Empirical Analysis of AMZN Microstructure Data

Sid Bhatia

12/3/2023

Importation

Import the required libraries.

```
library(highfrequency)
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.2     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(xts)
```

```
## Loading required package: zoo
##
## Attaching package: 'zoo'
##
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
##
##
## ################################### WARNING ###################################
## # We noticed you have dplyr installed. The dplyr lag() function breaks how    #
## # base R's lag() function is supposed to work, which breaks lag(my_xts).      #
## #                                                                             #
## # Calls to lag(my_xts) that you enter or source() into this session won't     #
## # work correctly.                                                             #
## #                                                                             #
## # All package code is unaffected because it is protected by the R namespace   #
## # mechanism.                                                                  #
## #                                                                             #
## # Set `options(xts.warn_dplyr_breaks_lag = FALSE)` to suppress this warning.  #
## #                                                                             #
## # You can use stats::lag() to make sure you're not using dplyr::lag(), or you #
## # can add conflictRules('dplyr', exclude = 'lag') to your .Rprofile to stop   #
## # dplyr from breaking base R's lag() function.                                #
```

```
## ################################### WARNING ###################################
##
## Attaching package: 'xts'
##
## The following objects are masked from 'package:dplyr':
##
##     first, last
```

```r
library(lubridate)
```

Data Pre-Processing

Load and pre-process the TAQ data.

```r
# Write file path for data.
file_path <- 'C://Users//sbhatia2//OneDrive - stevens.edu//Desktop//amzn_tick_history_01_04_2023.csv'

# Read in TAQ data.
data <- read.csv(file_path)

# head(data)
# tail(data)

# Display columns of dataset.
# colnames(data)

# head(data$Bid.Price)
# tail(data$Bid.Price)
#
# head(data$Ask.Price)
# tail(data$Ask.Price)

# Remove rows with NA Bid/Ask/Trade prices.
cleaned_data <- data[!is.na(data$Bid.Price) & !is.na(data$Ask.Price) & !is.na(data$Price), ]

# head(cleaned_data)
# tail(cleaned_data)

# Convert Exchange Time column to be compatible with xts object.
cleaned_data$Exch.Time <- as.POSIXct(paste("2000-01-01", cleaned_data$Exch.Time))

# Convert dataframe into xts object.
amzn_xts <- xts(cleaned_data[,-which(names(cleaned_data) == "Exch.Time")], order.by = cleaned_data$Exch

# Rename Bid, Ask, and Symbol columns to match correct format.
names(amzn_xts)[names(amzn_xts) == 'Bid.Price'] <- 'BID'
names(amzn_xts)[names(amzn_xts) == 'Ask.Price'] <- 'OFR'
names(amzn_xts)[names(amzn_xts) == 'X.RIC'] <- 'SYMBOL'

# head(amzn_xts)
# tail(amzn_xts)
```

Liquidity Analysis

Perform core liquidity analysis for the project.

Price Computation Retrieve and compute the bid, ask, trade, and mid prices.

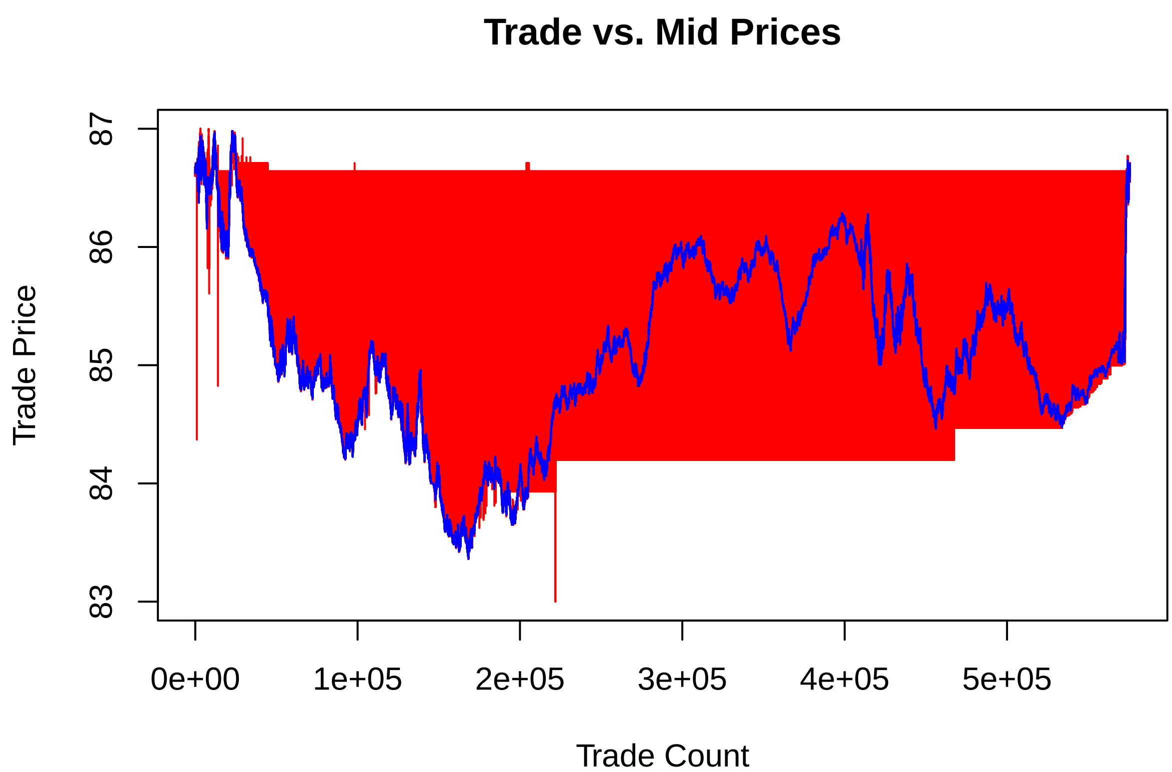
```r
# Determine number of trades in dataset.
num_of_trades <- nrow(amzn_xts)

paste("Total number of trades for AMZN on 01/04/2023:", num_of_trades)
```

```
## [1] "Total number of trades for AMZN on 01/04/2023: 575733"
```

```r
# Retrieve trade, bid, and ask prices.
prices <- as.numeric(amzn_xts$Price)
bids <- as.numeric(amzn_xts$BID)
asks <- as.numeric(amzn_xts$OFR)

# Compute the mid price as the average of the bid and ask prices.
mids <- (bids + asks) * 0.5
```

