Predictive Modeling

Author:Bakang Mogapi

Section 1, Part a: Monte Carlo Simulation for Stock Portfolio Valuation

In [14]:
```
# Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [15]:
```
# Define the stocks and weights in the portfolio
stocks = ['AAPL', 'MSFT', 'AMZN', 'GOOG']
weights = np.array([0.25, 0.25, 0.25, 0.25])
```

In [16]:
```
# Get the historical prices from Yahoo Finance
prices = pd.DataFrame()
for stock in stocks:
    prices[stock] = pd.read_csv(f'https://query1.finance.yahoo.com/v7/finan
ce/download/{stock}?period1=1577836800&period2=1640995200&interval=1d&event
s=history&includeAdjustedClose=true')['Adj Close']
```

In [17]:
```
# Calculate the daily returns
returns = prices.pct_change()
```

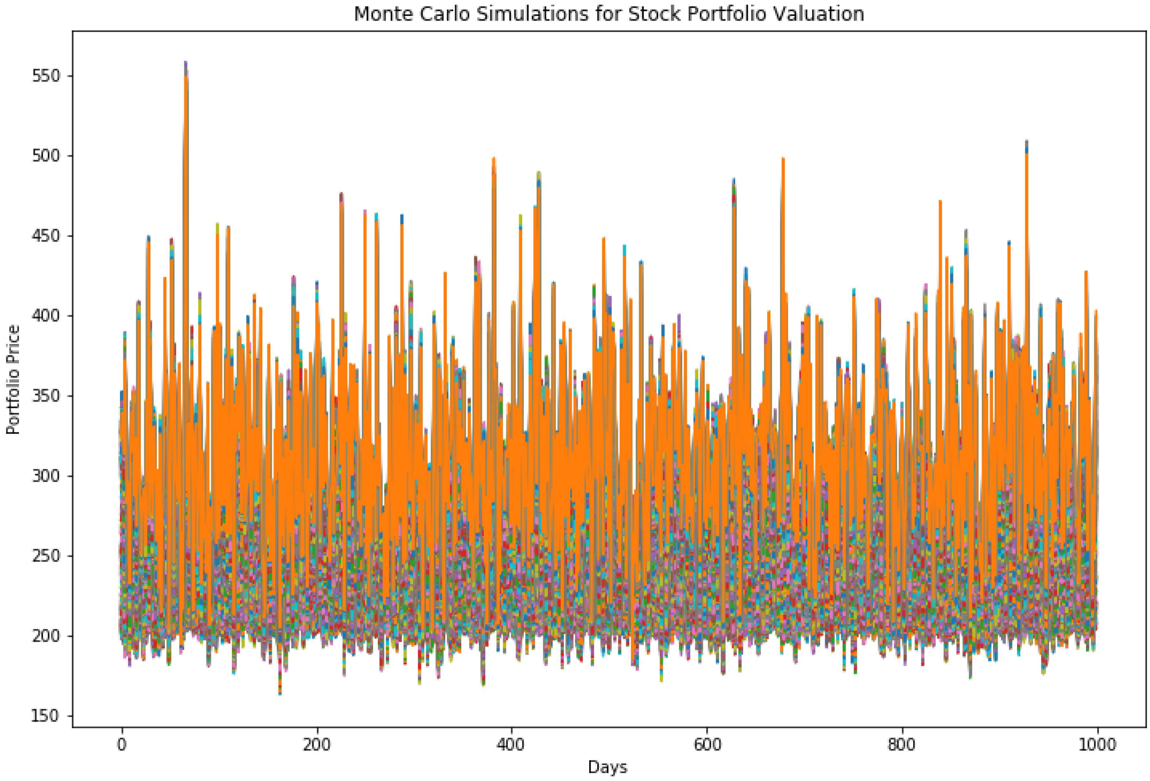
In [18]:
```
# Calculate the mean and standard deviation of returns
mean = returns.mean()
std = returns.std()
```

In [19]:
```
# Define the number of simulations and the time horizon
num_simulations = 1000
num_days = 252
```

In [20]:
```
# Create an empty array to store the simulation results
simulations = np.zeros((num_simulations, num_days))
```

In [21]:
```
# Loop over the number of simulations
for i in range(num_simulations):
    # Initialize the first price as the last price in the data
    simulations[i, 0] = prices.iloc[-1].dot(weights)
    # Loop over the number of days
    for j in range(1, num_days):
        # Generate a random vector of returns from a multivariate normal di
stribution
        z = np.random.multivariate_normal(mean, np.diag(std**2))
        # Calculate the portfolio return for that day
        r = np.sum(z * weights)
        # Update the portfolio price for that day
        simulations[i, j] = simulations[i, j-1] * (1 + r)
```

In [22]:
```
# Plot the simulations
plt.figure(figsize=(12,8))
plt.plot(simulations)
plt.title('Monte Carlo Simulations for Stock Portfolio Valuation')
plt.xlabel('Days')
plt.ylabel('Portfolio Price')
plt.show()
```



Part c: Monte Carlo Simulation for Valuing a Rainbow Option

In [23]:
```
# Import libraries
import numpy as np
import matplotlib.pyplot as plt
```

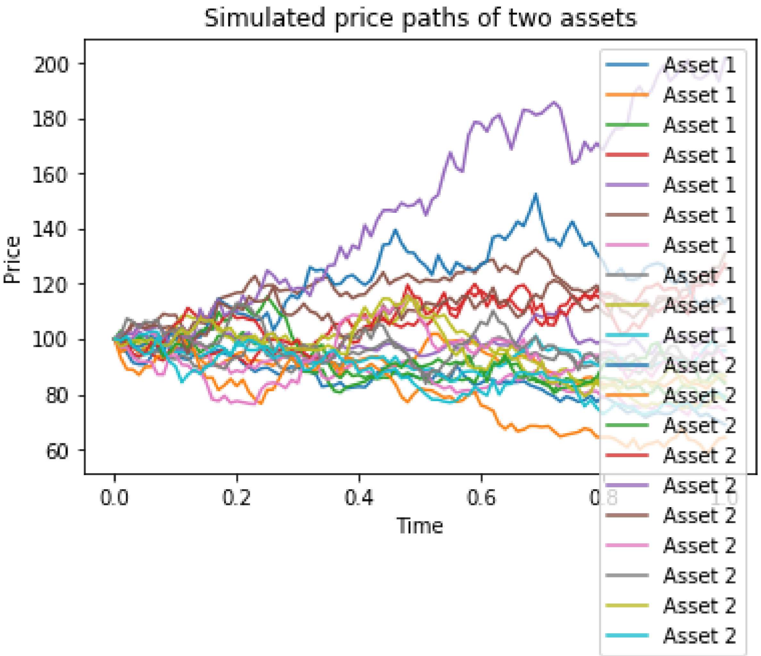
In [24]:
```
# Define parameters
S1 = 100 # Initial price of asset 1
S2 = 100 # Initial price of asset 2
K = 100 # Strike price
r = 0.05 # Risk-free interest rate
sigma1 = 0.2 # Volatility of asset 1
sigma2 = 0.3 # Volatility of asset 2
rho = 0.5 # Correlation between asset 1 and asset 2
T = 1 # Time to maturity
N = 1000 # Number of simulations
M = 100 # Number of time steps
dt = T/M # Time step
```

In [25]:
```
# Generate random scenarios
Z1 = np.random.normal(0, 1, (N, M)) # Standard normal random numbers for as
set 1
Z2 = rho*Z1 + np.sqrt(1-rho**2)*np.random.normal(0, 1, (N, M)) # Standard n
ormal random numbers for asset 2
S1_path = S1*np.exp(np.cumsum((r-0.5*sigma1**2)*dt + sigma1*np.sqrt(dt)*Z1,
axis=1)) # Simulated price path of asset 1
S2_path = S2*np.exp(np.cumsum((r-0.5*sigma2**2)*dt + sigma2*np.sqrt(dt)*Z2,
axis=1)) # Simulated price path of asset 2

