Appendix

- risk free rate obtained from treasury bills
- interpolate the missing price with average BID/ASK
- Problem with delta-vega hedging: true vega is far from approximation
- choose replicating option whose strike is closer to the underlying price

`{r setup, include=FALSE} knitr::opts_chunk$set(echo = TRUE) knitr::opts_chunk$set(warning = FALSE) knitr::opts_chunk$set(message = FALSE) knitr::opts_chunk$set(results='hide')`

```
# Load necessary libraries
if (!require("RQuantLib")) install.packages("RQuantLib", dependencies = TRUE)
library(RQuantLib)
library(dplyr)
library(stringr)
library(purrr)
library(fOptions)
library(ggplot2)
library(dplyr)
library(ggpubr)
```

```
process_treasury_rates <- function(data) {
  colnames(data)[colnames(data) == "X3.Mo"] <- "r"
  colnames(data)[1] <- "Date"
  data <- data[, c("Date","r"), drop = FALSE]
  data$Date <- as.Date(data$Date, format = "%m/%d/%Y")
  data$r <- data$r/100
  return(data)
}
```

```
# Function to calculate implied volatility
calculate_implied_volatility <- function(option_price, underlying_price, strike_price, time_to_maturit
  # Validate inputs
  if (is.na(option_price) || is.na(underlying_price) || is.na(strike_price) || is.na(time_to_maturity)
    return(NA)
  }

  # Calculate implied volatility
  implied_vol <- tryCatch({
    #EuropeanOptionImpliedVolatility(
    #  type = option_type,
    #  value = option_price,
    #  underlying = underlying_price,
    #  strike = strike_price,
    #  dividendYield = 0.0,  # Assuming no dividends
    #  riskFreeRate = risk_free_rate,
```

```
    #  maturity = time_to_maturity,
    #  volatility = 0.2
    #)
      GBSVolatility(
      price = option_price,
      S = underlying_price,
      X = strike_price,
      Time = time_to_maturity,
      r = risk_free_rate,
      b = 0,
      TypeFlag = "c" # "c" for call option, "p" for put option
    )
  }, error = function(e) {
    return(NA)  # Return NA if implied volatility cannot be calculated
  })

  return(implied_vol)
}

calculate_greeks <- function(greek, underlying_price, strike_price, time_to_maturity, risk_free_rate,
  if (is.na(implied_volatility) || time_to_maturity <= 0) {
    return(NA)
  }

  greeks <- GBSGreeks(
    Selection = greek,
    TypeFlag = option_type,
    S = underlying_price,
    X = strike_price,
    Time = time_to_maturity,
    r = risk_free_rate,
    b = 0,
    sigma = implied_volatility)
  return(greeks)
}

# Function to calculate delta
calculate_delta <- function(underlying_price, strike_price, time_to_maturity, risk_free_rate, implied_
  delta <- calculate_greeks("delta", underlying_price, strike_price, time_to_maturity, risk_free_rate,
  return(delta)
}

# Function to calculate vega
calculate_vega <- function(underlying_price, strike_price, time_to_maturity, risk_free_rate, implied_v
  vega <- calculate_greeks("vega", underlying_price, strike_price, time_to_maturity, risk_free_rate, i
  return(vega)
}

process_options_data <- function(data, strike_price, maturity_date, treasury_rates, historical_prices,
  treasury_rates <- process_treasury_rates(treasury_rates)
  # Convert the option maturity date to Date format
  maturity_date <- as.Date(maturity_date, format = "%Y-%m-%d")
```

```
# Add a column for time to maturity (in years)
colnames(data)[1] <- "Date"
data$Date <- as.Date(data$Date, format = "%Y-%m-%d")  # Assuming the column is named Unnamed..0
data$TimeToMaturity <- as.numeric(difftime(maturity_date, data$Date, units = "days")) / 365

historical_prices$Date <- as.Date(historical_prices$Date, format = "%Y-%m-%d")

data <- merge(data, treasury_rates, by="Date", all.x=TRUE)
data <- merge(data, historical_prices, by="Date", all.x=TRUE)

if (option_type == "call") {
  option_type <- "c"
  strike_column_name <- sprintf("C%i", strike_price)
} else {
  option_type <- "p"
  strike_column_name <- sprintf("P%i", strike_price)
}

# Initialize empty vectors for the results
implied_volatility <- numeric(nrow(data))
delta <- numeric(nrow(data))
vega <- numeric(nrow(data))


# Loop through each row of the data
for (i in 1:nrow(data)) {
  # Extract the relevant row data

  underlying_price <- data$Underlying[i]
  time_to_maturity <- data$TimeToMaturity[i]
  risk_free_rate <- data$r[i]
  option_price <- data[i, strike_column_name]


  # Calculate implied volatility
  implied_volatility[i] <- calculate_implied_volatility(
    option_price = option_price,
    underlying_price = underlying_price,
    strike_price = strike_price,
    time_to_maturity = time_to_maturity,
    risk_free_rate = risk_free_rate,
    option_type = option_type
  )

  # If use GREEKS from Refinitiv
  if (!is.na(data$DELTA[i]) & ref_greeks) {
    delta[i] <- data$DELTA[i]
    vega[i] <- data$VEGA[i]
    next
  }

  # Calculate delta
  delta[i] <- calculate_delta(
```

```
      underlying_price = underlying_price,
      strike_price = strike_price,
      time_to_maturity = time_to_maturity,
      risk_free_rate = risk_free_rate,
      implied_volatility = implied_volatility[i],
      option_type = option_type
    )

    # Calculate vega
    vega[i] <- calculate_vega(
      underlying_price = underlying_price,
      strike_price = strike_price,
      time_to_maturity = time_to_maturity,
      risk_free_rate = risk_free_rate,
      implied_volatility = implied_volatility[i],
      option_type = option_type
    )



  }

  # Add the results back to the data frame
  data$ImpliedVolatility <- implied_volatility
  data$DELTA <- delta
  data$VEGA <- vega

  return(data)
}
```

```
get_processed_options_data <- function(ref_greeks=FALSE, option_type="call") {
  # Define the path to your options data directory
  options_data_dir <- sprintf("./data/options_price/%s/", option_type)

  # List all CSV files in the options_data_dir matching the pattern "sp500_options_price_YYYY-MM-DD.cs
  options_files <- list.files(path = options_data_dir, pattern = "^sp500_options_price_\\d{4}-\\d{2}-\

  # Check if any files are found
  if(length(options_files) == 0){
    stop("No options data files found in the specified directory.")