Trade vs. Mid Price Plotting Plot the trade vs. mid prices.

```r
plot(prices, type="l", col="red", main="Trade vs. Mid Prices",
     xlab="Trade Count",
     ylab="Trade Price")
lines(mids, type="l", col="blue")
```



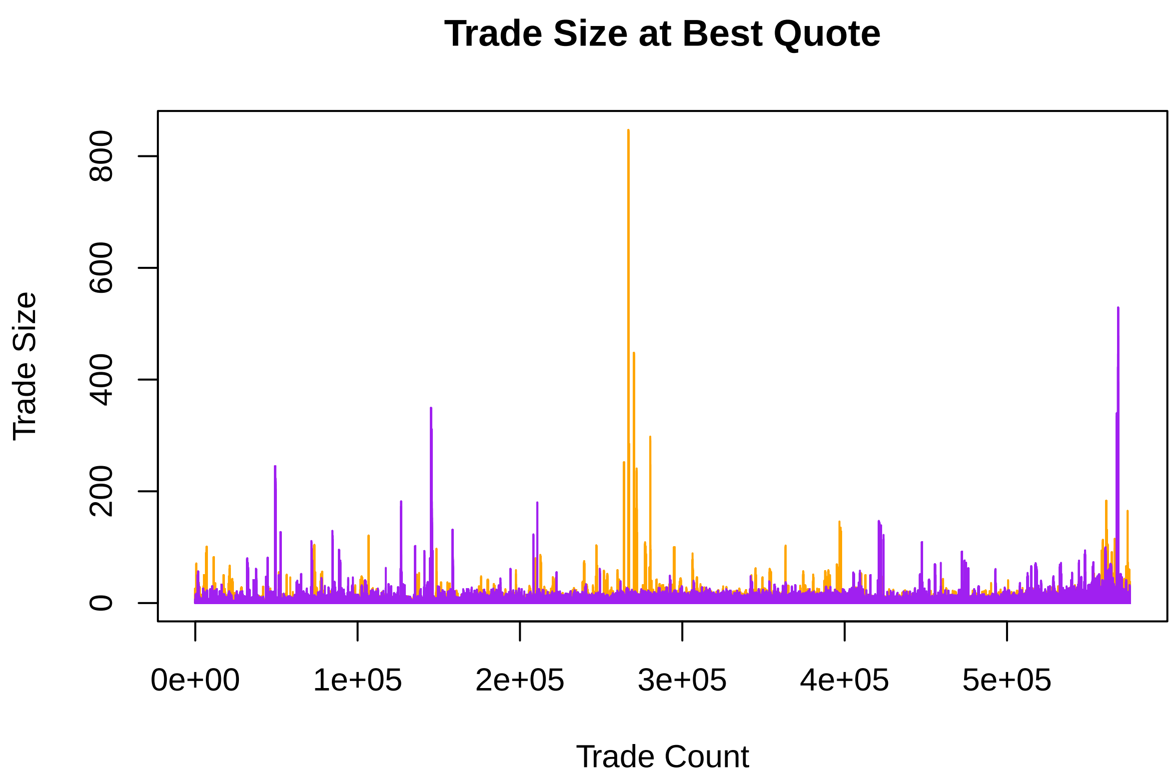
Size Computation Retrieve and compute the bid and ask sizes as well as the LOB imbalances.

```r
# Retrieve bid and ask sizes.
bid_size <- as.numeric(amzn_xts$Bid.Size)
```

```
ask_size <- as.numeric(amzn_xts$Ask.Size)

# Compute the LOB imbalance at each time step as the sum of the bid and ask sizes.
LOB_imbalance <- bid_size + ask_size
```

