.sqrt(t))

# cdf
phi_d1 = norm.cdf(d1)
phi_d2 = norm.cdf(d2)

# option call price
C = S*phi_d1 - c*np.exp(-p*t)*phi_d2
print(C)
```

```
575.3818054887142
```

Actual call option price is 577.70 (2025-09-24).

1.2 Formula for European Put Option:

$ P = c e^{-p t_0}, (-d_2) ;-; S_0 , (-d_1) $

where P is the put option price.

The last traded date of the put contract is 22/09/2025 and is set to expire in 17/04/2026. The last price 1.10.

[ ]:
```
# find contract expiry time
start = datetime(2025, 9, 22)
end = datetime(2026, 4, 17)

# difference in days
delta_days = (end - start).days

# convert to years (using 365-day convention)
t = delta_days / 365

sigma = 0.25
```

[ ]:
```
# cdf
phi_d1_p = norm.cdf(-d1)
phi_d2_p = norm.cdf(-d2)

# option call price
```

```
P = c*np.exp(-p*t)*phi_d2_p - S*phi_d1_p
print(P)
```

```
0.0
```

The actual price is 1.10.

1.3 Greeks for Call Options

In this section, we compute the main risk sensitivities of the European call option under the Black–Scholes model: **Delta**, **Gamma**, and **Vega**.

1.3.1 Delta (Call)

Delta measures the sensitivity of the option price to changes in the underlying stock price (S_0):

$$\Delta_{\text{call}} = \frac{\partial C}{\partial S_0}$$

For a European call option under the Black–Scholes model, Delta is given by:

$$\Delta_{\text{call}} = \Phi(d_1)$$

1.3.2 Gamma (Call)

Gamma measures the sensitivity of Delta to changes in the underlying stock price (S_0):

$$\Gamma = \frac{\partial^2 C}{\partial S_0^2}$$

For the Black–Scholes model, Gamma is:

$$\Gamma = \frac{\phi(d_1)}{S_0 \sigma \sqrt{t_0}}$$

Gamma is **identical for both call and put options**.

1.3.3 Vega (Call)

Vega measures the sensitivity of the option price to changes in volatility (sigma):

$$\text{Vega} = \frac{\partial C}{\partial \sigma}$$

Under the Black–Scholes model, Vega is given by:

$$\text{Vega} = S_0 \phi(d_1) \sqrt{t_0}$$

Vega is also **identical for call and put options**.

[ ]:
```
# Greeks for European call option
sigma = 1E-10
pdf_d1 = norm.pdf(d1)
delta_call = norm.cdf(d1)
gamma_call = pdf_d1 / (S * sigma * np.sqrt(t))
vega_call  = S * pdf_d1 * np.sqrt(t)
print (delta_call)
print (gamma_call)
print (vega_call)
```

```
1.0
0.0
0.0
```

1.4 Greeks for European Put Option

1.4.1 Delta (Put)

Delta for a put option measures the sensitivity of the put price to changes in the underlying stock price (S_0):

$$\Delta_{\text{put}} = \frac{\partial P}{\partial S_0}.$$

Under the Black–Scholes model, the Delta of a European put option is:

$$\Delta_{\text{put}} = \Phi(d_1) - 1 = -\Phi(-d_1)$$

1.4.2 Gamma (Put)

Gamma measures the sensitivity of Delta to changes in the stock price (S_0):

$$\Gamma_{\text{put}} = \frac{\partial^2 P}{\partial S_0^2}.$$

In the Black–Scholes model, Gamma is the **same** for call and put options:

$$\Gamma_{\text{put}} = \frac{\phi(d_1)}{S_0 \sigma \sqrt{t_0}}$$

1.4.3 Vega (Put)

Vega measures the sensitivity of the option price to changes in volatility ( ):

$$\text{Vega}_{\text{put}} = \frac{\partial P}{\partial \sigma}.$$

For both calls and puts, Vega is identical:

$$\text{Vega}_{\text{put}} = S_0 \phi(d_1) \sqrt{t_0}$$

Thus, while **Delta** differs between calls and puts (positive for calls, negative for puts), **Gamma** and **Vega** are identical for European call and put options under the Black–Scholes framework.

[ ]:
```
sigma = 0.25
delta_put = norm.cdf(d1) - 1
gamma_put = pdf_d1 / (S * sigma * np.sqrt(t))
vega_put  = S * pdf_d1 * np.sqrt(t)
print (delta_put)
print (gamma_put)
print (vega_put)
```

```
0.0
0.0
0.0
```

2 Monte Carlo Simulation

2.1 Theoretical Background

The underlying asset (ASML) is assumed to follow a **Geometric Brownian Motion (GBM)**, described by the stochastic differential equation:

$$dS_t = rS_t dt + \sigma S_t dB_t$$

Where: * r is the risk-free rate. * σ is the volatility. * dB_t is a Wiener process (Brownian motion).

Methodology: 1. **Simulate Paths:** We generate thousands of possible future prices for the stock at expiry (S_T) using the discretized solution to the GBM equation:

$$S_T = S_0 \cdot \exp\left((r - 0.5\sigma^2)T + \sigma\sqrt{T}Z\right)$$

where Z is a random variable from the standard normal distribution $\mathcal{N}(0,1)$. 2. **Calculate Payoff:** For each simulated path, we compute the option payoff: $\max(S_T - K, 0)$ for a call and $\max(K - S_T, 0)$ for a put. 3. **Discount:** The option price is the average of these payoffs, discounted back to the present at the risk-free rate r.

2.2 European Call Option

For call option, payoff $= \max(S_T - K, 0)$. Below is our implementation of Monte Carlo of call option pricing.

[ ]:
```
def monte_carlo_call(S, K, T, r, sigma, M):
    # generate M random number from distribution N(0,1)
    Z = np.random.standard_normal(M)

