Monte Carlo Simulation for Option Pricing

In this tutorial, we explore how Monte Carlo simulation can be applied to valuing financial derivatives.

Pricing derivatives through Monte Carlo methods relies on the principles of risk-neutral valuation, where asset prices are simulated under the risk-neutral measure to determine their fair value.

What is Monte Carlo Simulation

Monte Carlo simulation is a numerical technique that uses repeated random sampling to approximate quantities that are difficult or impossible to compute analytically, such as expectations, probabilities, and integrals. It originated in the 1940s at Los Alamos, where Stanislaw Ulam, John von Neumann, and Nicholas Metropolis developed it for neutron transport problems; the name references the Monte Carlo casino to highlight its reliance on chance.

In modern finance, it is used to value derivatives and assess portfolio risk by simulating large numbers of risk-neutral price paths and averaging discounted payoffs. The method naturally accommodates complex features like path dependence, multiple risk factors, and discontinuities (e.g., jumps or barriers), where closed-form solutions are unavailable.

Monte Carlo as a financial tool

The Monte Carlo simulation as briefly explained above is a numerical method for solving probabilistic problems by generating thousands or even millions of random scenarios to estimate statistical properties such as expected values, variances, and probabilities of certain outcomes.

Introduced to finance by Boyle in 1977, it became a powerful tool for valuing complex derivatives that lack analytical pricing formulas. By simulating numerous risk-neutral price paths, Monte Carlo methods approximate expected payoffs and discount them to present value.

This flexibility allows the modeling of multiple sources of uncertainty and realistic asset behaviors, including price jumps and path dependence, making it especially valuable for pricing exotic options and assessing portfolio risk.

Code

Step 1: Install necessary packages

```
%pip install -q numpy scipy matplotlib pytest
!pip install --upgrade pandas_datareader
```

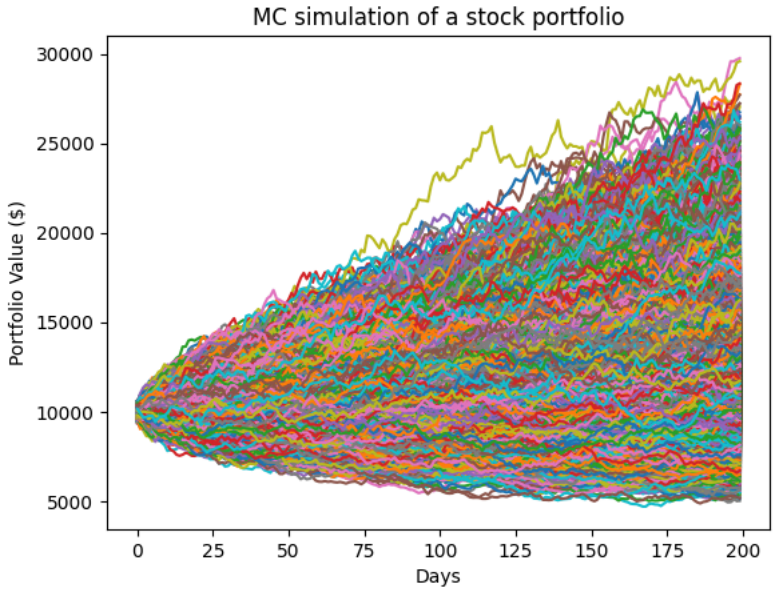
```
Requirement already satisfied: pandas_datareader in /usr/local/lib/python3.12/dist-packages (0.10.0)
Requirement already satisfied: lxml in /usr/local/lib/python3.12/dist-packages (from pandas_datareader) (5.4.0)
Requirement already satisfied: pandas>=0.23 in /usr/local/lib/python3.12/dist-packages (from pandas_datareader) (2.2.2)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.12/dist-packages (from pandas_datareader) (2.32.4)
Requirement already satisfied: numpy>=1.26.0 in /usr/local/lib/python3.12/dist-packages (from pandas>=0.23->pandas_datareade
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas>=0.23->pandas_
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas>=0.23->pandas_datareader
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas>=0.23->pandas_dataread
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests>=2.19.0->p
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests>=2.19.0->pandas_datare
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests>=2.19.0->pandas_
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests>=2.19.0->pandas_
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas>=0.2
```

```
import math
import numpy as np
import pandas as pd
import datetime
import scipy.stats as stats
import matplotlib.pyplot as plt
from pandas_datareader import data as pdr
```

```
# import data
import yfinance as yf

def get_data(stocks, start, end):
    stockData = yf.download(stocks, start=start, end=end)
```

```
    stockData = stockData['Close']
    returns = stockData.pct_change()
    meanReturns = returns.mean()
    covMatrix = returns.cov()
    return meanReturns, covMatrix

stockList = ['AAPL', 'TSLA', 'NVDA', 'MSFT', 'WMT', 'JPM']
stocks = stockList # Removed '.AX' suffix
endDate = datetime.datetime.now()
startDate = endDate - datetime.timedelta(days=300)

meanReturns, covMatrix = get_data(stocks, startDate, endDate)

weights = np.random.random(len(meanReturns))
weights /= np.sum(weights)

# Monte Carlo Method
mc_sims = 80000 # number of simulations
T = 200 #timeframe in days

meanM = np.full(shape=(T, len(weights)), fill_value=meanReturns)
meanM = meanM.T

portfolio_sims = np.full(shape=(T, mc_sims), fill_value=0.0)

initialPortfolio = 10000

for m in range(0, mc_sims):
    Z = np.random.normal(size=(T, len(weights)))#uncorrelated RV's
    L = np.linalg.cholesky(covMatrix) #Cholesky decomposition to Lower Triangular Matrix
    dailyReturns = meanM + np.inner(L, Z) #Correlated daily returns for individual stocks
    portfolio_sims[:,m] = np.cumprod(np.inner(weights, dailyReturns.T)+1)*initialPortfolio

plt.plot(portfolio_sims)
plt.ylabel('Portfolio Value ($)')
plt.xlabel('Days')
plt.title('MC simulation of a stock portfolio')
plt.show()

def mcVaR(returns, alpha=5):
    """ Input: pandas series of returns
        Output: percentile on return distribution to a given confidence level alpha
    """
    if isinstance(returns, pd.Series):
        return np.percentile(returns, alpha)
    else:
        raise TypeError("Expected a pandas data series.")

def mcCVaR(returns, alpha=5):
    """ Input: pandas series of returns
        Output: CVaR or Expected Shortfall to a given confidence level alpha
    """
    if isinstance(returns, pd.Series):
        belowVaR = returns <= mcVaR(returns, alpha=alpha)
        return returns[belowVaR].mean()
    else:
        raise TypeError("Expected a pandas data series.")


portResults = pd.Series(portfolio_sims[-1,:])

VaR = initialPortfolio - mcVaR(portResults, alpha=5)
CVaR = initialPortfolio - mcCVaR(portResults, alpha=5)

print('VaR_5 ${}'.format(round(VaR,2)))
print('CVaR_5 ${}'.format(round(CVaR,2)))
```

```
/tmp/ipython-input-76381793.py:5: FutureWarning: YF.download() has changed argument auto_adjust default to True
  stockData = yf.download(stocks, start=start, end=end)
[*********************100%***********************]  6 of 6 completed
```




```
VaR_5 $1517.33
CVaR_5 $2227.19
```