Trade Size Plotting Plot the trade sizes.

```
plot(ask_size, col="orange", type="h",
     ylab="Trade Size",
     xlab="Trade Count", main="Trade Size at Best Quote")
lines(bid_size, col="purple", type="h")
```



Liquidity Measures Compute the relevant liquidity measures.

```
# Retrieve liquidity measures.
liquidity_measures <- getLiquidityMeasures(amzn_xts)

# Compute the Quoted Spread.
quoted_spread <- mean(as.numeric(liquidity_measures$quotedSpread))

quoted_spread
```

```
## [1] 0.01512349
```

```
# Compute the Effective Spread.
effective_spread <- mean(as.numeric(liquidity_measures$effectiveSpread))
```

```r
effective_spread
```

```
## [1] 0.03655411
```

```r
# Compute the Realized Spread.
realized_spread <- mean(na.omit(as.numeric(liquidity_measures$realizedSpread)))

realized_spread
```

```
## [1] 0.02876561
```

```r
# Compute the Price Impact.
price_impact <- mean(na.omit(as.numeric(liquidity_measures$priceImpact)))

price_impact
```

```
## [1] 0.00389478
```

Time Bucket Analysis

Perform analysis on each hour in the dataset.

```r
## Split dataset into 24 buckets (24 hours) for analysis.

# amzn_data_list <- list()
#
# for (i in 0:23)
# {
#   # Construct the time range string.
#   start_hour <- sprintf("%02d:00", i)
#   end_hour <- end_hour <- ifelse(i == 23, "23:59:59", sprintf("%02d:00", i + 1))
#   time_range <- paste("T", start_hour, "/", "T", end_hour, sep = "")
#
#   # Subset amzn_xts and store in the list.
#   amzn_data_list[[paste("amzn_xts_", i + 1, sep = "")]] <- amzn_data_list[time_range]
# }
#
# amzn_data_list

amzn_xts_1 <- amzn_xts["T00:00/T01:00"]
amzn_xts_2 <- amzn_xts["T01:00/T02:00"]
amzn_xts_3 <- amzn_xts["T02:00/T03:00"]
amzn_xts_4 <- amzn_xts["T03:00/T04:00"]
amzn_xts_5 <- amzn_xts["T04:00/T05:00"]
amzn_xts_6 <- amzn_xts["T05:00/T06:00"]
amzn_xts_7 <- amzn_xts["T06:00/T07:00"]
amzn_xts_8 <- amzn_xts["T07:00/T08:00"]
amzn_xts_9 <- amzn_xts["T08:00/T09:00"]
amzn_xts_10 <- amzn_xts["T09:00/T10:00"]
amzn_xts_11 <- amzn_xts["T10:00/T11:00"]
amzn_xts_12 <- amzn_xts["T11:00/T12:00"]
amzn_xts_13 <- amzn_xts["T12:00/T13:00"]
amzn_xts_14 <- amzn_xts["T13:00/T14:00"]
amzn_xts_15 <- amzn_xts["T14:00/T15:00"]
amzn_xts_16 <- amzn_xts["T15:00/T16:00"]
```

```r
amzn_xts_17 <- amzn_xts["T16:00/T17:00"]
amzn_xts_18 <- amzn_xts["T17:00/T18:00"]
amzn_xts_19 <- amzn_xts["T18:00/T19:00"]
amzn_xts_20 <- amzn_xts["T19:00/T20:00"]
amzn_xts_21 <- amzn_xts["T20:00/T21:00"]
amzn_xts_22 <- amzn_xts["T21:00/T22:00"]
amzn_xts_23 <- amzn_xts["T22:00/T23:00"]
amzn_xts_24 <- amzn_xts["T23:00/T23:59:59"]

# head(amzn_xts_1)
# head(amzn_xts_24)
```

```r
## Calculate liquidity measures for each subset.

liquidity_1 <- getLiquidityMeasures(amzn_xts_1)
liquidity_2 <- getLiquidityMeasures(amzn_xts_2)
liquidity_3 <- getLiquidityMeasures(amzn_xts_3)
liquidity_4 <- getLiquidityMeasures(amzn_xts_4)
liquidity_5 <- getLiquidityMeasures(amzn_xts_5)
liquidity_6 <- getLiquidityMeasures(amzn_xts_6)
liquidity_7 <- getLiquidityMeasures(amzn_xts_7)
liquidity_8 <- getLiquidityMeasures(amzn_xts_8)
liquidity_9 <- getLiquidityMeasures(amzn_xts_9)
liquidity_10 <- getLiquidityMeasures(amzn_xts_10)
liquidity_11 <- getLiquidityMeasures(amzn_xts_11)
liquidity_12 <- getLiquidityMeasures(amzn_xts_12)
liquidity_13 <- getLiquidityMeasures(amzn_xts_13)
liquidity_14 <- getLiquidityMeasures(amzn_xts_14)
liquidity_15 <- getLiquidityMeasures(amzn_xts_15)
liquidity_16 <- getLiquidityMeasures(amzn_xts_16)
liquidity_17 <- getLiquidityMeasures(amzn_xts_17)
liquidity_18 <- getLiquidityMeasures(amzn_xts_18)
liquidity_19 <- getLiquidityMeasures(amzn_xts_19)
liquidity_20 <- getLiquidityMeasures(amzn_xts_20)
liquidity_21 <- getLiquidityMeasures(amzn_xts_21)
liquidity_22 <- getLiquidityMeasures(amzn_xts_22)
liquidity_23 <- getLiquidityMeasures(amzn_xts_23)
liquidity_24 <- getLiquidityMeasures(amzn_xts_24)

# # Initialize an empty list to store the liquidity measures.
# liquidity_measures_list <- list()

# # Loop through 24 subsets,
# for (i in 1:24) {
#   # Construct the variable name for each amzn_xts_*
#   xts_var_name <- paste("amzn_xts_", i, sep = "")
#
#   # Use get() to retrieve the xts object and calculate the liquidity measures.
#   liquidity_measures_list[[paste("liquidity_", i, sep = "")]] <- getLiquidityMeasures(get(xts_var_nam
# }
```

