Advanced Mathematics of Finance
Research Project Report

Computing VPIN: Bulk Classification vs. Lee-Ready

N.J. Murphy

Supervisor: Dr. T.Gebbie

Programme in Advanced Mathematics of Finance,
School of Computer Science and Applied Mathematics,
University of the Witwatersrand, Johannesburg.

UNIVERSITY OF THE
WITWATERSRAND,
JOHANNESBURG

3 December 2016

Abstract

We study the Volume-Synchronized Probability of Informed Trading (VPIN) metric as proposed by Easley et al. (2012)[1] to indicate order flow toxicity, a phenomenon in high-frequency trading where market makers may be unaware they are providing liquidity at a loss due to the adverse selection problem. We take you through the derivation of the original Probability of Informed Trading (PIN) model introduced by Easley et al. (1996)[2] which forms the foundation of the VPIN metric. The VPIN metric requires the classification of trading volume as being buyer- or seller-initiated in order to calculate the Order Imbalance which provides an indication of order flow toxicity within the market. We implement two approaches to classifying trading volume, the Bulk Classification algorithm and the Lee-Ready algorithm, for stocks listed on the Johannesburg Stock Exchange (JSE) and find no resemblance between the fluctuations of VPIN over time resulting from the two approaches.

Even though there is no agreement between the fluctuations of VPIN for the two approaches, we do however find a relation between the overall levels in VPIN for the two approaches for frequently traded stocks with high market capitalisations. Furthermore, we find that infrequently traded stocks with lower market capitalisations give much lower levels of VPIN for the Bulk Classification approach than for Lee-Ready, resulting in poor performance of the Bulk Classification algorithm for such stocks. We also find a relation between the change in the log-midquote around the same time that VPIN was computed for each stock.

Keywords: Order flow toxicity, VPIN, adverse selection, PIN, Bulk Classification, Lee-Ready, Order Imbalance

Contents

List of Figures

List of Tables

1 Introduction

Since the development of high-frequency trading, there have been significant affects on market liquidity and price discovery due to asymmetric information between active traders and market makers. When the one side of the market possesses information advantages, which could relate to both the fundamental information of a stock and to other factors which affect the overall nature of trading, it becomes a detriment to the other side of the market. Thus, the provision of liquidity due to asymmetric information has been closely studied in relation to the risks faced by market makers due to their possible or even probable exposure to entering into transactions with market participants with superior information. In such markets, the liquidity providers use the bid-ask spread as a buffer to avoid being picked off by informed traders and closely control the provision of liquidity to avoid losses. There are other determinants of the spread, such as order processing costs and inventory costs, however we will focus on the risks and cost of trading with a market participant with "inside" information.

In order to measure these risks, manage liquidity provision and measure the quality of order flow in the context of Market Microstructure, two major developments have been made in the past two decades. The focus of this paper will be on the concept of order flow toxicity introduced by Easley et al. (2012)[1] which was developed to detect adverse selection in the market by calculating the buy and sell imbalances of orders arriving into the market and use these imbalances to update a metric to match trade intensity. Flow toxicity is defined by the authors to be a phenomenon in high-frequency trading where market makers may be unaware they are providing liquidity at a loss due to the adverse selection problem. Periods of higher flow toxicity will indicate that a higher instance of informed trading is taking place and subsequently a higher instance of adverse selection. Adverse selection is broadly defined as informed traders taking liquidity from uninformed traders (market makers) and Easley et al. (2012)[1] define adverse selection as the "natural tendency for passive orders to fill quickly when they should fill slowly and fill slowly (or not at all) when they should fill quickly and vice versa.". This implies that market maker's limit orders are being matched at market depths that one would not usually expect to be matched at. The adverse selection problem not only encases asymmetric information but relates to any information that induces risks which may harm liquidity provision. To get an early warning signal leading up to potentially harmful events by estimating instances of flow toxicity in the market, Easley et al. (2012)[1] introduced the Volume-Synchronized Probability of Informed Trading (VPIN) metric.

VPIN has received widespread appraisal since its introduction. The most highly acclaimed application of VPIN was its ability to anticipate the Flash Crash on May 6, 2010. VPIN recorded some of its highest levels ever recorded hours before the *flash crash* took place indicating that order flow was highly toxic in the hours leading up to the crash. The authors argue that the metric was able to anticipate the liquidity-induced crisis that resulted from the crash which lead to many market makers widening their bid-ask spread and also the withdrawal of many market makers from the market leading to a large shortage of liquidity in the market. However, VPIN has also had its fair share of criticism too. Andersen and Bondarenko [8] boldly state that VPIN is not suitable for capturing order flow toxicity since it has no incremental predictive power for future volatility. At first the authors claimed that they were not able to reproduce the results of Easley et al. (2012)[1] but after minor tweaks found that VPIN is more in line with what happened during the flash crash rather than it being able to predict the crash.

VPIN is based upon a model previously introduced by Easley, Kiefer, O'Hara and Paperman (1996)[2] used to estimate the *Probability of INformed trading* (PIN) in a market. The original PIN model aims to model trading as an interaction between market makers and traders. We discuss the details of the model in section 3. The PIN measure incorporates a Poisson process to model arrival rates of traders and on each day calculates the buy and sell initiated trade volume imbalance to indicate the probability of informed trading. In order to estimate model parameters, numerical optimization techniques are used to maximize a likelihood function.

The VPIN measure estimates the original PIN measure by calculating the buy and sell volume imbalance, which the authors term *Order Imbalance* (OI), directly from the intraday data and is updated in stochastic time rather than being updated at the close of each trading day. The implication of this is that the VPIN measure is updated to match the arrival of new information into the market and is thus better suited for high-frequency trading environments. Thus, the major contrast between the two measures is this very fact that the PIN model uses *clock-time* and the VPIN model adopts *volume-time* by calculating the order imbalance each time a fixed volume threshold has been reached. The VPIN measure has obvious computational advantages over PIN since there is no need to implement numerical optimization techniques. In order to calculate the order imbalance, the transactions or transaction volumes need to be classified as being buyer- or seller-initiated since the tick data vendors do not provide such information. Easley et al. (2012)[1] introduce an algorithm called Bulk Classification which classifies the buy and sell percentages of volume over equally sized volume increments and which allows trading volume to be classified in volume time. This is in contrast to the more popular Lee-Ready algorithm which classifies each single transaction in the data as being buyer- or seller-initiated. In section 4, we look at the methodologies for implementing both of the trade classification algorithms in order to compute VPIN. In section 5, we compare the resulting VPIN values for both approaches and find there is no agreement between the two approaches in most cases. In some instances VPIN drops to a local minimum for the one approach and rises to a local maximum in the other approach. We also study the relationships between the market capitalisations of stocks and the levels of VPIN given by the two approaches.

2 Data

2.1 Data Description

Thomson Reuters Tick History is the source of all aggregated tick data used in the project. The electronic limit order book data corresponds to stocks listed on the top 40 of the JSE. For the purpose of this project, data for 16 stocks over a 6 month period from the 1^{st} of January 2013 until the 3^{rd} of June 2013 is utilised. The data provides a tick-by-tick series of real-time records of limit orders and executions which occur on the JSE over a given period.

There are three different types of messages which can be submitted to the exchange, namely-trade, quote and auction messages. The messages are organised in terms of their time-stamps and each time-stamp is accurate to microseconds. The data is stored in MAT-files, with each MAT-file containing 5 days worth of order book data. Within the MAT-file's, the data is contained in cell arrays which are useful in that they allow for string and double inputs. Once loaded into MATLAB, the table below shows the format of the data:

Column 1	RIC
Column 2	DateL
Column 3	TimeL
Column 4	DateTimeL
Column 5	Type
Column 6	Price
Column 7	Volume
Column 8	Market VWAP
Column 9	L1 Bid Price
Column 10	L1 Bid Size
Column 11	L1 Ask Price
Column 12	L1 Ask Size

Table 1: Descriptions of each column in the data once loaded into MATLAB

2.2 Data Processing

In order to access and process the order book data, we made use of a MATLAB Distributed Computing Server (MDCS) cluster which is a MATLAB based high performance computing cluster. The cluster is made up of one head node and 8 worker nodes where the head node is the pathway between the local machine and the worker nodes. The worker nodes are the computing resources which process script files which are submitted to the cluster using the MATLAB job scheduler. This requires running the MATLAB Submission Script (B.1) along with the necessary attached processing script (A.1 or A.2). The processing script will then be executed on the chosen data and can later be retrieved by running the Retrieval Script (B.2). The retrieved script will contain the processed data which can then be saved to the local machine and is saved as a MAT-file.