Explanation of the code

Import Libraries: The code starts by importing necessary libraries such as yfinance for downloading stock data, numpy for numerical operations, pandas for data manipulation, datetime for handling dates, scipy.stats for statistical functions, and matplotlib.pyplot for plotting.

get_data Function: This function downloads historical stock data for a given list of tickers and date range using yfinance. It then calculates the daily percentage change (returns), the mean of these returns, and the covariance matrix of the returns. These are essential inputs for the simulation.

Data Preparation: italicised textA list of stock tickers (stockList) is defined. The end date for data download is set to the current date, and the start date is set to 300 days prior. The get_data function is called to get the mean returns and covariance matrix for the specified stocks and date range. Random weights are assigned to each stock in the portfolio, and then normalized so they sum up to 1.

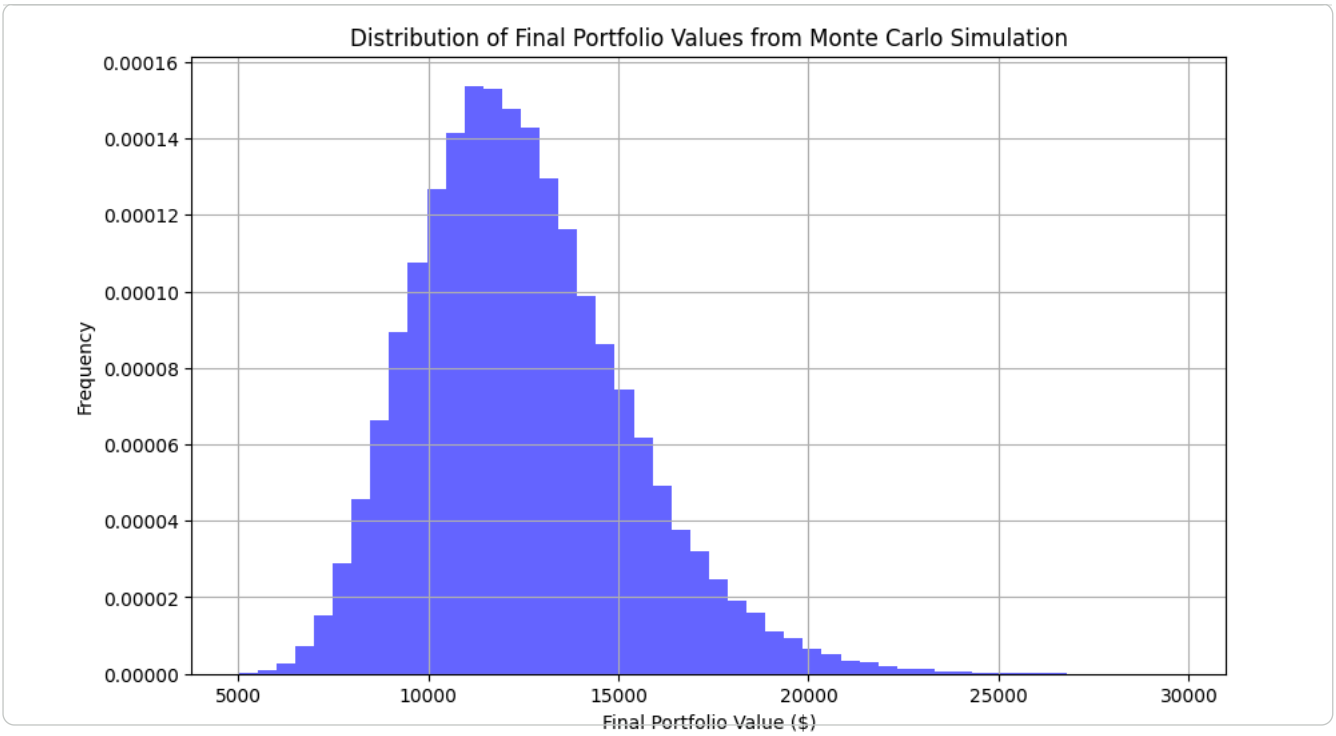
Monte Carlo Simulation Setup: mc_sims is set to 800000, representing the number of simulation runs. T is set to 200, representing the timeframe in days for the simulation. meanM is created as a matrix of mean returns, replicated for each day in the timeframe. portfolio_sims is initialized as a matrix to store the simulated portfolio values over time for each simulation run. initialPortfolio is set to 10000, the starting value of the portfolio.