  }

  if (option_type=="call") {
    strike_price <- 4525
  } else {
    strike_price <- 4450
  }

  # Function to extract date from filename using regex
  extract_date_from_filename <- function(filename) {
    # Extract the date part using regex
    date_str <- str_extract(basename(filename), "\\d{4}-\\d{2}-\\d{2}")
```

```r
    # Convert to Date object
    as.Date(date_str, format = "%Y-%m-%d")
  }

  # Load Treasury Rates and Historical Prices (assuming paths are correct)
  treasury_rates <- read.csv("./data/treasury_rates/daily-treasury-rates.csv")
  historical_prices <- read.csv("./data/underlying_price/historical_data_sp500.csv")

  # Initialize a list to store processed options data
  processed_options_list <- list()

  # Loop through each options data file
  for (i in seq_along(options_files)) {
    file <- options_files[i]

    # Extract maturity date from filename
    maturity_date <- extract_date_from_filename(file)

    if (is.na(maturity_date)) {
      warning(paste("Could not extract date from filename:", file))
      next  # Skip to the next file
    }

    # Read the options data CSV
    options_data <- read.csv(file, stringsAsFactors = FALSE)

    if (!"DELTA" %in% names(options_data)) {
      options_data$DELTA <- NaN
    }
    if (!"VEGA" %in% names(options_data)) {
      options_data$VEGA <- NaN
    }

    # Process the merged options data
    processed_data <- process_options_data(
      data = options_data,
      strike_price = strike_price,
      maturity_date = maturity_date,
      treasury_rates = treasury_rates,
      historical_prices = historical_prices,
      ref_greeks = ref_greeks,
      option_type = option_type
    )

    # Optionally, add a column to indicate the maturity date or source file
    # processed_data$MaturityDate <- maturity_date
    # processed_data$SourceFile <- basename(file)

    # Append the processed data to the list
    processed_options_list[[i]] <- processed_data
  }
  return(processed_options_list)
}
```

```
delta_hedging <- function(data, frequency = 1, fee_rate=0.0) {
  # Ensure the data is sorted by Date in ascending order
  data <- data[order(data$Date), ]

  differences_hedge <- numeric()  # Initialize a vector to store squared differences_hedge
  differences_no_hedge <- numeric()  # Initialize a vector to store squared differences__no_hedge

  n <- nrow(data)           # Total number of observations
  start_index <- 1          # Start from the first observation

  current_date <- data$Date[start_index]
  target_date <- current_date + frequency  # Define the next rehedging date based on calendar days
  # Calculate the absolute difference between each trading day and the target date
  date_diffs <- abs(difftime(data$Date, target_date, units = "days"))
  # Find the index of the trading day with the minimum difference
  hedge_index <- which.min(date_diffs)
  if (hedge_index <= start_index) {
    hedge_index <- start_index + 1
  }

  delta <- data$DELTA[start_index]
  total_trans_fee <- 0
  while (start_index < n) {
    # Calculate the change in option price (OP) between start_index and hedge_index
    OP <- data$C4525[start_index + 1] - data$C4525[start_index]

    # Calculate the change in underlying price (S) between start_index and hedge_index
    delta_S <- data$Underlying[start_index + 1] - data$Underlying[start_index]

    # Calculate the realized hedging error (RE)
    RE <- delta * delta_S

    # Compute the squared hedging error
    A <- (OP - RE)

    # Update the start_index to the hedge_index for the next iteration
    start_index <- start_index + 1

    if (start_index == hedge_index) {

      trans_fee <- abs(delta - data$DELTA[hedge_index]) * data$Underlying[hedge_index] * fee_rate
      A <- abs(A) + trans_fee
      total_trans_fee <- total_trans_fee + trans_fee

      delta <- data$DELTA[hedge_index]
      current_date <- data$Date[start_index]
      target_date <- current_date + frequency  # Define the next rehedging date based on calendar days

      # Calculate the absolute difference between each trading day and the target date
      date_diffs <- abs(difftime(data$Date, target_date, units = "days"))

      # Find the index of the trading day with the minimum difference
      hedge_index <- which.min(date_diffs)
```

```r
      if (hedge_index <= start_index) {
        hedge_index <- start_index + 1
      }
    }

    differences_hedge <- c(differences_hedge, A^2)
    differences_no_hedge <- c(differences_no_hedge, OP^2)

  }

  # Calculate Mean Squared Error (MSE) across all hedging intervals
  MSE <- sum(differences_hedge, na.rm = TRUE) / max(length(differences_hedge) - 1, 1)
  MSE_no_hedge <- sum(differences_no_hedge, na.rm = TRUE) / max(length(differences_no_hedge) - 1, 1)
  efficiency <- if (MSE_no_hedge == 0) NA else (1 - MSE / MSE_no_hedge) * 100

  result <- list(
    MSE = MSE,
    MSE_no_hedge = MSE_no_hedge,
    differences_no_hedge = differences_no_hedge,
    differences_hedge = differences_hedge,
    efficiency = efficiency,
    total_fees = total_trans_fee)

  # Print the results
  # cat(sprintf("MSE_no_hedge: %.2f\n", MSE_no_hedge))
  cat(sprintf("Rehedging Frequency: Every %d calendar day(s), MSE: %.2f (Efficiency: %.1f%%) and total

  return(result)
}
```

```r
processed_call_options_list <- get_processed_options_data(ref_greeks=FALSE)
max_freq <- 10
not_test_dates <- c(as.Date("2021-05-20", format = "%Y-%m-%d"), as.Date("2021-06-17", format = "%Y-%m-
n_options <- length(processed_call_options_list) - length(not_test_dates)

Es_no_fees <- as.data.frame(matrix(NA, nrow = n_options, ncol = max_freq))
colnames(Es_no_fees) <- paste0("F", 1:max_freq)

i <- 1
for (data in processed_call_options_list) {
  if (data$Date[nrow(data)] %in% not_test_dates) {
    next
  }
  cat(sprintf("Maturity: %s\n", data$Date[nrow(data)] + 1))
  data <- data[data$TimeToMaturity <= 0.124,]
  for (frequency in seq(max_freq)) {
    Es_no_fees[i, frequency]  <- delta_hedging(data, frequency=frequency)$efficiency
  }
  cat("\n")
  i <- i + 1
}
means_no_fees <- apply(Es_no_fees, 2, mean)
sds_no_fees <- apply(Es_no_fees, 2, sd)
```

```r
print(means_no_fees)
print(sds_no_fees)
```

```r
max_freq <- 10
n_options <- length(processed_call_options_list) - length(not_test_dates)
Es_fees <- as.data.frame(matrix(NA, nrow = n_options, ncol = max_freq))
colnames(Es_fees) <- paste0("F", 1:max_freq)

i <- 1
for (data in processed_call_options_list) {
  if (data$Date[nrow(data)] %in% not_test_dates) {
    next
  }
  data <- data[data$TimeToMaturity <= 0.124,]
  for (frequency in seq(max_freq)) {
    Es_fees[i, frequency]  <- delta_hedging(data, frequency=frequency, fee_rate=0.01)$efficiency
  }
  cat("\n")
  i <- i + 1
}
means_fees <- apply(Es_fees, 2, mean)
sds_fees <- apply(Es_fees, 2, sd)
print(means_fees)
print(sds_fees)
```

```r
merge_options_data <- function(ref_greeks=FALSE) {
  processed_call_options_list <- get_processed_options_data(ref_greeks=ref_greeks)
  # Initialize an empty list to store the merged datasets
  merged_options_list <- list()

  # Loop through each dataset in processed_call_options_list
  for (i in seq_along(processed_call_options_list)) {

    # Extract the current dataset
    current_data <- processed_call_options_list[[i]]

    # Rename the column
    current_data <- current_data %>%
      rename(
        "DELTAbs" = "DELTA",
        "VEGAbs"  = "VEGA",
        "Cbs"= "C4525"
      )

    # Iterate over all other datasets to join
    for (j in seq_along(processed_call_options_list)) {
      if (i != j) {  # Avoid joining the dataset with itself

        # Extract another dataset and rename "C4525" to "Crep"
        join_data <- processed_call_options_list[[j]]
        join_data <- join_data %>%
          rename(
            "DELTArep" = "DELTA",
```

```
            "VEGArep"  = "VEGA",
            "Crep"= "C4525"
          )

        # Perform a left join by "Date"
        merged_data <- left_join(current_data, join_data[, c("Date", "Crep", "DELTArep", "VEGArep")],

        # Update the current_data with the new merged_data
        current_data <- merged_data
      }
    }

    current_data <- current_data[current_data$TimeToMaturity <= 0.124,]
    current_data <- current_data %>%
      select_if(~ !any(is.na(.)))