3 The PIN Model

The PIN model (Easley et al. (1996)[2]) is based on the assumption that the market is collectively made up of a heterogeneous population of traders where some participants have superior information to others. More specifically, the aim is to model the interaction between market makers and traders (position takers). Within each trading day $i = 1, 2, , I$, individual traders enter into the trading of stocks or securities with a market maker. On a given day, the competitive market maker will buy or sell one unit of a given stock at his/her offered bid or ask quotes which are made publically available. Before the trading day begins, nature will determine whether an information event on the given stock occurs. The information events occur with a probability of alpha and are independent of one another. An information event can either be good news or bad news which occur with probability $1-\delta$ and δ, respectively. At the end of each trading day, each participant has complete information about the value of the stock. Consequently, at the end of the day, the informed traders will have realised value of the stock. The value of the stock $\bar{V}_i$ will be conditional on good news, $\underline{V}_i$ conditioned on bad news and V_i^* on no news. It is assumed that $\underline{V}_i < V_i^* < \bar{V}_i$. Thus, the market now consists of informed traders who have the benefit of asymmetric information about the stock value and uninformed traders who have received no information. After an information event does or does not occur, traders will arrive into the market according to an independent Poisson process. On a given day, orders from uninformed traders will arrive at rate ϵ. During days in which information events occur, informed traders will submit orders to buy a stock if they receive good news or otherwise sell a stock if they receive bad news, in both cases arriving into the market at rate μ. The trading process is more thoroughly explained in the trading tree shown in figure 1 below:

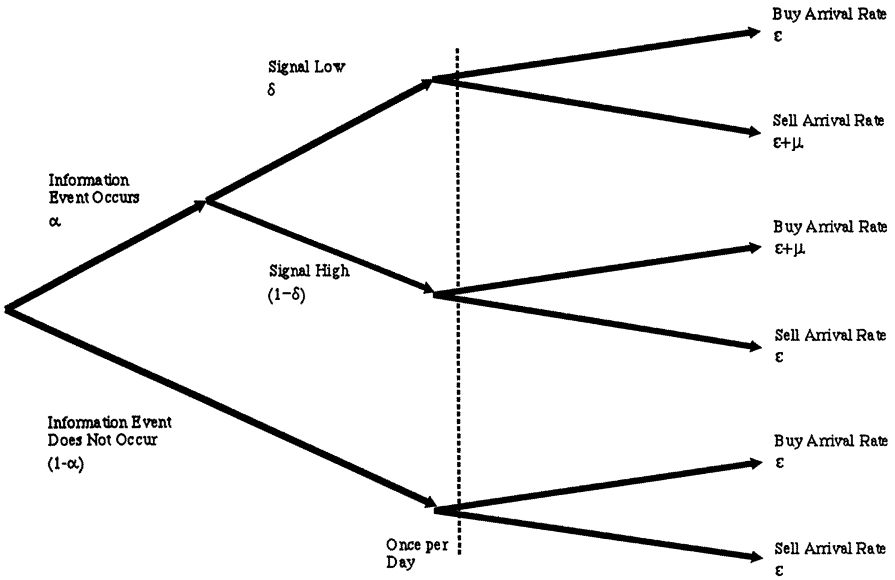


Figure 1. Tree diagram of the trading process. This figure gives the structure of the trading process, where α is the probability of an information event, δ is the probability of a low signal, μ is the rate of informed trade arrival, and ε is the rate of uninformed buy and sell trade arrivals. Nodes to the left of the dotted line occur once per day.

Figure 1: Diagram showing the trading process as explained by the PIN model. Image sourced from Easley et al. (1996)[2].

As stated previously, nature determines if an event occurs. As a consequence, it also determines whether the event is good or bad news. The 3 nodes- good news, bad news and no news which lie to the left of the dotted line will only occur once per day. Given the event for a particular day, traders arrive according to a Poisson process. On days in which good news occurs, traders arrive at a rate $\epsilon + \mu$ for buy orders and ϵ for sell orders. On days in which bad news occurs, traders arrive at rates ϵ for buy orders and $\epsilon + \mu$ for sell orders. On days in which no information event occurs, only uninformed traders are present in the market with an arrival rate of ϵ for both buy and sell orders. These parameters will thus determine the probability that informed trading is taking place in a stock.

The next step is to numerically estimate the above parameters through the maximization of a likelihood function. The function relates the observable market outcomes, which are the buy and sell volumes, to the unobservable parameters in the trading process: α, δ, μ, ϵ. The market maker will then use these unobservable parameters to determine his bid and ask prices. We will not discuss the details of the likelihood function and the implementation of the maximization of the function in this paper.

Once the parameters have been estimated, the market maker is able to determine the price at which he is willing to sell stock, the ask price, and the price at which he is willing to buy stock, the bid price. The market maker does not know which events will occur over a trading day and so it is assumed the market maker is Bayesian in such a way that his beliefs are updated depending on the information received from the arrival of orders. In this model, market makers will update their beliefs based on the toxicity of order flow which is described by the model parameters. By the assumption that each day's information is independent, each day has a separate set of beliefs. Let $P(t) = (P_n(t), P_b(t), P_g(t))$ be the time t representation of the market makers belief about "good news" (g),"bad news" (b) and "no news" (n). His belief at time 0 is given by $P(0) = (1 - \alpha, \alpha\delta, \alpha(1 - \delta))$.

In order to determine the bid and ask quotes at time t, the market makers beliefs are updated

conditional on the type of order that has arrived in the market. Both quotes are determined by the expected value of the stock conditional on the history of the process before time t, captured by $P(t)$ and the fact that a trader wants to buy or sell one unit of a given stock. Let S_t and B_t denote the arrival of a sell order and buy order at time t respectively. Let $P(t|S_t)$ denote the updated belief at time t conditional on a sell order arriving and $P(t|B_t)$ denote the updated belief at time t conditional on a buy order arriving. Given that an order to buy or sell a stock arrives at time t, by Bayes Rule for conditional probabilities, the market maker's probabilities conditioned on the type of order that arrives are given as follows:

Event	**Arrival of buy order**	**Arrival of sell order**
good news	$P_g(t\|B_t) = \frac{P_g(t)\epsilon}{\epsilon+\mu P_g(t)}$	$P_g(t\|S_t) = \frac{P_g(t)\epsilon}{\epsilon+\mu P_b(t)}$
bad news	$P_b(t\|B_t) = \frac{P_g(t)\epsilon}{\epsilon+\mu P_g(t)}$	$P_b(t\|S_t) = \frac{P_b(t)(\epsilon+\mu)}{\epsilon+\mu P_b(t)}$
no news	$P_n(t\|B_t) = \frac{P_g(t)\epsilon}{\epsilon+\mu P_g(t)}$	$P_n(t\|S_t) = \frac{P_n(t)\epsilon}{\epsilon+\mu P_b(t)}$

Table 2: Market maker's probabilities conditioned on the type of order that arrives for each given news event

Notice in each case that the probability is conditioned on the given arrival order. For the arrival of sell orders for instance, uninformed traders arrive at rate ϵ whether there is an information event or not and informed traders will sell when a bad news event occurs. Consequently, the probability of a sell order is given by $\epsilon + P_b(t)\mu$ which can be seen in the denominator of all 3 terms. A similar argument holds for the buy orders but in this case, the probability of a buy order will be the sum of the rate at which uninformed traders are arriving and the rate at which informed traders arrive into the market when a good news event occurs since informed traders will buy when they receive good news.
The *bid quote* at time t is the market maker's expected value of the stock at time t conditioned on trade history prior to time t and on the arrival of a sell order (using results in column 3 of table 2):

$$b(t) = \frac{V_i^* P_n(t)\epsilon + \underline{V}_i P_b(t)(\epsilon+\mu) + \bar{V}_i P_g(t)\epsilon}{\epsilon + \mu P_b(t)} \tag{1}$$

where $V_i^* = \delta \underline{V}_i + (1-\delta)\bar{V}_i$

Similarly, we can show that the *ask quote* at time t is the market makers expected value of the stock at time t conditioned on trade history prior to t and on the arrival of a buy order (using results in column 2 of table 2):

$$a(t) = \frac{V_i^* P_n(t)\epsilon + \underline{V}_i P_b(t)(\epsilon+\mu) + \bar{V}_i P_g(t)\epsilon}{\epsilon + \mu P_g(t)} \tag{2}$$

We can further relate these quote prices to the expected value of the stock conditional on the trade history before time t:

$$\mathbb{E}[V_i|t] = V_i^* P_n(t) + \underline{V}_i P_b(t) + \bar{V}_i P_g(t)$$

Which can be substituted into the above bid and ask quote equations, eq. (1) and eq. (2) respectively, and yield the final bid and ask quotes at time t. In the case of the bid quote, we have:

$$\begin{aligned} b(t) &= \frac{\mathbb{E}[V_i|t]\epsilon + \underline{V}_i P_b(t)\mu}{\epsilon + \mu P_b(t)} \\ &= \frac{\mathbb{E}[V_i|t]\epsilon + \mathbb{E}[V_i|t]\mu P_b(t) - \mathbb{E}[V_i|t]\mu P_b(t) + \underline{V}_i P_b(t)\mu}{\epsilon + \mu P_b(t)} \\ &= \frac{\mathbb{E}[V_i|t](\epsilon + \mu P_b(t)) - \mathbb{E}[V_i|t]\mu P_b(t) + \underline{V}_i P_b(t)\mu}{\epsilon + \mu P_b(t)} \end{aligned}$$

Hence, the bid quote is given by:

$$b(t) = \mathbb{E}[V_i|t] - \frac{\mu P_b(t)}{\epsilon + \mu P_b(t)}(\mathbb{E}[V_i|t] - \underline{V}_i) \tag{3}$$

Similarly, we can show that the ask quote is:

$$a(t) = \mathbb{E}[V_i|t] + \frac{\mu P_g(t)}{\epsilon + \mu P_g(t)}(\bar{V}_i - \mathbb{E}[V_i|t]) \tag{4}$$

It is clear by these equations that the presence of informed and uninformed traders plays a significant role in determining the quote prices at which the market maker must buy or sell his inventory.

