Project #3: Elementary Monte Carlo. More on portfolios. Put prices.

Samuel Kalisch

2024-03-23

```
library(ggplot2)
library(tidyverse)
library(lubridate)
```

Problem #1 (10 points)

A weighted spinner has four different regions: blue, red, green, and yellow. The blue is twice as likely as red, the red is twice as likely as green, and the green is twice as likely as yellow. You are invited to play the game with the following outcomes:

- if the spinner lands on blue, you lose $5;
- if the spinner lands on red, you win $2;
- if the spinner lands on green, you win $10;
- if the spinner lands on yellow, you win $20.

Use *Monte Carlo* simulation to figure out the fair price for entering this game. **Soultion:**

```
#Calculate the probabilities and winnings (b,r,g,y)
amt = c(-5,2,10,20)
weights = c(8,4,2,1)
probs = numeric(4)
for (i in 1:4) {probs[i]= weights[i]/sum(weights)}

#Monte Carlo
n = 100000000
mu.mc=mean(sample(amt, size=n, prob=probs,replace = TRUE))%>%round(4)

#Display answer
print(paste0("The fair price for entering this game is: ",mu.mc))
## [1] "The fair price for entering this game is: 0.5334"
```

Problem #2 (10 points)

Use *Monte Carlo* simulation to estimate the number π.

We know that the area of a circle with radius 1 is pi. We will esimate the probability that a random point on a square of side 2 land on the circle. This probability time the are of the square should yield pi.

Soultion:

```
#We assume that the side of the square is 2, which implies that it's center sits at (1,1)
a = 2
center=c(a/2,a/2)

#Monte Carlo
n = 100000000
sample_x = runif(n, min = 0, max = 1)
sample_y = runif(n, min = 0, max = 1)


#Distance function
distance <- function(x1,y1,x2,y2){
  return(sqrt((x2-x1)^2 + (y2-y1)^2))
}

#Calculate answer
dist= distance(center[1],center[2],sample_x,sample_y)
prob = sum(dist < 1)/length(dist)
ans = (a^2*prob) %>%round(4)

#Display answer
print(paste0("Pi has an approximate value of: ",ans))
## [1] "Pi has an approximate value of: 3.1415"
```

Problem #3 (10 points)

Use *Monte Carlo* to estimate the expected distance of a randomly chosen point in a square to the center of that square.

Soultion:

```
#We assume that the side of the square is 1, which implies that it's center sits at (1/2,1/2)
a = 1
center=c(a/2,a/2)

#Monte Carlo
n = 100000000
sample_x = runif(n, min = 0, max = 1)
sample_y = runif(n, min = 0, max = 1)


#Distance function
distance <- function(x1,y1,x2,y2){
  return(sqrt((x2-x1)^2 + (y2-y1)^2))
```

```
}

#Calculate answer
ans = mean(distance(center[1],center[2],sample_x,sample_y))%>%round(4)

#Display answer
print(paste0("The expected distance of on a square of side 1 is: ",ans))
## [1] "The expected distance of on a square of side 1 is: 0.3826"
```

Problem #4 (25 points)

Your initial wealth is exactly \$100. You are allowed to invest in shares of a particular stock. You are also allowed to both lend and borrow at the continuously compounded risk-free interest rate of 0.05. Keeping your money uninvested is **not allowed**.

You can rebalance your portfolio every morning, once you have observed the opening stock price. This means that you can change the number of shares you own (if you decide to do so) and accordingly adapt your risk-free investment.

You proceed to create a "rule'' according to which you will be rebalancing your portfolio. Here is a possible rule you can use:

I start by investing in one share of stock. Then, every day I will flip a fair coin. If the outcome is heads, I will buy one share of stock. If the outcome is tails, I will short one share. The balance of my wealth is to be invested at the risk-free rate (if needed, I will borrow at the continuously compounded risk-free interest rate).

Over the following 10 days, you observe the following stock prices for a non-dividend-paying stock:

Day	0	1	2	3	4	5	6	7	8	9	10
Stock price	100	80	64	80	64	80	100	80	64	80	100

As the time passes you follow investment rules above to rebalance your portfolio. Complete the following table describing your portfolio **just before and just after** the rebalancing is done. Even more precisely, for the 10 days, both for "morning" and "evening" print out:

- the number of shares of stock in the portfolio,
- the balance of the risk-free investment,
- the wealth in the stock,
- the total wealth.

