Computational Finance Assignment 1

Misho Yanakiev, Max Meijer, Diogo Franquinho

February 21, 2021

In this report, we will consider a European call option on a non-dividend-paying stock with a maturity of one year and a strike price of €99. The one-year continuously compounded interest rate is assumed to be 6%. The current price of the stock is assumed to be €100 and its volatility is assumed to be 20%.

1 Approximating the option price with a binomial tree program

We have written a binomial tree program to find the option price. The computed option price is 11.546434850755055. The algorithm is explained below.

First, we construct a binomial tree of the stock prices. To do so, we first need to compute u and d. These are given by the following formulas:

$$u = e^{r\sqrt{\Delta t}} \qquad d = e^{-r\sqrt{\Delta t}}$$

The binomial tree B is a 2-dimensional array containing at index (i, j) the stock price after i time steps if j up-moves happened in those i time steps. Note that the number of up-moves is sufficient to determine the stock price because multiplication is commutative and associative. This binomial tree can be efficiently calculated in $O(N^2)$ by using the fact that $B(i, j)$ can be written in terms of u, d, $B(i-1, j)$ and $B(i-1, j-1)$. For $i = 0$, we have that $B(0, 0) = S$ where S is the stock price at time 0. For $0 \leqslant j < i$, we can write $B(i, j) = B(i-1, j) \cdot d$ (because to go from $(i-1, j)$ to (i, j) we need a down-move to happen) and for $0 < j = i$, we can write $B(i, j) = B(i-1, j-1) \cdot u$ (because to go from $(i-1, j-1)$ to (i, j) we need an up-move to happen).

Next, we reason backwards to determine the option price, starting at the maturity time. To do this, we also need to compute the risk-neutral probability p which can be computed using equation (3):

$$p := \frac{e^{r\Delta t} - d}{u - d}$$

Furthermore, we compute the payoff of the option at maturity time based on the stock price using the formula $C(S_T) = \max(0, S_T - K)$ where S_T is the stock price at maturity time and K is the strike price. The stock price at maturity time can be found by looking in the last row of the binomial tree (it depends on the number of up-moves). We can then reason backwards, computing the fair price $f(i, j)$ at time i if j up-moves happened by using equation (4) from the appendix of the assignment:

$$f(i, j) = e^{-r\Delta t}(pf(i+1, j+1) + (1-p)f(i+1, j))$$

After calculating the valuation at every $f(i, j)$, we compute $f(0, 0)$ to get the value of the option at time 0, giving the result 11.546434850755055.

1.1 Complexity

Since we have two nested **for** loops of order N (since $N(N+1)/2 = O(N^2)$) with $O(1)$ operations in them we get that the algorithm has time complexity of $O(N^2)$.

2 Comparison between binomial tree and Black-Scholes models

The Black-Scholes model is another way to determine the value of the option price. We implemented the Black-Scholes formula in Python. For this exercise we needed to evaluate it at $t = 0$. The Black-Scholes formula is given by:

$$V(t) = S_t N(d_1) - e^{r\tau} K N(d_2), \text{ with } d_1 = \frac{\log(\frac{S_t}{K}) + (r + \frac{1}{2}\sigma)r}{\sigma\sqrt{\tau}}, \text{ and } d_2 = d_1 - \sigma\sqrt{\tau},$$

where $\tau = T - t$, and $N(x)$ is the c.d.f. of the standard normal. If we fill in the same parameters as in Question 1, we obtain 11.544280227051097 as a result. This a bit lower than the result from Question 1. This means that our simulation estimate is off by approximately 0.0021546237. This is unsurprising, because we lose some accuracy in the binomial tree program by only having the stock price move at a finite number of time points.

2.1 Rate of Convergence

We analyzed what the effect is of increasing N on the error of the simulation with respect to the result of applying the Black-Scholes formula. The results of this analysis are plotted in fig. 1. It can be seen that indeed, the error of the estimate seems to converge to zero. It is interesting to note that the error alternates between positive and negative values.

We have also created another version of this graph with log scales on both axes and the absolute value of the error instead of the error. This is shown in fig. 2. This graph makes it more clear that accuracy keeps increasing if we increase the number of steps, although the convergence does not seem to be monotone.

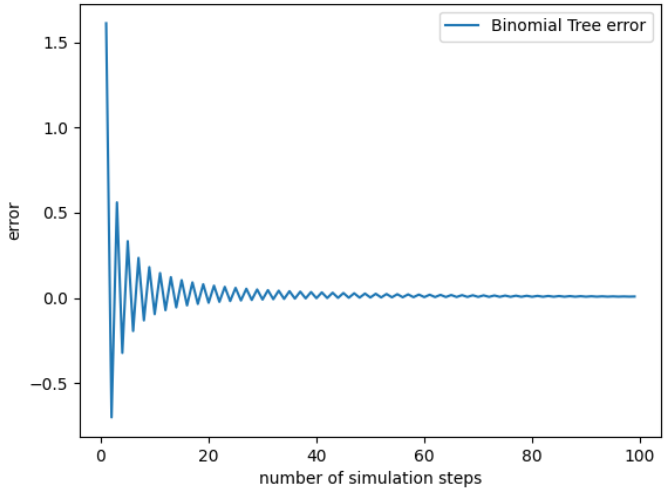

Figure 1: Error of European call option price estimation using the binomial tree program, for different numbers of steps used