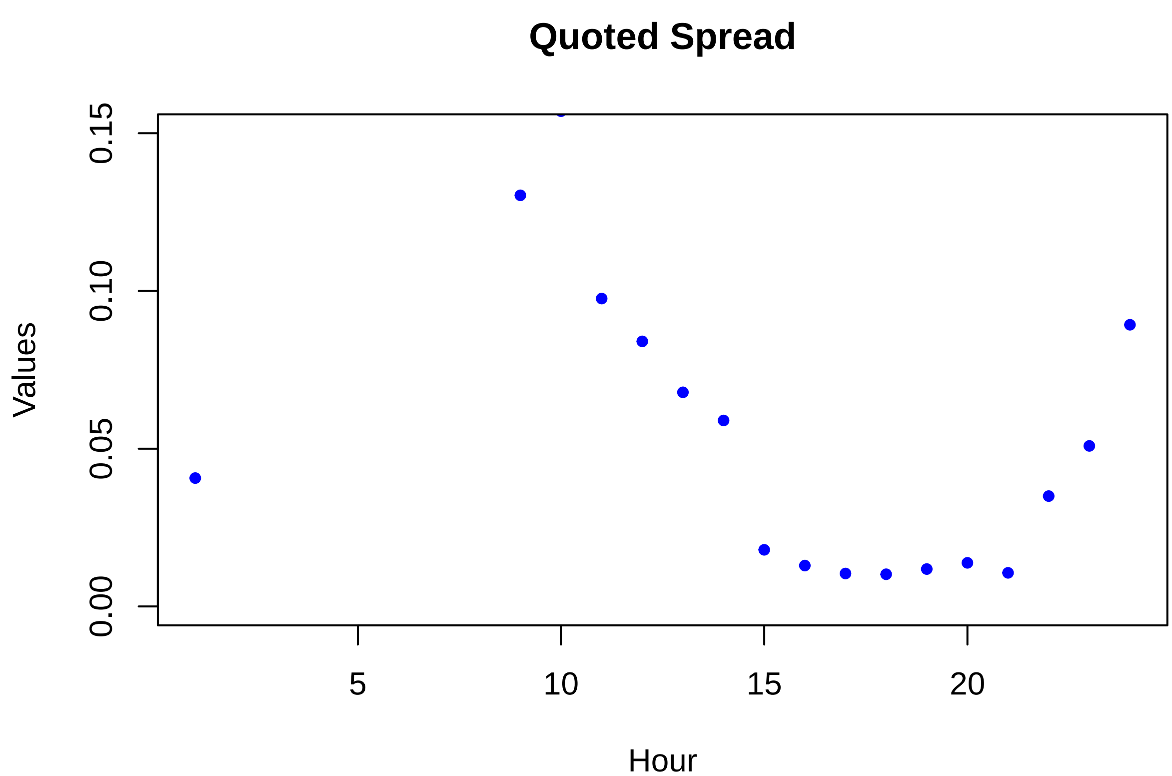
Quoted Spread Calculation Calculated quoted spread for each hour.

```r
## Calculate Quoted Spread for each bucket.

# Initialize an empty vector to store the quoted spread means.
quoted_spread_hr <- numeric(24)

# Loop through 24 hours.
for (i in 1:24) {
  # Construct the variable name as a string.
  var_name <- paste("liquidity_", i, sep = "")

  # Use get() to retrieve the dataframe and calculate the mean quoted spread.
  quoted_spread_hr[i] <- mean(as.numeric(get(var_name)$quotedSpread), na.rm = TRUE)
}

quoted_spread_hr
```

```
##  [1] 0.04068116        NaN        NaN        NaN        NaN        NaN
##  [7]        NaN        NaN 0.13031746 0.15699330 0.09759420 0.08401734
## [13] 0.06786622 0.05894439 0.01794931 0.01295927 0.01043155 0.01019991
## [19] 0.01184737 0.01382093 0.01064357 0.03497011 0.05088099 0.08928669
```