First take note of the continuously compounded, risk-free interest rate:

```
r=0.05
```

Then, I create a vectore containing the evolution of the stock price:

```
s= c(100, 80, 64, 80, 64, 80, 100, 80, 64, 80, 100)
```

Rule (assumed a coinflip at the start):

```r
#set up vectors
pi.v=numeric(11)
cash=numeric(11)
wealth=numeric(11)

#initialize
wealth[1]=100
pi.v[1]= sample(c(1, -1), 1)
cash[1]=wealth[1]-pi.v[1]*s[1]

print(sprintf("Day=%d, %s", 1, "morning"))
## [1] "Day=1, morning"
  print(sprintf("shares=%6.4f, cash=%6.4f, wealth in stock=%6.4f, total wealth=%6.4f",
                pi.v[1], cash[1], pi.v[1]*s[1], wealth[1]))
## [1] "shares=-1.0000, cash=200.0000, wealth in stock=-100.0000, total wealth=100.0000"
#dynamics of the rule
rebalance<-function(shares,s.prev,s.curr){
  return(sample(c(1, -1), 1))
  }

#we move forward thorugh time with this rule
for(i in 1:10){
  cash[i+1]=cash[i]*exp(r/365)
  wealth[i+1]=cash[i+1]+pi.v[i]*s[i+1]

  #before re-balancing
  print(sprintf("Day=%d, %s", i, "evening"))
  print(sprintf("shares=%6.4f, cash=%6.4f, wealth in stock=%6.4f, total wealth=%6.4f",
                pi.v[i], cash[i+1], pi.v[i]*s[i+1], wealth[i+1]))

  #now we re-balance
  pi.v[i+1]=rebalance(pi.v[i],s[i],s[i+1])
  cash[i+1]=wealth[i+1]-pi.v[i+1]*s[i+1]

  #after re-balancing
  print(sprintf("Day=%d, %s", i+1, "morning"))
  print(sprintf("shares=%6.4f, cash=%6.4f, wealth in stock=%6.4f, total wealth=%6.4f",
                pi.v[i+1], cash[i+1], pi.v[i+1]*s[i+1], wealth[i+1]))
}
## [1] "Day=1, evening"
## [1] "shares=-1.0000, cash=200.0274, wealth in stock=-80.0000, total wealth=120.0274"
## [1] "Day=2, morning"
## [1] "shares=1.0000, cash=40.0274, wealth in stock=80.0000, total wealth=120.0274"
## [1] "Day=2, evening"
## [1] "shares=1.0000, cash=40.0329, wealth in stock=64.0000, total wealth=104.0329"
## [1] "Day=3, morning"
## [1] "shares=1.0000, cash=40.0329, wealth in stock=64.0000, total wealth=104.0329"
## [1] "Day=3, evening"
## [1] "shares=1.0000, cash=40.0384, wealth in stock=80.0000, total wealth=120.0384"
## [1] "Day=4, morning"
## [1] "shares=1.0000, cash=40.0384, wealth in stock=80.0000, total wealth=120.0384"
## [1] "Day=4, evening"
## [1] "shares=1.0000, cash=40.0439, wealth in stock=64.0000, total wealth=104.0439"
```

```
## [1] "Day=5, morning"
## [1] "shares=-1.0000, cash=168.0439, wealth in stock=-64.0000, total wealth=104.0439"
## [1] "Day=5, evening"
## [1] "shares=-1.0000, cash=168.0669, wealth in stock=-80.0000, total wealth=88.0669"
## [1] "Day=6, morning"
## [1] "shares=-1.0000, cash=168.0669, wealth in stock=-80.0000, total wealth=88.0669"
## [1] "Day=6, evening"
## [1] "shares=-1.0000, cash=168.0899, wealth in stock=-100.0000, total wealth=68.0899"
## [1] "Day=7, morning"
## [1] "shares=-1.0000, cash=168.0899, wealth in stock=-100.0000, total wealth=68.0899"
## [1] "Day=7, evening"
## [1] "shares=-1.0000, cash=168.1129, wealth in stock=-80.0000, total wealth=88.1129"
## [1] "Day=8, morning"
## [1] "shares=1.0000, cash=8.1129, wealth in stock=80.0000, total wealth=88.1129"
## [1] "Day=8, evening"
## [1] "shares=1.0000, cash=8.1140, wealth in stock=64.0000, total wealth=72.1140"
## [1] "Day=9, morning"
## [1] "shares=-1.0000, cash=136.1140, wealth in stock=-64.0000, total wealth=72.1140"
## [1] "Day=9, evening"
## [1] "shares=-1.0000, cash=136.1327, wealth in stock=-80.0000, total wealth=56.1327"
## [1] "Day=10, morning"
## [1] "shares=-1.0000, cash=136.1327, wealth in stock=-80.0000, total wealth=56.1327"
## [1] "Day=10, evening"
## [1] "shares=-1.0000, cash=136.1513, wealth in stock=-100.0000, total wealth=36.1513"
## [1] "Day=11, morning"
## [1] "shares=-1.0000, cash=136.1513, wealth in stock=-100.0000, total wealth=36.1513"
```