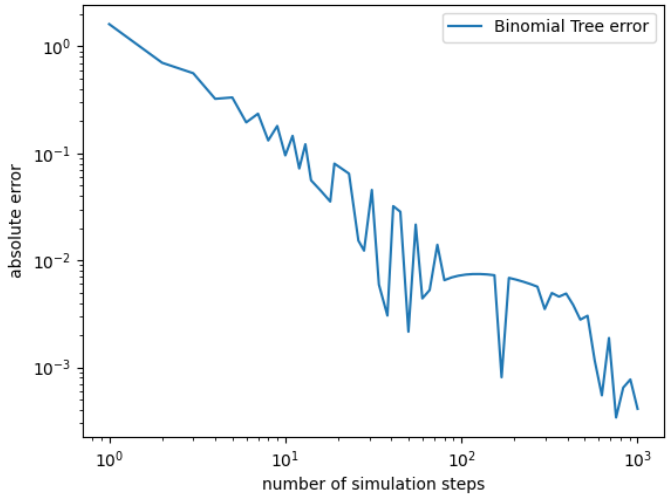

Figure 2: Absolute error of European call option price estimation using the binomial tree program, for different numbers of steps used (log scale on both axes)

3 Experiments with different values of the volatility

Furthermore we wanted to investigate how changing values of the volatility would change our stock price and moreover how our estimate price of the option would compare to the analytical value for also changing volatility values. Therefore we used our previous models and plotted the option price, and volatility values $(1\%, 2\%, \ldots, 99\%)$, see fig. 3. We plotted both our 50-step binomial tree estimate and the the Black-Scholes formula to calculate the option price. From the plot it becomes clear that the values are very similar.

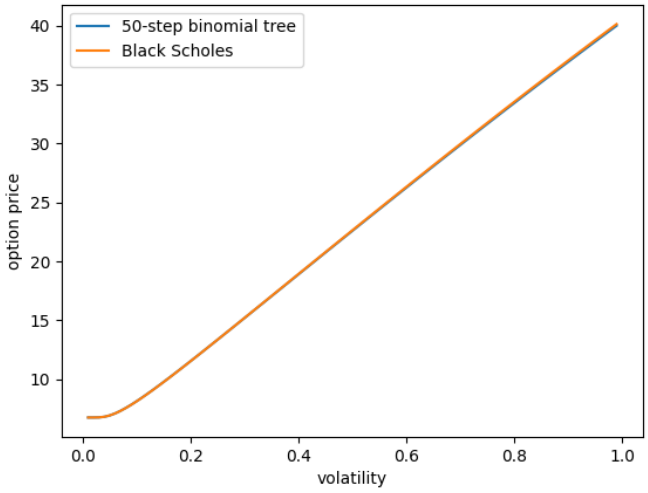

Figure 3: Price of the option in relation to volatility values

From fig. 3 we can see that as the volatility increases the option price will also increase. Intuitively this makes sense as lower values of volatility imply that the stock price won't deviate much from the initial value, making our option to buy the stock at maturity time less valuable.

4 Delta-parameter in the Black-Scholes model

We want to calculate $\Delta_t = \frac{\partial C_t}{\partial S_t}$, where C_t is the price of a European-Call option in the Black-Scholes model.

$$C_t = V(t) = S_t N(d_1) - e^{r\tau} K N(d_2)$$

where $N(x)$ is the c.d.f. of the standard normal distribution. Hence, its derivative $N'(x)$ is the p.d.f. of the standard normal distribution, i.e. $N'(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$. We also have that d_1 is a function of S_t and d_2 is a function of d_1. We can derive the following:

$$\begin{aligned}
\Delta_t &= N(d_1) + S_t N'(d_1) d_1' - e^{-r\tau} K N'(d_1 - \sigma\sqrt{\tau})(d_1 - \sigma\sqrt{\tau})' \\
\Delta_t &= N(d_1) + d_1'(S_t \frac{1}{\sqrt{2\pi}} e^{-\frac{d_1^2}{2}} - e^{-r\tau} K \frac{1}{\sqrt{2\pi}} e^{-\frac{d_1^2 - 2d_1\sigma\sqrt{\tau} + \sigma^2\tau}{2}}) \\
\Delta_t &= N(d_1) + d_1' \frac{1}{\sqrt{2\pi}} e^{-\frac{d_1^2}{2}} (S_t - e^{-r\tau} K e^{d_1\sigma\sqrt{\tau} - \frac{1}{2}\sigma^2\tau})
\end{aligned}$$

Now we just need to prove that

$$S_t - e^{-r\tau} K e^{d_1\sigma\sqrt{\tau} - \frac{1}{2}\sigma^2\tau} = 0 \tag{1}$$

From the definition of d_1 we have that $d_1 = \frac{\log(\frac{S_t}{K}) + (r + \frac{1}{2}\sigma)r}{\sigma\sqrt{\tau}}$, so by substitution we have:

$$\begin{aligned}
S_t - e^{-r\tau} K e^{d_1\sigma\sqrt{\tau} - \frac{1}{2}\sigma^2\tau} &= \\
S_t - e^{-r\tau} K e^{\log(\frac{S_t}{K}) + (r + \frac{1}{2}\sigma^2)\tau - \frac{1}{2}\sigma^2\tau} &= \\
S_t - e^{-r\tau} K e^{\log(S_t) - \log(K) + r\tau} &= \\
S_t - e^{-r\tau} K e^{\log(S_t)} e^{-\log(K)} e^{r\tau} &= \\
S_t - S_t = 0 &
\end{aligned}$$

