Case 3: Numéraires in a Binomial Tree and Monte Carlo

Asha de Meij - i6254733

November 14, 2024

Part 1: Numéraires in a Binomial Tree

Question 1. Use the money-market account B_t as a numéraire. Rewrite your binomial tree program to compute the expectation of a martingale in each binomial step. No discounting allowed, which means that during the tree-calculation you should use only the "expectation formula" $f[t,j] = p\, f[t+1, j+1] + (1-p)\, f[t+1, j]$. This requires that you divide the final payoff by the numéraire. At time $t = 0$, you convert the relative price back to € by multiplying with the numéraire. Check that your price is correct.

To do this in python i edited my "BinomialTree" method from week one and got the following.

```python
# CASE 1: Numeraire = Bank Account

def binomialTree_BankNum(r, sigma, T, guarantee, n):
    dt = T / n
    R_dt = np.exp(r * dt)

    u = R_dt * np.exp(sigma * np.sqrt(dt))
    d = R_dt * np.exp(-sigma * np.sqrt(dt))

    # Adjusted probability with bank as numéraire
    p_star = (R_dt - d) / (u - d)
    numMaturity = np.exp(r * T)


    # bank account values
    BankValue = 100 * R_dt ** np.arange(n + 1)

    # Stock prices
    stockValues = 100 * u**np.arange(n + 1) * d**(n - np.arange(n + 1))

    # option prices
    contractPrice = np.maximum(stockValues, guarantee)/numMaturity

    for i in np.arange(n, 0, -1):
        contractPrice = (p_star * contractPrice[1:i+1] + (1 - p_star) * contractPrice[0:i] )

    return contractPrice[0]
```

```python
price = binomialTree_BankNum(r, sigma, T, guarantee, n)
print("Option Price with bank account as numéraire:", price)
```

```
Option Price with bank account as numéraire: 113.45806426770213
```

Figure 1: Option price using Bank Account as Numéraire

Furthermore, by increasing the number of time steps we can see that the computed price approximates the the contract price more accurately (See Figure: 2).

```
Option Price with bank account as numéraire (n = 100): 113.45806426770213
Option Price with bank account as numéraire (n = 1000): 113.48815013916746
Option Price with bank account as numéraire (n = 10000): 113.48538436520136
Option Price with bank account as numéraire (n = 100000): 113.48500827520739
```

Figure 2: Option price with bank account as numéraire for different time steps

Question 2. Use the stock-price S_t as a numéraire and do the martingale calculation in the binomial tree (rewrite your code once more). Check that your price is still correct.

Similarly as in question 1, i made a second method that is also based on "BinomialTree" method from week one. The probabilities were adjusted since this time the stock is the numéraire and the computation of the option prices was also changed. This can be seen in the following code:

```python
def binomialTree_StockAsNumeraire(r, sigma, T, guarantee, n):
    dt = T/n
    R_dt = np.exp(r * dt)

    u = R_dt * np.exp(sigma * np.sqrt(dt))
    d = R_dt * np.exp(-sigma * np.sqrt(dt))


    p1_star = ((R_dt-d)*R_dt) / ((u-d)*R_dt)
    p2_star = ((u-R_dt)*R_dt) / ((u-d)*R_dt)

    # Stock price
    stock = 100 * u ** np.arange(0,n+1,1) * d ** np.arange(n,-1,-1)

    # option prices
    contractPrice = np.maximum(guarantee, stock)

    for i in np.arange(n,0,-1):
        contractPrice = (p1_star * contractPrice[1:i+1] + p2_star * contractPrice[0:i]) / R_dt

    return contractPrice[0]
```

```python
price = binomialTree_StockAsNumeraire(r, sigma, T, guarantee, n)
print("Option Price with Stock as numéraire:", price)
```

```
Option Price with Stock as numéraire: 113.45806426770045
```

Figure 3: Option price using Stock-price as Numéraire

Similarly as in question 1, by increasing the number of time steps we can see that the computed price approximates the the contract price more accurately (See Figure: 4).

```
Option Price with Stock-price as numéraire (n = 100): 113.45806426770045
Option Price with Stock-price as numéraire (n = 1000): 113.48815013916254
Option Price with Stock-price as numéraire (n = 10000): 113.48538436531159
Option Price with Stock-price as numéraire (n = 100000): 113.48500827500095
```

Figure 4: Option price with Stock-price as numéraire for different time steps

By comparing the the values we got by using the Stock-price and then the money-market account as Numéraire, we can see that the option prices are the same no matter which numéraire we chose(last 3 digits are not).

Question 3. Would it be possible to use the unit-linked contract itself as a numéraire?