    current_data <- current_data[, !(names(current_data) %in% c("r", "TimeToMaturity", "ImpliedVolatil


    # Use `gsub()` to Replace Matching Column Names with the Base Name
    base_names <- c("Crep", "DELTArep", "VEGArep")
    pattern <- paste0("^(", paste(base_names, collapse = "|"), ")(\\.x|\\.y)+$")
    names(current_data) <- gsub(pattern, "\\1", names(current_data))


    # Append the fully merged dataset to the list
    merged_options_list[[i]] <- current_data
  }
  merged_options_list <- Filter(function(options_data) 'Crep' %in% names(options_data), merged_options
  return(merged_options_list)
}
```

```
delta_vega_hedging <- function(data, frequency = 1, fee_rate=0.0) {
  # Ensure the data is sorted by Date in ascending order
  data <- data[order(data$Date), ]

  differences_hedge <- numeric()  # Initialize a vector to store squared differences_hedge
  differences_no_hedge <- numeric()  # Initialize a vector to store squared differences__no_hedge

  n <- nrow(data)           # Total number of observations
  start_index <- 1          # Start from the first observation

  current_date <- data$Date[start_index]
  target_date <- current_date + frequency  # Define the next rehedging date based on calendar days
  # Calculate the absolute difference between each trading day and the target date
  date_diffs <- abs(difftime(data$Date, target_date, units = "days"))
  # Find the index of the trading day with the minimum difference
  hedge_index <- which.min(date_diffs)
  if (hedge_index <= start_index) {
    hedge_index <- start_index + 1
  }

  eta <- data$VEGAbs[start_index] / data$VEGArep[start_index]
```

```
alpha <- data$DELTAbs[start_index] - eta * data$DELTArep[start_index]

total_trans_fee <- 0
while (start_index < n) {
  # Calculate the change in option price (OP) between start_index and hedge_index
  OP <- data$Cbs[start_index + 1] - data$Cbs[start_index]

  # Calculate the change in underlying price (S) between start_index and hedge_index
  delta_S <- data$Underlying[start_index + 1] - data$Underlying[start_index]
  delta_Crep <- data$Crep[start_index + 1] - data$Crep[start_index]

  # Calculate the realized hedging error (RE)
  RE <- alpha * delta_S + eta * delta_Crep

  # Compute the squared hedging error
  A <- (OP - RE)

  # Update the start_index to the hedge_index for the next iteration
  start_index <- start_index + 1

  if (start_index == hedge_index) {
    new_eta <- data$VEGAbs[hedge_index] / data$VEGArep[hedge_index]
    new_alpha <- data$DELTAbs[hedge_index] - new_eta * data$DELTArep[hedge_index]

    trans_fee <- (abs(alpha - new_alpha) * data$Underlying[hedge_index] +
                  abs(eta - new_eta) * data$Crep[hedge_index] ) * fee_rate
    A <- abs(A) + trans_fee
    total_trans_fee <- total_trans_fee + trans_fee

    eta <- new_eta
    alpha <- new_alpha

    current_date <- data$Date[start_index]
    target_date <- current_date + frequency  # Define the next rehedging date based on calendar days

    # Calculate the absolute difference between each trading day and the target date
    date_diffs <- abs(difftime(data$Date, target_date, units = "days"))

    # Find the index of the trading day with the minimum difference
    hedge_index <- which.min(date_diffs)
    if (hedge_index <= start_index) {
      hedge_index <- start_index + 1
    }
  }

  differences_hedge <- c(differences_hedge, A^2)
  differences_no_hedge <- c(differences_no_hedge, OP^2)

}

# Calculate Mean Squared Error (MSE) across all hedging intervals
MSE <- sum(differences_hedge, na.rm = TRUE) / max(length(differences_hedge) - 1, 1)
MSE_no_hedge <- sum(differences_no_hedge, na.rm = TRUE) / max(length(differences_no_hedge) - 1, 1)
```

```
  efficiency <- if (MSE_no_hedge == 0) NA else (1 - MSE / MSE_no_hedge) * 100

  result <- list(
    MSE = MSE,
    MSE_no_hedge = MSE_no_hedge,
    differences_no_hedge = differences_no_hedge,
    differences_hedge = differences_hedge,
    efficiency = efficiency,
    total_fees = total_trans_fee)

  # Print the results
  # cat(sprintf("MSE_no_hedge: %.2f\n", MSE_no_hedge))
  cat(sprintf("Rehedging Frequency: Every %d calendar day(s), MSE: %.2f (Efficiency: %.1f%%) and total

  return(result)
}
```

```
merged_options_list <- merge_options_data(ref_greeks=FALSE)
```

```
max_freq <- 10
not_test_dates <- c(as.Date("2021-05-20", format = "%Y-%m-%d"), as.Date("2021-06-17", format = "%Y-%m-

n_options <- length(merged_options_list) - length(not_test_dates)

Es_dv_no_fees <- as.data.frame(matrix(NA, nrow = n_options, ncol = max_freq))
colnames(Es_dv_no_fees) <- paste0("F", 1:max_freq)

i <- 1
for (data in merged_options_list) {
  if (data$Date[nrow(data)] %in% not_test_dates) {
    next
  }
  cat(sprintf("Maturity: %s\n", data$Date[nrow(data)] + 1))
  for (frequency in seq(max_freq)) {
    Es_dv_no_fees[i, frequency]  <- delta_vega_hedging(data, frequency=frequency)$efficiency
  }
  cat("\n")
  i <- i + 1
}
means_dv_no_fees <- apply(Es_dv_no_fees, 2, mean)
sds_dv_no_fees <- apply(Es_dv_no_fees, 2, sd)
print(means_dv_no_fees)
print(sds_dv_no_fees)
```

```
max_freq <- 10
n_options <- length(merged_options_list) - length(not_test_dates)

Es_dv_fees <- as.data.frame(matrix(NA, nrow = n_options, ncol = max_freq))
colnames(Es_dv_fees) <- paste0("F", 1:max_freq)

i <- 1
for (data in merged_options_list) {
  if (data$Date[nrow(data)] %in% not_test_dates) {
```

```
    next
  }
  cat(sprintf("Maturity: %s\n", data$Date[nrow(data)] + 1))
  for (frequency in seq(max_freq)) {
    Es_dv_fees[i, frequency]  <- delta_vega_hedging(data, frequency=frequency, fee_rate=0.01)$efficien
  }
  cat("\n")
  i <- i + 1
}
means_dv_fees <- apply(Es_dv_fees, 2, mean)
sds_dv_fees <- apply(Es_dv_fees, 2, sd)
print(means_dv_fees)
print(sds_dv_fees)
```

```
merge_put_call <- function(ref_greeks=FALSE) {
  processed_call_options_list <- get_processed_options_data(ref_greeks=ref_greeks, option_type="call")
  processed_put_options_list <- get_processed_options_data(ref_greeks=ref_greeks, option_type="put")

  # Initialize an empty list to store the merged datasets
  merged_options_list <- list()

  # Loop through each dataset in processed_call_options_list
  for (call_data in processed_call_options_list) {
    # Rename the column
    call_data <- call_data %>%
      rename(
        "DELTA_c" = "DELTA",
        "VEGA_c"  = "VEGA",
        "C"= "C4525"
      )

    # Iterate over all datasets in processed_put_options_list to join
    for (put_data in processed_put_options_list) {
      # Rename put option columns
      put_data <- put_data %>%
        rename(
          "DELTA_p" = "DELTA",
          "VEGA_p"  = "VEGA",
          "P"= "P4450"
        )

      # Perform a left join by "Date"
      merged_data <- left_join(call_data, put_data[, c("Date", "P", "DELTA_p", "VEGA_p")], by = "Date"

      # Update the call_data with the new merged_data
      call_data <- merged_data
    }

    # Filter rows based on TimeToMaturity
    call_data <- call_data %>%
      select_if(~ !any(is.na(.)))