# Compute payoffs
payoff = np.maximum(np.maximum(S1_path[:, -1], S2_path[:, -1]) - K, 0) # Pa
yoff of the best-of call option
price = np.exp(-r*T)*np.mean(payoff) # Discounted average payoff as the opt
ion price
print(f"The price of the best-of call option is {price:.2f}")
```

```
The price of the best-of call option is 18.40
```

In [26]:
```
# Plot some scenarios
plt.plot(np.linspace(0, T, M+1), np.insert(S1_path[:10, :], 0, S1, axis=1).
T, label="Asset 1") # Plot 10 scenarios of asset 1
plt.plot(np.linspace(0, T, M+1), np.insert(S2_path[:10, :], 0, S2, axis=1).
T, label="Asset 2") # Plot 10 scenarios of asset 2
plt.xlabel("Time")
plt.ylabel("Price")
plt.legend()
plt.title("Simulated price paths of two assets")
plt.show()
```



Section 2,Part a: Markov Chains for Srock Portfolio Valuation

In [28]:
```
# Import libraries
import numpy as np
import matplotlib.pyplot as plt
```

In [29]:
```
# Define the stock symbols and the initial portfolio value
symbols = ["AAPL", "MSFT", "AMZN", "GGOG"]
value = 10000
```

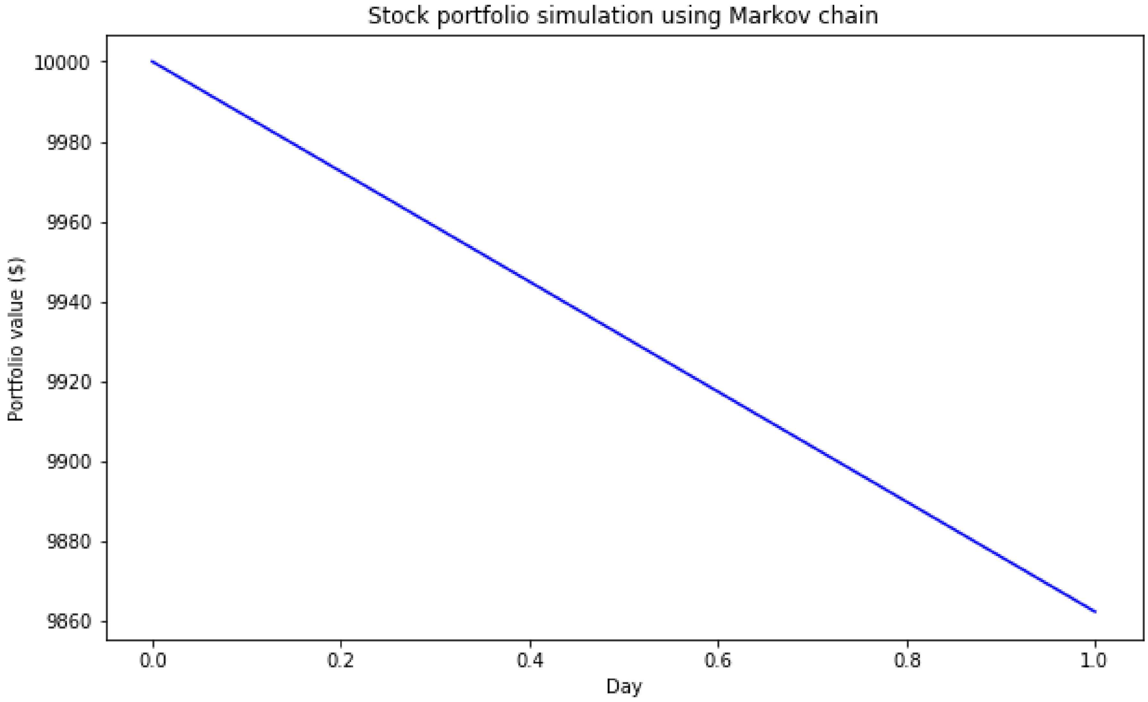
In [30]:
```
# Generate a random transition matrix for the Markov chain
# Each row represents the probability of switching from one stock to anothe
r
# The diagonal elements represent the probability of staying with the same
stock
np.random.seed(42) # For reproducibility
P = np.random.rand(4, 4)
P = P / P.sum(axis=1, keepdims=True) # Normalize the rows to sum to 1
print("Transition matrix:")
print(P)
```