In order for the market maker to protect himself from losses due to being adversely selected by informed traders, he will set a spread. In this model we assume the spread is present due to adverse selection but in real-world markets there are other determinants of the spread. This spread is known as the bid-ask spread and it has a major impact on market liquidity as discussed in section 1.
Denote the bid-ask spread at time t by $\Sigma(t)$ where:

$$\begin{aligned} \Sigma(t) &= a(t) - b(t) \\ &= \frac{\mu P_g(t)}{\epsilon + \mu P_g(t)}(\bar{V}_i - \mathbb{E}[V_i|t]) + \frac{\mu P_b(t)}{\epsilon + \mu P_b(t)}(\mathbb{E}[V_i|t] - \underline{V}_i) \end{aligned}$$

Since the bid and ask quotes in eq. (3) and eq. (4) respectively, are only affected by asymmetric information in the market then clearly the bid-ask spread must only be affected by asymmetric information. The 1st term in the spread is the probability that a buy order occurs in response to an information-revealing event and the 2^{nd} is the probability that a sell order occurs in response to an information-revealing event. Observing the bid-ask spread when trading begins at time zero is particularly important as the market maker needs to decide on a price for the inventory before the commencement of the trading day. At time zero, there is equal probability that a good or bad news event will occur and so we set $\delta = 1 - \delta = \frac{1}{2}$. We also know that at time zero, the probability

of bad news is $P_b(0) = \alpha\delta$ and hence $P_g(0) = \alpha\delta$. The bid-ask spread becomes:

$$\begin{aligned}
\Sigma(0) &= \frac{\mu\alpha\delta}{\epsilon + \mu\alpha\delta}(\bar{V}_i - \mathbb{E}[V_i|t]) + \frac{\mu\alpha\delta}{\epsilon + \mu\alpha\delta}(\mathbb{E}[V_i|t] - \underline{V}_i) \\
&= \frac{\alpha\mu\delta}{\epsilon + \alpha\mu\delta}(\bar{V}_i - \underline{V}_i) \\
&= \frac{\alpha\mu}{2\left(\epsilon + \alpha\mu\left(\frac{1}{2}\right)\right)}(\bar{V}_i - \underline{V}_i) \qquad \left(\delta = \frac{1}{2}\right) \\
&= \frac{\alpha\mu}{2\epsilon + \alpha\mu}(\bar{V}_i - \underline{V}_i)
\end{aligned} \tag{5}$$

The coefficient is the key component of the PIN model and is the probability that the opening order for a particular day arises from an informed trader. This is termed the *Probability of INformed trading* (PIN) and is given by:

$$PIN = \frac{\alpha\mu}{\alpha\mu + 2\epsilon} \tag{6}$$

Where $\alpha\mu + 2\epsilon$ is the rate at which all orders arrive into the market and $\alpha\mu$ is the arrival of information-based orders. We will now discuss the Volume Synchronized Probability of Informed Trading (VPIN) metric as a high-frequency estimate of PIN.

4 VPIN

The VPIN metric was introduced by Easley, de Prado and O'Hara (2012)[1] as an estimate of PIN which is more apt to high-frequency trading environments. The major modification made by the VPIN procedure to the PIN model is that VPIN is updated in volume-time while PIN is updated in clock-time. Easley et al. (2012)[1] argue that volume is a proxy for the arrival of new information into the market and consequently VPIN is updated over volume increments to match the arriving information into the market. This is in contrast to PIN which is updated on a daily basis.

The significant development that is made by the VPIN metric on the PIN model is that VPIN estimates flow toxicity analytically by approximating the numerator and the denominator of the PIN measure (eq. (6)) so that we can calculate VPIN directly from the data at a rate which matches the arrival of trade volume without use of numerical optimization. Another key development is the broader definition of information that underlies VPIN. Abad and Yagüe (2012) [3] compare the information underlying the PIN model to that of the VPIN model. They state that the PIN model focuses on fundamental information about the value of the stock with a certain probability and in response to this, informed traders become more active on one side of the market depending on whether the information was good (buy) or bad (sell) which unbalances trade activity. Whereas, VPIN measures order flow toxicity which is a wider concept focusing on the likelihood of market makers being adversely selected, which was discussed in detail in section 1. Adverse selection may include fundamental information but also any factors which affect the nature of trading or the provision of liquidity in the market. These factors result from any information which may cause unbalanced or increased intensity in trading activity and could include any information related to asset returns or systemic effects.

The first key component which is required to update the VPIN metric in volume-time is the size of the volume increments over which VPIN is updated, called *volume buckets*. The idea is to split all trade data into equally-sized volume buckets thus dividing trading sessions into samples which will contain comparable amounts of information. The objective is to compute the *Order Imbalance (OI)* over a fixed number of volume buckets, however this requires classification of the

trading volume as being buyer- or seller-initiated. Since data providers do not supply information on whether the messages sent to an exchange were triggered by buy or sell transactions, researchers have devised algorithms that determine if a given transaction has been initiated by the buying or the selling agent. Rather than using the well-known Tick-rule or the Lee-Ready trade classification algorithm, Easley et al. (2012)[1] propose a new classification algorithm, termed *Bulk Classification*, in which trades over short time intervals are aggregated and the standardized price change between the beginning and the end of the time interval is computed in order to assign a percentage of buy and sell volume to the time interval. In this paper, we will explore both the Bulk Classification and the Lee-Ready algorithm to classify trades, and discuss the details and methodologies of the corresponding methods. The original VPIN study (2012)[1] utilised aggregated data over small time intervals such as that provided by Bloomberg. The time interval the study uses is 1-minute. Since we only have access to tick-by-tick data, we are required to sort the data into 1-minute *time bars*. VPIN is then calculated for a particular period of the day, say τ, as measured in volume-time by computing the expected order imbalance over that period of events. This gives us the VPIN metric which we will now derive from the PIN model.

4.1 The link between PIN and VPIN

The PIN measure is estimated for each trading session (day) and analogously, to estimate VPIN over a trading session (as measured in volume-time), we are required to compute the order imbalance over a sample of a specified amount of volume buckets which make up the session. This requires choosing the number of buckets for which each VPIN will be calculated, termed the *sample size* and denoted by n, and the amount of trading volume needed to fill each bucket which will be kept fixed throughout the calculation and is called the *Volume Bucket Size (VBS)*. Each volume bucket is then split further into time bars. Easley et al. (2012)[1] denote the buy and sell volume of each bucket V_τ^B and V_τ^S, respectively. Then, given that we are aiming to compute the order imbalance over a sample of buckets (n) to estimate flow toxicity to match the arrival of trades, define the buy volumes [1] over each bucket as follows:

$$V_\tau^B = \sum_{i=t(\tau-1)+1}^{t(\tau)} V_i \cdot Z\Big(\frac{P_i - P_{i-1}}{\sigma_{\triangle P}}\Big) \tag{7}$$

Since each bucket has fixed size VBS, the τ^{th} bucket then has sell volume [1]:

$$V_\tau^S = VBS - V_\tau^B = \sum_{i=t(\tau-1)+1}^{t(\tau)} V_i \cdot \Big[1 - Z\Big(\frac{P_i - P_{i-1}}{\sigma_{\triangle P}}\Big)\Big] \tag{8}$$

Thus, in the case of buys (eq. (7)), the buy volume for the τ^{th} bucket is given by the sum of the volumes of all of the time bars in the bucket where $t(\tau-1)$ is the index of the last time bar in the $(\tau-1)^{th}$ bucket and $t(\tau)$ is the index of the last time bar in the τ^{th} bucket. Z is the cumulative normal distribution function where the normal distribution of the standardized price change is given by $Z\Big(\frac{P_i - P_{i-1}}{\sigma_{\triangle P}}\Big)$ where $P_i - P_{i-1}$ represents the price change between the last trade in the current time bar, i, and the last trade in the previous time bar, $i-1$. $\sigma_{\triangle P}$ is the standard deviation of price changes over all n buckets required to compute VPIN for one sample. With regard to PIN, we were interested in the ratio of informed traders ($\alpha\mu$) to that of all traders active in the market ($\alpha\mu + 2\epsilon$). Here, we aim to create analytical approximates to these 2 quantities for the VPIN calculation. As for the denominator of the VPIN metric, since each bucket is of fixed size and we have a sample of n buckets representing a trading session, the approximation of the denominator of PIN ($\alpha\mu + 2\epsilon$), is the expected value of the total volume traded over the sample of

n buckets:

$$\begin{aligned}\mathbb{E}[V_\tau^B + V_\tau^S] &= \frac{1}{n}\sum_{\tau=1}^{n} V_\tau^B + V_\tau^S \\ &= \frac{1}{n}\sum_{\tau=1}^{n} VBS \\ &= \frac{1}{n}(n * VBS) = VBS \end{aligned} \tag{9}$$

As for the numerator, since we are looking at approximating the probability of informed traders in the market, the order imbalance is a good approximate. If a larger volume of orders is arriving onto the one side of the market (for instance, the sell side) than the other side of the market (buy side), this indicates that informed traders may have superior information and are thus exploiting this advantage by submitting more orders given the infomation they possess. Hence, a reasonable approximate for this imbalance between buy and sell volumes over the trading session would be the expected absolute value of the order imbalance over the sample of n buckets:

$$\mathbb{E}[|V_\tau^S - V_\tau^B|] = \frac{1}{n}\sum_{\tau=1}^{n} OI_\tau \tag{10}$$

where $OI_\tau = |V_\tau^S - V_\tau^B|$

Easley, Engle, O'Hara and Wu (2008)[4] prove that for each period the expected order imbalance is $\mathbb{E}[|V_\tau^S - V_\tau^B|] \approx \alpha\mu$ and above we proved that the expected total volume traded is $E[V_\tau^B + V_\tau^S] \approx \alpha\mu + 2\epsilon$.

Hence, the VPIN metric is given by:

$$\begin{aligned} VPIN &= \frac{\mathbb{E}[|V_\tau^S - V_\tau^B|]}{\mathbb{E}[V_\tau^B + V_\tau^S]} \\ &= \frac{\frac{1}{n}\sum_{\tau=1}^{n} OI_\tau}{VBS} \qquad \text{(by (9) and (10))} \\ &= \frac{\sum_{\tau=1}^{n} OI_\tau}{n * VBS} \\ &\approx \frac{\alpha\mu}{\alpha\mu + 2\epsilon} \end{aligned} \tag{11}$$

Simply stated, VPIN is the average of all order imbalances in the sample length (n) which is computed by summing up the order imbalances for all the buckets in the given sample (expected trade imbalance) and dividing by the product of volume bucket size (VBS) and sample length (n) (expected total traded volume).

