Experiment 8

Name:Jhanvi Parekh

Sap;60009210033

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

```
[18] import numpy as np
     import plotly.graph_objs as go
     import plotly.io as pio
     import yfinance as yf

[19] # Ensure Plotly renders correctly in Colab
     pio.renderers.default = 'colab'

[20] # Step 1: Download historical stock data (e.g., Reliance Industries)
     ticker = 'RELIANCE.NS'
     data = yf.download(ticker, start='2020-01-01', end='2024-01-01')
     prices = data['Adj Close']

     [**********100%**********]  1 of 1 completed

     # Step 2: Calculate the historical drift and volatility of the stock
     log_returns = np.log(prices / prices.shift(1)).dropna()
     mu = log_returns.mean() * 252  # Annualized drift
     sigma = log_returns.std() * np.sqrt(252)  # Annualized volatility

[22] # Step 3: CPPI strategy parameters
     initial_portfolio_value = 100000  # Starting portfolio value
     floor_value = 90000  # Minimum acceptable portfolio value
     multiplier = 5  # Determines exposure to the risky asset
     risk_free_rate = 0.02  # 2% annual return on the risk-free asset
```

```
[22] risk_free_rate = 0.02  # 2% annual return on the risk-free asset
     T = 1  # Time horizon in years
     num_simulations = 100  # Number of simulations
     num_steps = 252  # Number of trading days in a year
```

```
[23] # Step 4: Monte Carlo simulation for the risky asset using GBM
     def gbm_simulation(S0, mu, sigma, T, num_simulations, num_steps):
         dt = T / num_steps
         simulations = np.zeros((num_steps, num_simulations))
         simulations[0] = S0

         for sim in range(num_simulations):
             for step in range(1, num_steps):
                 z = np.random.standard_normal()
                 simulations[step, sim] = simulations[step - 1, sim] * np.exp((mu - 0.5 * sigma**2) * dt + sigma * np.sqrt(dt) * z)

         return simulations
```

```
[24] # Simulate the risky asset paths
     S0 = prices[-1]  # Current stock price
     risky_asset_simulations = gbm_simulation(S0, mu, sigma, T, num_simulations, num_steps)
```

```
<ipython-input-24-f36ba0128704>:2: FutureWarning:

Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value
```

```
def cppi_simulation(initial_portfolio_value, floor_value, multiplier, risk_free_rate, risky_asset_simulations, num_steps):
    """Simulates the CPPI strategy.

    Args:
        initial_portfolio_value: The starting portfolio value.
        floor_value: The minimum acceptable portfolio value.
        multiplier: Determines the exposure to the risky asset.
        risk_free_rate: The risk-free rate of return.
        risky_asset_simulations: Simulated paths of the risky asset.
        num_steps: The number of time steps in the simulation.

    Returns:
        A NumPy array representing the portfolio value over time for each simulation.
    """
    num_simulations = risky_asset_simulations.shape[1]
    portfolio_values = np.zeros((num_steps, num_simulations))
    portfolio_values[0] = initial_portfolio_value

    for sim in range(num_simulations):
        for step in range(1, num_steps):
            # Calculate the cushion
            cushion = portfolio_values[step - 1, sim] - floor_value

            # Calculate the exposure to the risky asset
            exposure = multiplier * cushion

            # Allocate to risky and risk-free assets
            risky_allocation = min(exposure, portfolio_values[step - 1, sim])  # Ensure risky allocation doesn't exceed portfolio value
            risk_free_allocation = portfolio_values[step - 1, sim] - risky_allocation
```

✓ 0s completed at 20:39

+ Code + Text

Disk

✦ Gemini

[27]
```
# Run the CPPI simulation
cppi_simulations = cppi_simulation(initial_portfolio_value, floor_value, multiplier, risk_free_rate, risky_asset_simulations, num_steps)

# Step 6: Create an interactive Plotly plot
time_steps = np.arange(num_steps)
fig = go.Figure()

# Plot multiple CPPI simulations
for sim in range(num_simulations):
    fig.add_trace(go.Scatter(
        x=time_steps,
        y=cppi_simulations[:, sim],
        mode='lines',
        name=f'Simulation {sim+1}',
        opacity=0.5
    ))
```

```
# Run the CPPI simulation
cppi_simulations = cppi_simulation(initial_portfolio_value, floor_value, multiplier, risk_free_rate, risky_asset_simulations, num_steps)

# Step 6: Create an interactive Plotly plot
time_steps = np.arange(num_steps)
fig = go.Figure()
```