```
Transition matrix:
[[0.14102156 0.35796222 0.27560979 0.22540643]
 [0.12620081 0.1261813  0.04698284 0.70063506]
 [0.2613905  0.30790022 0.00895102 0.42175826]
 [0.59038015 0.15059391 0.12895285 0.13007308]]
```

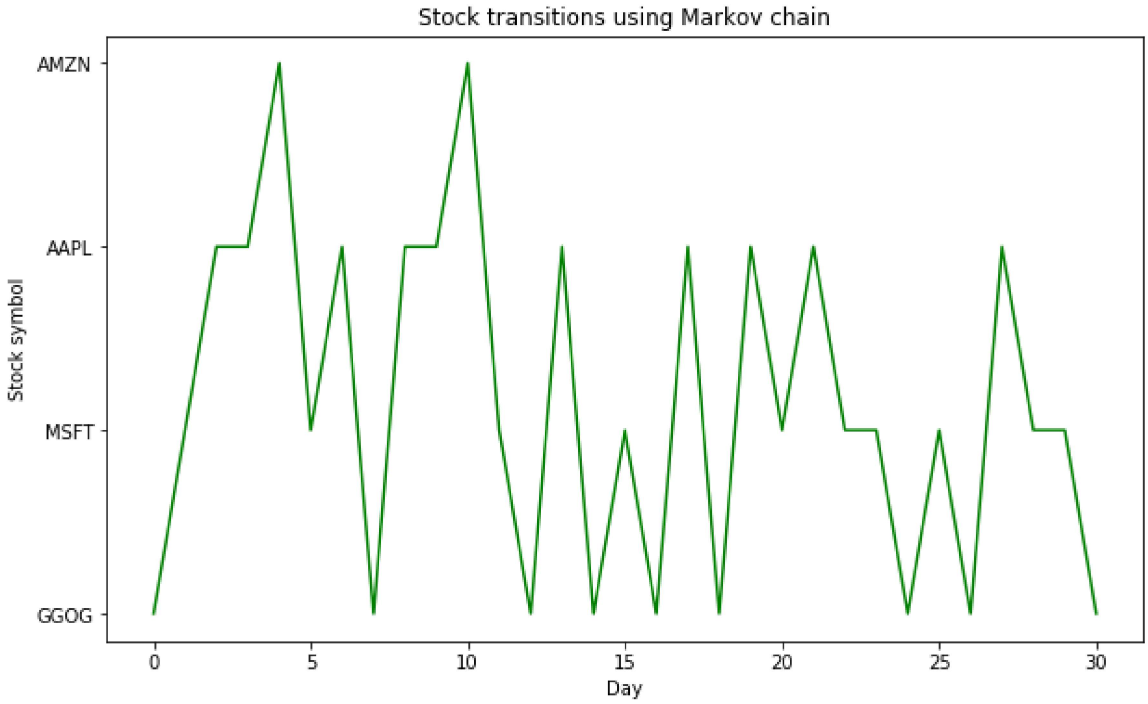
In [31]:
```
# Simulate the stock portfolio for 30 days using the Markov chain
# Assume that the portfolio value changes by a random percentage each day
# The percentage is drawn from a normal distribution with mean 0 and standa
rd deviation 0.01
days = 30
values = [value] # List to store the portfolio values
stocks = [np.random.choice(symbols)] # List to store the current stock
for i in range(days):
    # Choose the next stock according to the transition matrix
    current_stock = stocks[-1]
    current_index = symbols.index(current_stock)
    next_index = np.random.choice(4, p=P[current_index])
    next_stock = symbols[next_index]
    stocks.append(next_stock)
```

In [33]:
```
    # Update the portfolio value according to the random percentage change
    percentage_change = np.random.normal(0, 0.01)
    next_value = value * (1 + percentage_change)
    values.append(next_value)
    value = next_value# Plot the portfolio value over time
plt.figure(figsize=(10, 6))
plt.plot(values, color="blue")
plt.xlabel("Day")
plt.ylabel("Portfolio value ($)")
plt.title("Stock portfolio simulation using Markov chain")
plt.show()
```

In [34]:
```
# Plot the portfolio value over time
plt.figure(figsize=(10, 6))
plt.plot(values, color="blue")
plt.xlabel("Day")
plt.ylabel("Portfolio value ($)")
plt.title("Stock portfolio simulation using Markov chain")
plt.show()
```



In [35]:
```
# Plot the stock transitions over time
plt.figure(figsize=(10, 6))
plt.plot(stocks, color="green")
plt.xlabel("Day")
plt.ylabel("Stock symbol")