Problem #5 (20 points)

Repeat the above problem, but with a new investment "rule'' of your own. Make sure that there is an element of *randomness* in your rule. You can use the above rule or any of the rules you encountered in previous projects for inspiration, but do not just merely make simple (trivial!) adjustments.

Stochastic Oscillator Strategy:

The Stochastic Oscillator is a momentum indicator that compares the closing price of a security to its price range over a certain period of time. It consists of two lines: %K and %D. %K represents the current price's position relative to the highest high and lowest low over a specified look-back period. %D is a simple moving average of %K.

Buy Signal: When %K crosses below a certain threshold (typically 20), it suggests that the stock is oversold, indicating a potential buying opportunity.

Sell Signal: When %K crosses above another threshold (typically 80), it suggests that the stock is overbought, indicating a potential selling opportunity.

No Action: If %K remains within the thresholds, no action is take

Investopedia: Stochastic Oscillator

Implementation:

- I will implement a random treshold for the oscillator parameters.
- I will only use %K due to data constraints
- The strategy will begin after the 4th day of trading

```r
#set up vectors
pi.v=numeric(11)
cash=numeric(11)
wealth=numeric(11)

#initialize
wealth[1]=100
pi.v[1]= 0
cash[1]=wealth[1]-pi.v[1]*s[1]

print(sprintf("Day=%d, %s", 1, "morning"))
## [1] "Day=1, morning"
  print(sprintf("shares=%6.4f, cash=%6.4f, wealth in stock=%6.4f, total wealth=%6.4f",
                pi.v[1], cash[1], pi.v[1]*s[1], wealth[1]))
## [1] "shares=0.0000, cash=100.0000, wealth in stock=0.0000, total wealth=100.0000"
# Stochastic Oscillator Strategy with Randomized Parameters
rebalance <- function(threshold_low, threshold_high, percent_k) {

  # Generate buy signal if %K crosses above threshold_high and sell signal if %K crosses below threshol
  if (percent_k < threshold_low) {
    return(1)  # Buy signal
  } else if (percent_k > threshold_high) {
    return(-1)  # Sell signal
  } else {
    return(0)  # No action
  }
}

#we move forward thorugh time with this rule
for(i in 1:10){
  cash[i+1]=cash[i]*exp(r/365)
  wealth[i+1]=cash[i+1]+pi.v[i]*s[i+1]

  #before re-balancing
  print(sprintf("Day=%d, %s", i, "evening"))
  print(sprintf("shares=%6.4f, cash=%6.4f, wealth in stock=%6.4f, total wealth=%6.4f",
                pi.v[i], cash[i+1], pi.v[i]*s[i+1], wealth[i+1]))

  #rebalance, we don't trade in the first 4 days to gather info
  if(i+1==4){print("----------------------Stochasitc Oscillator strategy begins.----------------------")
  if(i+1>=4){

    # Randomly select thresholds for the oscillator parameters
    threshold_low <- sample(20:40, 1)  # Random low threshold between 20 and 40
    threshold_high <- sample(60:80, 1)  # Random high threshold between 60 and 80
    s_history <- s[(i-2):(i+1)] # History of last 4 prices
    percent_k = (s_history[4] - min(s_history)) / (max(s_history) - min(s_history)) * 100 # Calculate %

    print(sprintf("Threshold Low=%6.4f, Threshold High=%6.4f, K = %6.4f",threshold_low,threshold_high,pe