    # calculate terminal price
    ST = S * np.exp((r - 0.5 * sigma**2) * T + sigma * np.sqrt(T) * Z)

    # calculate payoff
    payoffs = np.maximum(ST - K, 0)
```

```
    # discounting to resent value
    discount_factor = np.exp(-r * T)
    prices = payoffs * discount_factor

    price_est = np.mean(prices)
    std_error = np.std(prices) / np.sqrt(M)

    return price_est, std_error, prices
```

[ ]:
```
# parameters
start_date = datetime(2025, 9, 24)
end_date = datetime(2026, 4, 17)
delta_days = (end_date - start_date).days
T = delta_days / 365

S0 = 946.94 # spot price (current stock price as of 2025-9-24)
K = 380 # strike price
sigma = 1E-10 # implied volatility
r = 0.04 # annual risk-free rate (Bank of England as of 2025-09-24)

# simulation settings
np.random.seed(42)
M = 100000 # number of simulations/paths

# Monte Carlo estimate
mc_price, se, _ = monte_carlo_call(S0, K, T, r, sigma, M)

print("Number of simulations (M): {}". format(M))
print("Estimated price based on Monte Carlo: {}".format(mc_price))
```

```
Number of simulations (M): 100000
Estimated price based on Monte Carlo: 575.3818054887828
```

2.3 European Put Option

The steps done here is similar to the steps for the call option. The only difference is how payoff is calculated. For put option, payoff $= \max(K - S_T, 0)$

[ ]:
```
def monte_carlo_put(S, K, T, r, sigma, M):
    # generate M random number from distribution N(0,1)
    Z = np.random.standard_normal(M)

    # calculate terminal price
    ST = S * np.exp((r - 0.5 * sigma**2) * T + sigma * np.sqrt(T) * Z)

    # calculate payoff
    payoffs = np.maximum(K - ST, 0)
```

```
    # discounting to resent value
    discount_factor = np.exp(-r * T)
    prices = payoffs * discount_factor

    price_est = np.mean(prices)
    std_error = np.std(prices) / np.sqrt(M)

    return price_est, std_error, prices
```

[ ]:
```
# parameters
start_date = datetime(2025, 9, 24)
end_date = datetime(2026, 4, 17)
delta_days = (end_date - start_date).days
T = delta_days / 365

S0 = 946.94 # spot price (current stock price as of 2025-9-24)
K = 380 # strike price
sigma = 0.25  # implied volatility
r = 0.04 # annual risk-free rate (Bank of England as of 2025-09-24)

# simulation settings
np.random.seed(42)
M = 100000 # number of simulations/paths

# Monte Carlo estimate
mc_price, se, _ = monte_carlo_put(S0, K, T, r, sigma, M)

print("Number of simulations (M): {}". format(M))
print("Estimated price based on Monte Carlo: {}".format(mc_price))
```

```
Number of simulations (M): 100000
Estimated price based on Monte Carlo: 0.0
```

2.4 Visualization of Monte Carlo Simulation Paths

To illustrate the stochastic process underlying our pricing model, we visualized the stock price evolution for both options. For clarity and rendering performance, we plotted a random subset of 50 paths, rather than the full 100,000 used in the calculation.

[ ]:
```
def simulate_paths(S, T, r, sigma, M, steps):
    dt = T / steps

    # initialize paths
    paths = np.zeros((M, steps + 1))
    paths[:, 0] = S

    # generate random shocks for all steps
```

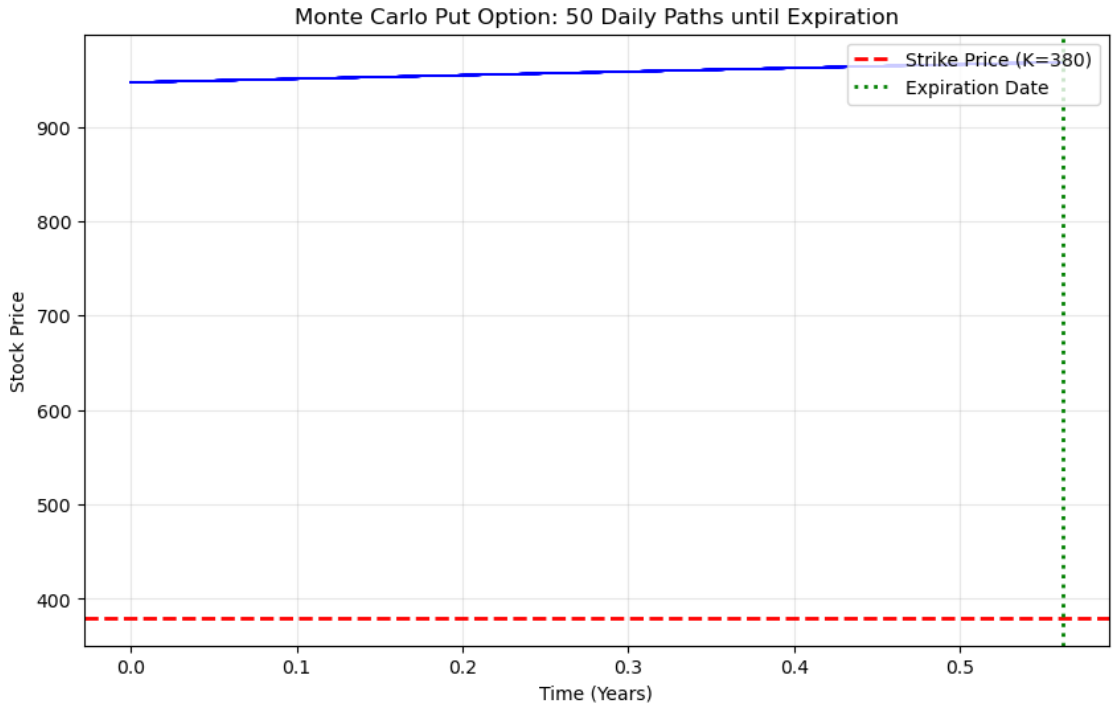
```
    Z = np.random.standard_normal((M, steps))