Quoted Spread Plotting Plot quoted spread for each hour.

```r
# Plot Quoted Spread for 24 hours.
plot(1:24, quoted_spread_hr, type="p", pch=20, col="blue", main="Quoted Spread",
     xlab = "Hour", ylab = "Values", ylim = c(0,0.15))
```



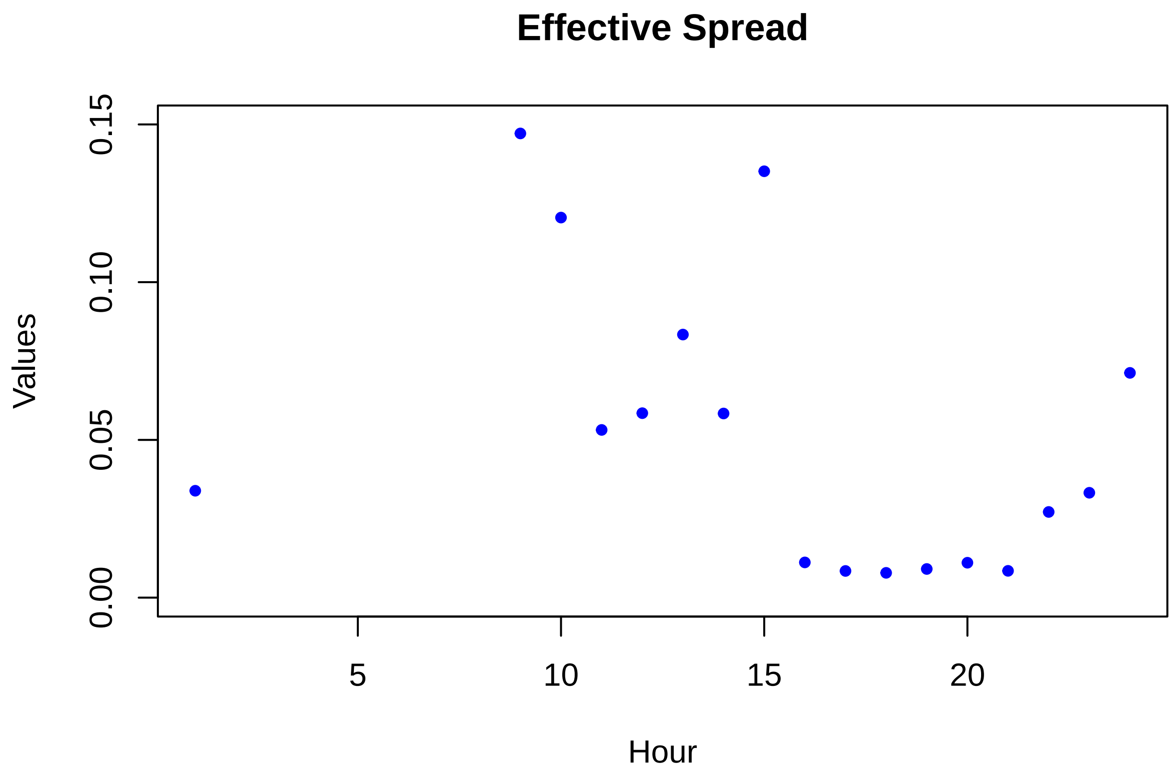
Effective Spread Calculation Calculated effective spread for each hour.

```
## Calculate Effective Spread for each bucket.

# Initialize an empty vector to store the effective spread means.
effective_spread_hr <- numeric(24)

# Loop through 24 hours.
for (i in 1:24) {
  # Construct the variable name as a string.
  var_name <- paste("liquidity_", i, sep = "")

  # Use get() to retrieve the dataframe and calculate the mean quoted spread.
  effective_spread_hr[i] <- mean(as.numeric(get(var_name)$effectiveSpread), na.rm = TRUE)
}

effective_spread_hr
```

```
##  [1] 0.033885700         NaN         NaN         NaN         NaN         NaN
##  [7]         NaN         NaN 0.147142857 0.120477387 0.053159420 0.058468208
## [13] 0.083390050 0.058367702 0.135155833 0.011162538 0.008454183 0.007855181
## [19] 0.009078966 0.011062543 0.008486380 0.027158412 0.033240495 0.071242091
```

Effective Spread Plotting Plot quoted spread for each hour.

