Experiment 8

Name: Bhuvi Ghosh
SAPID: 60009210191
Batch: D22

Aim: Analyze various Interactive plots of Monte Carlo Simulations of CPPI and GBM.

Objective:

- Learn about CPPI and drawdown constraints.
- Implement CPPI and drawdown constraints in Python.
- Test the CPPI and drawdown constraints on a specified data source.

Theory:

Monte Carlo Simulation: A Monte Carlo simulation is a mathematical technique that uses repeated random sampling to approximate the probability distribution of a random variable. This can be used to estimate the value of a financial asset, the risk of a financial portfolio, or the performance of a trading strategy.

The basic idea behind a Monte Carlo simulation is to generate a large number of random samples from the probability distribution of the random variable. These samples are then used to estimate the expected value, variance, and other statistical properties of the random variable.

For example, to estimate the value of a financial asset, we could generate a large number of random samples from the asset's price distribution. These samples would then be used to estimate the asset's expected price, its volatility, and its risk.

CPPI: CPPI stands for Constant Proportion Portfolio Insurance. It is a dynamic asset allocation strategy that aims to protect investors from downside risk while still allowing them to participate in market upside.

The basic idea behind CPPI is to maintain a constant proportion of the portfolio in risky assets. This proportion is called the floor value. The remaining part of the portfolio is invested in risk-free assets.

As the value of the risky assets rises, the floor value is increased. This means that more of the portfolio is invested in risky assets. As the value of the risky assets falls, the floor value is decreased. This means that less of the portfolio is invested in risky assets.

The CPPI strategy can be implemented using a Monte Carlo simulation. The simulation would start with a certain initial portfolio value and a certain floor value. The simulation would then generate a large number of random samples from the asset's price distribution. These samples would then be used to track the value of the portfolio over time.

GBM: GBM stands for geometric Brownian motion. It is a stochastic process that is used to model the evolution of asset prices over time.

The GBM process is characterized by two parameters: the drift rate and the volatility. The drift rate is the expected rate of return of the asset. The volatility is a measure of the uncertainty of the asset's returns.

The GBM process can be used to model the price of a financial asset over time. The price of the asset is modeled as a random walk with drift. The drift rate represents the expected rate of return of the asset, and the volatility represents the uncertainty of the asset's returns.

The GBM process can be implemented using a Monte Carlo simulation. The simulation would start with a certain initial asset price and a certain drift rate and volatility. The simulation would then generate a large number of random samples from the GBM process. These samples would then be used to track the price of the asset over time.

Formulae

Here are some of the formulae that are used in Monte Carlo simulations of CPPI and GBM:

- Expected value: The expected value of a random variable is the average of its possible values. For example, if a random variable can take on the values 1, 2, and 3, then the expected value is $(1 + 2 + 3) / 3 = 2$.
- Variance: The variance of a random variable is a measure of its uncertainty. The variance is calculated by taking the average of the squared deviations from the mean. For example, if a random variable can take on the values 1, 2, and 3, then the variance is $(1 - 2)^2 + (2 - 2)^2 + (3 - 2)^2 / 3 = 1/3$.
- Standard deviation: The standard deviation of a random variable is the square root of its variance. The standard deviation is a measure of the typical distance from the mean. For example, if a random variable has a variance of 1/3, then the standard deviation is $\sqrt{1/3} = \sim 0.577$.
- Drift rate: The drift rate of a GBM process is the expected rate of return of the asset. The drift rate is typically expressed as a percentage. For example, if the drift rate is 5%, then the asset is expected to increase in value by 5% on average each year.
- Volatility: The volatility of a GBM process is a measure of the uncertainty of the asset's returns. The volatility is typically expressed as a percentage. For example, if the volatility is 10%, then the asset's price is expected to fluctuate by 10% on average each year.

Lab Experiment to be done by students:

1. Implement a Monte Carlo simulation of a CPPI strategy.
2. Implement a Monte Carlo simulation of a GBM strategy.
3. Create interactive plots of the Monte Carlo simulations

```
!pip install yfinance numpy pandas matplotlib plotly
```

```
Requirement already satisfied: yfinance in /usr/local/lib/python3.10/dist-packages (0.2.44)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.26.4)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (2.2.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.7.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (5.24.1)
Requirement already satisfied: requests>=2.31 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2.32.3)
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.10/dist-packages (from yfinance) (0.0.11)
Requirement already satisfied: lxml>=4.9.1 in /usr/local/lib/python3.10/dist-packages (from yfinance) (4.9.4)
Requirement already satisfied: platformdirs>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from yfinance) (4.3.6)
Requirement already satisfied: pytz>=2022.5 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2024.2)
Requirement already satisfied: frozendict>=2.3.4 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2.4.5)
Requirement already satisfied: peewee>=3.16.2 in /usr/local/lib/python3.10/dist-packages (from yfinance) (3.17.6)
Requirement already satisfied: beautifulsoup4>=4.11.1 in /usr/local/lib/python3.10/dist-packages (from yfinance) (4.12.3)
Requirement already satisfied: html5lib>=1.1 in /usr/local/lib/python3.10/dist-packages (from yfinance) (1.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.3.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (24.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.4)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly) (9.0.0)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4>=4.11.1->yfinance) (2.6)
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.10/dist-packages (from html5lib>=1.1->yfinance) (1.16.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from html5lib>=1.1->yfinance) (0.5.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance) (2024.8.30)
```

```
[2]  import numpy as np
     import pandas as pd
     import yfinance as yf
     import matplotlib.pyplot as plt
     import plotly.graph_objects as go
```

```python
def get_stock_data(ticker, start, end):
    stock_data = yf.download(ticker, start=start, end=end)
    stock_data['Returns'] = stock_data['Adj Close'].pct_change()
    return stock_data
```

```python
def monte_carlo_cppi(stock_data, num_simulations, time_horizon, floor_value, multiplier, initial_wealth):
    stock_returns = stock_data['Returns'].dropna()
    mu = stock_returns.mean()
    sigma = stock_returns.std()

    dt = 1 / time_horizon
    wealth_paths = []

    for _ in range(num_simulations):
        wealth = initial_wealth
        floor = floor_value
        wealth_path = []

        for _ in range(time_horizon):
            random_shock = np.random.normal(0, 1)
            stock_return = mu * dt + sigma * random_shock * np.sqrt(dt)

            cushion = max(wealth - floor, 0)
            risky_allocation = multiplier * cushion

            bond_allocation = wealth - risky_allocation
            equity_return = stock_return * risky_allocation
            bond_return = 0.02 * bond_allocation
            wealth = bond_allocation + (risky_allocation * (1 + stock_return))
            wealth_path.append(wealth)
        wealth_paths.append(wealth_path)
    return np.array(wealth_paths)
```

```python
def monte_carlo_gbm(stock_data, num_simulations, time_horizon, initial_price):
    stock_returns = stock_data['Returns'].dropna()
    mu = stock_returns.mean()
    sigma = stock_returns.std()

    dt = 1 / time_horizon
    price_paths = []

    for _ in range(num_simulations):
        price_path = [initial_price]
        for _ in range(time_horizon):
            random_shock = np.random.normal(0, 1)
            price = price_path[-1] * np.exp((mu - 0.5 * sigma ** 2) * dt + sigma * random_shock * np.sqrt(dt))
            price_path.append(price)
        price_paths.append(price_path)
    return np.array(price_paths)
```