    # Drop any remaining unnecessary columns
```

```
    call_data <- call_data[, !(names(call_data) %in% c("r", "ImpliedVolatility"))]

    # Remove columns with names matching the pattern P.x, DELTA_p.x, VEGA_p.x
    base_names <- c("P", "DELTA_p", "VEGA_p")
    pattern <- paste0("^(", paste(base_names, collapse = "|"), ")(\\.x|\\.y)+$")
    names(call_data) <- gsub(pattern, "\\1", names(call_data))

    unique_cols <- !duplicated(names(call_data))
    call_data <- call_data[, unique_cols]

    # Append the cleaned dataset to the list
    merged_options_list <- append(merged_options_list, list(call_data))

  }
  merged_options_list <- Filter(function(options_data) 'P' %in% names(options_data), merged_options_li
  return(merged_options_list)
}
```

```
delta_hedging_portfolio <- function(data, frequency = 1, fee_rate=0.0) {
  # Ensure the data is sorted by Date in ascending order
  data <- data[order(data$Date), ]

  differences_hedge <- numeric()  # Initialize a vector to store squared differences_hedge
  differences_no_hedge <- numeric()  # Initialize a vector to store squared differences__no_hedge

  n <- nrow(data)           # Total number of observations
  start_index <- 1          # Start from the first observation

  current_date <- data$Date[start_index]
  target_date <- current_date + frequency  # Define the next rehedging date based on calendar days
  # Calculate the absolute difference between each trading day and the target date
  date_diffs <- abs(difftime(data$Date, target_date, units = "days"))
  # Find the index of the trading day with the minimum difference
  hedge_index <- which.min(date_diffs)
  if (hedge_index <= start_index) {
    hedge_index <- start_index + 1
  }

  delta <- data$DELTA_c[start_index] + data$DELTA_p[start_index]
  total_trans_fee <- 0
  while (start_index < n) {
    # Calculate the change in option price (OP) between start_index and hedge_index
    OP <- data$C[start_index + 1] - data$C[start_index] +
          data$P[start_index + 1] - data$P[start_index]

    # Calculate the change in underlying price (S) between start_index and hedge_index
    delta_S <- data$Underlying[start_index + 1] - data$Underlying[start_index]

    # Calculate the realized hedging error (RE)
    RE <- delta * delta_S

    # Compute the squared hedging error
    A <- (OP - RE)
```

```
    # Update the start_index to the hedge_index for the next iteration
    start_index <- start_index + 1

    if (start_index == hedge_index) {
      new_delta <- data$DELTA_c[hedge_index] + data$DELTA_p[hedge_index]
      trans_fee <- abs(delta - new_delta) * data$Underlying[hedge_index] * fee_rate
      A <- abs(A) + trans_fee
      total_trans_fee <- total_trans_fee + trans_fee

      delta <- new_delta
      current_date <- data$Date[start_index]
      target_date <- current_date + frequency  # Define the next rehedging date based on calendar days

      # Calculate the absolute difference between each trading day and the target date
      date_diffs <- abs(difftime(data$Date, target_date, units = "days"))

      # Find the index of the trading day with the minimum difference
      hedge_index <- which.min(date_diffs)
      if (hedge_index <= start_index) {
        hedge_index <- start_index + 1
      }
    }

    differences_hedge <- c(differences_hedge, A^2)
    differences_no_hedge <- c(differences_no_hedge, OP^2)

  }

  # Calculate Mean Squared Error (MSE) across all hedging intervals
  MSE <- sum(differences_hedge, na.rm = TRUE) / max(length(differences_hedge) - 1, 1)
  MSE_no_hedge <- sum(differences_no_hedge, na.rm = TRUE) / max(length(differences_no_hedge) - 1, 1)
  efficiency <- if (MSE_no_hedge == 0) NA else (1 - MSE / MSE_no_hedge) * 100

  result <- list(
    MSE = MSE,
    MSE_no_hedge = MSE_no_hedge,
    differences_no_hedge = differences_no_hedge,
    differences_hedge = differences_hedge,
    efficiency = efficiency,
    total_fees = total_trans_fee)

  # Print the results
  # cat(sprintf("MSE_no_hedge: %.2f\n", MSE_no_hedge))
  cat(sprintf("Rehedging Frequency: Every %d calendar day(s), MSE: %.2f (Efficiency: %.1f%%) and total

  return(result)
}
```

```
merged_call_put_list <- merge_put_call(ref_greeks = FALSE)
```

```
max_freq <- 10
n_options <- length(merged_call_put_list)
```

```r
Es_v_port_no_fees <- as.data.frame(matrix(NA, nrow = n_options, ncol = max_freq))
colnames(Es_v_port_no_fees) <- paste0("F", 1:max_freq)

i <- 1
for (data in merged_call_put_list) {
  data <- data[data$TimeToMaturity <= 0.124,]
  cat(sprintf("Maturity: %s\n", data$Date[nrow(data)] + 1))
  for (frequency in seq(max_freq)) {
    Es_v_port_no_fees[i, frequency]  <- delta_hedging_portfolio(data, frequency=frequency)$efficiency
  }
  cat("\n")
  i <- i + 1
}
means_v_port_no_fees <- apply(Es_v_port_no_fees, 2, mean)
sds_v_port_no_fees <- apply(Es_v_port_no_fees, 2, sd)
print(means_v_port_no_fees)
print(sds_v_port_no_fees)
```

```r
max_freq <- 10
n_options <- length(merged_call_put_list)

Es_v_port_fees <- as.data.frame(matrix(NA, nrow = n_options, ncol = max_freq))
colnames(Es_v_port_fees) <- paste0("F", 1:max_freq)

i <- 1
for (data in merged_call_put_list) {
  data <- data[data$TimeToMaturity <= 0.124,]
  cat(sprintf("Maturity: %s\n", data$Date[nrow(data)] + 1))
  for (frequency in seq(max_freq)) {
    Es_v_port_fees[i, frequency]  <- delta_hedging_portfolio(data, frequency=frequency, fee_rate = 0.0
  }
  cat("\n")
  i <- i + 1
}
means_v_port_fees <- apply(Es_v_port_fees, 2, mean)
sds_v_port_fees <- apply(Es_v_port_fees, 2, sd)
print(means_v_port_fees)
print(sds_v_port_fees)
```

```r
merge_portfolio_data <- function(ref_greeks=FALSE) {
  merged_call_put_list <- merge_put_call(ref_greeks = ref_greeks)
  # Initialize an empty list to store the merged datasets
  merged_portfolio_list <- list()

  # Loop through each dataset in merged_call_put_list
  for (i in seq_along(merged_call_put_list)) {

    # Extract the current dataset
    current_data <- merged_call_put_list[[i]]

    # Rename the column
    current_data <- current_data %>%
      rename(
```

```
        "Cbs"= "C",
        "DELTA_cbs" = "DELTA_c",
        "VEGA_cbs"  = "VEGA_c",
        "Pbs" = "P",
        "DELTA_pbs" = "DELTA_p",
        "VEGA_pbs"  = "VEGA_p"
      )

    # Iterate over all other datasets to join
    for (j in seq_along(merged_call_put_list)) {
      if (i != j) {  # Avoid joining the dataset with itself

        # Extract another dataset and rename "C4525" to "Crep"
        join_data <- merged_call_put_list[[j]]
        join_data <- join_data %>%
          rename(
            "Crep"= "C",
            "DELTA_crep" = "DELTA_c",
            "VEGA_crep"  = "VEGA_c",
            "Prep" = "P",
            "DELTA_prep" = "DELTA_p",
            "VEGA_prep"  = "VEGA_p"
          )

        # Perform a left join by "Date"
        merged_data <- left_join(current_data, join_data[, c("Date", "Crep", "DELTA_crep", "VEGA_crep"

        # Update the current_data with the new merged_data
        current_data <- merged_data
      }
    }

    current_data <- current_data[current_data$TimeToMaturity <= 0.124,]
    current_data <- current_data %>%
      select_if(~ !any(is.na(.)))