    # calculate price at each step
    for t in range(1, steps + 1):
        # S_t = S_{t-1} * exp(drift + diffusion)
        paths[:, t] = paths[:, t-1] * np.exp((r - 0.5 * sigma**2) * dt + sigma␣
↪* np.sqrt(dt) * Z[:, t-1])

    return paths

def visualize_paths(paths, vis_steps):
    plt.figure(figsize=(10, 6))

    # time axis (0 to T)
    time_axis = np.linspace(0, T, vis_steps + 1)

    # plot the paths
    plt.plot(time_axis, paths.T, color='blue', alpha=0.5, linewidth=1) #␣
↪Transpose paths so rows are time steps for plotting
    plt.axhline(y=K, color='red', linestyle='--', linewidth=2, label=f'Strike␣
↪Price (K={K})') # add the horizontal line for strike price
    plt.axvline(x=T, color='green', linestyle=':', linewidth=2,␣
↪label='Expiration Date') # add vertical line for expiration

    plt.title(f'Monte Carlo Put Option: 50 Daily Paths until Expiration')
    plt.xlabel('Time (Years)')
    plt.ylabel('Stock Price')
    plt.legend(loc='upper right')
    plt.grid(True, alpha=0.3)
    plt.show()
```

The first visualization is for the call option.

The plot displays paths that appear as a single upward line. This behavior is expected given the input parameter of $\sigma \approx 0$ (1×10^{-10}). With volatility that is practically zero, the random component of the Geometric Brownian Motion (dB_t) becomes negligible. Consequently, the model reduces to a deterministic exponential growth driven solely by the risk-free rate (r). The stock price simply drifts upward over time with no variance.

[ ]:
```
start_date = datetime(2025, 9, 24)
end_date = datetime(2026, 4, 17)
delta_days = (end_date - start_date).days
T = delta_days / 365

S0 = 946.94 # spot price (current stock price as of 2025-9-24)
K = 380 # strike price
sigma = 1E-10 # implied volatility
r = 0.04 # annual risk-free rate (Bank of England as of 2025-09-24)
```

```
vis_M = 50                  # Low number of paths (as requested)
vis_steps = delta_days      # One step per day until expiration
paths = simulate_paths(S0, T, r, sigma, vis_M, vis_steps)
visualize_paths(paths, vis_steps)
```



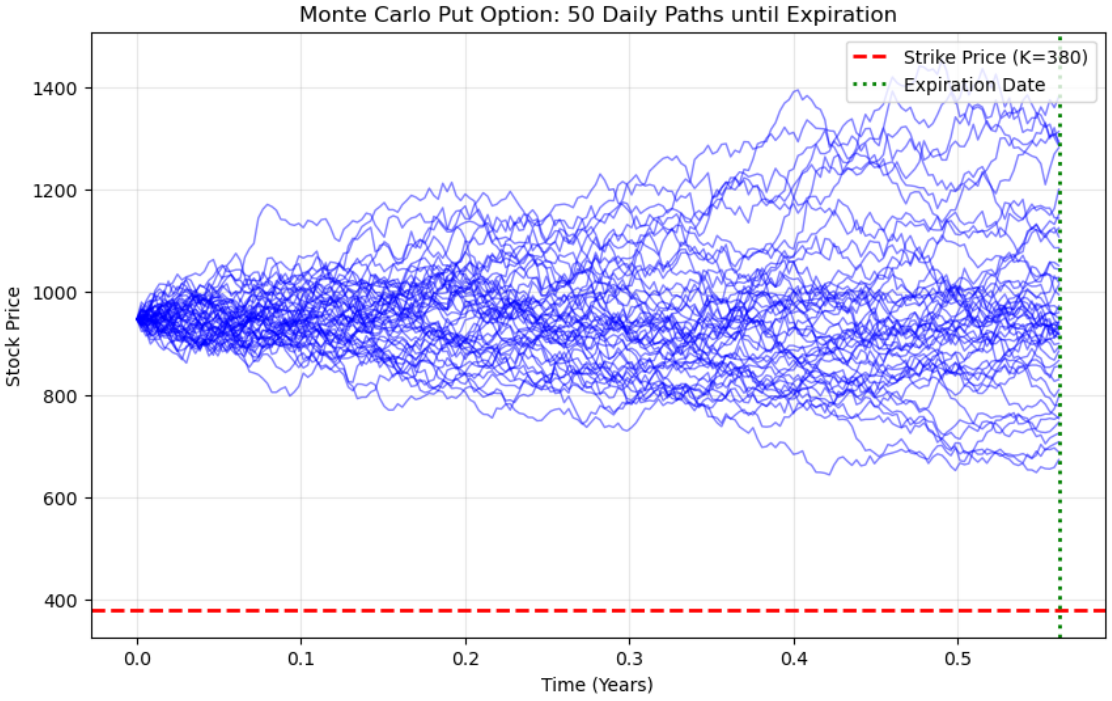
The second visualization is for the Put Option.

Here, we utilizes a realistic volatility of $\sigma = 0.25$, resulting in the characteristic "cone of uncertainty" where price paths diverge significantly over time.

One important thing is, this plot explains the valuation result of 0.00. The option is deeply Out-of-the-Money (OTM), that is, the initial spot price ($S_0 \approx 947$) is more than double the strike price ($K = 380$). For the (put) option to have value, the stock price would need to crash below strike, represented by the red dashed line ($K = 380$). As illustrated, even with significant volatility, none of the simulated paths decline steeply enough to reach the profit zone. Therefore, the terminal payoff $\max(K - S_T, 0)$ is zero for every path.

[ ]:
```
end_date = datetime(2026, 4, 17)
delta_days = (end_date - start_date).days
T = delta_days / 365