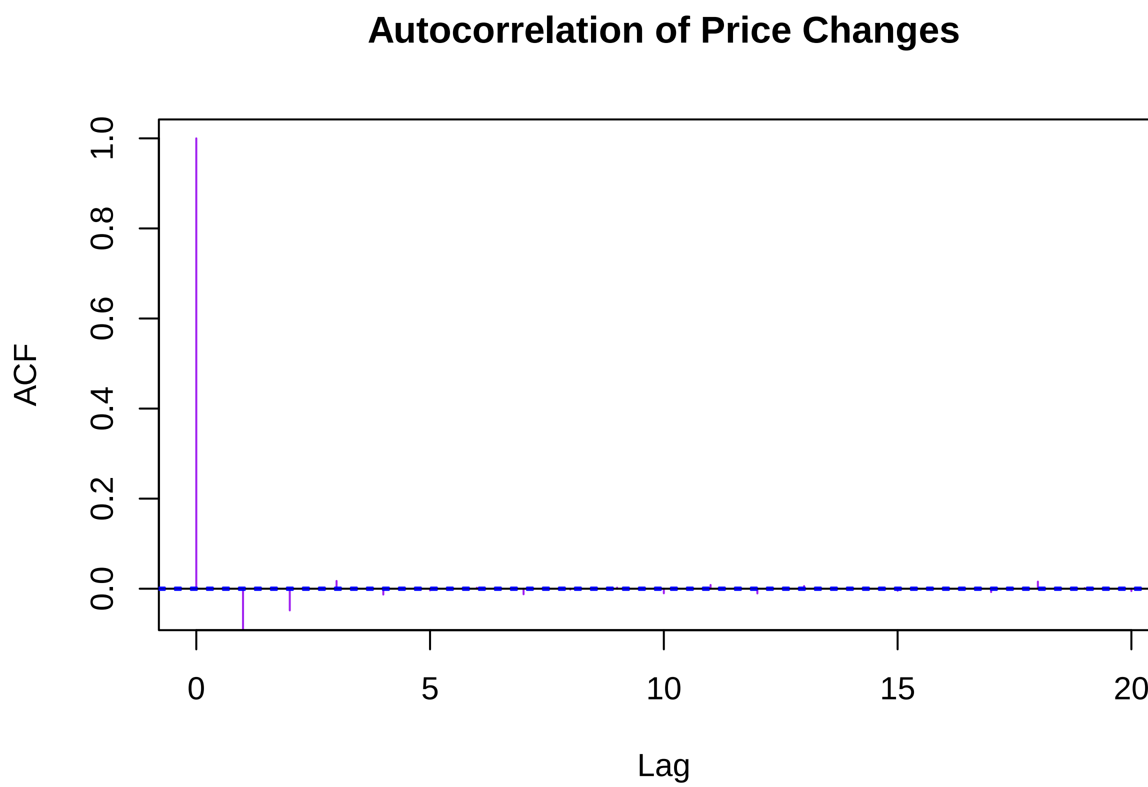
```
# Plot Quoted Spread for 24 hours.
plot(1:24, effective_spread_hr, type="p", pch=20, col="blue", main="Effective Spread",
     xlab = "Hour", ylab = "Values", ylim = c(0,0.15))
```



Volatility Estimate

The following section estimates the volatility using various methods.

```
prices <- as.numeric(amzn_xts$Price)

# Calculate price differences.
price_difference <- diff(prices)

# Display autocorrelation of price differences.
covpr <- acf(price_difference, lag.max=20, type="correlation", plot=FALSE)

plot(covpr, col="purple", ylim = c(-0.05,1), main = "Autocorrelation of Price Changes")
```



Roll Model Estimate

```
## Calculate parameters and estimates for Roll's Model.

covpr <- acf(price_difference, lag.max=20, type="covariance", plot=FALSE)

gamma0 <- sd(price_difference)^2

gamma0
```

```
## [1] 0.005261148
```

```
gamma1 <- covpr$acf[2]

gamma1
```

```
## [1] -0.00224291
```

