Home Assignment 1 - Quantitative Finance

Aleksandar Gradev (12321078)

2025-11-08

Task definition

In a frictionless market a stock-market index is traded at a price of EUR 4600. In addition, there is a risk-free money market account with an annualized rate of 1.9% (with continuous compounding).The volatility of the stock price is 10%. In addition, there are six options traded in the market, that derive their prices from the price of the underlying. Each has a time to expiry of two years and the following payoff structures:

- Option 1: European call with strike price 4800
- Option 2: American put with strike price 4000
- Option 3: Bermudan put pith strike price 3800

I will start by initializing the known parameters from the task definition:

```
# Init parameters
S0 <- 4600
sigma <- 0.1
T <- 2
N <- 40
r <- 0.019
```

Underlying's price

Next, we have to find the underlying's price at each time step and state. For this particular task a suitable way to store the data will be to use a list of vectors. The list of vectors has many properties that will help us. We can iterate over the items of the list - they will represent the time steps. We can also iterate over the elements of each vector - they will represent the state at a particular time step.

Let's first calculate the multiplicative constant for an up-movement with the formula

$$k = e^{\sigma \cdot \sqrt{\frac{T}{N}}}$$

```
k <- exp(sigma*sqrt(T/N))

cat("Multiplicative constant k =", k)
```

```
## Multiplicative constant k = 1.022613
```

The formula for the price in the up-state and down-state at time step 1, given the price S_0 at time 0, is:

$$S^u = S_0 \cdot k^1 \text{ and } S^d = S_0 \cdot k^{-1}$$

Then we can write the possible prices of all 3 states at time step 2 as:

$$S^{uu} = S^0 \cdot k^2 \text{ and } S^{ud} = S^{du} = S_0 \cdot k^0 \text{ and } S^{dd} = S_0 \cdot k^{-2}$$

Let's observe what is happening with the powers of k in the first 5 time steps

```
## [[1]]
## [1] 0
##
## [[2]]
## [1]  1 -1
##
## [[3]]
## [1]  2  0 -2
##
## [[4]]
## [1]  3  1 -1 -3
##
## [[5]]
## [1]  4  2  0 -2 -4
##
## [[6]]
## [1]  5  3  1 -1 -3 -5
```

Keep in mind that since R uses one-based indexing, time step i is represented as element *i+1* in the list.

We can easily spot a couple of things about the "development" of this pyramid. Firstly, the first (most left) element of each level of the pyramid (time step) represents the "only-up" state at that time step and its value is equal to the index of the time step *i*. More over, at time step i we have *i+1* possible states. The difference between each state from left to right, when we express the "only-up" state as first element in the vector and the "only-down" state as last element in the vector, is -2. Then this provides us with enough information to come up with a general formula.

Let i represent a time step from the set of all time steps N and j represent a state from the set of possible states J_i at a particular time step i. Then we know for the length/cardinality of the set $|A| = i + 1$. For the set J_i assume the first element j_1 represents the "only-up" state, i.e. the $u^{i+1} \cdot d^0$ state, the second element j_2 represents the element $u^i \cdot d^1$ and so on until the last element j_{i+1} representing the "only-one-down" state, i.e. the $u^0 \cdot d^{i+1}$ state. Then the price at any arbitrary time step i in state j is given as:

$$S = S_0 \cdot k^{i-2\cdot(j-1)}$$

Now, with this formula we can easily write a for loop that calculates the price of the underlying asset at any time step and in any state, given only the starting price S_0 and the number of time steps N.

```
# Init lists to store the results and include the value at time step 0
underlying <- list(S0)
powers <- list(0)

# Loop over each time step
for (i in 1:N){
  # Empty vectors to store values of all states for time step t
  powers_t <- c()
  prices_t <- c()

  # Loop over state at time step t
  for (j in 1:(i+1)){
    # Calculate the power of k in this state
    power_state <- (i-(j-1)*2)
    # Calculate the price in this state
    price_t_state <- S0*k^power_state
    # Append it to the time step vector
    prices_t <- c(prices_t, price_t_state)
    powers_t <- c(powers_t, power_state)
  }
  # Append the time step vector to the list with values
  underlying <- append(underlying, list(prices_t))
  powers <- append(powers, list(powers_t))
}

# Observe the prices in the first 4 steps
underlying[1:5]
```

```
## [[1]]
## [1] 4600
##
## [[2]]
## [1] 4704.018 4498.282
##
## [[3]]
## [1] 4810.388 4600.000 4398.814
##
## [[4]]
## [1] 4919.163 4704.018 4498.282 4301.545
##
## [[5]]
## [1] 5030.398 4810.388 4600.000 4398.814 4206.427
```

European Call Option

Now, let us consider an European call option with strike price 4800. An European call option allows exercising only at maturity. So now, when we have the underlying's price at maturity and we know the strike price, we can compute the option value at maturity. We can also compute the Martingale probability:

```
# Define strike price
K = 4800

# Compute Martingale probs
q <- (exp(r*T/N) - 1/k)/(k-1/k)
cat("The Martingale probability q =", q,"\n")
```

```
## The Martingale probability q = 0.515661
```

```
# Obtain call option at expiry
call_exercise_value <- sapply(underlying[[length(underlying)]],
                              function(X) round(max(X-K,0),2))

# Store it as the first element of a list
call_value <- list(call_exercise_value)

#cat("The value of the option at expiry is:\n", call_value[[1]])
```

Now we can recursively compute the option values at every state starting with step $N-1$. I will use the following formula:

$$C_n = e^{-r \cdot \frac{T}{N}} \cdot [q \cdot C_{n+1}^{up} + (1-q) \cdot C_{n+1}^{down}]$$