S0 = 946.94 # spot price (current stock price as of 2025-9-24)
K = 380 # strike price
sigma = 0.25 # implied volatility
```

```
r = 0.04 # annual risk-free rate (Bank of England as of 2025-09-24)

vis_M = 50                  # Low number of paths (as requested)
vis_steps = delta_days      # One step per day until expiration
paths = simulate_paths(S0, T, r, sigma, vis_M, vis_steps)
visualize_paths(paths, vis_steps)
```



3 Monte Carlo Convergence Analysis Against Black–Scholes Pricing

In this section, we analyse the convergence behaviour of the Monte Carlo estimator for pricing a European put option. The Monte Carlo results are compared with the analytical benchmark obtained from the Black–Scholes formula, using consistent model parameters.

To assess convergence, we compute simulated put prices at increasing numbers of simulation paths:

$$M \in \{10^2,\ 5\times10^2,\ 10^3,\ 5\times10^3,\ 10^4,\ 5\times10^4,\ 10^5\}.$$

3.1 Convergence of Monte Carlo Put Price

Across all simulation sizes, the estimated Put price remains at 0.00.

Reason: - The spot price is far above the strike price. - In all simulated paths, the stock price never falls below the strike at maturity. - Therefore, the payoff max(K − S_T, 0) is always zero.

Observation: As the number of simulations increases, the Monte Carlo estimate remains equal to the Black–Scholes theoretical price (which is also 0). This confirms that the option has no value under the given parameters.

3.2 Error Decay Between Monte Carlo and Black–Scholes

The second figure shows the absolute difference between the Monte Carlo Put estimate and the Black–Scholes analytical price.

Because both values are identically zero, the absolute error: | MC Price − BS Price | is also zero at every simulation size.

As a result, the error curve collapses to a horizontal line at 0.

3.3 Standard Error of the Monte Carlo Estimator

The last figure reports the estimated standard error of the Monte Carlo estimator.

For every simulation size, this value is also zero.

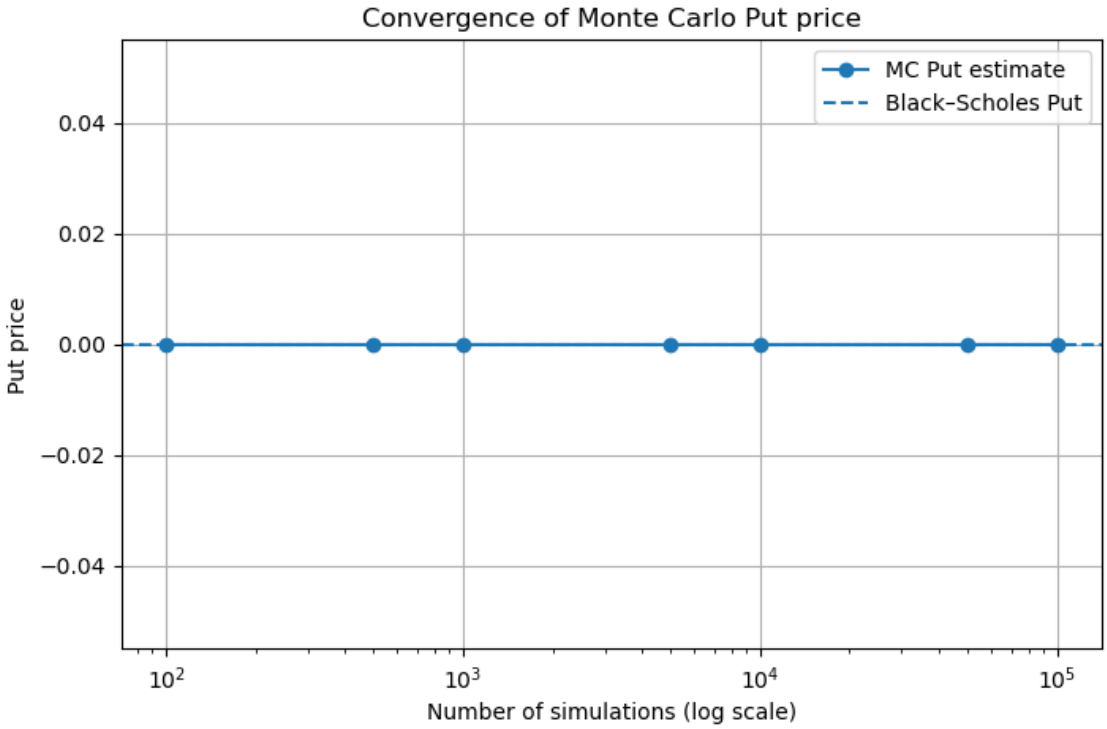
3.4 Final Conclusion

- The Put option is worthless under the current market parameters.
- Monte Carlo simulation and the Black–Scholes model both yield a Put price of 0.00.
- Absolute error and standard error remain exactly 0 across all simulation sizes.
- Convergence is trivial in this case because the payoff distribution is degenerate (always zero).

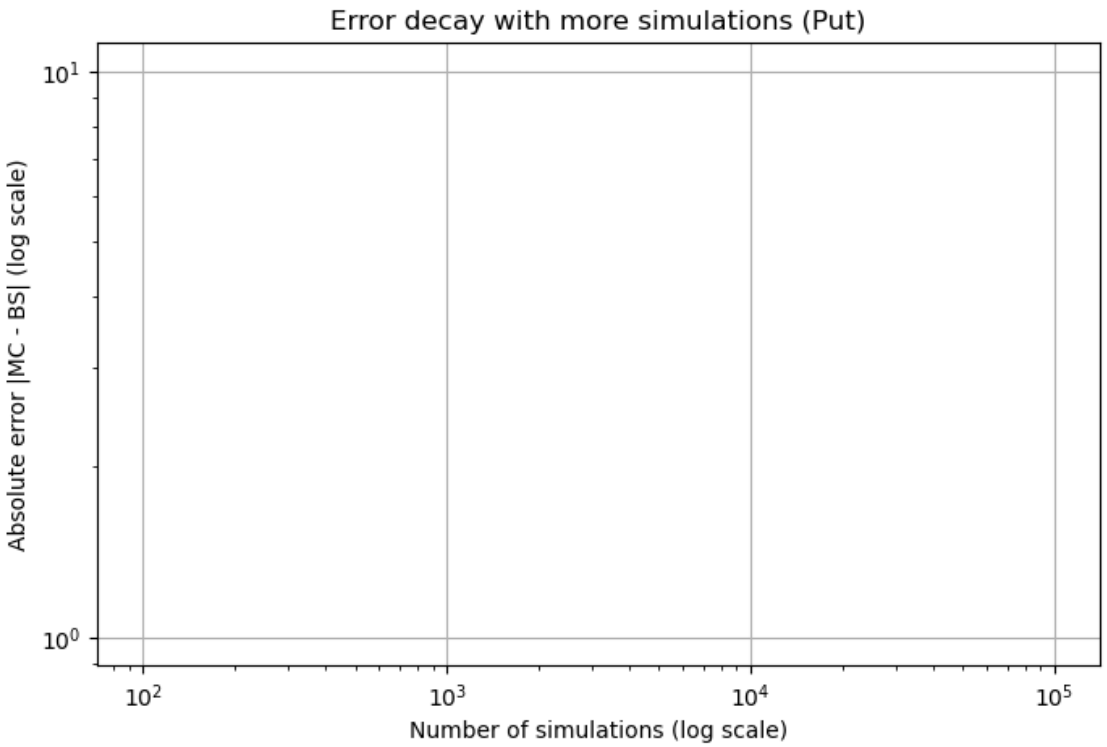
Although this scenario does not produce visible convergence dynamics, it does validate the correctness of the Monte Carlo implementation. For more meaningful convergence behaviour, parameters (strike, volatility, or spot price) would need to be adjusted to create a non-zero payoff probability.

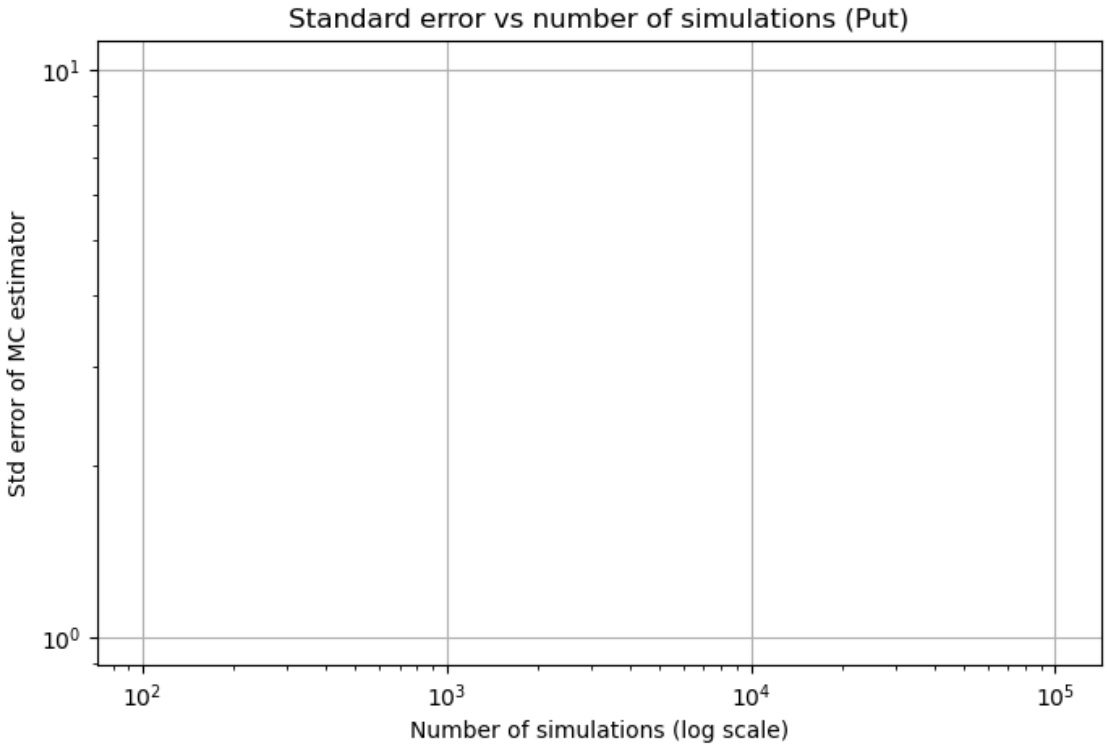
[ ]:
```
bs_call = C
bs_put = P