plt.title("Stock transitions using Markov chain")
plt.yticks(symbols)
plt.show()
```



Part b: Markov Chains for Rainbow Options Pricing

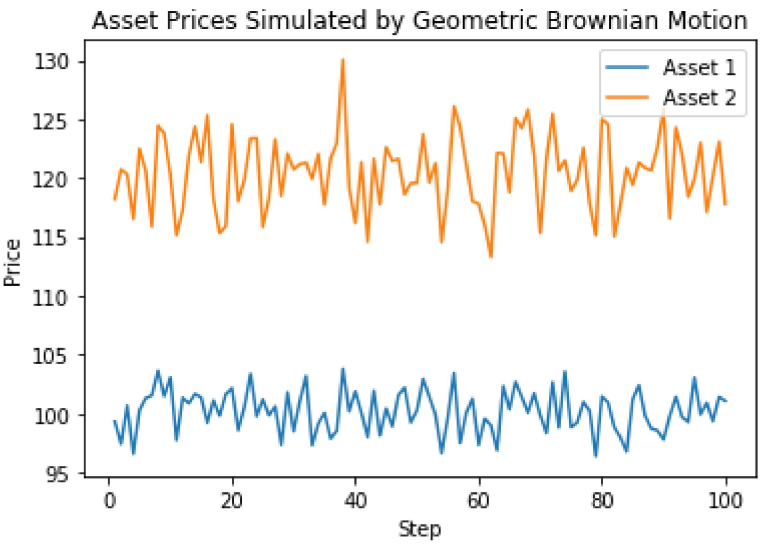
In [36]:
```
# Import libraries
import numpy as np
import matplotlib.pyplot as plt
```

In [37]:
```
# Define parameters
n = 100 # Number of steps
m = 2 # Number of assets
r = 0.05 # Risk-free rate
sigma = np.array([0.2, 0.3]) # Volatilities of assets
rho = 0.5 # Correlation between assets
S0 = np.array([100, 120]) # Initial prices of assets
K = 110 # Strike price of option
T = 1 # Time to maturity of option
```

In [38]:
```
# Generate correlated random variables
np.random.seed(42) # Set seed for reproducibility
Z = np.random.multivariate_normal(mean=[0, 0], cov=[[1, rho], [rho, 1]], si
ze=n)
W1 = Z[:, 0]
W2 = Z[:, 1]
```

In [39]:
```
# Simulate asset prices using geometric Brownian motion
dt = T / n # Time step
S1 = S0[0] * np.exp((r - 0.5 * sigma[0] ** 2) * dt + sigma[0] * np.sqrt(dt)
* W1)
S2 = S0[1] * np.exp((r - 0.5 * sigma[1] ** 2) * dt + sigma[1] * np.sqrt(dt)
* W2)
```

In [40]:
```
# Plot asset prices
plt.plot(np.arange(n) + 1, S1, label="Asset 1")
plt.plot(np.arange(n) + 1, S2, label="Asset 2")
plt.xlabel("Step")
plt.ylabel("Price")
plt.title("Asset Prices Simulated by Geometric Brownian Motion")
plt.legend()
plt.show()
```



In [41]:
```
# Calculate payoff of rainbow option
# Assume the option is a call on the maximum of the two assets
payoff = np.maximum(np.maximum(S1, S2) - K, 0)
```

In [42]:
```
# Discount payoff to present value
PV = payoff * np.exp(-r * T)
```

In [43]:
```
# Estimate option price as the mean of the present values
price = np.mean(PV)
```

In [44]:
```
# Print option price
print(f"The estimated price of the rainbow option is {price:.2f}")
```