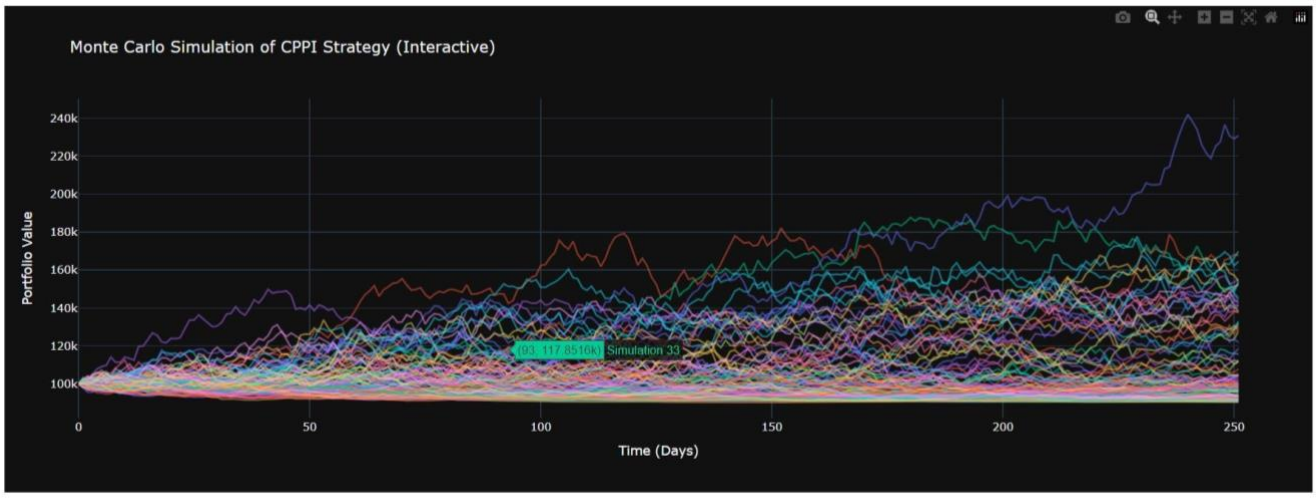
2s [29]
```
# Step 6: Create an interactive Plotly plot
time_steps = np.arange(num_steps)
```

✓ 0s completed at 20:39

+ Code + Text

RAM Disk

✦ Gemini



+ Code + Text

RAM Disk

✦ Gemini

2.Implement a Monte Carlo simulation of a GBM strategy

[31]
```
import numpy as np
import pandas as pd
import plotly.graph_objs as go
import plotly.io as pio
import yfinance as yf
```

[32]
```
pio.renderers.default = 'colab'
```

```
# Step 1: Define parameters for the GBM simulation
def gbm_simulation(S0, mu, sigma, T, num_simulations, num_steps):
    dt = T / num_steps  # Time increment
    simulations = np.zeros((num_steps + 1, num_simulations))  # Matrix to hold stock prices
    simulations[0] = S0  # Initial stock price

    for sim in range(num_simulations):
        for step in range(1, num_steps + 1):
            z = np.random.standard_normal()  # Generate random standard normal variable
            # GBM formula
            simulations[step, sim] = simulations[step - 1, sim] * np.exp((mu - 0.5 * sigma**2) * dt + sigma * np.sqrt(dt) * z)

    return simulations
```

[34]
```
# Step 2: Get historical stock data to calculate mu and sigma
ticker = 'RELIANCE.NS'  # You can change this to any other stock symbol
```

