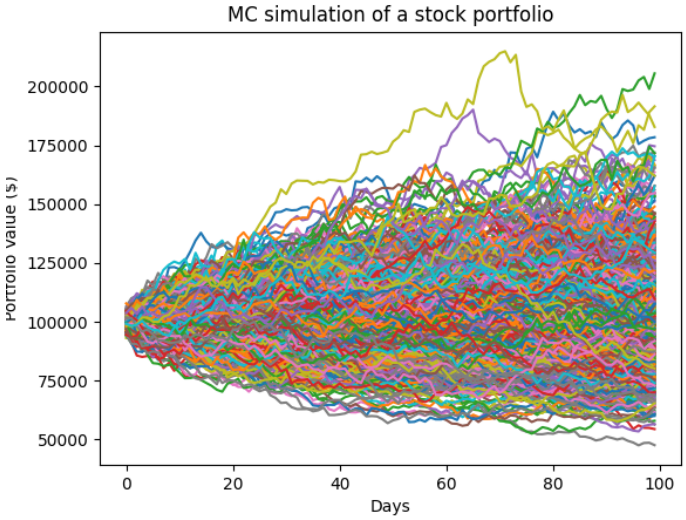
Montecarlo Method

Riccardo Pondini

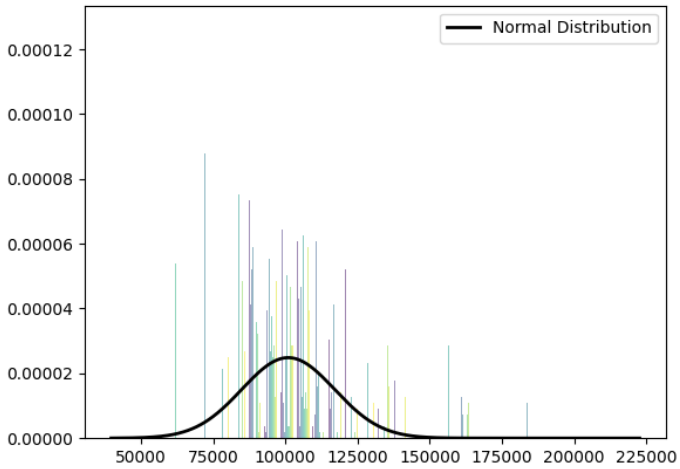
January 2024

```python
import pandas as pd
import numpy as np
import datetime as dt
from pandas_datareader import data as pdr
from scipy.stats import norm, t
import matplotlib.pyplot as plt
import yfinance as yf

# Import data
def getData(stocks, start, end):
    stockData = yf.download(stocks, start=start, end=end)

    # Extracting adjusted closing prices for each stock
    adjClose = stockData['Adj Close']

    # Handling missing values before calculating returns
    adjClose = adjClose.dropna()

    # Calculate daily returns
    returns = adjClose.pct_change(fill_method=None)

    # Calculate mean returns and covariance matrix
    meanReturns = returns.mean()
    covMatrix = returns.cov()

    return returns, meanReturns, covMatrix

# Expected returns and standard deviation
def portfolioPerformance(weights, meanReturns, covMatrix, Time):
    returns = np.sum(meanReturns*weights)*Time
    # Std dev formula with an equity portfolio
    std = np.sqrt( np.dot(weights.T, np.dot(covMatrix, weights)) ) * np.sqrt(Time)
    return returns, std

# Defining asset classes and dates
stockList = ['GOOGL', 'AMZN', 'TSLA', 'NVDA', 'META', 'AAPL']
stocks = [stock for stock in stockList]
endDate = dt.datetime.now()
startDate = endDate - dt.timedelta(days=800)
returns, meanReturns, covMatrix = getData(stocks, start=startDate, end=endDate)

# Giving random weights and making their sum equal to 1
weights = np.random.random(len(returns.columns))
weights /= np.sum(weights)

# Portfolio daily returns (scalar product between returns matrix and weights array)
returns['portfolio'] = returns.dot(weights)
```

```
# Monte Carlo Method
mc_sims = 1000 # number of simulations
T = 100 #timeframe in days

# Matrix of avarage returns for every stock
meanM = np.full(shape=(T, len(weights)), fill_value=meanReturns)
meanM = meanM.T
# Matrix that will store all the simulated portfolio returns for each day
portfolio_sims = np.full(shape=(T, mc_sims), fill_value=0.0)

initialPortfolio = 100000

for m in range(0, mc_sims):
    # MC loops
    Z = np.random.normal(size=(T, len(weights)))
    # Cholesky decomposition
    L = np.linalg.cholesky(covMatrix)
    # Computing simulated returns by adding also the correlation between stocks
    dailyReturns = meanM + np.inner(L, Z)
    # Computing portfolio simulated returns
    portfolio_sims[:,m] = np.cumprod(np.inner(weights, dailyReturns.T)+1)*initialPortfolio

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

# Creating the histogram of the simulations
plt.figure()
plt.hist(portfolio_sims, bins=30, density=True, alpha=0.5, color=plt.cm.viridis(np.linspace(0, 1, 1000)))

# Mean and Std dev of the simulated distribution
mean_sim = np.mean(portfolio_sims)
std_dev_sim = np.std(portfolio_sims)

# Plotting the curve of the normal (Gaussian) distribution based on the simulated mean and standard deviation
xmin, xmax = plt.xlim()
# 1000 evenly distributed values (x-axis points for calculating the probability density function)
x = np.linspace(xmin, xmax, 1000)
# Values associated to the normal distribution on the previous x-axis points
p = norm.pdf(x, mean_sim, std_dev_sim)

plt.plot(x, p, 'k', linewidth=2, label='Normal Distribution')
plt.legend()
plt.show()
```



(a) 1000 simulasions of a 100000$ equity portfolio.



(b) Distribution of the simulations.