M_list = [100, 500, 1000, 5000, 10000, 50000, 100000]

mc_call_est, mc_put_est = [], []
mc_call_err, mc_put_err = [], []
mc_call_se, mc_put_se = [], []

for M in M_list:
    np.random.seed(42)
    c_price, c_se, _ = monte_carlo_call(S0, K, T, r, sigma, M)
    p_price, p_se, _ = monte_carlo_put(S0, K, T, r, sigma, M)

    mc_call_est.append(c_price)
    mc_put_est.append(p_price)
    mc_call_err.append(abs(c_price - bs_call))
    mc_put_err.append(abs(p_price - bs_put))
    mc_call_se.append(c_se)
    mc_put_se.append(p_se)
```

```
sigma= 0.25
# As M increases, price converges to bs_put
plt.figure(figsize=(8,5))
plt.plot(M_list, mc_put_est, marker='o', label='MC Put estimate')
plt.axhline(bs_put, linestyle='--', label='Black-Scholes Put')
plt.xscale('log')
plt.xlabel('Number of simulations (log scale)')
plt.ylabel('Put price')
plt.title('Convergence of Monte Carlo Put price')
plt.legend()
plt.grid(True)
plt.show()

# As M increases, Absolute error changes
plt.figure(figsize=(8,5))
plt.loglog(M_list, mc_put_err, marker='o')
plt.xlabel('Number of simulations (log scale)')
plt.ylabel('Absolute error |MC - BS| (log scale)')
plt.title('Error decay with more simulations (Put)')
plt.grid(True)
plt.show()

# As M increases, Standard error changes
plt.figure(figsize=(8,5))
plt.loglog(M_list, mc_put_se, marker='o')
plt.xlabel('Number of simulations (log scale)')
plt.ylabel('Std error of MC estimator')
plt.title('Standard error vs number of simulations (Put)')
plt.grid(True)
plt.show()
```