4.2 Process for Calculating VPIN via the Bulk Classification Algorithm

4.2.1 Time Bars

In the original study of VPIN, Easley et al. (2012)[1] make use of aggregated trade data which is already sorted into short time intervals (1-minute time bars). We make use of tick data and thus have trade and quote and trade data for each entry on the limit order book at the exact time the trade was processed by the JSE. Thus, the first major step is to aggregate the trade data into 1-minute time bars. We begin by extracting only trade entries from the data. We will only consider trades which occur in the Continuous Trading Session (09:00 - 16:50) of the JSE. Any data occurring during the Opening Auction Call Session (08:30 - 09:00) and the Closing Auction Call

Session (16:50 - 17:00) is removed. The associated algorithm for this process is shown in Table 3 below:

Algorithm 1 Extract trade data for the Continuous Trading Session

Inputs: data
Output: trade data

1: Extract trade only data
2: Create a vector containing the year, month, day, hour, minute and second of each entry in the trade data
3: Convert all entries of the vector to serial date numbers of the date and time
4: **Romove rows** $\rightarrow$ hour < 9 **AND** (hour=16 & minute>50)

Table 3: Pseudo code for extracting data corresponding to the Continuous Trading Session

Once we have all the intraday trade data for the entire period of the 6 months we are considering, we create a continuous block of trade data where each entry contains the Reuters Instrument Code (RIC), the time and date of the trade, the price at which it was executed and the amount of volume traded. We then construct a vector of time bars for each day, where the total number of days will be the same as that in the block of trade data where each day will have the first time bar at 9:00 when the continuous trading session starts, and the final time bar will be complteted at 16:50 when the continuous trading session ends. Once all the time bars have been created, we sort each trade into the time bar which contains the same date on which the trade took place and bounds the time at which the trade occurred. Table 4 below gives the algorithm required for this process:

Algorithm 2 Create 1-min time bars

Inputs: trade data
Output: time bars

1: Compute the indices for the start and end of each day from trade data for entire period
2: Create vector of 1-minute times from 9:00 until 16:50 $\Rightarrow$ 470 time bars per day
3: Create array which will store time bars for each day
4: **for** $i = 1$ **do** number of days
5: **for** $j = 1$ **do** number of time bars per day
6: Insert date into column 1, beginning of time bar into column 2 and end of time bar into column 3
7: **end for**
8: **end for**
9:
10: Create a vector containing the year, month, day, hour, minute and second of each entry in the time bars
11: Convert all entries of the vector to serial date numbers of the date and time
12:

Please see next page for continuation of algorithm

13: **for** $i = 1$ **do** number of days
14: **for** $j = 1$ **do** number of time bars per day
15: Locate indices of trade data for which: serial date number j^{th} of time bar $\leq$ serial date numbers of trade data $\leq$ serial date number of $(j+1)^{th}$ time bar
16: Capture the data for all such trades in column 4 of time bar array
17: **end for**
18: **end for**

Table 4: Pseudo code for creating 1-minute time bars and capturing all trade date for each corresponding time bar

The objective is to assign the total volume in each time bar a buy volume and a sell volume and so we first require price shifts between consecutive bars in order to implement eq. (7) and eq. (8). As discussed previously, in order to accomplish this, we compute the difference between the price of the last transaction which occurred in the current time bar and the price of the most recent transaction in the preceding time bar. We acknowledge that this may not be the optimal approach since prices of trades from the beginning of the time bar towards the middle of the time bar, for instance, may have been considerably larger or smaller than the prices towards the end of the time bar and so vital information on the magnitude of prices may be ignored. We propose that a change in the Volume Weighted Average Price (VWAP) between time bars may be a more suitable approach, however we do not explore this in this paper. Now, it is important that we avoid comparing price shifts between two consecutive days as it may drastically skew results since the prices at which a stock was trading at on the close of the trading day is often dissimilar to the price at which the stock begins the trading at the following day. Once the trades have been aggregated into the relevant time bars and the price shifts have been computed, the next step is to add up the volumes of the individual trades in each time bar so that we are able to split this volume into buy and sell volumes accordingly. Table 5 below illustrates the algorithm for the above process:

Algorithm 3 Compute price shifts and total volume for time bars

Inputs: time bars
Output: time bars

1: Compute the indices for the start and end of each day from time bar data for entire period
2:
3: **for** $i = 1$ **do** number of days
4: Set price shift of first time bar of each day to zero
5: Compute total volume for first time bar of each day
6: **for** $j =$ beginning of day **do** end of day
7: Compute the difference between the price of the last trade in the $(j+1)^{th}$ time bar and the last price in the j^{th} time bar: $\triangle P_{j+1} = P(j+1) - P(j)$
8: Compute the total volume of all trades in the j^{th} time bar: $\sum_i V_i(j+1)$
9: **end for**
10: **end for**

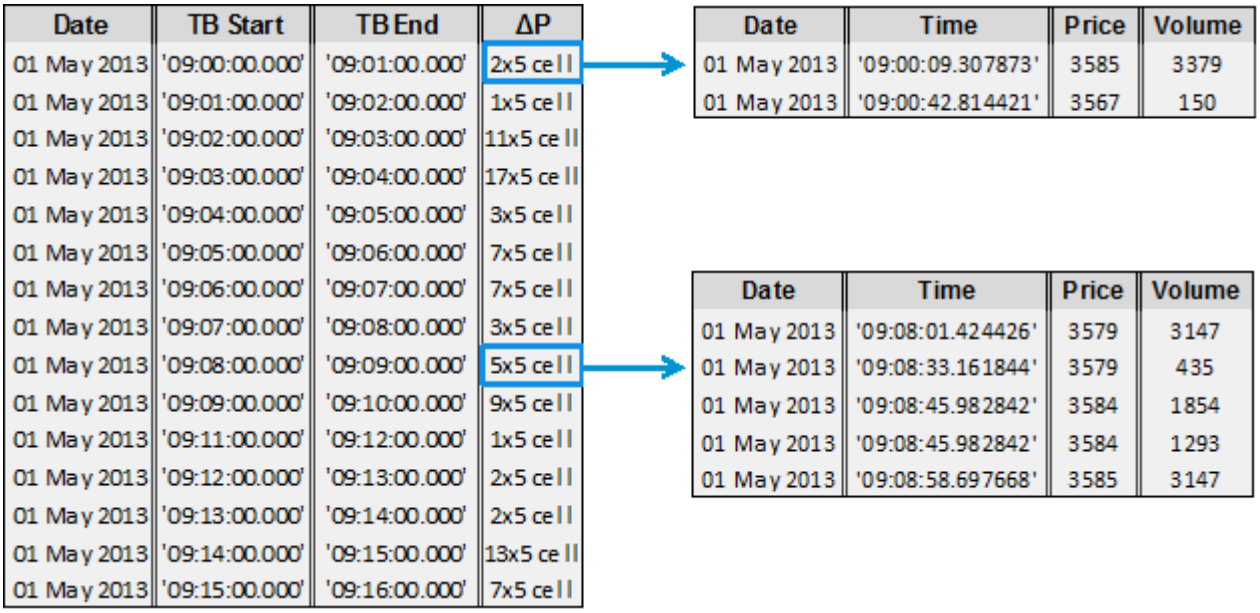
Table 5: Pseudo code for computing price shifts and the total trade volume in each time bar

On the following page are figures showing an example of FirstRand Limited trade data before time bar aggregation (Figure 6) and the time bars after the trade data is aggregated into the relevant bar (Figure 7):

Date	Time	Price	Volume
01 May 2013	'09:00:09.307873'	3585	3379
01 May 2013	'09:00:42.814421'	3567	150
01 May 2013	'09:01:29.592092'	3568	2484
01 May 2013	'09:02:27.820628'	3576	1397
01 May 2013	'09:02:27.850639'	3576	2235
01 May 2013	'09:02:27.850639'	3575	369
01 May 2013	'09:02:27.900630'	3575	1397
01 May 2013	'09:02:29.870777'	3580	560
01 May 2013	'09:02:29.870777'	3580	1050
01 May 2013	'09:02:37.841371'	3582	2393
01 May 2013	'09:02:57.621858'	3583	1396
01 May 2013	'09:02:57.631870'	3583	1283
01 May 2013	'09:02:58.446967'	3583	1303
01 May 2013	'09:02:59.642017'	3583	1160
01 May 2013	'09:03:00.992130'	3577	1396
01 May 2013	'09:03:06.447611'	3577	980
01 May 2013	'09:03:06.452621'	3576	1961
01 May 2013	'09:03:08.537715'	3579	2121
01 May 2013	'09:03:08.542755'	3578	548
01 May 2013	'09:03:14.583197'	3579	2121
01 May 2013	'09:03:14.593188'	3578	1387
01 May 2013	'09:03:28.814335'	3579	2121
01 May 2013	'09:03:28.814335'	3578	3402

Date	Time	Price	Volume
01 May 2013	'09:03:28.824322'	3578	2121
01 May 2013	'09:03:29.814430'	3576	1457
01 May 2013	'09:03:34.569722'	3576	4972
01 May 2013	'09:03:34.569722'	3576	28500
01 May 2013	'09:03:35.404830'	3575	2679
01 May 2013	'09:03:45.285561'	3576	2106
01 May 2013	'09:03:45.285561'	3576	2419
01 May 2013	'09:03:59.310696'	3575	555
01 May 2013	'09:04:13.311719'	3569	1410
01 May 2013	'09:04:29.332986'	3569	1980
01 May 2013	'09:04:55.279019'	3569	1744
01 May 2013	'09:05:01.564524'	3568	1447
01 May 2013	'09:05:39.506454'	3571	2059
01 May 2013	'09:05:41.626614'	3570	2000
01 May 2013	'09:05:45.548914'	3566	3156
01 May 2013	'09:05:47.067044'	3568	234
01 May 2013	'09:05:47.797069'	3568	2347
01 May 2013	'09:05:47.917056'	3568	247
01 May 2013	'09:06:18.364460'	3571	1202
01 May 2013	'09:06:18.758476'	3572	1400
01 May 2013	'09:06:34.164696'	3576	1358
01 May 2013	'09:06:35.090856'	3575	1400
01 May 2013	09:06:42.430265'	3575	1397