Running the Simulation: A for loop runs for the specified number of simulations (mc_sims). Inside the loop: Z generates random numbers from a normal distribution, representing uncorrelated random shocks. L is the result of the Cholesky decomposition of the covariance matrix. This is used to introduce correlation between the stock returns in the simulation, matching the historical relationships. dailyReturns calculates the correlated daily returns for each stock by combining the mean returns with the correlated random shocks (np.inner(L, Z)). portfolio_sims is updated by calculating the cumulative product of the daily portfolio returns (which are the weighted sum of individual stock returns) plus 1, and multiplying by the initial portfolio value. This simulates the portfolio value path over time for each simulation. Plotting Results: The code plots all the simulated portfolio value paths over the T days.

VaR and CVaR Calculation: mcVaR and mcCVaR functions are defined to calculate the Value at Risk (VaR) and Conditional Value at Risk (CVaR) respectively, at a given confidence level (alpha). portResults takes the final portfolio values from all simulations. VaR and CVaR are calculated by subtracting the respective risk metrics from the initial portfolio value.

Printing Results: The calculated VaR and CVaR values are printed. In essence, the code simulates many possible future scenarios for the stock prices based on their historical mean returns and covariance. By running thousands of these simulations, it generates a distribution of potential portfolio outcomes, from which the VaR and CVaR are calculated to quantify the potential downside risk.

```
plt.figure(figsize=(10, 6))
plt.hist(portResults, bins=50, density=True, alpha=0.6, color='b')
plt.xlabel('Final Portfolio Value ($)')
plt.ylabel('Frequency')
plt.title('Distribution of Final Portfolio Values from the Monte Carlo Simulation')
plt.grid(True)
plt.show()
```



Double-click (or enter) to edit

```
import numpy as np # Import numpy again to ensure it's available
import pandas as pd # Import pandas to use pd.Series

# Recalculate portResults if it's not available
# This assumes portfolio_sims is available from a previous run of the simulation cell.
# If portfolio_sims is also not defined, the main simulation cell needs to be run first.
if 'portfolio_sims' in locals():
    portResults = pd.Series(portfolio_sims[-1,:])
    print(f"Minimum Final Portfolio Value: ${np.min(portResults):.2f}")
    print(f"Maximum Final Portfolio Value: ${np.max(portResults):.2f}")
    print(f"Average Final Portfolio Value: ${np.mean(portResults):.2f}")
    print(f"Median Final Portfolio Value (50th percentile): ${np.percentile(portResults, 50):.2f}")
    print(f"5th Percentile Final Portfolio Value (used for VaR): ${np.percentile(portResults, 5):.2f}")
else:
    print("Error: 'portfolio_sims' is not defined. Please run the main Monte Carlo simulation cell first.")
```

```
Minimum Final Portfolio Value: $5019.88
Maximum Final Portfolio Value: $29747.45
Average Final Portfolio Value: $12474.12
Median Final Portfolio Value (50th percentile): $12178.76
5th Percentile Final Portfolio Value (used for VaR): $8482.67
```

Let's break it down and go through what every portfolio value means:

Minimum Final Portfolio Value at $5019.88:

This represents the lowest final portfolio value recorded across all 80,000 Monte Carlo simulations. In the most unfavorable scenario modeled, the initial investment of $10,000 ended with a final value of $5,019.88 after 200 days. This outcome reflects the worst-case scenario observed in the simulation.

Maximum Final Portfolio Value at $29747.45

This is the highest final portfolio value recorded across all 80,000 Monte Carlo simulations. In the most favorable scenario simluated by the model, the initial $10,000 portfolio grew to $29,747.45 after 200 days. This represents the best-case ooutcome observed in the simulation

Average Final Portfolio Value at $12474.12

This is the arithemtic mean of all 80,000 simulated final portfolio values. Meaning on average across all simulations, the initial $10,000 portfolio is expected to reach a value of approximately $12,474.12 after 200 days based on the historical data and the simulation model

Median Final Portfolio Value (50th percentile) at $12178.76

This is the middle value of the simulated final portfolio values when they are sorted from lowest to highest. Fifty percent of the simulated outcomes were below this value, and fifty percent were above it. Half of the time, the initial $10,000 portfolio is expected to end up with a value less than or equal to $12,178.76 after 200 days, and half of the time it's expected to be greater than or equal to this value. The fact that the median $12,178.76 is slightly lower than the average $12,474.12 suggests that the distribution of outcomes is slightly skewed, with some larger positive outcomes pulling the average up.