```
/opt/anaconda3/lib/python3.12/site-packages/IPython/core/pylabtools.py:170:
UserWarning: Data has no positive values, and therefore cannot be log-scaled.
  fig.canvas.print_figure(bytes_io, **kw)
```


Error decay with more simulations (Put)
Absolute error |MC - BS| (log scale)
10^1
10^0
10^2
10^3
10^4
10^5
Number of simulations (log scale)


Standard error vs number of simulations (Put)
Std error of MC estimator
10^1
10^0
10^2
10^3
10^4
10^5
Number of simulations (log scale)

4 Monte Carlo Convergence Analysis Against Black–Scholes Pricing

In this section, we analyse the convergence behaviour of the Monte Carlo estimator for pricing a European call option. The Monte Carlo results are compared with the analytical benchmark obtained from the Black–Scholes formula, using consistent model parameters.

To assess convergence, we compute simulated call prices at increasing numbers of simulation paths:

$$M \in \{10^2,\ 5\times 10^2,\ 10^3,\ 5\times 10^3,\ 10^4,\ 5\times 10^4,\ 10^5\}.$$

4.1 Monte Carlo Call Price vs. Black–Scholes Value

The first plot illustrates how the Monte Carlo estimate approaches the analytical Black–Scholes price as the number of simulations increases.

For small values of M, the simulated call prices exhibit noticeable fluctuations due to high sampling variance. As M grows, these fluctuations diminish and the Monte Carlo estimates stabilise near the theoretical value.

This behaviour is expected: the Monte Carlo estimator is unbiased, meaning that

$$\mathbb{E}[\hat{C}_{\mathrm{MC}}] = C_{\mathrm{BS}},$$

and therefore converges in probability to the true Black–Scholes price.

> As the number of simulated paths increases, the Monte Carlo estimates converge towards the Black–Scholes benchmark.

4.2 Error Decay: $|C_{\mathbf{MC}} - C_{\mathbf{BS}}|$ vs. Number of Simulations