Table 6: A snippet of FirstRand Limited trade data corresponding to a short period of time at the beginning of the trading day on the 1^{st} of November 2013



Date	TB Start	TB End	ΔP
01 May 2013	'09:00:00.000'	'09:01:00.000'	2x5 cell
01 May 2013	'09:01:00.000'	'09:02:00.000'	1x5 cell
01 May 2013	'09:02:00.000'	'09:03:00.000'	11x5 cell
01 May 2013	'09:03:00.000'	'09:04:00.000'	17x5 cell
01 May 2013	'09:04:00.000'	'09:05:00.000'	3x5 cell
01 May 2013	'09:05:00.000'	'09:06:00.000'	7x5 cell
01 May 2013	'09:06:00.000'	'09:07:00.000'	7x5 cell
01 May 2013	'09:07:00.000'	'09:08:00.000'	3x5 cell
01 May 2013	'09:08:00.000'	'09:09:00.000'	5x5 cell
01 May 2013	'09:09:00.000'	'09:10:00.000'	9x5 cell
01 May 2013	'09:11:00.000'	'09:12:00.000'	1x5 cell
01 May 2013	'09:12:00.000'	'09:13:00.000'	2x5 cell
01 May 2013	'09:13:00.000'	'09:14:00.000'	2x5 cell
01 May 2013	'09:14:00.000'	'09:15:00.000'	13x5 cell
01 May 2013	'09:15:00.000'	'09:16:00.000'	7x5 cell

Date	Time	Price	Volume
01 May 2013	'09:00:09.307873'	3585	3379
01 May 2013	'09:00:42.814421'	3567	150

Date	Time	Price	Volume
01 May 2013	'09:08:01.424426'	3579	3147
01 May 2013	'09:08:33.161844'	3579	435
01 May 2013	'09:08:45.982842'	3584	1854
01 May 2013	'09:08:45.982842'	3584	1293
01 May 2013	'09:08:58.697668'	3585	3147

Table 7: FirstRand Limited trade data after the trade data has been aggregated into time bars with each transaction sorted into the time bar which bounds the time at which the transaction took place

4.2.2 Volume Buckets

The next major step is to organise the time bars into equally sized volume buckets. This requires choosing the volumes bucket size (VBS) which Easley et al. (2012)[1] specify as being the average daily volume (ADV) of the given stock over the period being considered divided by the number of buckets required to complete a sample over which VPIN is calculated i.e. the sample size (n). The authors choose the sample size (n) to be 50 buckets. On a day of average volume, this would be equivalent to computing the daily VPIN. Table 8 below gives the algorithm for computing the ADV:

Algorithm 4 Compute VBS and ADV

Inputs:

- trade data
- daystart: index for the start of each day
- dayend: index for the end of each day

Output: VBS

1: **for** $i = 1$ **do** number of days
2: sum up volumes from the beginning of the i^{th} day to the end of the i^{th} day:
3: dailyvol(i) $= \sum_{daystart(i)}^{dayend(i)} V_i$
4: **end for**
5:
6: ADV $= \sum_i$ dailyvol(i)/(no. of days)
7: VBS = **round**(ADV,0)
8: Initiate the number of buckets required to compute each update of VPIN: n=50

Table 8: Pseudo code for calculating the volume bucket size (VBS) and the average daily volume (ADV)

Once 50 volume buckets have been filled completely, the VPIN metric will be computed by implementing eq. (11) and thereafter it will be updated at the completion of each bucket. For instance, when bucket 51 has been completed, the first bucket is dropped and a new VPIN is calculated based on buckets 2-51.

The idea is to iterate through all of the 1 minute time bars and sum up the volumes of the bars until the VBS threshold is reached. We then assign the current set of time bars for which the threshold was reached the relevant bucket index. If the last time bar in the given set of time bars, which make up the volume of the τ^{th}, bucket is of volume greater than that required to fill the volume bucket, the last time bar is split further into two time bars where the first bar will contain the amount of volume required fill the most recent bucket and the second bar will contain the excess volume. This results in the excess volume being transferred into the $(\tau+1)^{th}$ bucket and will contribute towards the volume calculation for that bucket. The same price change is used for the time bar which contains excess volume as was for the time bar used to complete the previous bucket. The table below illustrates the algorithm for this process:

Algorithm 5 Assign each time bar a bucket index

Inputs: time bars
Output: time bars

Set bucket counter, τ =1, and index counter, ind = 0
Set $i = 0$
while $i <$ number of time bars **do**
    locate index for time bar where a bucket will be completed
    if $\sum_{ind(\tau+1)}^{i} V_i \geq$ VBS **then**
        Compute the excess volume of the set of time bars which give a volume above that required to complete a bucket: excess = $\sum_{ind(\tau)+1}^{i} - \sum_{ind(\tau)+1}^{i-1}$
        Compute the volume in the i^{th} time bar which is of the volume needed to complete the bucket: timebar(i) = VBS - $\sum_{ind(\tau)+1}^{i-1}$
        Shift all the time bars below the i^{th} time bar down and create a new time bar which will contatin the excess volume and the same price change as the i^{th} time bar
        assign all the time bars in the current set the associated bucket counter τ: timebars((ind(tau)+1):i) = τ
    end if
    $i = i + 1$
end while

Table 9: Pseudo code for assigning each time bar the relevant bucket index

Below is an example of the data after the price changes are computed and the time bars are assigned bucket indices. This data is related to the data given in figure 6.

Date	TB Start	TB End	ΔP	TB Volume	#Bucket
01 May 2013	'09:00:00.000'	'09:01:00.000'	0	3529	1
01 May 2013	'09:01:00.000'	'09:02:00.000'	1	2484	1
01 May 2013	'09:02:00.000'	'09:03:00.000'	15	14543	1
01 May 2013	'09:03:00.000'	'09:04:00.000'	-8	60846	1
01 May 2013	'09:04:00.000'	'09:05:00.000'	-6	5134	1
01 May 2013	'09:05:00.000'	'09:06:00.000'	-1	11490	1
01 May 2013	'09:06:00.000'	'09:07:00.000'	7	8154	1
01 May 2013	'09:07:00.000'	'09:08:00.000'	4	4945	1
01 May 2013	'09:08:00.000'	'09:09:00.000'	6	9876	1
01 May 2013	'09:09:00.000'	'09:10:00.000'	-15	18939	1
01 May 2013	'09:11:00.000'	'09:12:00.000'	1	1398	1
01 May 2013	'09:12:00.000'	'09:13:00.000'	3	6413	1
01 May 2013	'09:13:00.000'	'09:14:00.000'	6	4113	1
01 May 2013	**'09:14:00.000'**	**'09:15:00.000'**	**0**	**10935**	**1**
01 May 2013	'09:14:00.000'	'09:15:00.000'	0	5590	2
01 May 2013	'09:15:00.000'	'09:16:00.000'	0	9668	2
-	-	-	-	-	-
01 May 2013	'09:39:00.000'	'09:40:00.000'	0	4252	2
01 May 2013	**'09:41:00.000'**	**'09:42:00.000'**	**-4**	**45551**	**2**
01 May 2013	'09:41:00.000'	'09:42:00.000'	-4	66324	3
01 May 2013	'09:42:00.000'	'09:43:00.000'	-2	3272	3
-	-	-	-	-	-

Table 10: An example of FirstRand Limited data after time bars are assigned bucket indices

4.2.3 Bulk Classification and VPIN Metric Calculation

Given that the data is now sorted into appropriate buckets, we are able to implement the formulas given by eq. (7) and eq. (8) in order to assign buy and sell volumes, respectively, to each bucket. Each time bar within the given sample of 50 buckets over which we are computing VPIN is assigned a buy (sell) volume by multiplying the volume of each time bar by the normal distribution of the standardised price change of the time bar as implied by eq. (7) (eq. (8)) for each time bar index $t(\tau)$ for the τ^{th} bucket. The price change is standardized by the standard deviation of all price changes in the given sample of 50 buckets. This sets all price changes in the sample to be proportional to one another and allows the normal distribution of these price changes to be comparable. If a time bar has zero price change, as is the case with a number of time bars in the sample, half of the volume is allocated to buyer-initiated volume and the other half to seller-initiated volume. Time bars with positive price changes will allocate more volume to buys which makes sense since it implies the price went up between the beginning and the end of the time bar, indicating the presence of more buyer interest in the stock. On the other hand, time bars with a negative price change will allocate more volume to sells since there were more seller-initiated transactions driving the price of the stock down. The further the price change is from zero in either direction, the greater effect it will have on the order imbalance, indicating higher instances of flow toxicity in the market.

Once each time bar has an assigned buy and sell volume, we iterate over the buckets and compute the absolute value of the order imbalance for the τ_{th} bucket ($OI_\tau = |V_\tau^S - V_\tau^B|$) at the same time as the last associated time bar in that bucket. We then sum up these imbalances for the sample of 50 buckets and obtain the order imbalance for the sample which we then divide by the product of the sample size (n=50) and the VBS. This is exactly the VPIN equation (eq. (11)). In a rolling window process, we compute the VPIN for 50 buckets at a time, upon each iteration

dropping the first bucket and adding another bucket onto the end.