5th Percentile Final Portfolio Value (used for VaR) at $8482.67

This is the value below which 5% of the simulated final portfolio values fall This means that there is a 5% chance, according to this simulation, that the initial $10,000 portfolio could end up with a value of $8482.67 or less after 200 days. This is a key number for calculating the Value at Risk (VaR), which represents a potential loss threshold. In this case, the 5% VaR is the initial portfolio value minus this 5th percentile value $8482.67 or less after 200 days. This is a key number for calculating the Value at Risk (VaR), which represents a potential loss threshold. In this case, the 5% VaR is the initial portfolio value minus this 5th percentile value ($10,000 - $8482.67 = $$8482.67 = $1517.33), meaning our portfolio could expect to lose at most $1517.33 with 95% confidence over 200 days, based on this model.

Explaining the Application of Monte Carlo Simulation to Option Pricing

Describing the theoretical basis for using Monte Carlo methods to price options, including the risk-neutral valuation framework.

```
%%markdown

# Monte Carlo Simulation for Option Pricing: Theoretical Basis

Monte Carlo simulation is a powerful tool for pricing financial options, especially those with complex features or path dep

The theoretical foundation for using Monte Carlo simulation in option pricing is the **risk-neutral valuation framework**.

Here's how Monte Carlo simulation is applied within the risk-neutral valuation framework to price an option:

1.  **Model the Underlying Asset Price:** Choose a stochastic process to model the behavior of the underlying asset price u
2.  **Generate Price Paths:** Simulate a large number of independent price paths for the underlying asset from the present
3.  **Calculate Payoff for Each Path:** For each simulated price path, calculate the option's payoff at expiration. The pay
4.  **Calculate the Average Payoff:** Compute the average of the payoffs calculated in the previous step across all simulat
5.  **Discount to Present Value:** Discount the average payoff back to the present time using the risk-free rate. This disc

Monte Carlo methods are particularly well-suited for pricing **exotic options**, which often have path-dependent payoffs (n
```

Monte Carlo Simulation for Option Pricing: Theoretical Basis

Monte Carlo simulation is a powerful tool for pricing financial options, especially those with complex features or path dependencies that make analytical solutions difficult or impossible. The core idea is to simulate a large number of possible future price paths for the underlying asset and then calculate the option's payoff for each path. The average of these payoffs, discounted back to the present at the risk-free rate, provides an estimate of the option's fair value.

The theoretical foundation for using Monte Carlo simulation in option pricing is the **risk-neutral valuation framework**. This fundamental concept in financial mathematics states that the price of any derivative security is equal to the expected value of its future payoff, discounted at the risk-free rate, under a hypothetical probability measure called the risk-neutral measure (often denoted by Q). Under this measure, all assets are expected to grow at the risk-free rate.

Here's how Monte Carlo simulation is applied within the risk-neutral valuation framework to price an option:

1. **Model the Underlying Asset Price:** Choose a stochastic process to model the behavior of the underlying asset price under the risk-neutral measure. A common choice is Geometric Brownian Motion, which assumes that the logarithm of the asset price follows a Brownian motion with a drift equal to the risk-free rate and a volatility equal to the asset's volatility.
2. **Generate Price Paths:** Simulate a large number of independent price paths for the underlying asset from the present time to the option's expiration date, using the chosen stochastic process and the risk-neutral drift and volatility.
3. **Calculate Payoff for Each Path:** For each simulated price path, calculate the option's payoff at expiration. The payoff depends on the type of option (e.g., for a call option, the payoff is the maximum of zero and the difference between the asset price at expiration and the strike price).
4. **Calculate the Average Payoff:** Compute the average of the payoffs calculated in the previous step across all simulated paths.
5. **Discount to Present Value:** Discount the average payoff back to the present time using the risk-free rate. This discounted average payoff is the Monte Carlo estimate of the option's price.

Monte Carlo methods are particularly well-suited for pricing **exotic options**, which often have path-dependent payoffs (meaning the payoff depends not just on the final price but on the price movements over the life of the option) or complex features that are difficult to handle with analytical models. Examples include Asian options (whose payoff depends on the average price of the underlying asset over a period) and barrier options (whose payoff

Implement a basic monte carlo option pricing model

Written below is a written code to price a simple option (e.g., a European call or put) using Monte Carlo simulation, building upon the existing simulation framework.