The second figure presents, on a log–log scale, the absolute difference between the Monte Carlo price and the Black–Scholes analytical value.

We observe that the pricing error declines approximately linearly with respect to M in log–log space, consistent with the theoretical Monte Carlo convergence rate:

$$|C_{\mathrm{MC}} - C_{\mathrm{BS}}| = O\left(\frac{1}{\sqrt{M}}\right).$$

> When the number of simulated paths increases, random estimation noise averages out, resulting in increasingly accurate pricing estimates.

4.3 Standard Error of the Monte Carlo Estimator

The final plot shows the standard error of the Monte Carlo estimator as a function of M, also in log–log scale.

Again, the error declines close to linearly, which is consistent with

$$\mathrm{Std}(\hat{C}_{\mathrm{MC}}) = O\left(\frac{1}{\sqrt{M}}\right).$$

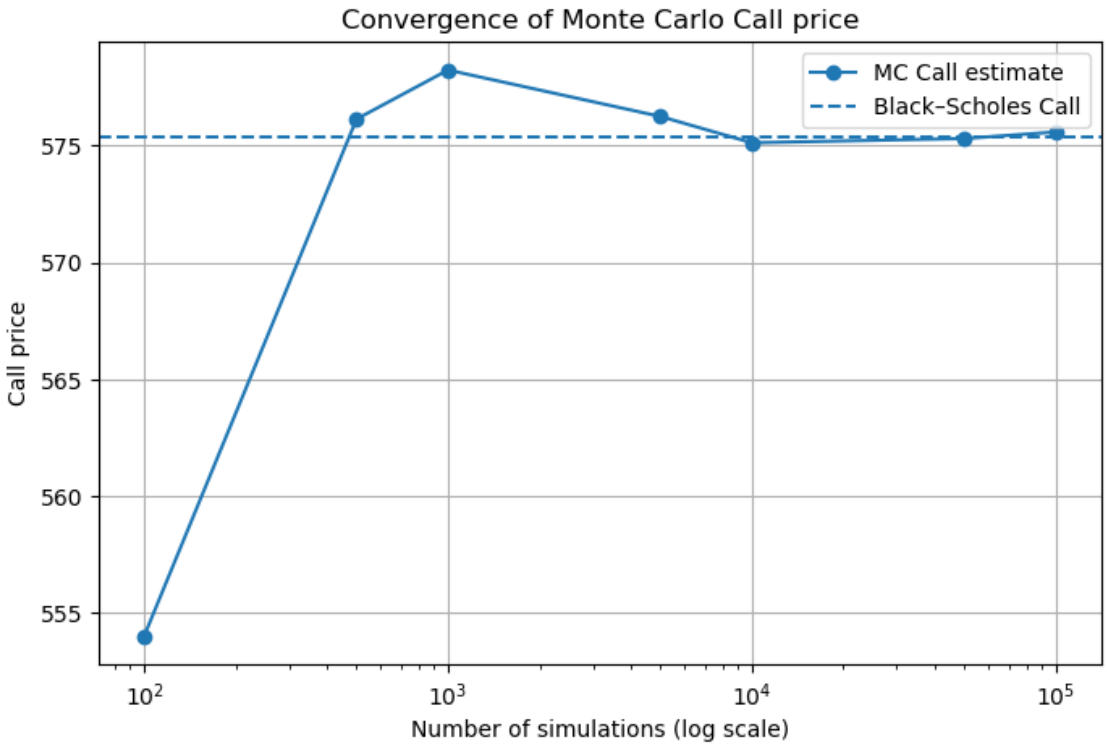
This supports the empirical evidence that Monte Carlo accuracy improves when the number of simulated paths increases. Lower standard errors indicate greater stability and reliability in the estimator.

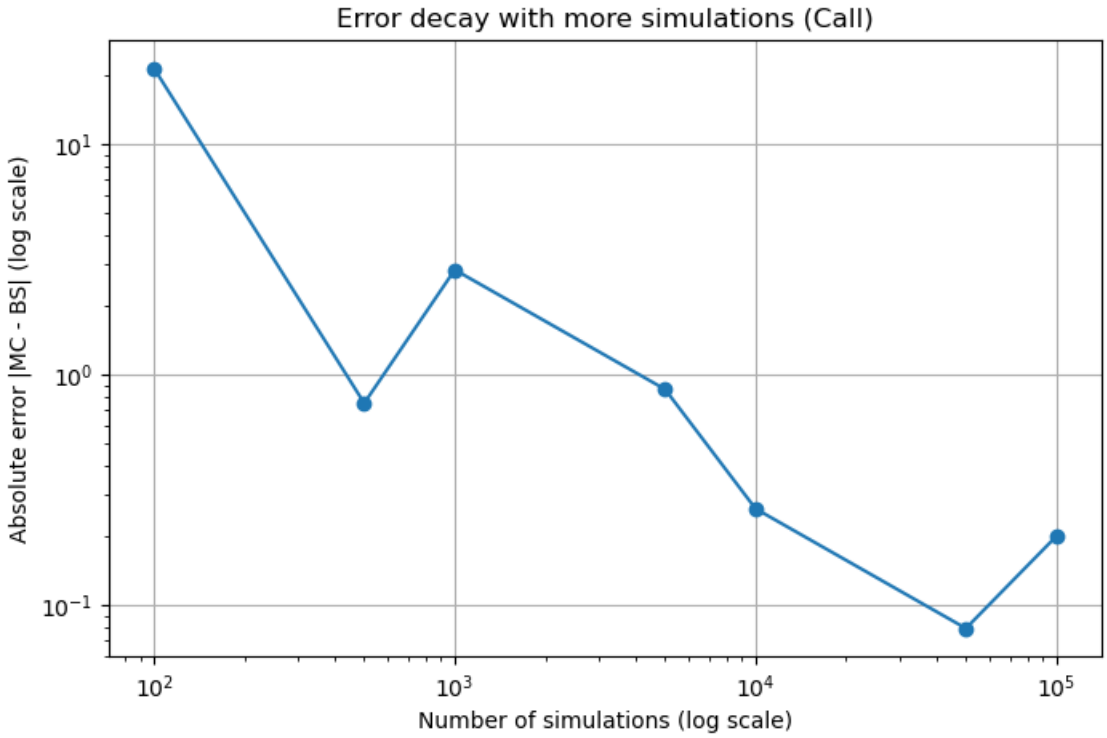
4.4 Summary of Convergence Findings

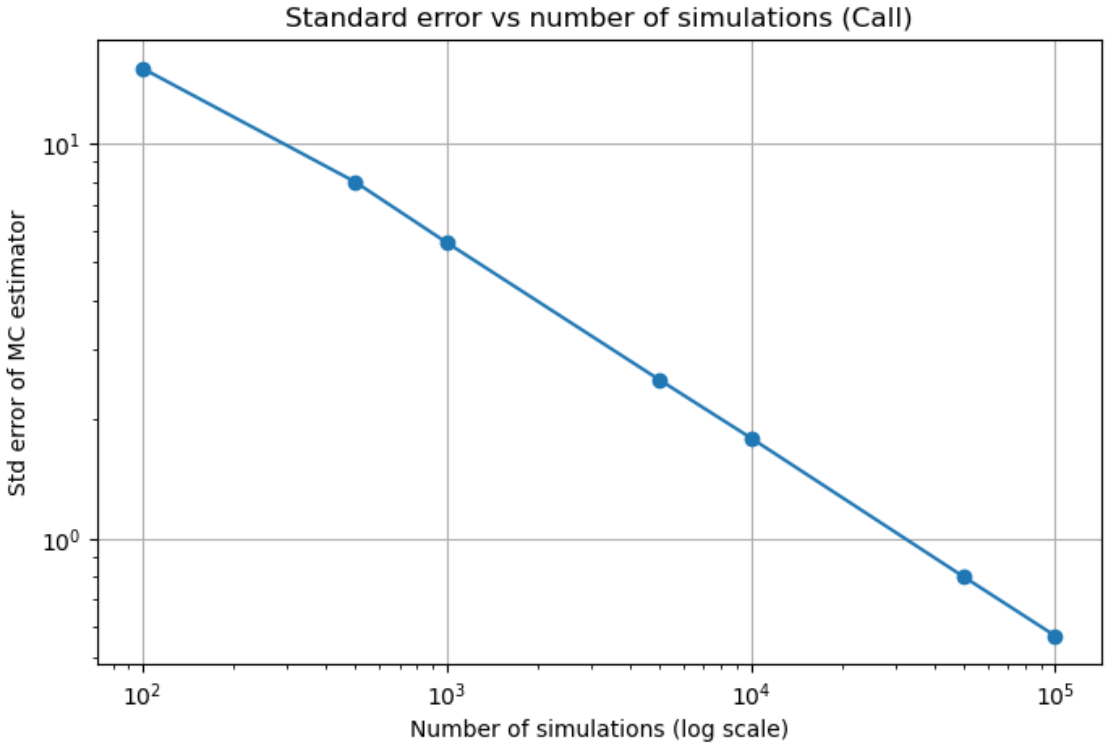
- **Consistent convergence:**
  The Monte Carlo estimated prices approach the analytical Black–Scholes call price as the number of simulated paths increases.
- **Expected convergence rate:**
  The absolute price error decays at approximately $1/\sqrt{M}$, matching theoretical results from Monte Carlo estimation.
- **Estimator precision improvement:**
  The standard error of the estimator decreases with increasing M, confirming better numerical stability at higher simulation counts.
- **Practical implication:**
  Achieving significantly higher precision requires a disproportionately large increase in simulation paths. For example, obtaining a $10\times$ improvement in accuracy typically requires around $100\times$ more paths.

[ ]:
```
sigma = 0