The unit-linked contract is essentially has a strictly positive price. If we would use the unit-linked contract as the numéraire, then the martingale pricing formula would be:

$$\frac{C[0]}{C[0]} = \mathbb{E}^* \left[\frac{C[1]}{C[1]} \right]$$

If we multiply both sides by C[0], we get:

$$C[0] = \mathbb{E}^* \left[\frac{C[1]}{C[1]} \right] C[0]$$

Now we can see that it does not make sense using the unit-linked contract as a numéraire, since we would express the price of the contract in terms of the UL contracts.

Question 4. Would it be possible to use a call-option (payoff $\max(S_T - K, 0)$) as a numéraire?

No a call option can not be used as a numéraire in the binomial model. The reason for this is that the call option's payoff depends on the underlying asset price relative to the strike price, and if the asset price falls below the strike price, the call option's value becomes zero. This means that the call option is not strictly positive because it can also take the value of zero. This is why a call option isn't suitable as a numéraire.

Question 5. Would it be possible to use cash (a constant process $C_t \equiv 1$) as a numéraire?

When interest rates are positive, holding cash becomes less attractive compared to depositing it in a bank to earn interest, as the bank account would grow over time while cash remains constant. Conversely, when interest rates are negative, holding cash outside of banks becomes more attractive, as it avoids the cost of negative interest. Because of these different shifts in attractiveness based on interest rate changes, cash is generally not practical as a numéraire in financial models. In this case if the cash remains constant it cannot be used as numéraire in the martingale pricing.

Part 2: Monte Carlo Simulation

Write a computer program to compute the price of a UL contract with a Monte Carlo simulation.

```python
def MonteCarloSimulation(S0, T, r, sigma, N, M, guarantee):

    dt = T/N
    Z = np.random.normal(0, 1, (M, N)) # random paths

    # Stock prices
    S = np.zeros((M, N+1))
    S[:, 0] = S0

    # Monte Carlo simulation
    for t in range(1, N+1):
        S[:, t] = S[:, t-1] * np.exp((r - 0.5*sigma**2)*dt + sigma*np.sqrt(dt)*Z[:, t-1])

    # Calculate payoff
    optionPrice = np.maximum(S[:, -1], guarantee)

    # Discount payoffs
    optionPriceFinal = np.exp(-r*T) * np.mean(optionPrice)
    SE = np.exp(-r*T) * np.std(optionPrice) / np.sqrt(M)

    return optionPriceFinal, SE
```

Figure 5: Monte Carlo Simulation code in Python

Question 6. Investigate the convergence behaviour for increasing number of paths M. Confirm the $\mathcal{O}(M^{-1/2})$ convergence rate.

To answer this question, I plotted the standard error against the number of paths M, as shown below in Figure 4. By observing this plot, we can see that the standard error decreases as M increases, following a $\frac{1}{\sqrt{M}}$-shaped curve which confirms the $\mathcal{O}(M^{-1/2})$ convergence rate.

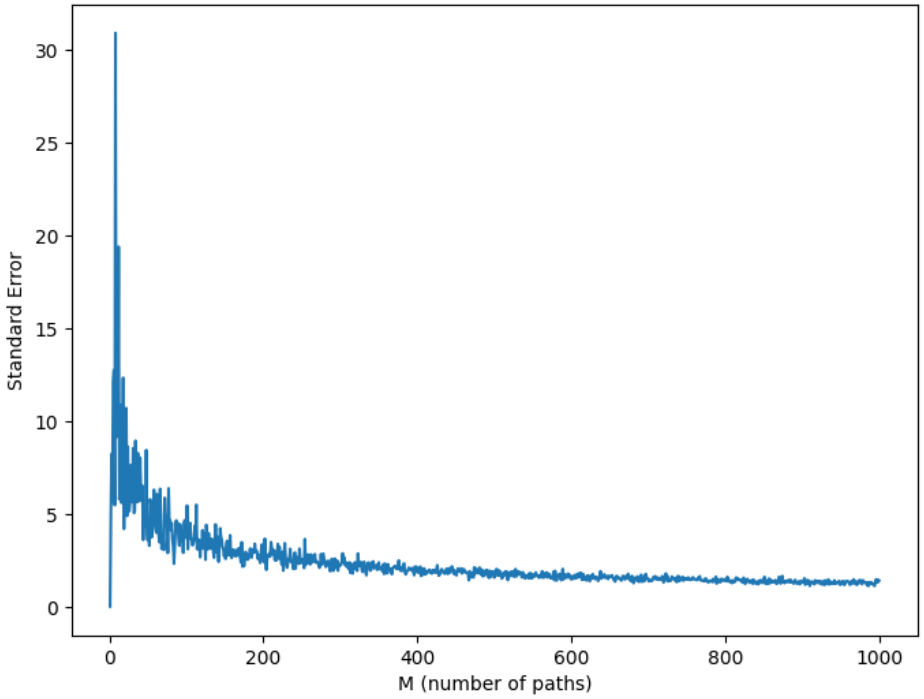

Figure 6: Standard Error vs. Number of Paths

Question 7. Re-run your MC calculation multiple times for $M = 1000$ (using different seed-values for the random number generator!). Plot the empirical distribution function of the estimator $\hat{E}$. Compare this distribution to the asymptotic approximation $\hat{E} \approx \mathcal{N}(\mathbb{E}[f], \mathrm{Var}[f]/M)$.