Reasoning: The goal is to price a European call option using Monte Carlo simulation. This requires adapting the existing simulation to a risk-neutral framework, calculating payoffs, averaging, and discounting. These steps can be combined into a single code block.

```
# Define option parameters
strike_price = 150  # Example strike price
time_to_expiration = 1 # Time to expiration in years
risk_free_rate = 0.05 # Example risk-free rate
# Volatility will be derived from the historical data (covMatrix)

# Adapt simulation for risk-neutral pricing
# We need to calculate the expected future price under the risk-neutral measure.
# For a single asset (or effectively a portfolio treated as a single asset for this purpose)
# the risk-neutral drift is the risk-free rate.
# However, since our current simulation generates portfolio paths based on historical
# mean returns and covariance, we'll adapt this by changing the drift to the risk-free rate.
# The volatility is implicitly handled by the Cholesky decomposition of the covariance matrix.

# For simplicity in this example, let's assume we are pricing an option on the portfolio
# value itself, with the initial portfolio value as the current "asset price".
# A more rigorous approach would be to simulate individual stock prices under risk-neutrality
# and then calculate the portfolio value, but this adaptation demonstrates the principle.

# Use the risk-free rate as the drift for the portfolio simulation under risk-neutrality
# We can approximate the daily risk-free rate
daily_risk_free_rate = risk_free_rate / 252 # Assuming 252 trading days in a year

# Recalculate meanM with the risk-free rate as the drift
meanM_rn = np.full(shape=(T, len(weights)), fill_value=daily_risk_free_rate)
meanM_rn = meanM_rn.T


# Re-run the Monte Carlo simulation with the risk-neutral drift
portfolio_sims_rn = np.full(shape=(T, mc_sims), fill_value=0.0)
initialPortfolio_rn = initialPortfolio # Starting portfolio value remains the same

for m in range(0, mc_sims):
    Z = np.random.normal(size=(T, len(weights))) # Uncorrelated RV's
    # L = np.linalg.cholesky(covMatrix) # Already calculated
    dailyReturns_rn = meanM_rn + np.inner(L, Z) # Correlated daily returns with risk-neutral drift
    portfolio_sims_rn[:,m] = np.cumprod(np.inner(weights, dailyReturns_rn.T)+1)*initialPortfolio_rn


# Calculate option payoff at expiration for each path (European Call Option)
# The payoff is max(0, final_portfolio_value - strike_price)
final_portfolio_values_rn = portfolio_sims_rn[-1, :]
call_payoffs = np.maximum(0, final_portfolio_values_rn - strike_price)

# Calculate the average payoff
average_payoff = np.mean(call_payoffs)

# Discount the average payoff back to present value
# Present Value = Average Payoff * exp(-risk_free_rate * time_to_expiration)
option_price = average_payoff * np.exp(-risk_free_rate * time_to_expiration)

# Store the calculated option price
european_call_price = option_price

print(f"Estimated European Call Option Price: ${european_call_price:.2f}")
```

```
Estimated European Call Option Price: $9747.75
```

Discuss the impact of simulation parameters on option price

The Monte Carlo simulation provides an estimate of the option price, and the parameters used in the simulation significantly influence this estimate. Let's discuss the key parameters and their impact:

- **Number of Simulations (`mc_sims`)**: This is perhaps the most direct parameter related to the Monte Carlo method itself. A higher number of simulations generally leads to a more accurate and stable estimate of the option price. This is because increasing the number of paths reduces the standard error of the Monte Carlo estimate, bringing it closer to the true theoretical value. However, increasing `mc_sims` also directly increases the computational time required to run the simulation.
- **Timeframe in Days (`T`)**: This parameter, when combined with the daily risk-free rate, determines the total time horizon of the simulation, which should align with the `time_to_expiration` of the option. A longer timeframe (larger `T`) allows for potentially larger movements in the underlying asset price. For a call option, a longer timeframe generally increases the probability of significant positive payoffs, leading to a higher option price, assuming other factors remain constant.

- **Initial Portfolio Value (`initialPortfolio`)**: This is the starting point for all simulated price paths. A higher initial portfolio value means the simulated final portfolio values are likely to be higher, increasing the probability that the option finishes in-the-money (final value > strike price) and thus increasing the call option price.
- **Strike Price (`strike_price`)**: This is a fundamental option parameter, not strictly a *simulation* parameter, but it directly impacts the calculated payoff in each simulation. A lower strike price increases the likelihood of a positive payoff at expiration (since final value - strike price is more likely to be positive), leading to a higher average payoff and thus a higher option price.
- **Risk-Free Rate (`risk_free_rate`)**: In the risk-neutral simulation, the risk-free rate acts as the drift for the asset price paths. A higher risk-free rate leads to higher expected future asset prices under the risk-neutral measure. This increases the expected payoff of a call option. Additionally, the risk-free rate is used to discount the expected future payoff back to the present. A higher risk-free rate means a larger discount, reducing the present value. For a call option, the positive effect on the expected future value typically outweighs the negative discounting effect, resulting in a positive relationship between the risk-free rate and the call option price.
- **Underlying Portfolio Volatility (derived from `covMatrix`)**: The volatility of the underlying asset (or portfolio, as modeled by the covariance matrix in our case) is a crucial determinant of option price. Higher volatility means the simulated price paths have larger fluctuations. For a call option, this increased fluctuation increases the potential for large positive payoffs while the downside is limited (payoff is zero if the option is out-of-the-money). This asymmetry means higher volatility increases the expected payoff and thus the call option price.

In summary, adjusting these parameters allows us to explore different market conditions and option specifications and observe their impact on the estimated option price through the Monte Carlo simulation. The number of simulations primarily affects the *accuracy* of the estimate, while the financial parameters (initial value, strike, time, rate, volatility) primarily affect the *value* of the option itself.

Explain the option pricing code

Subtask:

Provide a clear explanation of the code implemented in the previous step.