Therefore we have that $\Delta_t = N(d_1)$

5 Delta-parameter in the binomial tree model

We calculate first the values for a volatility $\sigma = 20\%$. The theoretical delta given by the formula:

$$\Delta_t = N(d_1)$$

turned out to be 0.67373551 while the tree model delta was 0.67255696 for $t = 0$. The relative error between the two is 0.17%. We then tested for a set of 99 volatility values in $[1\%, 99\%]$. Figure 4 shows how the two deltas compare over this and it there seems to be a lower bound for the delta at around 0.65, and both the theoretical delta and the binomial tree delta seem to follow the same pattern. Figure 5 gives the relative percentage error at each point, based on these results we can conclude that the binomial tree is a very effective approximation of the Black-Scholes equation for a European option in discrete time, as the highest error given is around 0.30%. From the graph we can see that the error is minimal at low volatilities and peaks at around 15%. From then on there is a sharp decrease and a slowly varying error between 0.10% and 0.15%.

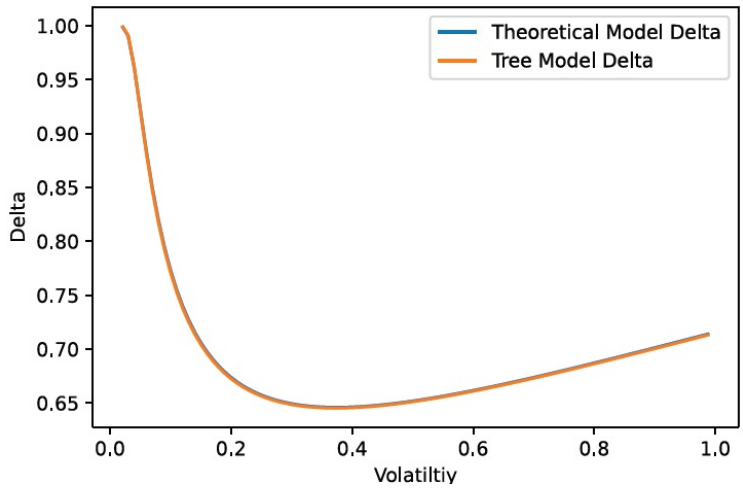

Figure 4: Comparison of Black Scholes and Binomial Tree Delta Values

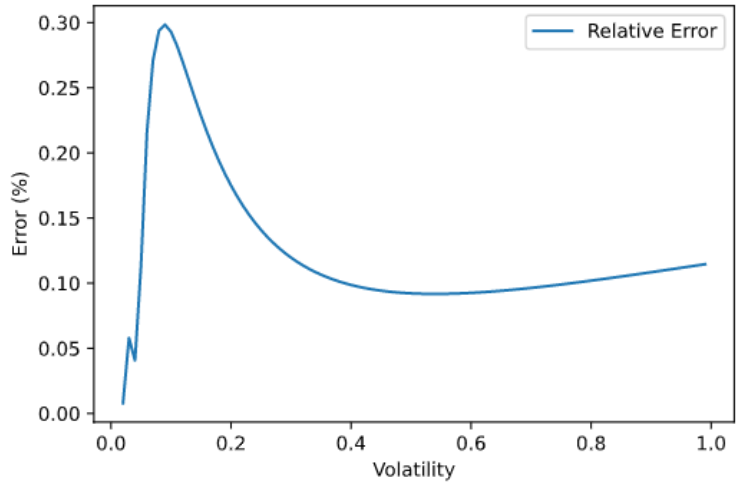

Figure 5: Percentage Error Between Black Scholes and Bin. Tree Deltas

6 American call and put options

We changed the algorithm so that it can also handle American call / put options. With the European call option we only needed to compute a maximum in the final time step because that is the only moment that the option holder has a decision. For the American call option at any time step, there are now two possible choices for the holder of the option: keep the option, or execute it (call or put, depending on the type). Thus the payoff is then calculated as max(keep value, use value). The keep value is computed in the same way as for the European option, i.e. $\exp(-r\Delta t) * (p * f_u + (1-p) * f_d)$. The computation of the use value depends on the type of the option for a call option it will be the current stock price minus the strike price, while for a put option it will be the strike price minus the current stock price.

Using the same parameters as in Question 1, we obtain an American call option price of 11.546434850755055 and an American put option price of 5.347763498417657. Note that the price for the American call option is exactly the same as for the European one. This is because at any point in time, regardless of what happened before, the expected value of the stock in the future is greater than its current value. Therefore, it is always at least as good to wait before using the option, even if you are certain that you will use it. But for the American put option the price is gonna be different from the European one, as the value we get using the Black-Scholes formula is 4.778971812289001 for the European put option, and the same argument applies.

6.1 Experiments with different values of the volatility

We also experimented the effect of volatility changes to the American call and put options. This is plotted in fig. 6. It can be seen that the difference between the graphs remains roughly the same as the volatility changes. To confirm this we plotted the difference between the price of the American call and put options in the same graph. For both the call and the put option we see that the price increases as the volatility is increased. This makes sense, because if the stock price doesn't move much then there is not much value in being able to decide whether to use the option or not.