    current_data <- current_data[, !(names(current_data) %in% c("r", "TimeToMaturity", "ImpliedVolatil


    # Use `gsub()` to Replace Matching Column Names with the Base Name
    base_names <- c("Crep", "DELTA_crep", "VEGA_crep", "Prep", "DELTA_prep", "VEGA_prep")
    pattern <- paste0("^(", paste(base_names, collapse = "|"), ")(\\.x|\\.y)+$")
    names(current_data) <- gsub(pattern, "\\1", names(current_data))


    # Append the fully merged dataset to the list
    merged_portfolio_list[[i]] <- current_data
  }
  merged_portfolio_list <- Filter(function(options_data) 'Crep' %in% names(options_data), merged_portf
  return(merged_portfolio_list)
}
```

```r
merged_portfolio_list <- merge_portfolio_data(ref_greeks=FALSE)
```

```r
delta_vega_hedging_portfolio <- function(data, frequency = 1, fee_rate=0.0, call_strike=4525, put_stri
  # Ensure the data is sorted by Date in ascending order
  data <- data[order(data$Date), ]

  differences_hedge <- numeric()  # Initialize a vector to store squared differences_hedge
  differences_no_hedge <- numeric()  # Initialize a vector to store squared differences__no_hedge

  n <- nrow(data)            # Total number of observations
  start_index <- 1           # Start from the first observation

  current_date <- data$Date[start_index]
  target_date <- current_date + frequency  # Define the next rehedging date based on calendar days
  # Calculate the absolute difference between each trading day and the target date
  date_diffs <- abs(difftime(data$Date, target_date, units = "days"))
  # Find the index of the trading day with the minimum difference
  hedge_index <- which.min(date_diffs)
  if (hedge_index <= start_index) {
    hedge_index <- start_index + 1
  }

  eta <- (data$VEGA_cbs[start_index] + data$VEGA_pbs[start_index]) / (data$VEGA_crep[start_index] + da
  alpha <- (data$DELTA_cbs[start_index] + data$DELTA_pbs[start_index]) -
          eta * (data$DELTA_crep[start_index] + data$DELTA_prep[start_index])

  underlying_0 <- data$Underlying[1]
  if (abs(call_strike - underlying_0) <= abs(put_strike - underlying_0)) {
    rep_option <- data$Crep
  } else {
    rep_option <- data$Prep
  }

  total_trans_fee <- 0
  while (start_index < n) {
    # Calculate the change in option price (OP) between start_index and hedge_index
    OP <- (data$Cbs[start_index + 1] - data$Cbs[start_index]) +
      (data$Pbs[start_index + 1] - data$Pbs[start_index])

    # Calculate the change in underlying price (S) between start_index and hedge_index
    delta_S <- data$Underlying[start_index + 1] - data$Underlying[start_index]
    delta_rep_option <- rep_option[start_index + 1] - rep_option[start_index]

    # Calculate the realized hedging error (RE)
    RE <- alpha * delta_S + eta * delta_rep_option

    # Compute the squared hedging error
    A <- (OP - RE)

    # Update the start_index to the hedge_index for the next iteration
    start_index <- start_index + 1

    if (start_index == hedge_index) {
```

```
      new_eta <- (data$VEGA_cbs[hedge_index] + data$VEGA_pbs[hedge_index]) / (data$VEGA_crep[hedge_ind
      new_alpha <- (data$DELTA_cbs[hedge_index] + data$DELTA_pbs[hedge_index]) -
                   new_eta * (data$DELTA_crep[hedge_index] + data$DELTA_prep[hedge_index])

      trans_fee <- (abs(alpha - new_alpha) * data$Underlying[hedge_index] +
                   abs(eta - new_eta) * rep_option[hedge_index]) * fee_rate
      A <- abs(A) + trans_fee
      total_trans_fee <- total_trans_fee + trans_fee

      eta <- new_eta
      alpha <- new_alpha

      current_date <- data$Date[start_index]
      target_date <- current_date + frequency  # Define the next rehedging date based on calendar days

      # Calculate the absolute difference between each trading day and the target date
      date_diffs <- abs(difftime(data$Date, target_date, units = "days"))

      # Find the index of the trading day with the minimum difference
      hedge_index <- which.min(date_diffs)
      if (hedge_index <= start_index) {
        hedge_index <- start_index + 1
      }
    }

    differences_hedge <- c(differences_hedge, A^2)
    differences_no_hedge <- c(differences_no_hedge, OP^2)

  }

  # Calculate Mean Squared Error (MSE) across all hedging intervals
  MSE <- sum(differences_hedge, na.rm = TRUE) / max(length(differences_hedge) - 1, 1)
  MSE_no_hedge <- sum(differences_no_hedge, na.rm = TRUE) / max(length(differences_no_hedge) - 1, 1)
  efficiency <- if (MSE_no_hedge == 0) NA else (1 - MSE / MSE_no_hedge) * 100

  result <- list(
    MSE = MSE,
    MSE_no_hedge = MSE_no_hedge,
    differences_no_hedge = differences_no_hedge,
    differences_hedge = differences_hedge,
    efficiency = efficiency,
    total_fees = total_trans_fee)

  # Print the results
  # cat(sprintf("MSE_no_hedge: %.2f\n", MSE_no_hedge))
  cat(sprintf("Rehedging Frequency: Every %d calendar day(s), MSE: %.2f (Efficiency: %.1f%%) and total

  return(result)
}
```

```
max_freq <- 10
n_options <- length(merged_portfolio_list)
```

```
Es_dv_port_no_fees <- as.data.frame(matrix(NA, nrow = n_options, ncol = max_freq))
colnames(Es_dv_port_no_fees) <- paste0("F", 1:max_freq)

i <- 1
for (data in merged_portfolio_list) {
  cat(sprintf("Maturity: %s\n", data$Date[nrow(data)] + 1))
  for (frequency in seq(max_freq)) {
    Es_dv_port_no_fees[i, frequency]  <- delta_vega_hedging_portfolio(data, frequency=frequency, fee_r
  }
  cat("\n")
  i <- i + 1
}
means_dv_port_no_fees <- apply(Es_dv_port_no_fees, 2, mean)
sds_dv_port_no_fees <- apply(Es_dv_port_no_fees, 2, sd)
print(means_dv_port_no_fees)
print(sds_dv_port_no_fees)
```

```
max_freq <- 10
n_options <- length(merged_portfolio_list)

Es_dv_port_fees <- as.data.frame(matrix(NA, nrow = n_options, ncol = max_freq))
colnames(Es_dv_port_fees) <- paste0("F", 1:max_freq)

i <- 1
for (data in merged_portfolio_list) {
  cat(sprintf("Maturity: %s\n", data$Date[nrow(data)] + 1))
  for (frequency in seq(max_freq)) {
    Es_dv_port_fees[i, frequency]  <- delta_vega_hedging_portfolio(data, frequency=frequency, fee_rate
  }
  cat("\n")
  i <- i + 1
}
means_dv_port_fees <- apply(Es_dv_port_fees, 2, mean)
sds_dv_port_fees <- apply(Es_dv_port_fees, 2, sd)
print(means_dv_port_fees)
print(sds_dv_port_fees)
```

```
welsch_t_test <- function(delta, delta_vega) {
  mean1 <- mean(delta_vega)
  mean2 <- mean(delta)

  sd1 <- sd(delta_vega)
  sd2 <- sd(delta)

  n1 <- length(delta_vega)
  n2 <- length(delta)

  t_stat <- (mean1 - mean2) / sqrt((sd1^2 / n1) + (sd2^2 / n2))  # T-statistic
  df <- ((sd1^2 / n1 + sd2^2 / n2)^2) /
    (((sd1^2 / n1)^2 / (n1 - 1)) + ((sd2^2 / n2)^2 / (n2 - 1)))  # Degrees of freedom
  p_value <- pt(t_stat, df, lower.tail = FALSE)  # One-tailed test
  return(c(t_stat = t_stat, df = df, p_value = p_value))
}
```

```
significant_performance <- function(delta, delta_vega) {
  mean_delta <- mean(delta)
  mean_delta_vega <- mean(delta_vega)

  p_value <- t.test(delta, delta_vega, alternative = "less", var.equal = FALSE)$p.value
  if (p_value < 0.05) {
    return("Rejected")
  } else {
    return("No evidence")
  }
}

compare_delta_vs_dv <- function(delta_df, dv_df) {
  max_freq <- ncol(delta_df)

  # Perform Welch's t-test for each frequency
  results <- data.frame(
    Frequency = 1:max_freq,
    Result = ""
  )

  for (freq in seq(max_freq)) {
    delta_array <- delta_df[, freq]
    delta_vega_array <- dv_df[, freq]

    results$Result[freq] <- significant_performance(delta_array, delta_vega_array)