Below is the algorithm illustrating the procedure we used to calculate the VPIN metric:

Algorithm 6 Calculate VPIN

Inputs: time bars

Output: VPIN

Compute indices of rows where bucket starts and where bucket ends for each bucket index τ
Compute the number of buckets over the entire period being considered
Compute the number of VPIN's for the given period: no. of VPIN's = no. of buckets - 50 + 1
Set the size of the window for which each VPIN will be calculated: samplesize = 50

for $i = 1$ **do** no. of VPIN's
    create vector of price changes which correspond to the price changes computed in time bar data
    calculate the standard deviation of the price changes: $\sigma_{\triangle P}$

    if $i = 1$ **then** (check if we are computing buy/sell volume for 1st time bar)
        index = 1
    else if $i > 1$ **then**
        index = 0
    end if

    for j = bucketstart(i)+ind **do** bucketend(i+samplesize-1)
        compute the buy volume for the j^{th} time bar: $V_j^B = V_j.\Phi\left(\frac{\triangle P_j}{\sigma_{\triangle P}}\right)$
        compute the sell volume for the j^{th} time bar: $V_j^S = V_j.\left(1 - \Phi\left(\frac{\triangle P_j}{\sigma_{\triangle P}}\right)\right)$
    end for

    for $j = i$ **do** i + samplesize - 1
        compute Order Imbalance (OI) for each bucket for i^{th} calculation of VPIN: $OI_j = \left|\sum_{bucketstart(j)}^{bucketend(j)} V_j^S - \sum_{bucketstart(j)}^{bucketend(j)} V_j^B\right|$
    end for

    VPIN(i) $= \sum_{bucketstart(i)}^{bucketend(i+samplesize-1)} OI_j$
end for

Table 11: Pseudo code for computing the buy and sell volume for each time bar, each bucket and computing VPIN in a rolling window process

An associated example relating to the data in figure 10 after each time bar has been assigned a buy and sell volume and the order imbalance for each bucket is computed is illustrated in figure 12 below:

Date	TB Start	TB End	ΔP	TB Volume	#Bucket	V_{Buy}	V_{Sell}	OI
01 May 2013	'09:00:00.000'	'09:01:00.000'	0	3529	1	-	-	-
01 May 2013	'09:01:00.000'	'09:02:00.000'	1	2484	1	1532.93	951.07	-
01 May 2013	'09:02:00.000'	'09:03:00.000'	15	14543	1	14542.78	0.22	-
01 May 2013	'09:03:00.000'	'09:04:00.000'	-8	60846	1	0.00	60846.00	-
01 May 2013	'09:04:00.000'	'09:05:00.000'	-6	5134	1	3718.89	1415.11	-
01 May 2013	'09:05:00.000'	'09:06:00.000'	-1	11490	1	10706.86	783.14	-
01 May 2013	'09:06:00.000'	'09:07:00.000'	7	8154	1	8084.07	69.93	-
01 May 2013	'09:07:00.000'	'09:08:00.000'	4	4945	1	918.38	4026.62	-
01 May 2013	'09:08:00.000'	'09:09:00.000'	6	9876	1	7153.83	2722.17	-
01 May 2013	'09:09:00.000'	'09:10:00.000'	-15	18939	1	0.00	18939.00	-
01 May 2013	'09:11:00.000'	'09:12:00.000'	1	1398	1	1398.00	0.00	-
01 May 2013	'09:12:00.000'	'09:13:00.000'	3	6413	1	4645.35	1767.65	-
01 May 2013	'09:13:00.000'	'09:14:00.000'	6	4113	1	3349.14	763.86	-
01 May 2013	**'09:14:00.000'**	**'09:15:00.000'**	**0**	**10935**	**1**	**403.75**	**10531.25**	**46362.06**
01 May 2013	'09:14:00.000'	'09:15:00.000'	0	5590	2	2795.00	2795.00	-
01 May 2013	'09:15:00.000'	'09:16:00.000'	0	9668	2	4834.00	4834.00	-
01 May 2013	'09:16:00.000'	'09:17:00.000'	-3	11339	2	2036.40	9302.60	-
01 May 2013	'09:17:00.000'	'09:18:00.000'	1	1918	2	1705.59	212.41	-
01 May 2013	'09:18:00.000'	'09:19:00.000'	-1	2614	2	707.11	1906.89	-
01 May 2013	'09:19:00.000'	'09:20:00.000'	3	1000	2	889.25	110.75	-
01 May 2013	'09:20:00.000'	'09:21:00.000'	-2	12516	2	791.40	11724.60	-
01 May 2013	'09:21:00.000'	'09:22:00.000'	1	4100	2	3363.67	736.33	-
01 May 2013	'09:23:00.000'	'09:24:00.000'	-1	3356	2	907.82	2448.18	-
01 May 2013	'09:25:00.000'	'09:26:00.000'	-3	400	2	108.20	291.80	-
01 May 2013	'09:26:00.000'	'09:27:00.000'	2	1706	2	1598.13	107.87	-
01 May 2013	'09:28:00.000'	'09:29:00.000'	-2	6073	2	672.57	5400.43	-
-	-	-	-	-	-	-	-	-
01 May 2013	'09:38:00.000'	'09:39:00.000'	3	1569	2	1516.69	52.31	-
01 May 2013	'09:39:00.000'	'09:40:00.000'	0	4252	2	763.63	3488.37	-
01 May 2013	**'09:41:00.000'**	**'09:42:00.000'**	**-4**	**45551**	**2**	**5044.67**	**40506.33**	**90578.76**
-	-	-	-	-	-	-	-	-

Table 12: An example of FirstRand Limited data after buy and sell volumes for each time bar are computed and order imbalances for each bucket have been computed

4.3 The Process for Calculating VPIN via the Lee-Ready Algorithm

The key difference between the Lee-Ready approach and the Bulk Classification approach is that we do not require time bars for the Lee-Ready apporach since we are classifying each single trade and not a bulk of trades at once. Thus, no price changes need to be calculated and no cumulative distribution quantities are needed since we can calculate the buy and sell volumes in each bucket directly from the trade data after it has been classified into buyer- and seller-initiated data. The process also differs from the Bulk Classification approach in that we now require both the trade and 'quote' data whereas in the Bulk Classification case we only needed trade data. This is because the Lee-Ready makes use a combination of the so called Quote rule and Tick rule, which we will discuss in detail below, and so quote entries are required in the implementation of the algorithm. Since we don't require the aggregation of data into time bars, we can immediately begin the trade

classification process.

4.3.1 Lee-Ready Trade Classification

Before we can implement the Lee-Ready algorithm, we need to create a block of tick data for the entire period we are considering where we will only consider trades and quotes which occur in the Continuous Trading Session (09:00 - 16:50) of the JSE. From the resulting data, we are required to calculate the midquote for each quote entry in the data which is just the average between the level 1 ask price and level 1 bid price. The order book data from Thomson Reuters does not provide both a bid and an ask update for each quote entry and so we first drop down the most recent bid (ask) quote into the cells which only have an ask (bid) update so that we are able to take an average of the bid and ask quotes for each and every quote entry. The midquote is then calculated for all quote entries. With this, we can now implement the Lee-Ready algorithm to classify each trade event or transaction.

The first step is to check whether the transaction obeys the Quote rule and if it fails, we implement the Tick rule. The Quote rule states: Transactions occurring at prices higher than the previous midquote are classified as being buyer-initiated (table 13), and transactions occurring at prices lower than the previous midquote are classified as being seller-initiated (table 14). Below are examples of the Quote Rule for both buyer- and seller-initiated transactions:

Date	Time	Type	Price	Volume	L1 Bid Price	L1 Ask Price	Midquote	Buy(1)/Sell(-1)
01-MAY-2013'	'09:03:45.285561'	'Quote'	0	0	3575	3576	3575.5	-
01-MAY-2013'	'09:03:45.285561'	'Trade'	3576	2419	99609375	0	0	1
01-MAY-2013'	'09:03:45.285561'	'Quote'	0	0	3575	3580	3577.5	-

Table 13: Implementation of Quote rule for a buyer-initiated transaction

Date	Time	Type	Price	Volume	L1 Bid Price	L1 Ask Price	Midquote	Buy(1)/Sell(-1)
01-MAY-2013'	'09:03:00.074141'	'Quote'	0	0	3577	3585	3581	-
01-MAY-2013'	'09:03:00.992130'	'Trade'	3577	1396	89990234375	0	0	-1
01-MAY-2013'	'09:03:00.992130'	'Quote'	0	0	3577	3585	3581	-

Table 14: Implementation of Quote rule for a seller-initiated transaction

If the transaction does not obey the Quote rule we move onto the Tick rule, which states: Transactions occurring at a price that equals the previous midquote but is higher than the previous transaction price are classified as being buyer-initiated (table 15) and transactions occurring at a price lower than the previous transaction price are classified as seller-initiated. If the price equals the previous midquote and there is no difference in prices between the current and previous transaction but the previous tick change was up, then the trade is classified as being buyer-initiated, and if the tick was down then it is classified as being seller-initiated. Finally, if two or more trade events happen consecutively then the first trade is classified using the Quote or Tick rule and the successive trades are classified the same as the first trade. Below is an example of the Tick rule for a buyer-initiated transaction:

Date	Time	Type	Price	Volume	L1 Bid Price	L1 Ask Price	Midquote	Buy(1)/Sell(-1)
01-MAY-2013'	'10:05:05.348064'	'Quote'	0	0	3568	3569	3568.5	-
01-MAY-2013'	'10:05:05.432999'	'Trade'	3568	1102	-	0	0	-1
01-MAY-2013'	'10:05:05.432999'	'Quote'	0	0	3567	3569	3568	-
01-MAY-2013'	'10:05:05.602982'	'Quote'	0	0	3567	3569	3568	-
01-MAY-2013'	'10:05:05.608063'	'Quote'	0	0	3568	3570	3569	-
01-MAY-2013'	'10:05:06.043063'	'Trade'	3569	900	-	0	0	1