    pi.v[i+1]=rebalance(threshold_low,threshold_high,percent_k)
    cash[i+1]=wealth[i+1]-pi.v[i+1]*s[i+1]
  }
```

```r
  #after re-balancing
  print(sprintf("Day=%d, %s", i+1, "morning"))
  print(sprintf("shares=%6.4f, cash=%6.4f, wealth in stock=%6.4f, total wealth=%6.4f",
                pi.v[i+1], cash[i+1], pi.v[i+1]*s[i+1], wealth[i+1]))

}
## [1] "Day=1, evening"
## [1] "shares=0.0000, cash=100.0137, wealth in stock=0.0000, total wealth=100.0137"
## [1] "Day=2, morning"
## [1] "shares=0.0000, cash=100.0137, wealth in stock=0.0000, total wealth=100.0137"
## [1] "Day=2, evening"
## [1] "shares=0.0000, cash=100.0274, wealth in stock=0.0000, total wealth=100.0274"
## [1] "Day=3, morning"
## [1] "shares=0.0000, cash=100.0274, wealth in stock=0.0000, total wealth=100.0274"
## [1] "Day=3, evening"
## [1] "shares=0.0000, cash=100.0411, wealth in stock=0.0000, total wealth=100.0411"
## [1] "----------------------Stochasitc Oscillator strategy begins.----------------------"
## [1] "Threshold Low=30.0000, Threshold High=64.0000, K = 44.4444"
## [1] "Day=4, morning"
## [1] "shares=0.0000, cash=100.0411, wealth in stock=0.0000, total wealth=100.0411"
## [1] "Day=4, evening"
## [1] "shares=0.0000, cash=100.0548, wealth in stock=0.0000, total wealth=100.0548"
## [1] "Threshold Low=21.0000, Threshold High=78.0000, K = 0.0000"
## [1] "Day=5, morning"
## [1] "shares=1.0000, cash=36.0548, wealth in stock=64.0000, total wealth=100.0548"
## [1] "Day=5, evening"
## [1] "shares=1.0000, cash=36.0597, wealth in stock=80.0000, total wealth=116.0597"
## [1] "Threshold Low=31.0000, Threshold High=62.0000, K = 100.0000"
## [1] "Day=6, morning"
## [1] "shares=-1.0000, cash=196.0597, wealth in stock=-80.0000, total wealth=116.0597"
## [1] "Day=6, evening"
## [1] "shares=-1.0000, cash=196.0866, wealth in stock=-100.0000, total wealth=96.0866"
## [1] "Threshold Low=33.0000, Threshold High=63.0000, K = 100.0000"
## [1] "Day=7, morning"
## [1] "shares=-1.0000, cash=196.0866, wealth in stock=-100.0000, total wealth=96.0866"
## [1] "Day=7, evening"
## [1] "shares=-1.0000, cash=196.1135, wealth in stock=-80.0000, total wealth=116.1135"
## [1] "Threshold Low=39.0000, Threshold High=64.0000, K = 44.4444"
## [1] "Day=8, morning"
## [1] "shares=0.0000, cash=116.1135, wealth in stock=0.0000, total wealth=116.1135"
## [1] "Day=8, evening"
## [1] "shares=0.0000, cash=116.1294, wealth in stock=0.0000, total wealth=116.1294"
## [1] "Threshold Low=37.0000, Threshold High=79.0000, K = 0.0000"
## [1] "Day=9, morning"
## [1] "shares=1.0000, cash=52.1294, wealth in stock=64.0000, total wealth=116.1294"
## [1] "Day=9, evening"
## [1] "shares=1.0000, cash=52.1365, wealth in stock=80.0000, total wealth=132.1365"
## [1] "Threshold Low=25.0000, Threshold High=78.0000, K = 44.4444"
## [1] "Day=10, morning"
## [1] "shares=0.0000, cash=132.1365, wealth in stock=0.0000, total wealth=132.1365"
## [1] "Day=10, evening"
## [1] "shares=0.0000, cash=132.1546, wealth in stock=0.0000, total wealth=132.1546"
```

```
## [1] "Threshold Low=20.0000, Threshold High=78.0000, K = 100.0000"
## [1] "Day=11, morning"
## [1] "shares=-1.0000, cash=232.1546, wealth in stock=-100.0000, total wealth=132.1546"
```