  }

  return(results)
}



compare_delta_vs_dv(Es_no_fees, Es_dv_no_fees)
compare_delta_vs_dv(Es_fees, Es_dv_fees)
compare_delta_vs_dv(Es_v_port_no_fees, Es_dv_port_no_fees)
```

```
# Function to identify the best frequency and perform Welch's t-tests
plot_efficiency_comparison <- function(data_matrix) {
  # Number of frequencies
  max_freq <- ncol(data_matrix)

  # Compute mean and standard deviation for each frequency
  means <- colMeans(data_matrix, na.rm = TRUE)
  sds <- apply(data_matrix, 2, sd, na.rm = TRUE)

  # Identify the best frequency by highest mean efficiency
  best_freq <- which.max(means)

  # Initialize list to store Welch's t-test results
  t_test_results <- list()
```

```r
# Perform Welch's t-tests for all other frequencies against the best frequency
best_vec <- data_matrix[, best_freq]
for (freq in 1:max_freq) {
  if (freq == best_freq) {
    t_test_results[[freq]] <- NA
    next
  }
  freq_vec <- data_matrix[, freq]
  t_res <- t.test(freq_vec, best_vec,
                  alternative = "less",  # H1: mu(freq) < mu(best)
                  var.equal = FALSE,     # Welch's t-test
                  )
  t_test_results[[freq]] <- t_res
}

# Determine significance for each frequency
significance_vec <- rep("Not tested", max_freq)
for (freq in 1:max_freq) {
  if (freq == best_freq) {
    significance_vec[freq] <- "Best frequency"
  } else {
    pval <- t_test_results[[freq]]$p.value
    if (!is.na(pval) && pval < 0.05) {
      significance_vec[freq] <- "Significantly worse"
    } else {
      significance_vec[freq] <- "Not significantly worse"
    }
  }
}

# Create a data frame for plotting
df_plot <- data.frame(
  Frequency = 1:max_freq,
  MeanEff   = means,
  SDEff     = sds,
  Significance = significance_vec,
  IsBest    = 1:max_freq == best_freq
)

# Plot the results
plot <- ggplot(df_plot, aes(x = SDEff, y = MeanEff, color = Significance)) +
  geom_point(aes(color = Significance), size = 3) +
  geom_text(
    aes(label = sprintf("1/%i", Frequency)),
    vjust = -1,  # adjust vertical position
    size = 3,
    color = "black"
  ) +
  scale_color_manual(values = c(
    "Best frequency"              = "red",
    "Significantly worse"    = "black",
    "Not significantly worse" = "blue"
  )) +
```

```
    labs(
      title = "Efficiency Comparison Across Frequencies",
      x = "Std Dev of Effectiveness (%)",
      y = "Mean Effectiveness (%)",
      color = "Significance"
    ) +
    theme_minimal()

  return(plot)
}

# Example Usage
# Assume `Es_no_fees` is the data matrix of size n_options x n_frequencies
plot_efficiency_comparison(Es_no_fees)
ggsave("scatter_plot_single.png", width = 7, height = 5)
plot_efficiency_comparison(Es_fees)
ggsave("scatter_plot_single_delta_fee.png", width = 7, height = 5)
plot_efficiency_comparison(Es_dv_no_fees)
ggsave("scatter_plot_single_dv.png", width = 7, height = 5)
plot_efficiency_comparison(Es_dv_fees)
ggsave("scatter_plot_single_dv_fee.png", width = 7, height = 5)
plot_efficiency_comparison(Es_v_port_no_fees)
ggsave("scatter_plot_port_delta.png", width = 7, height = 5)
plot_efficiency_comparison(Es_dv_port_no_fees)
ggsave("scatter_plot_port_delta_vega.png", width = 7, height = 5)
```

```
find_pareto_points <- function(df, sd_col = "SDEfficiency", mean_col = "MeanEfficiency") {
  # We'll create a boolean vector indicating Pareto status
  n <- nrow(df)
  pareto_vec <- rep("Pareto Optimal", n)  # start with all TRUE

  for (i in 1:n) {
    for (j in 1:n) {
      if (i != j) {
        # If point j is strictly better in both criteria, i is NOT Pareto optimal
        if ((df[[sd_col]][j] < df[[sd_col]][i]) &&
            (df[[mean_col]][j] > df[[mean_col]][i])) {
          pareto_vec[i] <- "Non Pareto Optimal"
          break
        }
      }
    }
  }
  return(pareto_vec)
}
```

```
df_single_delta <- data.frame(
  Frequency      = 1:max_freq,
  MeanEfficiency = means_no_fees,
  SDEfficiency   = sds_no_fees
)

# Mark Pareto
```

```
df_single_delta$IsPareto <- find_pareto_points(df_single_delta)

df_single_delta_fees <- data.frame(
  Frequency      = 1:max_freq,
  MeanEfficiency = means_fees,
  SDEfficiency   = sds_fees
)

# Mark Pareto
df_single_delta_fees$IsPareto <- find_pareto_points(df_single_delta_fees)


df_single_dv <- data.frame(
  Frequency      = 1:max_freq,
  MeanEfficiency = means_dv_no_fees,
  SDEfficiency   = sds_dv_no_fees
)

df_single_dv$IsPareto <- find_pareto_points(df_single_dv)

df_strangle_delta <- data.frame(
  Frequency      = 1:max_freq,
  MeanEfficiency = means_v_port_no_fees,
  SDEfficiency   = sds_v_port_no_fees
)

df_strangle_delta$IsPareto <- find_pareto_points(df_strangle_delta)

df_strangle_dv <- data.frame(
  Frequency      = 1:max_freq,
  MeanEfficiency = means_dv_port_no_fees,
  SDEfficiency   = sds_dv_port_no_fees
)

df_strangle_dv$IsPareto <- find_pareto_points(df_strangle_dv)
```

```
p_single_delta <- ggplot(df_single_delta,
                         aes(x = SDEfficiency, y = MeanEfficiency)) +
  #geom_point(aes(color = IsPareto), size = 3) +
  geom_point(size = 3) +
  geom_text(
    aes(label = sprintf("1/%i", Frequency)),
    vjust = -1,          # adjust vertical position
    size = 3,
    color = "black"      # or you can color by IsPareto if you prefer
  ) +
  #scale_color_manual(values = c("Non Pareto Optimal" = "black", "Pareto Optimal" = "red")) +
  labs(
    x = "Std Dev of Effectiveness (%)",
    y = "Mean Effectiveness (%)",
    color = "Pareto Optimality"
  ) +
  theme_minimal()
```

```r
ggsave("scatter_plot_single.png", plot = p_single_delta, width = 7, height = 5)
p_single_delta
```

```r
p_single_delta_fees <- ggplot(df_single_delta_fees,
                         aes(x = SDEfficiency, y = MeanEfficiency)) +
  #geom_point(aes(color = IsPareto), size = 3) +
  geom_point(size = 3) +
  geom_text(
    aes(label = sprintf("1/%i", Frequency)),
    vjust = -1,          # adjust vertical position
    size = 3,
    color = "black"      # or you can color by IsPareto if you prefer
  ) +
  #scale_color_manual(values = c("Non Pareto Optimal" = "black", "Pareto Optimal" = "red")) +
  labs(
    x = "Std Dev of Effectiveness (%)",
    y = "Mean Effectiveness (%)",
    color = "Pareto Optimality"
  ) +
  theme_minimal()

ggsave("scatter_plot_single_delta_fee.png", plot = p_single_delta_fees, width = 7, height = 5)
p_single_delta_fees
```

```r
p_single_delta_vega <- ggplot(df_single_dv,
                         aes(x = SDEfficiency, y = MeanEfficiency)) +
  #geom_point(aes(color = IsPareto), size = 3) +
  geom_point(size = 3) +
  geom_text(
    aes(label = sprintf("1/%i", Frequency)),
    vjust = -1,          # adjust vertical position
    size = 3,
    color = "black"      # or you can color by IsPareto if you prefer
  ) +
  #scale_color_manual(values = c("Non Pareto Optimal" = "black", "Pareto Optimal" = "red")) +
  labs(
    x = "Std Dev of Effectiveness (%)",
    y = "Mean Effectiveness (%)",
    color = "Pareto Optimality"
  ) +
  theme_minimal()

ggsave("scatter_plot_single_dv.png", plot = p_single_delta_vega, width = 7, height = 5)
p_single_delta_vega
```

```r
# Run delta_hedging for different frequencies and collect results
run_delta_hedging <- function(data, frequencies, fee_rate = 0.0) {
  results <- list()

  for (freq in frequencies) {
    data <- data[data$TimeToMaturity <= 0.124,]
    result <- delta_hedging(data, frequency = freq, fee_rate = fee_rate)
```

```