Let's calculate the European call value and observe the option price at $t \in [0, 4]$.

```
# Calculate option intrinsic value across time steps
for (i in 1:N){
  # Store the values at t=i+1
  values <- call_value[[i]]
  # Init vector for option values at time t=i
  current_values <- c()

  # Loop over states at time step i
  for (j in 1:(length(values)-1)){
    # Calculate the option value via the formula for time t=i, state j
    current_value <- (q*values[j] + (1-q)*values[j+1])*exp(-r*T/N)
    current_value <- round(current_value,2)
    # Store the state j value to the time i vector
    current_values <- c(current_values, current_value)
  }
  # Store price vector at time t=i to the list
  call_value <- append(call_value, list(current_values))
}

# Observe the value at steps 0 to 4
tail(call_value,5)
```

```
## [[1]]
## [1] 499.15 346.59 226.89 138.96  79.02
##
## [[2]]
## [1] 424.86 288.34 184.13 109.82
##
## [[3]]
## [1] 358.40 237.64 148.00
##
## [[4]]
## [1] 299.63 194.04
##
## [[5]]
## [1] 248.25
```

American Put Option

Now, let us consider an American put option with strike price 4000. American options allow for early exercise at each time step. Therefore, it is important to check the early exercise value at each time step and step. The martingale probability does not change.

I will start with calculating the value at the last step and then I will proceed with computing the option price until time step 0 using the following formula:

$$P_n = max[\ K - S_n;\ e^{-r\cdot\frac{T}{N}} \cdot (q \cdot P_{n+1}^{up} + (1-q) \cdot P_{n+1}^{down})\ ]$$

Let's calculate the American put value and observe the option price at $t \in [0, 4]$.

```
# Set strike price
K = 4000

# Option value at expiry
put_exercise_value <- sapply(underlying[[length(underlying)]],
                             function(X) max(K-X,0))
put_value <- list(put_exercise_value)

# Calculate option intrinsic value across time steps
for (i in 1:N){
  # Store the values of the underlying at time step i
  underlying_values <- underlying[[length(underlying) - i]]
  # Store the option values at time step i+1
  values <- put_value[[i]]
  # Init vector for option values at time t=i
  current_values <- c()

  # Loop over states at time step i
  for (j in 1:(length(values)-1)){
    # Calculate value at t=i
    current_value <- (q*values[j] + (1-q)*values[j+1])*exp(-r*T/N)
    # Check early exercise of american option
    current_value_checked <- max(K - underlying_values[j], current_value)
    current_value_checked <- round(current_value_checked, 2)
    # Append the value at state j to the vector for values at time i
    current_values <- c(current_values, current_value_checked)
```

```
  }
  # Store the values at time i
  put_value <- append(put_value, list(current_values))
}

# Observe the value at steps 0 to 4
tail(put_value,5)
```

```
## [[1]]
## [1]  5.48 12.75 26.94 52.13 93.00
##
## [[2]]
## [1]  8.99 19.60 39.10 71.86
##
## [[3]]
## [1] 14.12 29.02 54.91
##
## [[4]]
## [1] 21.32 41.52
##
## [[5]]
## [1] 31.07
```

Bermudian Put Option

A Bermudan option is a type of option, which gives the owner of this contract the right to exercise the option every six months during its lifetime of the option. In this example, I have to price a Bermudian put option with strike price of 3800.

Since the duration of the options is 2 years and we have 40 time steps, we find that a one time step is equal to exactly 0.05 years. For the case of a Bermudian option, we will compare the options intrinsic value with the exercise value every 0.5 years or 10 steps.

```
# Set strike price
K = 3800

# Option value at expiry
bermudian_put_exercise_value <- sapply(underlying[[length(underlying)]],
                                       function(X) max(K-X,0))
bermudian_put_value <- list(bermudian_put_exercise_value)
```

Now, since in my for loop we iterate over time steps 39 to 0, we find that the index of the loop $i = N - t$ or rewritten the for time step as $t = N - i$. This means that we will compare the intrinsic value to the exercise value at index $i \in \{10, 20, 30, 40\}$, which corresponds to $t \in \{30, 20, 10, 0\}$.

```
for (i in 1:N){
  # Store the values of the underlying at time step i
  underlying_values <- underlying[[length(underlying) - i]]
  # Store the option values at time step i+1
  values <- bermudian_put_value[[i]]
  # Init vector for option values at time t=i
  current_values <- c()
```

```
  # Loop over states at time step i
  for (j in 1:(length(values)-1)){
    # Calculate value at t=i
    current_value <- (q*values[j] + (1-q)*values[j+1])*exp(-r*T/N)

    # Check early exercise of the bermudian option
    if(i%%10 == 0){
      # Compare intrinsic value to early exercise value
      current_value_checked <- max(K - underlying_values[j], current_value)
      current_value_checked <- round(current_value_checked, 2)

      # Append the value at state j to the vector for values at time i
      current_values <- c(current_values, current_value_checked)
    }
    else{
      # Append the value at state j to the vector for values at time i
      current_value <- round(current_value, 2)
      current_values <- c(current_values, current_value)
    }
  }
  # Store the values at time i
  bermudian_put_value <- append(bermudian_put_value, list(current_values))
}

# Observe the value at steps 0 to 4
tail(bermudian_put_value,5)
```

```
## [[1]]
## [1]  1.76  4.58 10.74 22.88 44.61
##
## [[2]]
## [1]  3.12  7.56 16.60 33.37
##
## [[3]]
## [1]  5.27 11.93 24.70
##
## [[4]]
## [1]  8.49 18.10
##
## [[5]]
## [1] 13.13
```