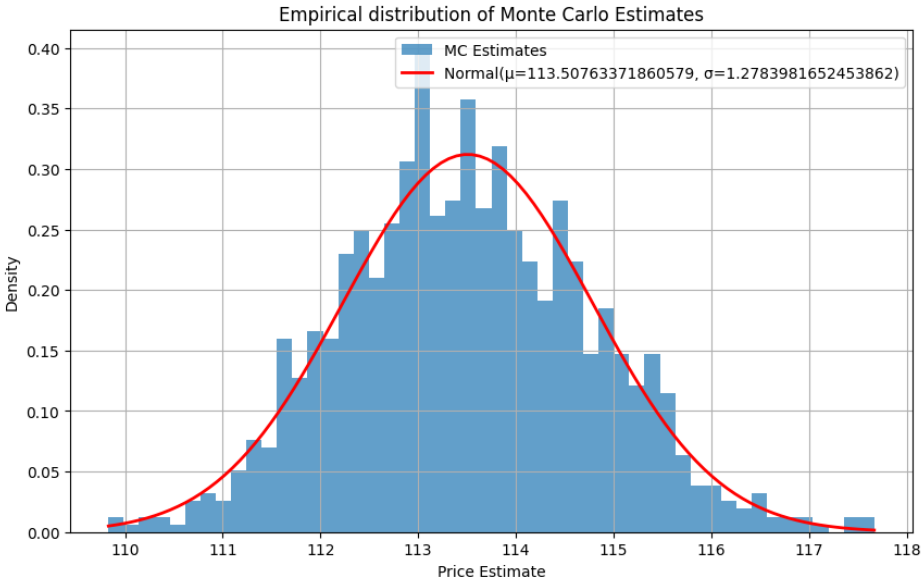

Figure 7: Distribution of Monte Carlo Estimates

By examining the graph, we can observe that the distribution of the Monte Carlo estimates closely resembles a normal distribution, as evidenced by the asymptotic distribution shown as the red line.

Question 8. If you generate M paths with N time-steps, then you need MN random numbers in total. Suppose you have a fixed budget of 10^6 random numbers. Is it better to choose a large M (with smaller N) or choose a large N (with smaller M) to obtain the most accurate MC estimate for a fixed "budget" $MN \leq 10^6$?

To answer this question, I performed an analysis using Monte Carlo simulations with different values of M (number of paths) and N (number of time steps) under the constraint that $MN \leq 10^6$, as follows:

- **High M with Low N**: In this scenario, I performed the simulation by allocating a high number of paths M and a smaller number of time steps N, such as $M = 10000$ and $N = 100$.
- **Low M with High N**: In this case, I used a small number of paths M and a high number of time steps N, such as $M = 100$ and $N = 10,000$.

The results of this experiment can be found in the image below:

```
M=10000, N=100: Mean Price=113.39179462762517, Std Dev=0.40459422528421607
M=5000, N=100: Mean Price=113.5874663267173, Std Dev=0.41147403342845096
M=2000, N=500: Mean Price=113.26138501332203, Std Dev=0.7230369287539823
M=100, N=10000: Mean Price=112.39922689798777, Std Dev=2.8515058284973382
M=100, N=5000: Mean Price=115.06664324582525, Std Dev=4.305421484383523
M=500, N=2000: Mean Price=113.70823322398124, Std Dev=1.356851468526291
```

Figure 8: Fixed Budget Analysis

From these results, it is evident that using a higher number of paths M while keeping the number of time steps N lower has better results (= better mean approximate and lower SE) for a fixed random number budget. For example using the combination $M = 10{,}000$ and $N = 100$ has the smallest standard deviation and produces a mean price closest to the actual price, which suggests that it's the most reliable Monte Carlo estimate.

In contrast, when M is low and N is high (e.g., $M = 100$ and $N = 10{,}000$), the estimate is highly variable (large SE) and deviates more from the actual price. This results is quite intuitive in my opinion since the since increasing the number of time steps does not "stabilise" this mean value, only by allocating more paths result in a better mean value. Furthermore, another reason why this makes sense is due to the simplicity of the geometric Brownian motion used in this process. Since the SDE doesn't require approximation, we can directly simulate the variables at the final time step without needing multiple intermediate steps. This is possible because Brownian motion scales with time and has independent increments.