    dates <- data$Date[1:length(result$differences_hedge)] # Match the dates to the results

    results[[as.character(freq)]] <- data.frame(
      Date = dates,
      Error = result$differences_hedge,
      Frequency = paste0("1/", freq)
    )
  }

  return(do.call(rbind, results)) # Combine all results into one data frame
}


frequencies <- list(1, 5, 10)  # Rehedging frequencies to test

hedging_results <- run_delta_hedging(processed_call_options_list[[10]], frequencies)

# Plot the results
ggplot(hedging_results, aes(x = Date, y = Error, color = Frequency)) +
  geom_line() +
  labs(
    title = "Single Call Option of S&P 500 expiring on 20-05-2022",
    x = "Time",
    y = "Daily Hedging Error",
    color = "Frequency"
  ) +
  theme_minimal() +
  theme(
    legend.position = "bottom",
    legend.title = element_text(size = 10),
    legend.text = element_text(size = 8)
  )

ggsave("mse_daily_delta_single.png", width=10, height=5)
```

```
# Run delta_hedging for different frequencies and collect results
run_delta_vega_hedging <- function(data, frequencies, fee_rate = 0.0) {
  results <- list()

  for (freq in frequencies) {
    result <- delta_vega_hedging(data, frequency = freq, fee_rate = fee_rate)
    dates <- data$Date[1:length(result$differences_hedge)] # Match the dates to the results
    results[[as.character(freq)]] <- data.frame(
      Date = dates,
      Error = result$differences_hedge,
      Frequency = paste0("1/", freq)
    )
  }

  return(do.call(rbind, results)) # Combine all results into one data frame
}
```

```
frequencies <- list(1, 5, 10)  # Rehedging frequencies to test
# Assuming `data` is preloaded with the required option data
hedging_results <- run_delta_vega_hedging(merged_options_list[[9]], frequencies)

# Plot the results
ggplot(hedging_results, aes(x = Date, y = Error, color = Frequency)) +
  geom_line() +
  labs(
    title = "Single Call Option of S&P 500 expiring on 20-05-2022",
    x = "Time",
    y = "Daily Hedging Error",
    color = "Frequency"
  ) +
  theme_minimal() +
  theme(
    legend.position = "bottom",
    legend.title = element_text(size = 10),
    legend.text = element_text(size = 8)
  )

ggsave("mse_daily_delta_vega_single.png", width=10, height=5)
```

```
# Run delta_hedging for different frequencies and collect results
run_delta_hedging_port <- function(data, frequencies, fee_rate = 0.0) {
  results <- list()

  for (freq in frequencies) {
    data <- data[data$TimeToMaturity <= 0.124,]
    result <- delta_hedging_portfolio(data, frequency = freq, fee_rate = fee_rate)
    dates <- data$Date[1:length(result$differences_hedge)] # Match the dates to the results

    results[[as.character(freq)]] <- data.frame(
      Date = dates,
      Error = result$differences_hedge,
      Frequency = paste0("1/", freq)
    )
  }

  return(do.call(rbind, results)) # Combine all results into one data frame
}


frequencies <- seq(10)  # Rehedging frequencies to test

hedging_results <- run_delta_hedging_port(merged_call_put_list[[10]], frequencies)

# Plot the results
ggplot(hedging_results, aes(x = Date, y = Error, color = Frequency)) +
  geom_line() +
  labs(
    title = "Call and Put Option of S&P 500 expiring on 19-05-2023",
    x = "Time",
    y = "Daily Hedging Error",
```

```
    color = "Frequency"
  ) +
  theme_minimal() +
  theme(
    legend.position = "bottom",
    legend.title = element_text(size = 10),
    legend.text = element_text(size = 8)
  )

ggsave("mse_daily_delta_portfolio.png", width=10, height=5)
```

```
# Run delta_hedging for different frequencies and collect results
run_delta_vega_hedging_port <- function(data, frequencies, fee_rate = 0.0) {
  results <- list()

  for (freq in frequencies) {
    result <- delta_vega_hedging_portfolio(data, frequency = freq, fee_rate = fee_rate)
    dates <- data$Date[1:length(result$differences_hedge)] # Match the dates to the results

    results[[as.character(freq)]] <- data.frame(
      Date = dates,
      Error = result$differences_hedge,
      Frequency = paste0("1/", freq)
    )
  }

  return(do.call(rbind, results)) # Combine all results into one data frame
}


frequencies <- seq(10)  # Rehedging frequencies to test

hedging_results <- run_delta_vega_hedging_port(merged_portfolio_list[[8]], frequencies)

# Plot the results
ggplot(hedging_results, aes(x = Date, y = Error, color = Frequency)) +
  geom_line() +
  labs(
    title = "Call and Put Option of S&P 500 expiring on 19-05-2023",
    x = "Time",
    y = "Daily Hedging Error",
    color = "Frequency"
  ) +
  theme_minimal() +
  theme(
    legend.position = "bottom",
    legend.title = element_text(size = 10),
    legend.text = element_text(size = 8)
  )

ggsave("mse_daily_delta_vega_portfolio.png", width=10, height=5)
```

```r
# Function to calculate efficiency for multiple datasets and average across them
calculate_sample_average_efficiency <- function(data_list, frequencies, fee_rates) {
  not_test_dates <- c(as.Date("2021-05-20", format = "%Y-%m-%d"), as.Date("2021-06-17", format = "%Y-%
  results <- data.frame()

  for (freq in frequencies) {
    for (fee_rate in fee_rates) {
      efficiencies <- c()  # To store efficiencies for each dataset

      for (data in data_list) {
        if (data$Date[nrow(data)] %in% not_test_dates) {
          next
        }
        data <- data[data$TimeToMaturity <= 0.124,]
        result <- delta_hedging(data, frequency = freq, fee_rate = fee_rate)
        efficiencies <- c(efficiencies, result$efficiency)  # Collect efficiency
      }

      # Calculate the sample average efficiency across datasets
      avg_efficiency <- mean(efficiencies, na.rm = TRUE)

      # Append results to the data frame
      results <- rbind(results, data.frame(
        Frequency = paste0("1/", freq),
        FeeRate = fee_rate,
        AvgEfficiency = avg_efficiency
      ))
    }
  }

  return(results)
}


frequencies <- c(1, 2, 5, 10)  # Rehedging frequencies
fee_rates <- seq(0, 0.02, by = 0.0001)  # Fee rates from 0 to 0.05 in steps of 0.01

sample_average_results <- calculate_sample_average_efficiency(processed_call_options_list, frequencies

# Plot the results
ggplot(sample_average_results, aes(x = FeeRate, y = AvgEfficiency, color = as.factor(Frequency))) +
  geom_line(size = 1) +
  labs(
    title = "Average Effectiveness by Fee Rates for Delta Hedging of a Single S&P 500 Call Option",
    x = "Fee Rate",
    y = "Average Effectiveness (%)",
    color = "Frequency"
  ) +
  theme_minimal() +
  theme(
    legend.position = "bottom",
    legend.title = element_text(size = 10),
    legend.text = element_text(size = 8)
```

```
  )
ggsave("fee_single_delta.png", width=10, height=5)
```

```
# Function to calculate efficiency for multiple datasets and average across them
calculate_sample_average_efficiency_dv <- function(data_list, frequencies, fee_rates) {
  not_test_dates <- c(as.Date("2021-05-20", format = "%Y-%m-%d"), as.Date("2021-06-17", format = "%Y-%
  results <- data.frame()

  for (freq in frequencies) {
    for (fee_rate in fee_rates) {
      efficiencies <- c()  # To store efficiencies for each dataset

      for (data in data_list) {
        if (data$Date[nrow(data)] %in% not_test_dates) {
          next
        }
        result <- delta_vega_hedging(data, frequency = freq, fee_rate = fee_rate)
        efficiencies <- c(efficiencies, result$efficiency)  # Collect efficiency
      }

      # Calculate the sample average efficiency across datasets