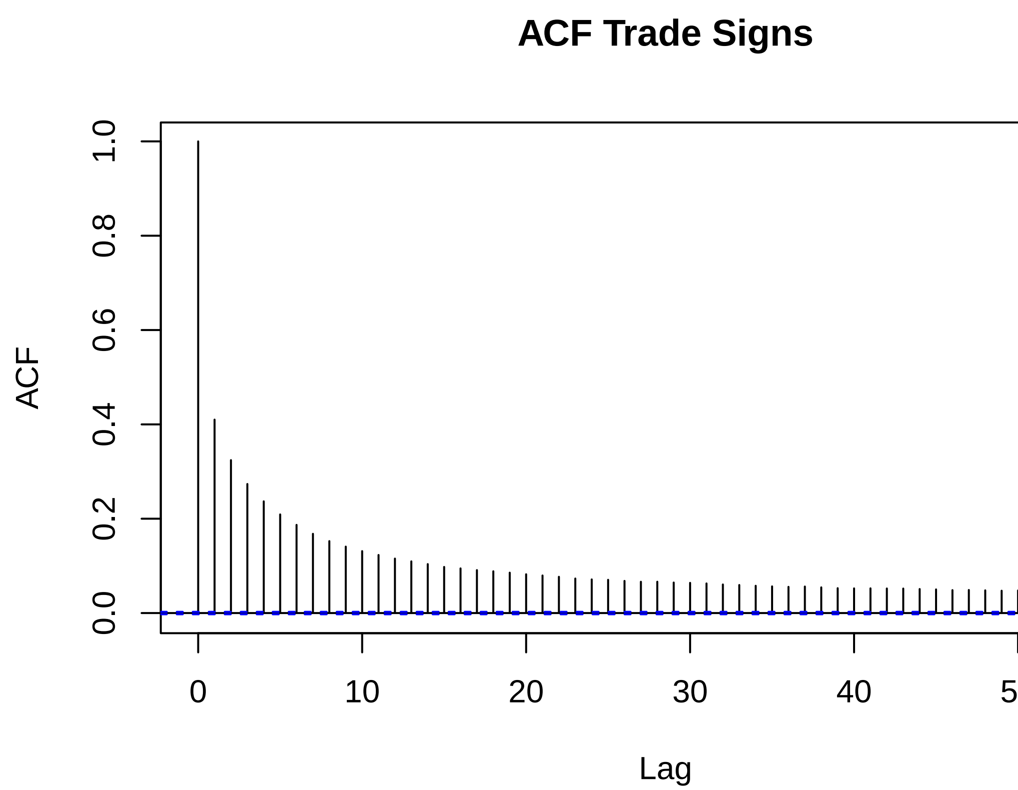
```
cparam <- sqrt(-covpr$acf[2])

cparam
```

```
## [1] 0.04735937
```

```
sig2u <- gamma0 + 2*gamma1

sigu <- sqrt(sig2u)

sigu
```

```
## [1] 0.02784472
```

```r
# Retrieve trade signs.
trade_signs <- getTradeDirection(amzn_xts)

acf_TS <- acf(trade_signs, main="ACF Trade Signs")
```



Improved Roll Model Estimate

```r
trade_signs_diff <- diff(trade_signs)

mid_diff <- diff(mids)

fit_lm <- lm(price_difference ~ mid_diff + trade_signs_diff)

fit_lm$coeff[3]
```

```
## trade_signs_diff
##       0.00953869
```

```r
realized_var <- function(q)
{
  rCov(diff(prices, lag = q, differences = 1)) / q
}

realized_var_df <- NULL

for(q in 1:200)
```

```
{
  realized_var_df <- c(realized_var_df, realized_var(q))
}

trades_per_five <- num_of_trades * 5 / 1440
realized_var_five <- realized_var(trades_per_five)

realized_var_five
```

Signature Plot

```
## [1] 11.41892
```

```
# Realized volatility sampling trades per five minutes.
sqrt(realized_var_five)
```

```
## [1] 3.379189
```

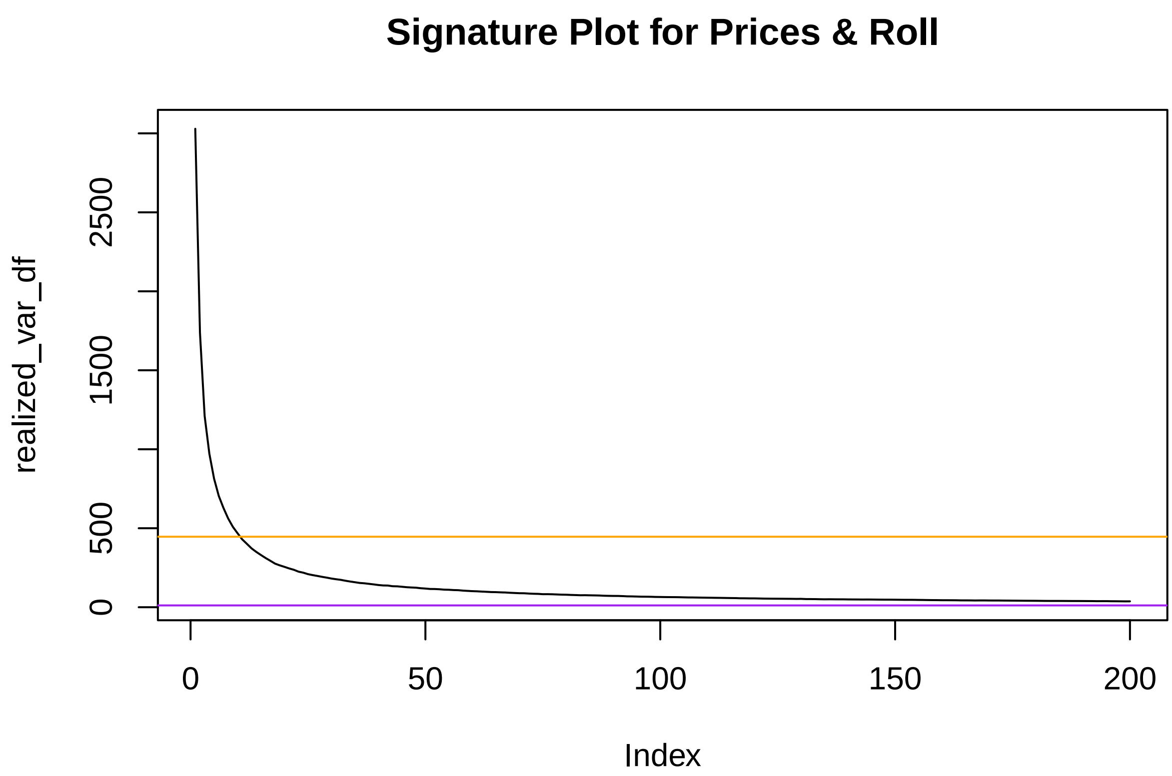
```
realized_volatility_roll <- sig2u * num_of_trades
realized_volatility_roll
```

```
## [1] 446.3821
```

```
sqrt(realized_volatility_roll)
```

```
## [1] 21.12776
```

```
plot(realized_var_df, type ="l",  main="Signature Plot for Prices & Roll")
abline(h = realized_var_five,col="purple")
abline(h = realized_volatility_roll,col="orange")
```