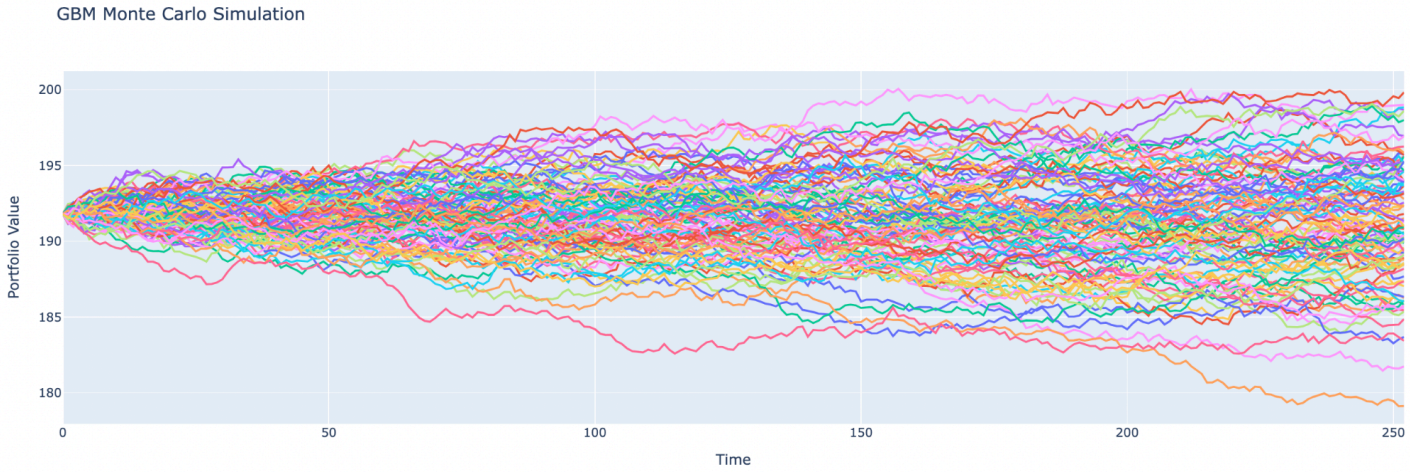
```python
def plot_monte_carlo_simulations(simulations, title, initial_value):
    fig = go.Figure()
    for sim in simulations:
        fig.add_trace(go.Scatter(x=list(range(len(sim))), y=sim, mode='lines'))

    fig.update_layout(
        title=title,
        xaxis_title='Time',
        yaxis_title='Portfolio Value',
        showlegend=False
    )
    fig.show()
```

```python
if __name__ == "__main__":
    ticker = 'AAPL'
    start_date = '2020-01-01'
    end_date = '2024-01-01'
    stock_data = get_stock_data(ticker, start_date, end_date)
    num_simulations = 100
    time_horizon = 252
    floor_value = 80
    multiplier = 5
    initial_wealth = 100
    cppi_simulations = monte_carlo_cppi(stock_data, num_simulations, time_horizon, floor_value, multiplier, initial_wealth)
    initial_price = stock_data['Adj Close'][-1]
    gbm_simulations = monte_carlo_gbm(stock_data, num_simulations, time_horizon, initial_price)
    plot_monte_carlo_simulations(cppi_simulations, "CPPI Strategy Monte Carlo Simulation", initial_wealth)
    plot_monte_carlo_simulations(gbm_simulations, "GBM Monte Carlo Simulation", initial_price)
```

```
[**********100%**********]  1 of 1 completed
<ipython-input-7-341c1e65e453>:12: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future vers
  initial_price = stock_data['Adj Close'][-1]
```


CPPI Strategy Monte Carlo Simulation
Portfolio Value
104
102
100
98
96
0
50
100
150
200
250
Time


GBM Monte Carlo Simulation
Portfolio Value
200
195
190
185
180
0
50
100
150
200
250
Time