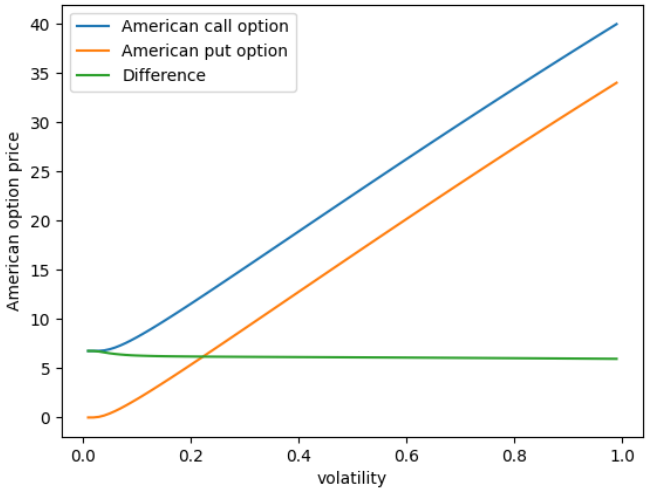

Figure 6: Price of American call and put options, computed using a 50-step binomial tree simulation

7 Derivation of the Black-Scholes put price

Using the formula for $C_t = S_t N(d_1) - e^{-r\tau} N(d_2)$ and the put-call parity $C_t - P_t = S_t - e^{-r\tau} K$, we get:

$$\begin{aligned} P_t &= C_t - S_t + e^{-r\tau} K \\ &= S_t N(d_1) - e^{-r\tau} N(d_2) - S_t + e^{-r\tau} K \\ &= S_t (N(d_1) - 1) + e^{-r\tau} K (1 - N(d_2)) \\ &= e^{-r\tau} K (1 - N(d_2)) - S_t (1 - N(d_1)) \\ &= e^{-r\tau} K N(-d_2) - S_t N(-d_1) \end{aligned}$$

8 Volatility Based Hedging Simulations

In this section, we discuss an algorithm / strategy that can hedge / mimic the European call option with only a portfolio of stocks and bonds. This requires changing the composition of the portfolio as time progresses.

The basic idea behind the algorithm is that the $\Delta = \frac{\partial C_t}{\partial S_t}$ that we computed earlier describes how many stocks we need to hold to get the same volatility as the stock. Note that this does depend on how volatile the stock is. If our estimate of stock were to be wrong, then the hedging quality would suffer.

First, we discuss the effect of how frequently we update the composition of the portfolio on the hedging performance. For this purpose, we maintain a stock and delta volatility of 0.2. Secondly, we examine the effect of using the wrong volatility during the hedging process.

8.1 Comparison between daily and weekly hedging

We expect that the hedging becomes more accurate if the composition of the portfolio is updated more frequently. For this experiment, we use the same value for the stock's volatility as for the hedging. We ran the Euler-Maruyama algorithm with the discretization of:

$$S_i = S_{i-1} + rS_{i-1}\Delta t + \sigma S_{i-1}\sqrt{\Delta t}Z_i$$

where $Z_i \sim \mathcal{N}(0,1)$ and $i \in \{0, 1, ..., N\}$. We compare the effect of daily hedging (over 365 days) and weekly hedging (over 52 weeks). At each time that we reallocate our portfolio (i.e. every day or every week), we compute the current Δ using the formula $\Delta = N(d_1)$. In fig. 7, we have plotted the Δ (i.e. the amount of stocks we hold) over time to illustrate the difference between daily and weekly hedging. By construction the weekly graph is the same as the daily graph, except that it is linearly interpolated on days at which the portfolio composition is not changed.

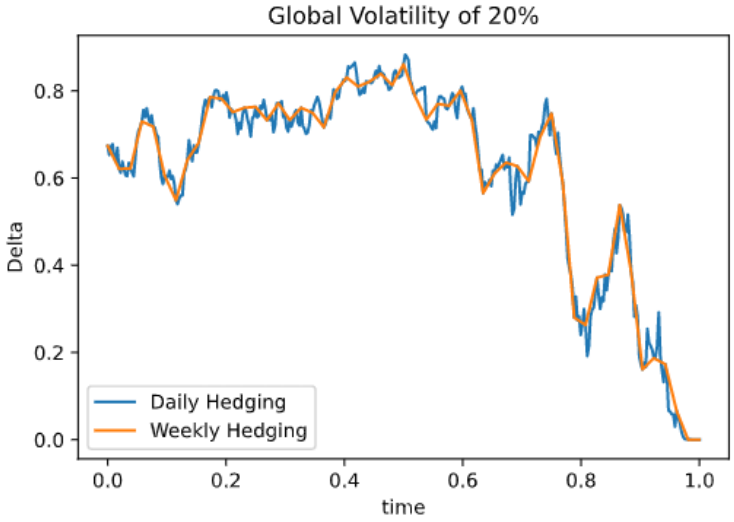

Figure 7: Comparison of weekly and daily hedging using a volatility of 20% for the stock process and the hedge

Ideally, we would like to have that the hedging portfolio provides the exact same returns as the payoffs of the actual option would have been. Thus, we are interested in the distribution of the hedge error, the difference between the returns of the hedging portfolio and the payoff of the option. To investigate this, we simulated 1000 sample paths of the stock price process and applied the hedging strategies along these and computed the hedge error for each simulation. In fig. 8, we have plotted the (normalized) histogram of our results. The hedge error was converted to the relative hedge error by dividing it by the option price.