plt.figure(figsize=(8,5))
plt.plot(M_list, mc_call_est, marker='o', label='MC Call estimate')
plt.axhline(bs_call, linestyle='--', label='Black-Scholes Call')
plt.xscale('log')
plt.xlabel('Number of simulations (log scale)')
plt.ylabel('Call price')
plt.title('Convergence of Monte Carlo Call price')
plt.legend()
plt.grid(True)
plt.show()


plt.figure(figsize=(8,5))
plt.loglog(M_list, mc_call_err, marker='o')
plt.xlabel('Number of simulations (log scale)')
plt.ylabel('Absolute error |MC - BS| (log scale)')
plt.title('Error decay with more simulations (Call)')
plt.grid(True)
plt.show()


plt.figure(figsize=(8,5))
plt.loglog(M_list, mc_call_se, marker='o')
plt.xlabel('Number of simulations (log scale)')
```

```
plt.ylabel('Std error of MC estimator')
plt.title('Standard error vs number of simulations (Call)')
plt.grid(True)
plt.show()
```




Error decay with more simulations (Call)
Absolute error |MC - BS| (log scale)
10^{1}
10^{0}
10^{-1}
10^{2}
10^{3}
10^{4}
10^{5}
Number of simulations (log scale)


Standard error vs number of simulations (Call)
Std error of MC estimator
10^{1}
10^{0}
10^{2}
10^{3}
10^{4}
10^{5}
Number of simulations (log scale)