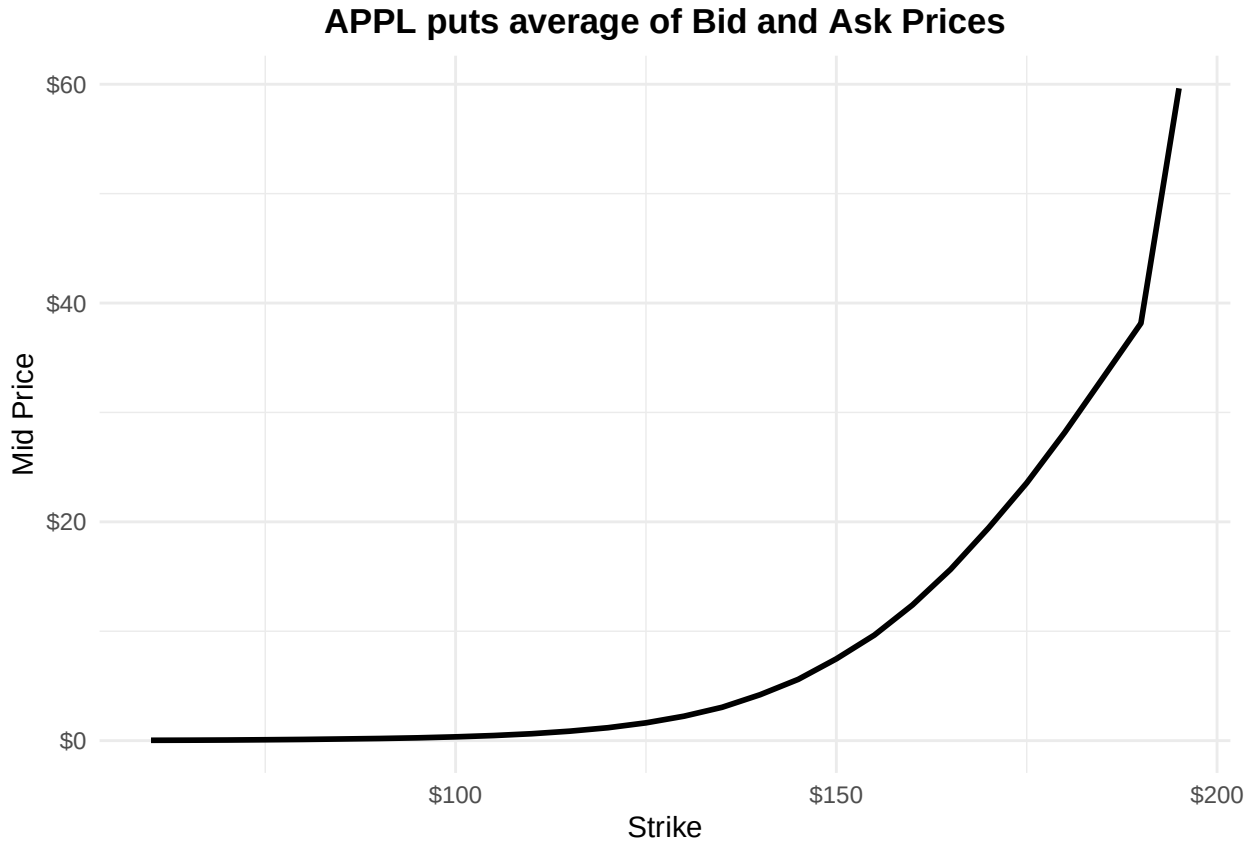
Problem #6 (25 points)

Apple puts

The file "apple-puts.csv" contains put prices observed on February 14th for European call options on the stocks of Apple Inc with expiry on May 19th and with varying strike prices.