We used two different versions of the hedging strategy, namely daily hedging and weekly hedging. The difference is in the frequency with which we adjust the hedging portfolio. The higher the frequency of the hedging the higher we expect the accuracy of the hedging to be, since we will be closer to a continuously (perfectly) hedged portfolio. This difference can also be seen in the fig. 8. The relative hedge error for the daily hedging is clustered more tightly around zero.

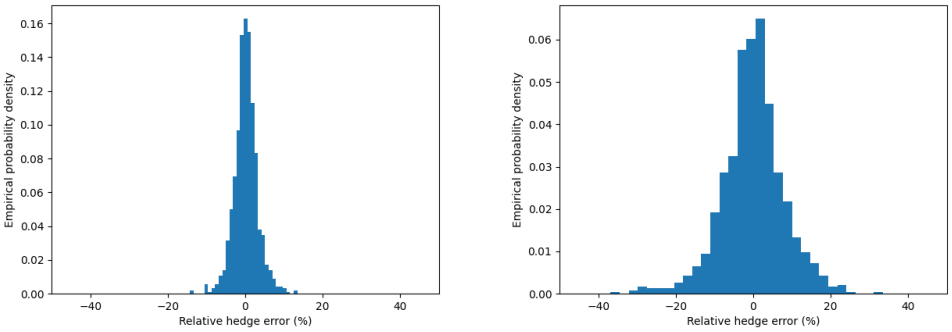

Figure 8: Distribution of hedge error when using the correct volatility (volatility: 20%). Left: daily hedging. Right: weekly hedging.

8.2 Variable Delta Volatility

Here, we follow the same procedure as in the previous scenario with the key exception that now the delta volatility (i.e. the volatility that is used within the model) does not match the stock volatility. The resultant histograms can be seen in fig. 9. This time, we can see that the spread in the error density is much larger than in the case of matching with the true volatility. This was to be expected since our model does not have a correct interpretation of the market and as such is more likely to miscalculate the hedging strategy. The extreme values for the relative error are rather large at 40% which is definitely a cause for concern, but it is also to be noted that this corresponds to the case where we have a discrepancy of 100% between the true and the hedge volatility. A conjecture can be made, which we do not investigate here, on whether the maximal error bounds for the model have a direct correspondence to the relative discrepancy between the stock and hedge volatilities.

It can also be seen that if the wrong stock volatility is used then the difference between weekly and daily hedging isn't as clear anymore. This makes sense, because the hedging is going to be wrong anyway (because the volatility is wrong) and doing the wrong thing more precisely won't lead to significantly better results.

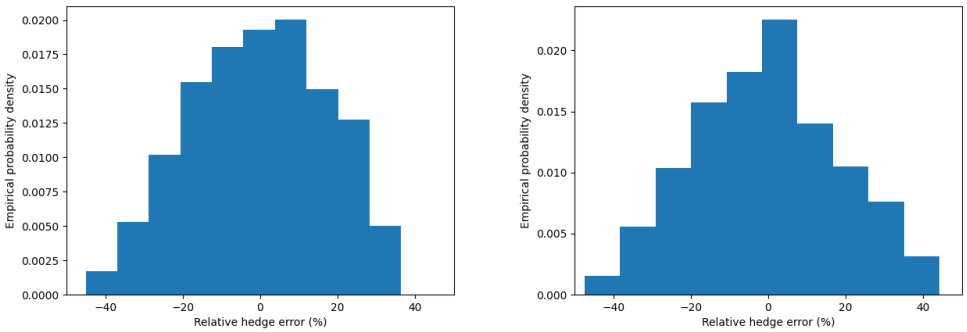

Figure 9: Distribution of hedge error when hedged based on assumption of 40% volatility while the actual volatility is 20%. Left: daily hedging. Right: weekly hedging.

In fig. 10 on the next page we vary the range of the delta volatility with 400 values in the span of $[0.01, 0.40]$ with 100 simulations each. From left to right, it can be seen that the relative error approaches zero as the delta volatility approaches the stock volatility of 20% and then the relative error diverges from zero again as the the delta volatility increases beyond 20%. This is unsurprising as the larger the difference in true and model volatility, the greater the error in pricing we can expect.

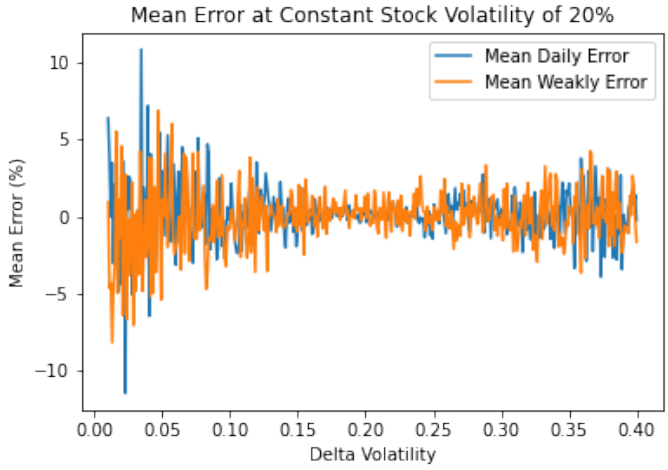

Figure 10: Comparison of the daily and weekly hedge error with different values of delta volatility for a fixed stock volatility of 20%