PIN (Probability of Informed Trading)

Compute the Probability of Informed Trading (PIN) based on this formula:

$$PIN = \frac{\alpha\mu}{\alpha\mu + \epsilon_B + \epsilon_S}$$

```r
library(InfoTrad)
```

```
##
## Attaching package: 'InfoTrad'

## The following object is masked from 'package:base':
##
##     print
```

```r
head(trade_signs)
```

```
## [1] -1  1  1  1  1  1
```

```r
tail(trade_signs)
```

```
## [1]  1  1 -1 -1 -1  1
```

```r
trade_direction <- matrix(trade_signs)

buy_side <- which(trade_direction > 0)

head(buy_side)
```

```
## [1] 2 3 4 5 6 7
```

```r
buy_trades_df <- data.frame(
  BuyIndex <- buy_side,
  BuyPrice <- amzn_xts$Price[buy_side]
)

head(buy_trades_df)
```

```
##                        BuyIndex....buy_side    Price
## X2000.01.01.00.00.06                      2 86.65000
## X2000.01.01.00.00.08                      3 86.65000
## X2000.01.01.00.00.11                      4 86.65000
## X2000.01.01.00.00.15                      5 86.65000
## X2000.01.01.00.00.16                      6 86.65000
## X2000.01.01.00.00.21                      7 86.68000
```

```r
sell_side <- which(trade_direction < 0)

head(sell_side)
```

```
## [1]  1  9 11 12 13 14
```

```r
sell_trades_df <- data.frame(
  SellIndex <- sell_side,
  SellPrice <- amzn_xts$Price[sell_side]
)

head(sell_trades_df)
```

```
##                          SellIndex....sell_side    Price
## X2000.01.01.00.00.03                          1 86.65000
## X2000.01.01.00.00.22                          9 86.65000
## X2000.01.01.00.00.25                         11 86.65000
## X2000.01.01.00.00.25.1                       12 86.65000
## X2000.01.01.00.00.27                         13 86.60000
## X2000.01.01.00.00.27.1                       14 86.65000
```

```r
num_buy_side <- length(matrix(buy_side))

num_buy_side
```

```
## [1] 305231
```

```r
num_sell_side <- length(trade_direction) - length(matrix(buy_side))

num_sell_side
```

```
## [1] 270502
```

```r
num_of_trades <- cbind(num_buy_side, num_sell_side)

num_of_trades
```

```
##      num_buy_side num_sell_side
## [1,]       305231        270502
```

```r
# EHO_out <- EHO(as.numeric(buy_trades_df$Price), as.numeric(sell_trades_df$Price))
#
```

```r
# model <- optim(initial_param, EHO_out, gr = NULL, method = c("Nelder-Mead"), hessian = FALSE)

pin_likelihood <- function(params)
{
    epsilon <- params[1]
    mu <- params[2]
    alpha <- params[3]
    delta <- params[4]

    lambda_b <- alpha * mu + epsilon
    lambda_s <- (1 - alpha) * mu + epsilon

    # Calculating the Poisson probabilities
    L_b <- dpois(num_of_trades[,1], lambda_b)
    L_s <- dpois(num_of_trades[,2], lambda_s)

    # To avoid log(0), we add a small number
    L <- L_b * L_s + 1e-10

    # Negative log-likelihood
    result <- -sum(log(L))

    return(result)
}

initial_params <- c(0.5, 0.5, 0.5, 0.5)

result <- optim(initial_params, pin_likelihood, method = "L-BFGS-B",
                lower = c(0,0,0,0), upper = c(1,1,Inf,Inf))

optim_params <- result$par
alpha_hat <- optim_params[3]
mu_hat <- optim_params[2]
epsilon_hat <- optim_params[1]

pin <- (alpha_hat * mu_hat) / (alpha_hat * mu_hat + 2 * epsilon_hat)

pin
```

```
## [1] 0.2
```