Reasoning: Write a markdown cell to explain the code implemented in the previous step, detailing the option parameters, risk-neutral adaptation, simulation, payoff calculation, averaging, and discounting.

```
%%markdown

### Explanation of the Monte Carlo Option Pricing Code

This section of the code builds upon the previous portfolio simulation to price a European call option using the Monte Carl

1.  **Define Option Parameters:**
    *   `strike_price = 150`: This variable sets the strike price of the European call option. This is the price at which t
    *   `time_to_expiration = 1`: This represents the time remaining until the option expires, measured in years.
    *   `risk_free_rate = 0.05`: This is the annual risk-free interest rate, used for discounting future payoffs back to th

2.  **Adapt Simulation for Risk-Neutral Pricing:**
    *   The core principle of risk-neutral valuation is that the expected return of any asset under the risk-neutral measur
    *   `daily_risk_free_rate = risk_free_rate / 252`: Since the simulation operates on daily steps (`T` is in days), the a
    *   `meanM_rn = np.full(shape=(T, len(weights)), fill_value=daily_risk_free_rate)`: A new matrix `meanM_rn` is created.

3.  **Re-run the Monte Carlo Simulation under Risk-Neutrality:**
    *   `portfolio_sims_rn = np.full(shape=(T, mc_sims), fill_value=0.0)`: A new array `portfolio_sims_rn` is initialized t
    *   `initialPortfolio_rn = initialPortfolio`: The initial portfolio value remains the same.
    *   The `for` loop iterates `mc_sims` times, just like the previous simulation.
    *   Inside the loop, `dailyReturns_rn = meanM_rn + np.inner(L, Z)` calculates the daily returns for each stock. The key
    *   `portfolio_sims_rn[:,m] = np.cumprod(np.inner(weights, dailyReturns_rn.T)+1)*initialPortfolio_rn`: The cumulative p

4.  **Calculate European Call Option Payoff:**
    *   `final_portfolio_values_rn = portfolio_sims_rn[-1, :]`: After running the simulations, this line extracts the final
    *   `call_payoffs = np.maximum(0, final_portfolio_values_rn - strike_price)`: For each simulated final portfolio value,

5.  **Calculate the Average Payoff:**
    *   `average_payoff = np.mean(call_payoffs)`: The average of all the calculated call option payoffs across the `mc_sims

6.  **Discount to Present Value:**
    *   `option_price = average_payoff * np.exp(-risk_free_rate * time_to_expiration)`: According to the risk-neutral valua

In summary, this code adapts the Monte Carlo simulation to the risk-neutral world by changing the expected return to the ri
```

Explanation of the Monte Carlo Option Pricing Code

This section of the code builds upon the previous portfolio simulation to price a European call option using the Monte Carlo method within the risk-neutral valuation framework.

1. **Define Option Parameters:**
    - `strike_price = 150`: This variable sets the strike price of the European call option. This is the price at which the holder of the option can buy the underlying asset (in this case, the portfolio) at expiration.
    - `time_to_expiration = 1`: This represents the time remaining until the option expires, measured in years.
    - `risk_free_rate = 0.05`: This is the annual risk-free interest rate, used for discounting future payoffs back to their present value under the risk-neutral measure.
2. **Adapt Simulation for Risk-Neutral Pricing:**
    - The core principle of risk-neutral valuation is that the expected return of any asset under the risk-neutral measure is the risk-free rate. In the previous portfolio simulation, the drift of the asset prices was based on historical mean returns. For option pricing under risk-neutrality, this drift is replaced by the risk-free rate.
    - `daily_risk_free_rate = risk_free_rate / 252`: Since the simulation operates on daily steps (`T` is in days), the annual risk-free rate is converted to a daily rate by dividing by the assumed number of trading days in a year (252).
    - `meanM_rn = np.full(shape=(T, len(weights)), fill_value=daily_risk_free_rate)`: A new matrix `meanM_rn` is created. Instead of using historical mean returns, each element in this matrix is filled with the calculated `daily_risk_free_rate`. This ensures that the simulated daily returns are now centered around the risk-free rate, consistent with the risk-neutral measure. The volatility, represented by the `covMatrix` and its Cholesky decomposition `L`, remains the same as it is assumed to be the same under both the real and risk-neutral measures.
3. **Re-run the Monte Carlo Simulation under Risk-Neutrality:**
    - `portfolio_sims_rn = np.full(shape=(T, mc_sims), fill_value=0.0)`: A new array `portfolio_sims_rn` is initialized to store the portfolio values over time for each simulation run under the risk-neutral drift.
    - `initialPortfolio_rn = initialPortfolio`: The initial portfolio value remains the same.
    - The `for` loop iterates `mc_sims` times, just like the previous simulation.
    - Inside the loop, `dailyReturns_rn = meanM_rn + np.inner(L, Z)` calculates the daily returns for each stock. The key difference here is the use of `meanM_rn` (based on the risk-free rate) instead of the historical `meanM`. The Cholesky decomposition `L` and the random shocks `Z` introduce the correlation and volatility.
    - `portfolio_sims_rn[:,m] = np.cumprod(np.inner(weights, dailyReturns_rn.T)+1)*initialPortfolio_rn`: The cumulative product of the daily portfolio returns (weighted sum of individual stock returns with the risk-neutral drift) is calculated to get the portfolio value path over time for the current simulation.
4. **Calculate European Call Option Payoff:**
    - `final_portfolio_values_rn = portfolio_sims_rn[-1, :]`: After running the simulations, this line extracts the final portfolio value from each of the `mc_sims` simulated paths. These are the simulated portfolio values at the option's expiration.
    - `call_payoffs = np.maximum(0, final_portfolio_values_rn - strike_price)`: For each simulated final portfolio value, the European call option payoff is calculated. The payoff is the maximum of zero and the difference between the final portfolio value and the `strike_price`. This reflects that a call option is only exercised if the underlying asset price at expiration is greater than the strike price.
5. **Calculate the Average Payoff:**
    - `average_payoff = np.mean(call_payoffs)`: The average of all the calculated call option payoffs across the `mc_sims` simulations is computed. This average represents the expected payoff of the option at expiration under the risk-neutral measure.
6. **Discount to Present Value:**
    - `option_price = average_payoff * np.exp(-risk_free_rate * time_to_expiration)`: According to the risk-neutral valuation principle, the fair price of the option today is the expected future payoff discounted back to the present at the risk-free rate. This line applies the continuous compounding discount formula: `e^(-rT)`, where `r` is the risk-free rate and `T` is the time to expiration. The result is the estimated price of the European call option.