Table 15: Implementation of Tick rule given that previous mid-quote = trade price and previous trade price < current trade price ⇒ buyer-initiated

There are instances where the Lee-Ready algorithm misclassifies trades. By misclassify, we mean, trades that were buyer-initiated but had negative price shifts and the trades that were seller-initiated but had positive price shifts. In the FirstRand sample data, 316 trades were buyer-initiated but had negative shifts and 221 were seller-initiated but had positive shifts. So roughly 4% of trades were misclassified. According to Theissen [6], the Lee-Ready algorithm classifies only 72.8% of the transactions correctly. Below is the algorithm we used for implementing the Lee-Ready algorithm:

Algorithm 7 Lee-Ready

Inputs:

- data
- uniqueday: index of each unique day
- daystart: index for the start of each day
- dayend: index for the end of each day

Output:

- data

1: **for** $i = 1$ **do** number of days
2: **for** j = Beginning of i^{th} day **do** End of i^{th} day
3: **if** j^{th} entry is a 'Trade' **then**
4: Check **Quote** rule
5: **if** $(j-1)^{th}$ entry is a 'Quote' **AND** j^{th} trade price > $(j-1)^{th}$ midquote **then**
6: Set classify column = 1 since classified as buy by Quote rule
7:
8: **else if** $(j-1)^{th}$ entry is a 'Quote' **AND** j^{th} trade price < $(j-1)^{th}$ midquote **then**
9: Set classify column = -1 since classified as sell by Quote rule
10: was set to
11: **end if**
12: Check **Tick** rule
13: **else if** $(j-1)^{th}$ entry is a 'Quote' **AND** j^{th} trade price = $(j-1)^{th}$ midquote **then**

Please see next page for continuation of algorithm

```
14:             if j^th trade price > (j − 1)^th trade price then
15:                 Set classify column = 1 since classified as buy by Tick rule
16:
17:             else if j^th trade price < (j − 1)^th trade price then
18:                 Set classify column = -1 since classified as sell by Tick rule
19:
20:             else if j^th trade price = (j − 1)^th trade price then
21:                 Set classify column to the same value that the most recent trade
22:
23:             else if (j − 1)^th entry is a 'Trade' then
24:                 if j^th trade price = (j − 1)^th trade price then
25:                     Set classify column to the same value that the most recent trade
26:                         was set to
27:                 else if j^th trade price > (j − 1)^th trade price then
28:                     Set classify column = 1 as it must be a buy
29:                 else if j^th trade price < (j − 1)^th trade price then
30:                     Set classify column = -1 as it must be a sell
31:                 end if
32:             end if
33:         end if
34:     end for
35: end for
```

Table 16: Pseudo code for the Lee-Ready Algorithm

4.3.2 Volume Buckets

Now that each trade has been classified accordingly, we can extract only the trade data. As in the Bulk Classification approach, we calculate the average daily volume for the trade data for the entire period and specify the sample size (n) over which VPIN is calculated which was chosen to be 50 buckets. The average daily volume and the associated VBS (ADV/50) will then be identical to that calculated in the Bulk Classification approach and so in both approaches we are ensured that we are calculating the VPIN over the same sequence of trade events. What will may differ is the amount of buy and sell volume in each bucket which we will now discuss.

As we did in the bulk approach, we iterate through the trades and sum up the volumes of the entries until the VBS threshold is reached. We assign the given set of trades the current bucket index. If the last trade in the given set of trades which make up the total volume required for the τ^{th} bucket is of volume greater than that required to fill the volume bucket, the last trade is broken up into two parts where one will fill the most recent bucket and the second part will contain the excess volume. Both parts will be classified the same as what they were before being split into separate parts and will have the same corresponding price. The excess volume will then contribute to the volume calculation for the next bucket.

4.3.3 VPIN Metric Calculation

Having assigned each trade the appropriate bucket index, we are now able to compute the VPIN over each sample of 50 buckets. We locate the indices for the start and end of each bucket in the entire block of data and then iterate through the block of data and update the VPIN in a rolling

window process. Given the current sample of 50 buckets, the buy and sell volumes are added up for each bucket in the sample and printed at the completion each bucket. We then compute the absolute difference between the buy and sell volume for each of the buckets in the sample. We then sum up these absolute differences between the volumes for all of the buckets which corresponds to the numerator of the VPIN metric (eq. (11)). The order imbalance is then divided by the product of the sample size ($n = 50$) and the VBS. This gives us the VPIN metric for the given sample. After iterating through the entire block of data, we are able to generate the all of VPIN's for the entire period being considered and plot the appropriate graphs which we illustrate in the next section. It is clear that the algorithm to compute VPIN via the Lee-Ready approach is very similar to that used to calculate it using the Bulk Classification approach and so we do not provide the algorithm here.

5 Results

Figures [2]-[17] show VPIN vs. time for both the Lee-Ready (orange) and the Bulk Classification (blue) approach. In this case, the graphs have been plotted in calendar time by plotting equi-spaced dates on the x-axis vs. VPIN on the y-axis. In most instances, the dates may not line up since the VPIN metric is updated in volume time however if we were to plot VPIN in volume time we would not be able to interpret the graphs and get an idea of the levels of toxicity in the market. This is because the graphs ill-behaved when plotted in volume time. In each of the figures, we have plotted the minimum and maximum VPIN to observe whether the VPIN's peak or trough at the same times for each approach. The figures also illustrate the relationship between time and the change in the log-midquote for both approaches. The midquotes are extracted at the same time that each VPIN is computed and so we take the logarithms of each of these values and take the differences between consecutive pairs of these values. Finally, the figures show the standard deviations of the price changes over each sample for which VPIN is computed ($\sigma_{\triangle P}$) vs. time. $\sigma_{\triangle P}$ correspond to the denominator in eq. (7) and eq. (8). The stocks are constituted into the three main sectors of the JSE: Financials (JSE-FINI), Resources (JSE-RESI) and Industrials (JSE-INDI) as seen below. The stocks in each sector are further arranged in descending order with regard to their market capitalisations.

5.1 Financials (JSE-FINI)

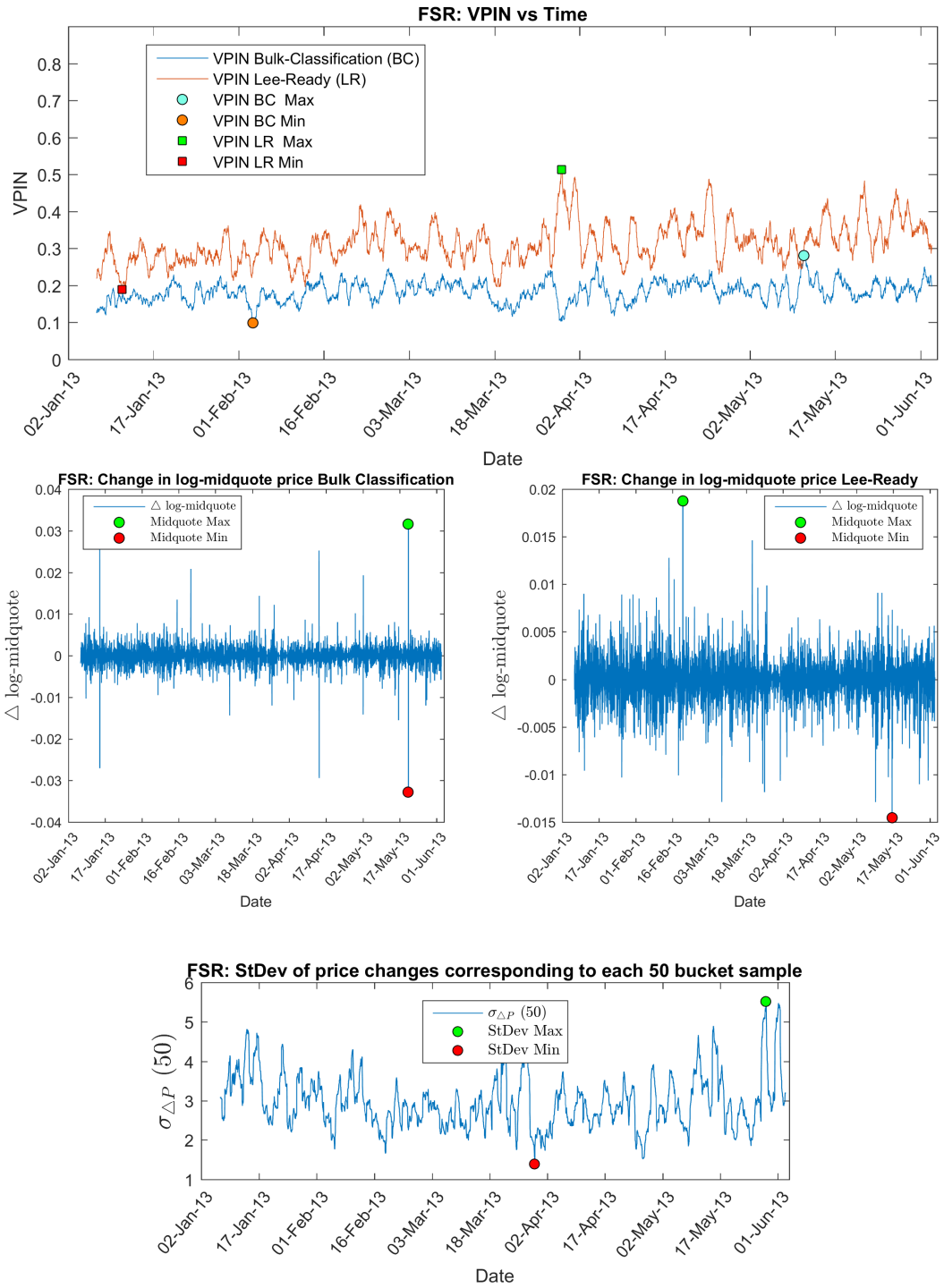


Figure 2: FirstRand Ltd (FSR) for the period 1^{st} January 2013 until 3^{rd} June 2013: The top pane shows VPIN vs calendar time for the Bulk Classification and Lee-Ready approach, the middle two plots show the corresponding change in log-midquotes and the last plot shows the standard deviation of price changes corresponding to each 50 bucket sample over which VPIN is calculated.