```
The estimated price of the rainbow option is 9.96
```

Section 3,Part a: Branching Process for S&P 500 Daily Movements

In [45]:
```
# Import libraries
import numpy as np
import matplotlib.pyplot as plt
```

In [46]:
```
# Set parameters
n = 500 # number of stocks in the index
p = 0.55 # probability of a stock increasing in price
a = 0.01 # factor for positive change
b = 0.01 # factor for negative change
T = 100 # number of days to simulate
```

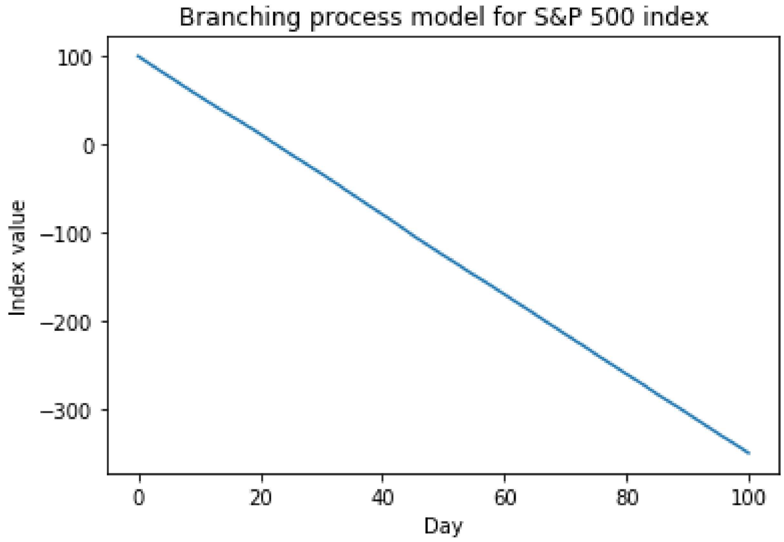
In [47]:
```
# Define offspring distribution
def offspring():
  # Generate a random number between 0 and 1
  u = np.random.uniform()
  # Return the change in the index value
  if u < p:
    return a - b
  else:
    return -(a + b)
```

In [48]:
```
# Initialize the index value
index = 100
```

In [49]:
```
# Initialize an empty list to store the index values
index_values = [index]
```

In [50]:
```
# Simulate the branching process for T days
for t in range(T):
  # Update the index value by adding the sum of n offspring
  index += sum(offspring() for i in range(n))
  # Append the index value to the list
  index_values.append(index)
```

In [51]:
```
# Plot the index values over time
plt.plot(range(T+1), index_values)
plt.xlabel('Day')
plt.ylabel('Index value')
plt.title('Branching process model for S&P 500 index')
plt.show()
```



Part b: Branching Process for Risk of Stock Portfolio

In [52]:
```
# Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