In summary, this code adapts the Monte Carlo simulation to the risk-neutral world by changing the expected return to the risk-free rate. It then simulates

Discuss the results

Subtask:

Analyze the results of the option pricing simulation and discuss the factors that influence the option price.

Reasoning: Analyze the results of the option pricing simulation and discuss the factors that influence the option price based on the calculated `european_call_price` and the parameters used in the simulation.

```
# Discuss the results and factors influencing the option price

print(f"Estimated European Call Option Price: ${european_call_price:.2f}\n")

print("Factors influencing the European Call Option Price:")
print("- Initial Portfolio Value ($10000): A higher initial value generally leads to higher expected final portfolio values
print(f"- Strike Price (${strike_price}): The strike price is inversely related to the call option price. A lower strike pr
print(f"- Time to Expiration ({time_to_expiration} year): Generally, a longer time to expiration increases the call option
print(f"- Risk-Free Rate ({risk_free_rate}): The risk-free rate has two main effects on the call option price in this risk-
print("- Underlying Portfolio Volatility (derived from covMatrix): Volatility is a measure of the uncertainty or fluctuatio
print(f"- Number of Monte Carlo Simulations ({mc_sims}): The number of simulations affects the accuracy and stability of th

print("\nLimitations of this basic model and potential sources of error:")
print("- Constant Risk-Free Rate and Volatility: The model assumes that the risk-free rate and the underlying portfolio's v
print("- Choice of Stochastic Process: Geometric Brownian Motion (implicitly used for portfolio value simulation here) migh
print("- Path Dependence: While Monte Carlo is good for path-dependent options, this example priced a European option which
print("- Correlation Assumption: The model relies on the historical covariance matrix to model the correlation between asse
print("- Efficiency: For simple European options, analytical formulas like Black-Scholes are generally more computationally
```

```
Estimated European Call Option Price: $9747.75

Factors influencing the European Call Option Price:
- Initial Portfolio Value ($10000): A higher initial value generally leads to higher expected final portfolio values, increa
- Strike Price ($150): The strike price is inversely related to the call option price. A lower strike price means the optior
- Time to Expiration (1 year): Generally, a longer time to expiration increases the call option price. More time allows for
- Risk-Free Rate (0.05): The risk-free rate has two main effects on the call option price in this risk-neutral framework. Fi
- Underlying Portfolio Volatility (derived from covMatrix): Volatility is a measure of the uncertainty or fluctuation in the
- Number of Monte Carlo Simulations (80000): The number of simulations affects the accuracy and stability of the estimated c

Limitations of this basic model and potential sources of error:
- Constant Risk-Free Rate and Volatility: The model assumes that the risk-free rate and the underlying portfolio's volatilit
- Choice of Stochastic Process: Geometric Brownian Motion (implicitly used for portfolio value simulation here) might not fu
- Path Dependence: While Monte Carlo is good for path-dependent options, this example priced a European option which is not
- Correlation Assumption: The model relies on the historical covariance matrix to model the correlation between assets, assu
- Efficiency: For simple European options, analytical formulas like Black-Scholes are generally more computationally efficie
```

Explore pricing different types of options

Subtask:

Discuss how the Monte Carlo method can be extended to price more complex options (e.g., American options, exotic options).

Reasoning: Discuss how Monte Carlo simulation can be extended to price more complex options, addressing American options, exotic options, and relevant techniques.

```
%%markdown

### Extending Monte Carlo to Price Complex Options

Monte Carlo simulation is particularly valuable for pricing options that do not have simple closed-form analytical solutior

#### Pricing American Options

American options give the holder the right to exercise the option at any time up to and including the expiration date. This

To price American options with Monte Carlo, techniques that determine the optimal exercise strategy are required. The most

**Least Squares Monte Carlo (LSM):**

1.  **Simulate Price Paths:** Generate a large number of price paths for the underlying asset under the risk-neutral measur
2.  **Work Backwards from Expiration:** Starting from the period just before expiration, and working backwards in time, at
    *   **Calculate the Immediate Exercise Value:** Determine the payoff if the option were exercised at that moment.
    *   **Estimate the Continuation Value:** This is the expected value of *not* exercising immediately, but continuing to
    *   **Make the Exercise Decision:** Compare the immediate exercise value to the estimated continuation value. If the im
3.  **Calculate Option Value:** The estimated American option price is the average of the discounted payoffs from all paths

LSM allows for the estimation of the continuation value at each point in time and state, enabling the determination of the

#### Pricing Exotic Options

Exotic options are options with features that go beyond standard European or American options. They often have complex payo

Examples of exotic options where Monte Carlo is widely used include:

*   **Asian Options:** The payoff depends on the average price of the underlying asset over a specified period, not just th
*   **Barrier Options:** The payoff depends on whether the underlying asset price reaches a certain "barrier" level during
*   **Lookback Options:** The payoff depends on the maximum or minimum price of the underlying asset over the option's life
*   **Path-Dependent Options (general):** Many other exotics have payoffs that depend on the entire path or specific points

**Why Analytical Methods are Insufficient for Exotic Options:**

Analytical solutions typically rely on solving partial differential equations (PDEs) or finding closed-form expressions for

**Techniques and Considerations for Complex Options:**

*   **Variance Reduction Techniques:** To improve the efficiency and accuracy of Monte Carlo for both American and exotic o
*   **Modeling Multi-Asset Options:** For options on multiple underlying assets (like basket options), Monte Carlo is a nat
*   **Computational Cost:** While flexible, Monte Carlo can be computationally intensive, especially for options requiring

In summary, Monte Carlo simulation, especially with extensions like the LSM method for American options, provides a flexibl
```

options. This includes **American options** and various **exotic options**.