(5 points) Import the data into a data frame. Create a line plot of the mid price (defined as the midpoint between the bid and the ask price) as it depends on the strike.

```
appl_puts<-read.csv("apple-puts.csv")

appl_puts = appl_puts%>%
  mutate(Mid = (Bid+Ask)/2)

# Plotting and Formatting
ggplot(appl_puts, aes(x = Strike)) +
  geom_line(aes(y = Mid), linewidth = 1) +
  labs(title = "APPL puts average of Bid and Ask Prices",
       x = "Strike",
       y = "Mid Price") +
  theme_minimal()+theme(plot.title = element_text(hjust = 0.5, face = "bold"))+
  scale_x_continuous(labels = scales::dollar_format(big.mark = ",")) +
  scale_y_continuous(labels = scales::dollar_format(big.mark = ","))
```



(5 points) What do you notice about the monotonicity and convexity of the above curve?

The curve is monotonically increasing and it is convex. The increasing trend of the curve suggests a consistent increase in values. Additionally, the convex shape indicates that the rate of increase is increasing over time.

(15 points) Prove that the same conclusions regarding the monotonicity and convexity of the put prices with respect to the strike must hold in general **under the no arbitrage assumption**.

Mathematically:

We assume no arbitrage for all arguments.

First, I will show that it is monotonically increasing:

Claim. Assume that one Put has a higher strike price. If K2 > K1 then V2(0)>=V1(0),

Proof. By definition, the value of a put at time T is Vp(T) = (K- S(T))+.

We have that V1(T) = (K1 - S(T))+ and V2(T) = (K2 - S(T))+. Since K2 is greater than K1, there is no scenario where (K2 - S(T))+ is less than (K1 - S(T))+, implying V2(T) is greater than or equal to V1(T).

Since the future value of the option is higher or equal at any point in time, the present value of the option is also higher, i.e., V2(0) is greater than or equal to V1(0). Therefore, the function is monotonically increasing as a function of the strike, as desired.

Now, I will show that it is convex:

Claim. A monotonically decreasing function of the price of a Call as a function of the strike is convex.

Proof. Let K1 < K2 < K3, assume that Vp(K2) > lambda*Vp(K1) + (1-lambda)*Vp(K3). We will show that there exist an arbitrage portfolio. We define lambda as the following:

$$\lambda = \frac{K_3 - K_2}{K_3 - K_1}$$

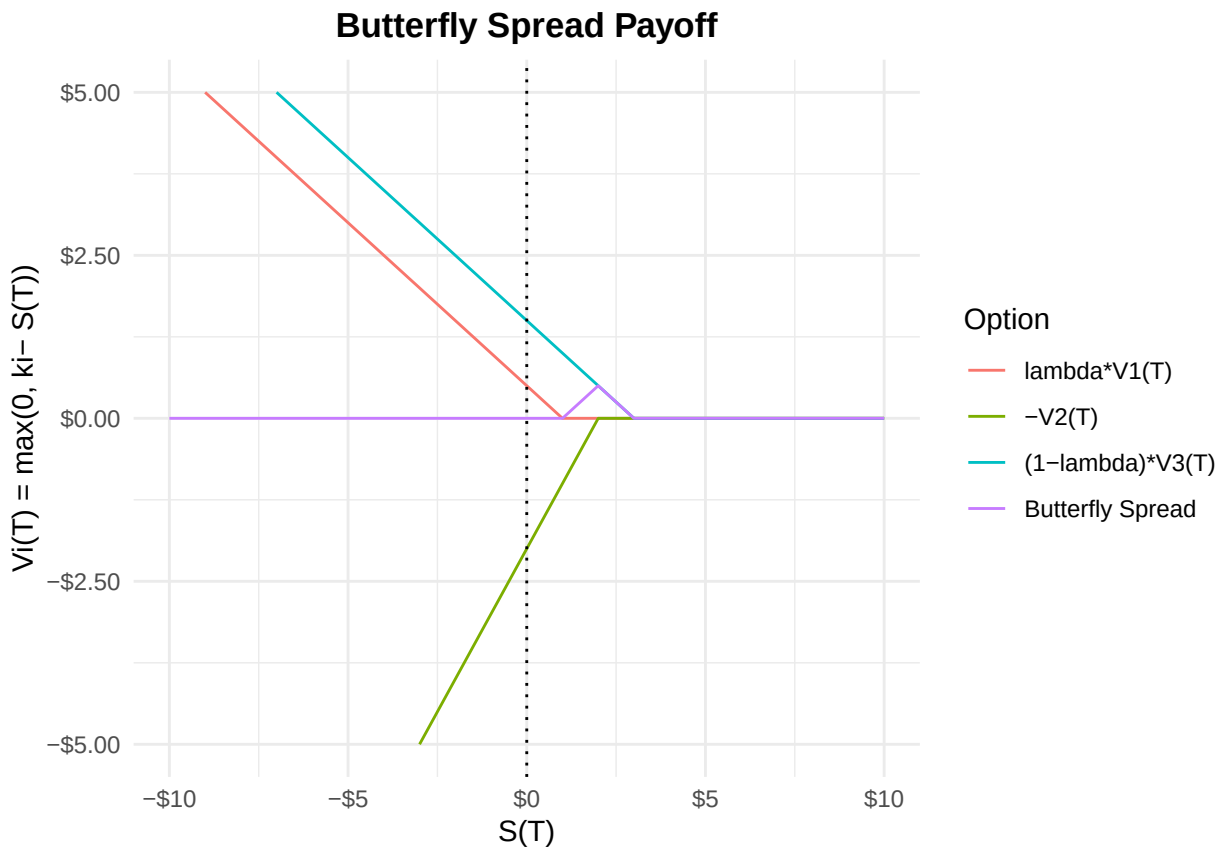
Firstly, we will show that the payoff is strictly positive. Let us construct a Butterfly spread. Visual Representation of Butterfly Spread. We define the butterfly as longing lambda and (1-lambda) options of K1 and K3 respectively and shorting one of K2. Thus the initial cost and profit are:

$$payoff = \lambda \cdot V_1(T) - V_2(T) + (1-\lambda) \cdot V_3(T)$$

$$InitialCost = \lambda \cdot V_1(0) - V_2(0) + (1-\lambda) \cdot V_3(0)$$

By our definition of lambda and the payoff function of the butterfly, it's payoff function is strictly positive.

```
# Function to calculate max(0, x - ki)
max_function <- function(x, k) {pmax(0, k - x )}

# Values for x-axis
x_values <- seq(-10, 10, by = 0.1)

# Values for k1, k2, k3
k1 <- 1
k2 <- 2
k3 <- 3

lambda = (k3 - k2) / (k3 - k1)

# Create a data frame with x values and corresponding y values for each function
data <- data.frame(
  x = rep(x_values, 4),
  y = c(lambda*max_function(x_values, k1), -max_function(x_values, k2), (1-lambda)*max_function(x_values
        lambda*max_function(x_values, k1) -max_function(x_values, k2) + (1-lambda)*max_function(x_values
  Option = rep(c("lambda*V1(T)", "-V2(T)", "(1-lambda)*V3(T)","Butterfly Spread"), each = length(x_valu
)

# Reorder the levels of Option so that Butterfly Spread comes last
data$Option <- factor(data$Option, levels = c("lambda*V1(T)", "-V2(T)", "(1-lambda)*V3(T)", "Butterfly S

# Create the ggplot
ggplot(data, aes(x = x, y = y, color = Option)) +
  geom_line() +
  labs(title = "Butterfly Spread Payoff", x = "S(T)", y = "Vi(T) = max(0, ki- S(T))") +
  geom_vline(xintercept = 0, linetype = "dotted", color = "black") +
  theme_minimal()+theme(plot.title = element_text(hjust = 0.5, face = "bold"))+
   scale_x_continuous(labels = scales::dollar_format(big.mark = ",")) +
  scale_y_continuous(labels = scales::dollar_format(big.mark = ","),limits = c(-5, 5))
## Warning: Removed 110 rows containing missing values or values outside the scale range
## (`geom_line()`).
```



Now, we will show that the cost is negative. Implying that it has a strictly positive profit. From our assumption:

$$Vp(k2) > \lambda \cdot Vp(k1) + (1 - \lambda) \cdot Vp(k3)$$

Implying that:

$$0 > \lambda \cdot Vp(k1) - Vp(k2) + (1 - \lambda) \cdot Vp(k3)$$

By our Initial cost function we have that:

$$0 > InitialCost$$

Since the payoff function is strictly positive and the Initial Cost is strictly negative by assumption, we have that the butterfly spread has a strictly positive profit. Therefore, it contradicts our no arbitrage assumption, as desired.