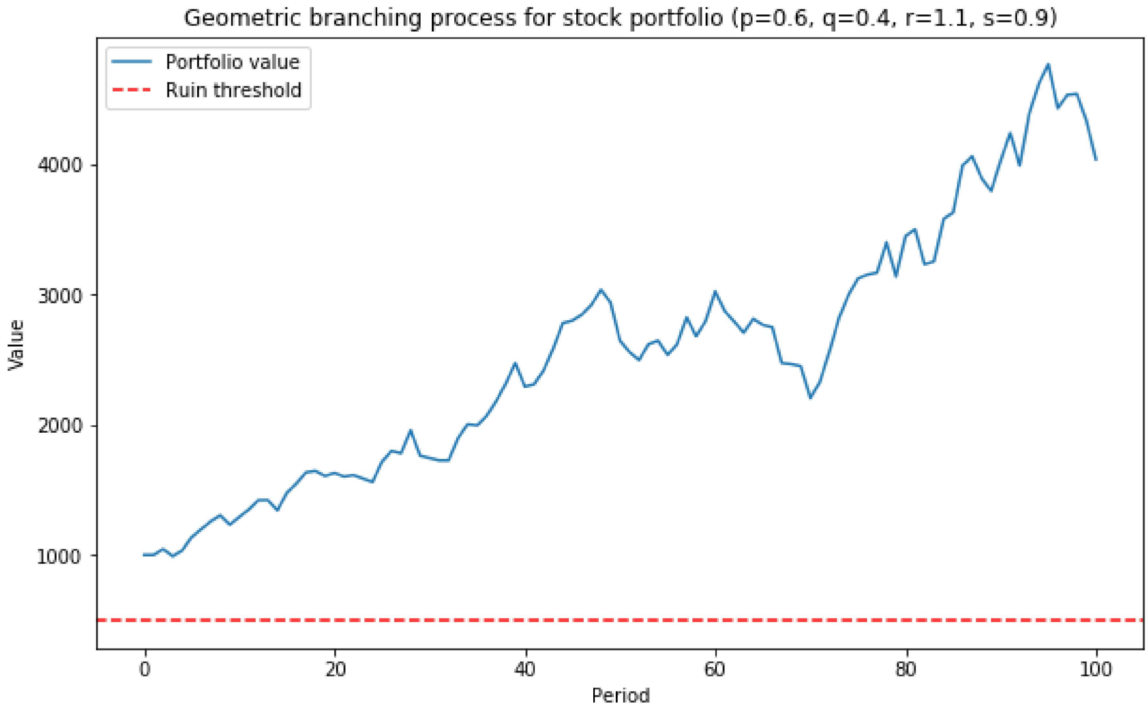
In [53]:
```
# Set random seed for reproducibility
np.random.seed(42)
```