Pricing American Options

American options give the holder the right to exercise the option at any time up to and including the expiration date. This early exercise feature introduces a significant challenge for Monte Carlo simulation because the decision to exercise early depends on the option's value at each point in time, which in turn depends on future expected values. Standard Monte Carlo, which typically simulates paths forward and calculates payoffs only at expiration, cannot directly handle this.

To price American options with Monte Carlo, techniques that determine the optimal exercise strategy are required. The most common method is the **Least Squares Monte Carlo (LSM) method**, developed by Longstaff and Schwartz (2001).

Least Squares Monte Carlo (LSM):

1. **Simulate Price Paths:** Generate a large number of price paths for the underlying asset under the risk-neutral measure, similar to the European option case.
2. **Work Backwards from Expiration:** Starting from the period just before expiration, and working backwards in time, at each time step and for each simulated path where the option is still alive:
    - **Calculate the Immediate Exercise Value:** Determine the payoff if the option were exercised at that moment.
    - **Estimate the Continuation Value:** This is the expected value of *not* exercising immediately, but continuing to hold the option. This expectation is conditional on the current state (e.g., the underlying asset price). LSM uses least squares regression to estimate this conditional expected value. It regresses the *discounted* future payoffs (calculated from subsequent time steps assuming the option was not exercised early) against a set of basis functions of the underlying asset price at the current time step.
    - **Make the Exercise Decision:** Compare the immediate exercise value to the estimated continuation value. If the immediate exercise value is greater, the optimal decision is to exercise.
3. **Calculate Option Value:** The estimated American option price is the average of the discounted payoffs from all paths, where the payoff for each path is recorded at the time of optimal exercise (either early or at expiration).

LSM allows for the estimation of the continuation value at each point in time and state, enabling the determination of the optimal early exercise boundary within the Monte Carlo framework.

Pricing Exotic Options

Exotic options are options with features that go beyond standard European or American options. They often have complex payoff structures or dependencies on the path taken by the underlying asset price, making analytical solutions extremely difficult or impossible. Monte Carlo simulation is particularly well-suited for these options because it can directly model these complex features by simulating the price paths and calculating the payoff according to the specific rules of the exotic option.

Examples of exotic options where Monte Carlo is widely used include:

- **Asian Options:** The payoff depends on the average price of the underlying asset over a specified period, not just the price at expiration. Monte Carlo can easily track the average price along each simulated path to calculate the payoff.
- **Barrier Options:** The payoff depends on whether the underlying asset price reaches a certain "barrier" level during the option's life. Monte Carlo simulations naturally generate paths that can be monitored to see if the barrier is hit.
- **Lookback Options:** The payoff depends on the maximum or minimum price of the underlying asset over the option's life. Monte Carlo paths allow tracking of these extreme values.
- **Path-Dependent Options (general):** Many other exotics have payoffs that depend on the entire path or specific points along the path. Monte Carlo's ability to simulate full price paths makes it ideal for these.

Why Analytical Methods are Insufficient for Exotic Options:

Analytical solutions typically rely on solving partial differential equations (PDEs) or finding closed-form expressions for expected values. The complex dependencies and path-dependent features of exotic options often lead to PDEs without known analytical solutions or make the calculation of the expected payoff analytically intractable. Monte Carlo simulation bypasses the need for solving these complex equations by directly simulating the underlying process and averaging the outcomes.

Techniques and Considerations for Complex Options:

- **Variance Reduction Techniques:** To improve the efficiency and accuracy of Monte Carlo for both American and exotic options, various variance reduction techniques are often employed. These include antithetic variates (using paired positive and negative random numbers), control variates (using a similar option with a known price to reduce variance), and importance sampling (focusing simulations on more relevant areas of the probability distribution).
- **Modeling Multi-Asset Options:** For options on multiple underlying assets (like basket options), Monte Carlo is a natural choice as it can easily handle the simulation of correlated price paths, as demonstrated in the portfolio simulation section.
- **Computational Cost:** While flexible, Monte Carlo can be computationally intensive, especially for options requiring a large number of simulations or complex path calculations. This is why efficiency improvements through variance reduction and parallel processing are important.

In summary, Monte Carlo simulation, especially with extensions like the LSM method for American options, provides a flexible and powerful approach to pricing complex financial derivatives where analytical solutions are not available. By simulating a large number of potential future scenarios, it allows for the direct incorporation of complex payoff structures and path dependencies.

Summary:

Data Analysis Key Findings

- Monte Carlo simulation is a powerful tool for option pricing, especially for complex options without analytical solutions, based on the risk-neutral valuation framework.
- Under the risk-neutral measure, the expected return of any asset is the risk-free rate, which is used as the drift in the simulation of asset price paths.
- The Monte Carlo option price is calculated by averaging the discounted payoffs across a large number of simulated risk-neutral paths.
- For a European call option, the price is positively influenced by the initial asset value, time to expiration, risk-free rate, and volatility, and negatively influenced by the strike price.
- The number of simulations affects the accuracy of the estimated price; more simulations lead to higher accuracy but also increased computation time.
- Pricing American options with Monte Carlo requires specialized techniques like the Least Squares Monte Carlo (LSM) method to account for the possibility of early exercise.