+ Code + Text

RAM Disk

✦ Gemini

[34]
```
# Step 2: Get historical stock data to calculate mu and sigma
ticker = 'RELIANCE.NS'  # You can change this to any other stock symbol
data = yf.download(ticker, start='2020-01-01', end='2024-01-01')
prices = data['Adj Close']
```

```
[**********100%**********]  1 of 1 completed
```

[35]
```
# Step 3: Calculate drift (mu) and volatility (sigma)
log_returns = np.log(prices / prices.shift(1)).dropna()
mu = log_returns.mean() * 252  # Annualized drift
sigma = log_returns.std() * np.sqrt(252)  # Annualized volatility
```

[36]
```
# Step 4: Define GBM parameters
S0 = prices[-1]  # Latest stock price
T = 1  # Time horizon in years
num_simulations = 1000  # Number of simulations
num_steps = 252  # Number of trading days in a year
```

```
<ipython-input-36-93552ff1c939>:2: FutureWarning:

Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value
```

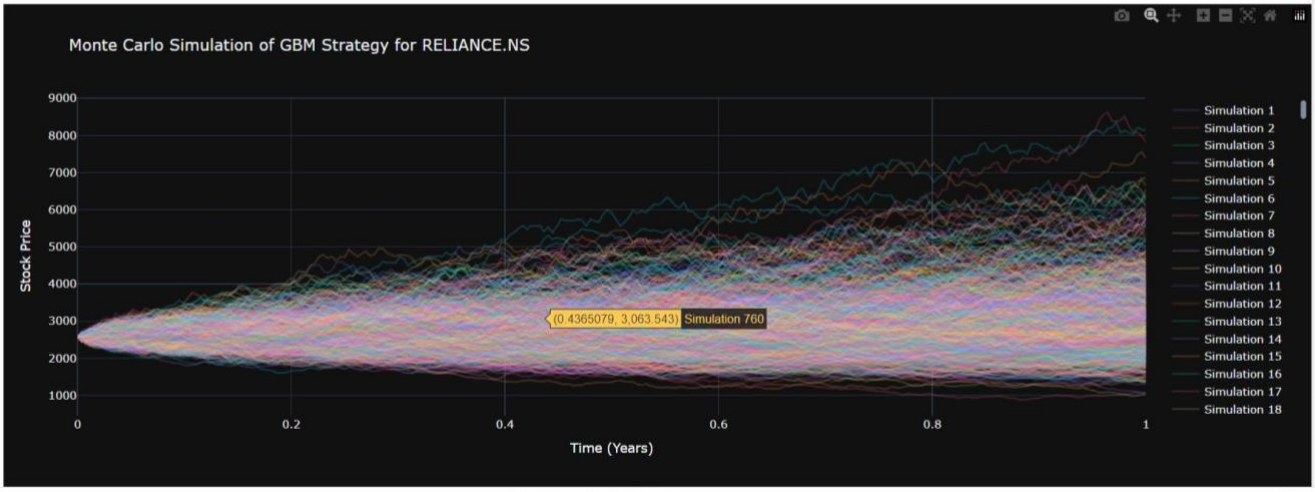
[37]
```
# Step 5: Run the GBM simulation
gbm_simulations = gbm_simulation(S0, mu, sigma, T, num_simulations, num_steps)
```

```
[37] # Step 5: Run the GBM simulation
     gbm_simulations = gbm_simulation(S0, mu, sigma, T, num_simulations, num_steps)

[38] # Step 6: Create an interactive Plotly plot
     time_steps = np.linspace(0, T, num_steps + 1)
     fig = go.Figure()

16s
# Plot multiple GBM simulations
for sim in range(num_simulations):
    fig.add_trace(go.Scatter(
        x=time_steps,
        y=gbm_simulations[:, sim],
        mode='lines',
        name=f'Simulation {sim + 1}',
        opacity=0.2
    ))

# Layout details
fig.update_layout(
    title=f'Monte Carlo Simulation of GBM Strategy for {ticker}',
    xaxis_title='Time (Years)',
    yaxis_title='Stock Price',
    template='plotly_dark'
)
```



✓ 0s completed at 20:39