      avg_efficiency <- mean(efficiencies, na.rm = TRUE)

      # Append results to the data frame
      results <- rbind(results, data.frame(
        Frequency = paste0("1/", freq),
        FeeRate = fee_rate,
        AvgEfficiency = avg_efficiency
      ))
    }
  }

  return(results)
}


frequencies <- c(1, 2, 5, 10)  # Rehedging frequencies
fee_rates <- seq(0, 0.02, by = 0.0001)  # Fee rates from 0 to 0.05 in steps of 0.01

sample_average_results_dv <- calculate_sample_average_efficiency_dv(merged_options_list, frequencies,

# Plot the results
ggplot(sample_average_results_dv, aes(x = FeeRate, y = AvgEfficiency, color = as.factor(Frequency))) +
  geom_line(size = 1) +
  labs(
    title = "Average Effectiveness by Fee Rates for Delta-Vega Hedging of a Single S&P 500 Call Option
    x = "Fee Rate",
    y = "Average Effectiveness (%)",
    color = "Frequency"
  ) +
  theme_minimal() +
  theme(
    legend.position = "bottom",
```

```
    legend.title = element_text(size = 10),
    legend.text = element_text(size = 8)
  )
ggsave("fee_single_delta_vega.png", width=10, height=5)
```

```
# Function to calculate efficiency for multiple datasets and average across them
calculate_sample_average_efficiency_port <- function(data_list, frequencies, fee_rates) {
  results <- data.frame()

  for (freq in frequencies) {
    for (fee_rate in fee_rates) {
      efficiencies <- c()  # To store efficiencies for each dataset

      for (data in data_list) {
        data <- data[data$TimeToMaturity <= 0.124,]
        result <- delta_hedging_portfolio(data, frequency = freq, fee_rate = fee_rate)
        efficiencies <- c(efficiencies, result$efficiency)  # Collect efficiency
      }

      # Calculate the sample average efficiency across datasets
      avg_efficiency <- mean(efficiencies, na.rm = TRUE)

      # Append results to the data frame
      results <- rbind(results, data.frame(
        Frequency = paste0("1/", freq),
        FeeRate = fee_rate,
        AvgEfficiency = avg_efficiency
      ))
    }
  }

  return(results)
}


frequencies <- c(1, 2, 5, 10)  # Rehedging frequencies
fee_rates <- seq(0, 0.02, by = 0.0001)  # Fee rates from 0 to 0.05 in steps of 0.01

sample_average_results_port <- calculate_sample_average_efficiency_port(merged_call_put_list, frequenc

# Plot the results
ggplot(sample_average_results_port, aes(x = FeeRate, y = AvgEfficiency, color = as.factor(Frequency)))
  geom_line(size = 1) +
  labs(
    title = "Average Effectiveness by Fee Rates for Delta Hedging of a Strangle Position",
    x = "Fee Rate",
    y = "Average Effectiveness (%)",
    color = "Frequency"
  ) +
  theme_minimal() +
  theme(
    legend.position = "bottom",
    legend.title = element_text(size = 10),
```

```
    legend.text = element_text(size = 8)
  )
ggsave("fee_port_delta.png", width=10, height=5)
```

```
# Function to calculate efficiency for multiple datasets and average across them
calculate_sample_average_efficiency_dv_port <- function(data_list, frequencies, fee_rates) {
  results <- data.frame()

  for (freq in frequencies) {
    for (fee_rate in fee_rates) {
      efficiencies <- c()  # To store efficiencies for each dataset

      for (data in data_list) {
        result <- delta_vega_hedging_portfolio(data, frequency = freq, fee_rate = fee_rate)
        efficiencies <- c(efficiencies, result$efficiency)  # Collect efficiency
      }

      # Calculate the sample average efficiency across datasets
      avg_efficiency <- mean(efficiencies, na.rm = TRUE)

      # Append results to the data frame
      results <- rbind(results, data.frame(
        Frequency = paste0("1/", freq),
        FeeRate = fee_rate,
        AvgEfficiency = avg_efficiency
      ))
    }
  }

  return(results)
}


frequencies <- c(1, 2, 5, 10)  # Rehedging frequencies
fee_rates <- seq(0, 0.02, by = 0.0001)  # Fee rates from 0 to 0.05 in steps of 0.01

sample_average_results_dv_port <- calculate_sample_average_efficiency_dv_port(merged_portfolio_list, f

# Plot the results
ggplot(sample_average_results_dv_port, aes(x = FeeRate, y = AvgEfficiency, color = as.factor(Frequency
  geom_line(size = 1) +
  labs(
    title = "Average Effectiveness by Fee Rates for Delta-Vega Hedging of a Strangle Position",
    x = "Fee Rate",
    y = "Average Effectiveness (%)",
    color = "Frequency"
  ) +
  theme_minimal() +
  theme(
    legend.position = "bottom",
    legend.title = element_text(size = 10),
    legend.text = element_text(size = 8)
  )
```

```
ggsave("fee_port_delta_vega.png", width=10, height=5)
```

```
# Combine Means and SDs for Visualization
df_no_fees <- data.frame(
  Frequency = 1:max_freq,
  Delta_Mean = means_no_fees,
  Delta_SD = sds_no_fees,
  Delta_Vega_Mean = means_dv_no_fees,
  Delta_Vega_SD = sds_dv_no_fees
)

df_fees <- data.frame(
  Frequency = 1:max_freq,
  Delta_Mean = means_fees,
  Delta_SD = sds_fees,
  Delta_Vega_Mean = means_dv_fees,
  Delta_Vega_SD = sds_dv_fees
)

df_no_fees_port <- data.frame(
  Frequency = 1:max_freq,
  Delta_Mean = means_v_port_no_fees,
  Delta_SD = sds_v_port_no_fees,
  Delta_Vega_Mean = means_dv_port_no_fees,
  Delta_Vega_SD = sds_dv_port_no_fees
)

# Plot 1: Mean Efficiency vs Rehedging Frequency (No Fees and Fees)
ggplot(df_no_fees, aes(x = Frequency)) +
  geom_line(aes(y = Delta_Mean, color = "Delta Hedging (No Fees)")) +
  geom_line(aes(y = Delta_Vega_Mean, color = "Delta-Vega Hedging (No Fees)")) +
  geom_errorbar(aes(ymin = Delta_Mean - Delta_SD, ymax = Delta_Mean + Delta_SD, color = "Delta Hedging
  geom_errorbar(aes(ymin = Delta_Vega_Mean - Delta_Vega_SD, ymax = Delta_Vega_Mean + Delta_Vega_SD, co
  labs(title = "Mean Effectiveness vs Rehedging Frequency (No Fees)",
       x = "Rehedging Frequency (Days)",
       y = "Effectiveness (%)",
       color = "Legend") +
  theme_minimal() +
  scale_x_continuous(breaks = scales::pretty_breaks(n = 10)) # Adjust 'n' to control number of breaks

ggsave("single_delta_vs_dv.png", width=8, height=5)


ggplot(df_fees, aes(x = Frequency)) +
  geom_line(aes(y = Delta_Mean, color = "Delta Hedging (With Fees)")) +
  geom_line(aes(y = Delta_Vega_Mean, color = "Delta-Vega Hedging (With Fees)")) +
  geom_errorbar(aes(ymin = Delta_Mean - Delta_SD, ymax = Delta_Mean + Delta_SD, color = "Delta Hedging
  geom_errorbar(aes(ymin = Delta_Vega_Mean - Delta_Vega_SD, ymax = Delta_Vega_Mean + Delta_Vega_SD, co
  labs(title = "Mean Effectiveness vs Rehedging Frequency (With Fees)",
       x = "Rehedging Frequency (Days)",
       y = "Effectiveness (%)",
       color = "Legend") +
```

```
  theme_minimal()+
  scale_x_continuous(breaks = scales::pretty_breaks(n = 10)) # Adjust 'n' to control number of breaks
ggsave("single_delta_vs_dv_fee.png", width=8, height=5)


ggplot(df_no_fees_port, aes(x = Frequency)) +
  geom_line(aes(y = Delta_Mean, color = "Delta Hedging (No Fees)")) +
  geom_line(aes(y = Delta_Vega_Mean, color = "Delta-Vega Hedging (No Fees)")) +
  geom_errorbar(aes(ymin = Delta_Mean - Delta_SD, ymax = Delta_Mean + Delta_SD, color = "Delta Hedging
  geom_errorbar(aes(ymin = Delta_Vega_Mean - Delta_Vega_SD, ymax = Delta_Vega_Mean + Delta_Vega_SD, co
  labs(title = "Mean Effectiveness vs Rehedging Frequency (No Fees)",
       x = "Rehedging Frequency (Days)",
       y = "Effectiveness (%)",
       color = "Legend") +
  theme_minimal()+
  scale_x_continuous(breaks = scales::pretty_breaks(n = 10)) # Adjust 'n' to control number of breaks
ggsave("port_delta_vs_dv.png", width=8, height=5)
```