A Appendix

```
# %%
import numpy as np
import matplotlib.pyplot as plt
from math import exp, log, sqrt
from scipy.stats import norm

# Compute u and d using formulas at the end of A.2.4
def computeUD(vol, dt):
  u = exp(vol * sqrt(dt))
  d = exp(-vol * sqrt(dt))
  return u, d

def buildTree(S, vol, T, N):
  dt = T / N
  matrix = np.zeros((N+1, N+1))

  # up and down move multiplier
  u, d = computeUD(vol, dt)

  matrix[0][0] = S

  # Iterate over the lower triangle
  for i in np.arange(1, N+1): # i is the time
    for j in np.arange(i): # j is the number of up moves
      # matrix[i,j] is the same thing as at the previous time (i-1),
      # but with an additional down-move
      # (Because the number of up-moves stays the same)
      # This means that we multiply by d.
      matrix[i, j] = matrix[i-1,j] * d
    # In case we have i up-moves within i time steps,
    # we have the same thing as i-1 up-moves within i-1 time steps
```

```
    # but with an additional up-move
    # This means that we multiply by u.
    matrix[i,i] = matrix[i-1,i-1] * u

  return matrix

#%%

# Compute the risk-neutral probability p
def computeP(r, dt, u, d):
  # Use formula (3) in A.2
  return (exp(r*dt)-d) / (u-d)

def valueOptionMatrix(tree, T, r, K, vol, N, american = False, put = False):
  dt = T / N
  u, d = computeUD(vol, dt)
  p = computeP(r, dt, u, d)

  columns = tree.shape[1]
  rows = tree.shape[0]

  # Walk backward, we start in the last row of the matrix

  # Add the payoff function in the last row
  for c in np.arange(columns):
    S = tree[rows - 1, c] # value of the stock
    tree[rows - 1, c] = max(0, K - S) if put else max(0, S - K)

  # For all other rows, we need to combine from previous rows
  # We walk backwards, from the last row to the first row

  for i in np.arange(rows - 1)[::-1]:
    for j in np.arange(i + 1):
      currentStockPrice = tree[i,j]
      down = tree[i+1, j]
      up = tree[i+1, j+1]
      # Use equation (4) from A.2.3
      keep_value = exp(-r*dt) * (p*up + (1-p)*down)
      use_value = (K - currentStockPrice) if put else (currentStockPrice - K)
      tree[i, j] = max(keep_value, use_value) if american else keep_value
  return tree

def computeCallOptionPrice(S, sigma, T, N, r, K, american = False, put = False
  ↪ ):
  tree = buildTree(S, sigma, T, N)
  matrix = valueOptionMatrix(tree, T, r, K, sigma, N, american, put)
  return matrix[0, 0]

def Normal(x):
  return norm.cdf(x)

def computeAnalytically(vol):
  d1 = (log(S / K) + (r + vol * vol / 2) * T)/vol / sqrt(T)
  d2 = d1 - vol * sqrt(T)
```

```
  return S * Normal(d1) - exp(-r*T)*K*Normal(d2)

sigma = 0.20 # 20% volatility
S = 100. # stock price in euros
T = 1. # 1 year maturity
N = 50 # Number of steps
K = 99. # Strike price
r = 0.06 # 6% interest rate

# Question 1
print("Estimated price:", computeCallOptionPrice(S, sigma, T, N, r,K))
print("Black Scholes:", computeAnalytically(sigma))

# Question 2
# Volatility
vols = np.linspace(0.01, 0.99, 99)
def computePriceByVolatility(vol):
  return computeCallOptionPrice(S, vol, T, N, r, K)
results = list(map(computePriceByVolatility, vols))
theoretical = list(map(computeAnalytically, vols))

plt.plot(vols, results, label='50-step binomial tree')
plt.plot(vols, theoretical, label='Black Scholes')
plt.xlabel('volatility')
plt.ylabel('option price')
plt.legend()
plt.savefig('volatility-european-price.png')
plt.close()

plt.plot(vols, (np.array(results) - np.array(theoretical))/np.array(
    ↪ theoretical)*100, label='Relative error (%)')
plt.xlabel('volatility')
plt.ylabel('Error (%)')
plt.legend()
plt.savefig('volatility-european-price-simulation-error.png')
plt.close()

# Changing number of steps (convergence)
Ns = np.arange(1,100)
def computePriceByN(N):
  return computeCallOptionPrice(S, sigma, T, N, r, K)
results = list(map(computePriceByN, Ns))
theoretical = list(map(lambda _ : computeAnalytically(sigma), Ns))
plt.plot(Ns, np.array(results) - np.array(theoretical), label='Binomial Tree
    ↪ error')
plt.xlabel('number of simulation steps')
plt.ylabel('error')
plt.legend()
plt.savefig('steps-european-price-error.png')
plt.close()

Ns = np.logspace(0,3, 75, dtype='int')
def computePriceByN(N):
  return computeCallOptionPrice(S, sigma, T, N, r, K)
```

```
results = list(map(computePriceByN, Ns))