In [54]:
```
# Define parameters
n = 100 # Number of periods
m = 4 # Number of stocks in the portfolio
p = 0.6 # Probability of increase
q = 0.4 # Probability of decrease
r = 1.1 # Increase factor
s = 0.9 # Decrease factor
v0 = 1000 # Initial portfolio value
vmin = 500 # Ruin threshold
```

In [55]:
```
# Simulate the geometric branching process
v = np.zeros((n+1,m)) # Portfolio value matrix
v[0,:] = v0/m # Initial value for each stock
for i in range(1,n+1):
    for j in range(m):
        x = np.random.binomial(1,p) # Bernoulli random variable
        if x == 1:
            v[i,j] = v[i-1,j] * r # Increase by factor r
        else:
            v[i,j] = v[i-1,j] * s # Decrease by factor s
```

In [56]:
```
# Compute the total portfolio value and the probability of ruin
vt = np.sum(v, axis=1) # Total portfolio value
pr = np.mean(vt < vmin) # Probability of ruin
```

In [57]:
```
# Plot the portfolio value and the ruin threshold
plt.figure(figsize=(10,6))
plt.plot(vt, label='Portfolio value')
plt.axhline(vmin, color='red', linestyle='--', label='Ruin threshold')
plt.xlabel('Period')
plt.ylabel('Value')
plt.title(f'Geometric branching process for stock portfolio (p={p}, q={q}, r={r}, s={s})')
plt.legend()
plt.show()
```



In [58]:
```
# Print the probability of ruin
print(f'The probability of ruin is {pr:.2f}')
```

```
The probability of ruin is 0.00
```