theoretical = list(map(lambda _ : computeAnalytically(sigma), Ns))
plt.plot(Ns, np.abs(np.array(results) - np.array(theoretical)), label='
    ↪ Binomial Tree error')
plt.xscale('log')
plt.yscale('log')
plt.xlabel('number of simulation steps')
plt.ylabel('absolute error')
plt.legend()
plt.savefig('steps-european-price-error-log.png')
plt.close()

def computeAmerican(vol):
  return computeCallOptionPrice(S, vol, T, N, r, K, True)
vols = np.linspace(0.01, 0.99, 99)
resultsCall = list(map(computeAmerican, vols))

plt.plot(vols, resultsCall, label='American call option')

def computeAmericanPut(vol):
  return computeCallOptionPrice(S, vol, T, N, r, K, True, True)
vols = np.linspace(0.01, 0.99, 99)
results = list(map(computeAmericanPut, vols))

plt.plot(vols, results, label='American put option')
plt.plot(vols, np.abs(np.array(results) - np.array(resultsCall)), label='
    ↪ Difference')
plt.xlabel('volatility')
plt.ylabel('American option price')
plt.legend()
plt.ticklabel_format(useOffset=False)
plt.savefig('volatility-american-price-put.png')

print("American call option price:", computeAmerican(sigma))
print("American put option price:", computeAmericanPut(sigma))

# %%
#Delta computation using the tree model
def Delta_Tree(S, sigma, T, N, r, K):
      dt = T/N
      u = np.exp(sigma*sqrt(dt))
      d = 1/u

      tree = buildTree(S, sigma, T, N)
      matrix = valueOptionMatrix(tree, T, r, K, sigma, N)
      delta = (matrix[1,1] - matrix[1,0])/(S*u-S*d)
      return delta

#define 98 volatilities in the range of [0.02, 0.99)
vols = np.linspace(0.02, 0.99, 98)

def comput_delta_volatility(vol):
  return Delta_Tree(S, vol, T, N, r, K)
```

```
#list of tree delta computations
deltas_tree = list(map(comput_delta_volatility, vols))

#Delta computation using the theoretical model
def theoretical_delta(S_0, K, r, vol):
    deltas = np.zeros(len(vol))

    for n in np.arange(len(vol)):
        deltas[n] = norm.cdf(( np.log(S_0/K) + (r+(vols[n])**2/2) ) / vols[n])
    return deltas

#list of Black-Scholes delta computations
deltas_theory = theoretical_delta(100, 99, 0.06, vols)

#relative error
delta_error = 100*np.abs(deltas_theory - deltas_tree)/ deltas_theory

#Single value delta computation for sigma =0.2
vols_1 = [0.2]
theory_1 = theoretical_delta(100, 99, 0.06, vols_1)
""" print(theory_1) """

tree_1 = Delta_Tree(100, 0.2, 1, 50, 0.06, 99)
""" print(tree_1) """

plt.plot(vols, deltas_theory, label = "Theoretical Model Delta")
plt.plot(vols, deltas_tree, label = "Tree Model Delta")
plt.xlabel("Volatiltiy")
plt.ylabel("Delta")
plt.legend()
plt.show()
plt.savefig('volatility_comparison.png')

plt.plot(vols, delta_error, label = "Relative Error")
plt.xlabel("Volatility")
plt.ylabel("Error (%)")
plt.legend()
plt.show()
plt.savefig('volatility_absolute_error.png')
# %%

#Black-Scholes Option Price
def BlackScholes(S, K, r,T, vol):
  d1 = (log(S / K) + (r + vol * vol / 2) * T)/vol / sqrt(T)
  d2 = d1 - vol * sqrt(T)
  return S * Normal(d1) - exp(-r*T)*K*Normal(d2)
#%%
np.random.seed(123)

#The Hedge Simulation function, returns the delta and the portfolio values at
    ↪ time N

def Hedge_Sim(S_0, N, M, r, K, sigma, V):
    dt = 1/N #time step
```

```
    S = np.zeros(N) #initiate empty stock price array
    total_value = np.zeros(N) #empty portfolio array
    option_price = np.zeros(N)
    cash = np.zeros(N) #empty cash array
    S[0] = S_0 #initiate the first stock price
    d_1 = np.zeros((N-1)//M+1) #initiate the d_1 array
    deltas = np.zeros((N-1)//M +1) #initiate the delta array
    deltas[0] = norm.cdf(( np.log( S[0] / K ) + (r + 0.5*sigma**2 ) * (1-1/N)
        ↪ ) / (sigma * np.sqrt( 1 - 1/N))) #initiate the delta
    total_value[0] = BlackScholes(S_0, K, r, 1, sigma) #initial portfolio
        ↪ value
    cash[0] = total_value[0] - deltas[0] * S[0] #initial case
    Zs = np.random.normal(size=N) #random walk
    option_price[0] = total_value[0]

    for i in np.arange(1,N):

        z = Zs[i]
        S[i]=S[i-1]+r*S[i-1]*dt+sigma*S[i-1]*np.sqrt(dt)*z #stock price
            ↪ according to the Euler discretization

        cash[i] = cash[i-1] * exp(r*dt) # get interest on the cash
        total_value[i] = cash[i] + deltas[(i-1) // M] * S[i] # total value is
            ↪ value of cash + value of stock
        option_price[i] = BlackScholes(S[i], K, r, (N-i)/N, sigma)
        if i%M == 0:
          d_1[i// M] = ( np.log( S[i] / K ) + (r + 0.5*V**2 ) * (1-i/(N)) ) /
              ↪ (V * np.sqrt( 1 - i/(N)))
          deltas[i//M] = norm.cdf(d_1[i//M])

          # change composition of the portfolio
          cash[i] = total_value[i] - deltas[i//M] * S[i]

    return deltas, total_value[-1] - max(0, S[-1] - K), total_value,
        ↪ option_price
#%%
np.random.seed(12346)
total_value_daily = Hedge_Sim(100, 365, 1, 0.06, 99, 0.2, 0.2)
np.random.seed(12346)
total_value_weekly = Hedge_Sim(100, 365, 7, 0.06, 99, 0.2, 0.2)
plt.plot(np.arange(0,365), total_value_daily[3], label="Option price")
plt.plot(np.arange(0, 365), total_value_daily[2], label = "Daily hedging")
plt.plot(np.arange(0, 365), total_value_weekly[2], label = "Weekly hedging")
plt.legend()
plt.xlabel("Time (days)")
plt.ylabel("Value (  )")
plt.savefig('total_value.png')
plt.show()
np.random.seed(12346)
total_value_daily = Hedge_Sim(100, 365, 1, 0.06, 99, 0.2, 0.2)
np.random.seed(12346)
total_value_weekly = Hedge_Sim(100, 365, 7, 0.06, 99, 0.2, 0.2)
plt.plot(np.arange(0, 365), total_value_daily[2]-total_value_daily[3], label =
    ↪ "Daily hedging")
```

```
plt.plot(np.arange(0, 365), total_value_weekly[2]-total_value_daily[3], label
    ↪ = "Weekly hedging")
plt.legend()
plt.xlabel("Time (days)")
plt.ylabel("Hedge error")
plt.savefig('total_value_error.png')
plt.show()

#%%
#Run the function for sims
def MultiHedge(S_0, N, M, r, K, sigma, V, sims):
  results = np.zeros(sims)
  for i in range(sims):
    results[i] = Hedge_Sim(S_0, N, M, r, K, sigma, V)[1]
  return results
#%%
#%%
np.random.seed(1234)
results = MultiHedge(100, 365, 1, 0.06, 99, 0.2, 0.2, 1000)
#%%
np.random.seed(1234)
resultsWeekly = MultiHedge(100, 365, 7, 0.06, 99, 0.2, 0.2, 1000)
#%%
print('20-20')
plt.hist(results / BlackScholes(S, K, r, 1, sigma) * 100, density=True, bins
    ↪ =30)
plt.xlim(-50, 50)
plt.xlabel('Relative hedge error (%)')
plt.ylabel('Empirical probability density')
plt.savefig('hedge-20-20-daily.png')
plt.close()
plt.hist(resultsWeekly / BlackScholes(S, K, r, 1, sigma)* 100, density=True,
    ↪ bins=30)
plt.xlim(-50, 50)
plt.xlabel('Relative hedge error (%)')
plt.ylabel('Empirical probability density')
plt.savefig('hedge-20-20-weekly.png')
plt.close()
print('Mean daily', np.mean(results))
print('Mean weekly', np.mean(resultsWeekly))
print('MSE daily', np.mean(results**2))
print('MSE weekly', np.mean(resultsWeekly**2))
#%%
np.random.seed(1234)
results = MultiHedge(100, 365, 1, 0.06, 99, 0.2, 0.4, 1000)
#%%
np.random.seed(1234)
resultsWeekly = MultiHedge(100, 365, 7, 0.06, 99, 0.2, 0.4, 1000)
#%%
print('20-40')
plt.hist(results / BlackScholes(S, K, r, 1, sigma)* 100, density=True)
plt.xlim(-50, 50)
plt.xlabel('Relative hedge error (%)')
plt.ylabel('Empirical probability density')
```

```
plt.savefig('hedge-20-40-daily.png')
plt.close()
plt.hist(resultsWeekly / BlackScholes(S, K, r, 1, sigma)* 100, density=True)
plt.xlim(-50, 50)
plt.xlabel('Relative hedge error (%)')
plt.ylabel('Empirical probability density')
plt.savefig('hedge-20-40-weekly.png')
plt.close()
print('Mean daily', np.mean(results))
print('Mean weekly', np.mean(resultsWeekly))
print('MSE daily', np.mean(results**2))
print('MSE weekly', np.mean(resultsWeekly**2))
#%%
#Compute the mean hedging errors
def means(S_0, N, M, r, K, sigma, sims, num_vols):
    contract_vols = np.linspace(0.01, 0.4, num_vols) #
    mean = np.zeros(num_vols)
    for i in np.arange(num_vols):
        mean[i] = np.mean(MultiHedge(S_0, N, M, r, K, sigma, contract_vols[i],
            ↪ sims))
    return (100 / 11.544280227051097)*mean #compute the relative error as a
        ↪ percentage of the fair price

c_vols = np.linspace(0.01, 0.4, 400)
Mean_returns_daily = means(100, 365, 1, 0.06, 99, 0.2, 100, 400)
Mean_returns_weekly = means(100, 365, 7, 0.06, 99, 0.2, 100, 400)

#%%
#Plot the daily vs weekly mean error as a function of the contract volatility
plt.plot(c_vols, Mean_returns_daily, label = "Mean Daily Error")
plt.plot(c_vols, Mean_returns_weekly, label = "Mean Weakly Error")
plt.title("Mean Error at Constant Stock Volatility of 20%")
plt.xlabel("Delta Volatility")
plt.ylabel("Mean Error (%)")
plt.legend()
plt.savefig("DailyvsWeeklyTry.png")
plt.show()
```