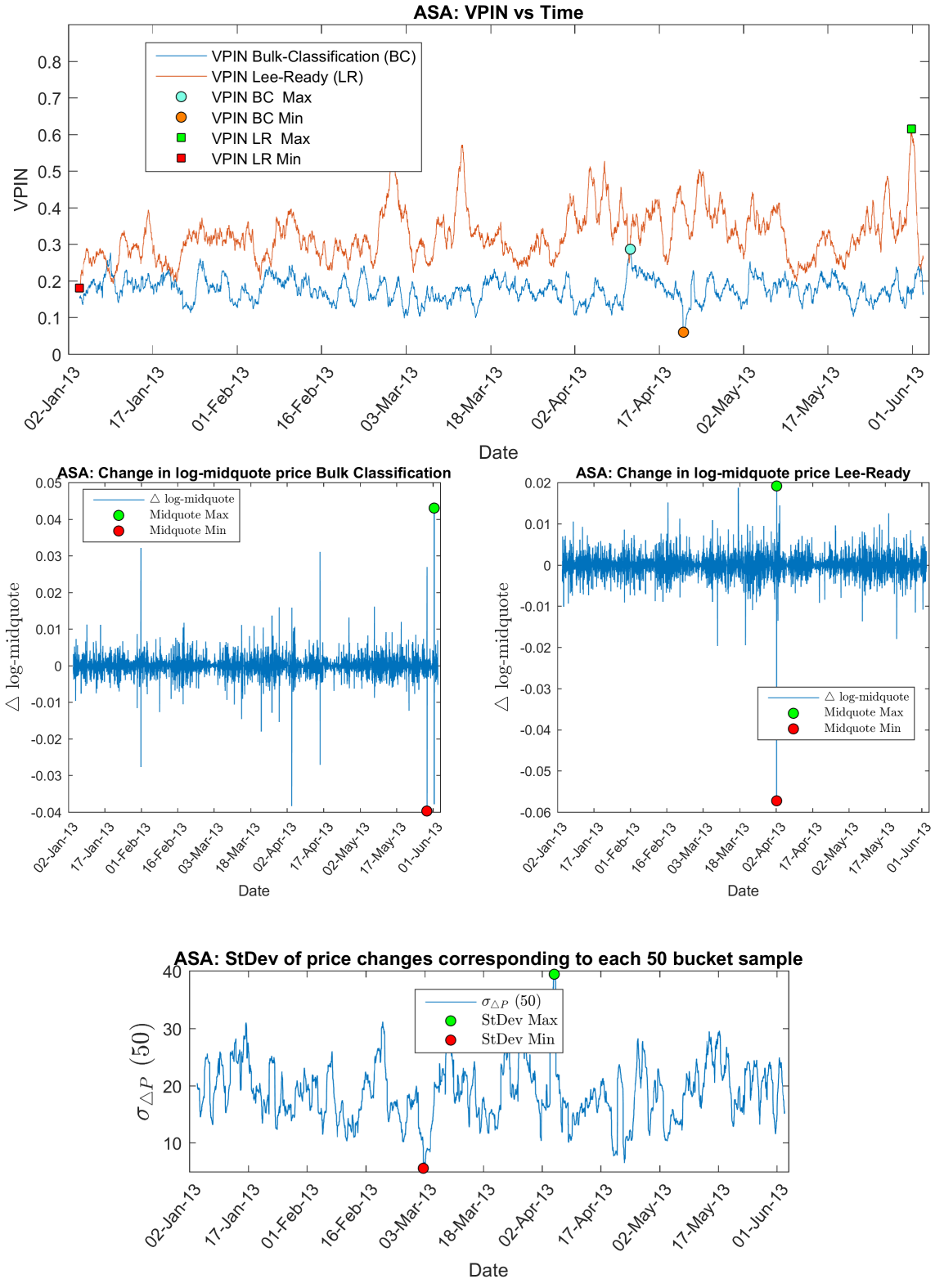


Figure 3: ABSA Group Ltd (ASA) for the period 1^{st} January 2013 until 3^{rd} June 2013: The top pane shows VPIN vs calendar time for the Bulk Classification and Lee-Ready approach, the middle two plots show the corresponding change in log-midquotes and the last plot shows the standard deviation of price changes ($\sigma_{\triangle P}$) over each 50 bucket sample.

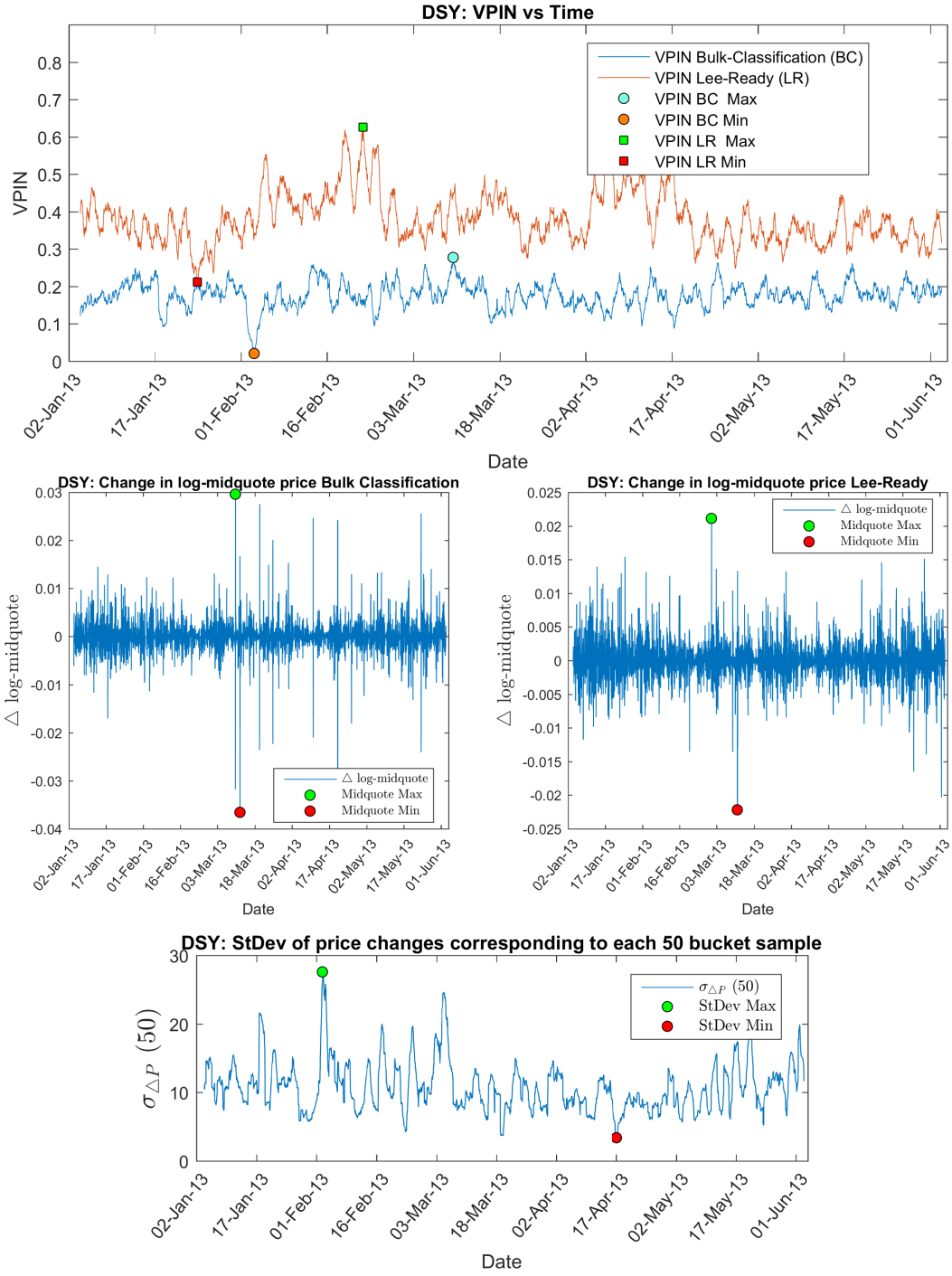


Figure 4: Discovery Ltd (DSY) for the period 1^{st} January 2013 until 3^{rd} June 2013: The top pane shows VPIN vs calendar time for the Bulk Classification and Lee-Ready approach, the middle two plots show the corresponding change in log-midquotes and the last plot shows the standard deviation of price changes ($\sigma_{\triangle P}$) over each 50 bucket sample.

An interesting observation of the Discovery Ltd figure, we can see that the change in the log-midquote for Bulk Classification (BC) obtains its minimum and maximum between the $3^{rd} - 18^{th}$ March and in the VPIN plot, we see VPIN for BC reaches its maximum around the same period. Another observation is that $\sigma_{\triangle P}$ reaches a maximum just after 1^{st} February and VPIN for BC reaches its minumum around the same date.

Another observation is that as the market capitalisation decreases, starting with FirstRand and ending with Discovery, there is an increase in the difference between the overall levels of VPIN between the Bulk Classification and Lee-Ready approach.

5.2 Resources (JSE-RESI)

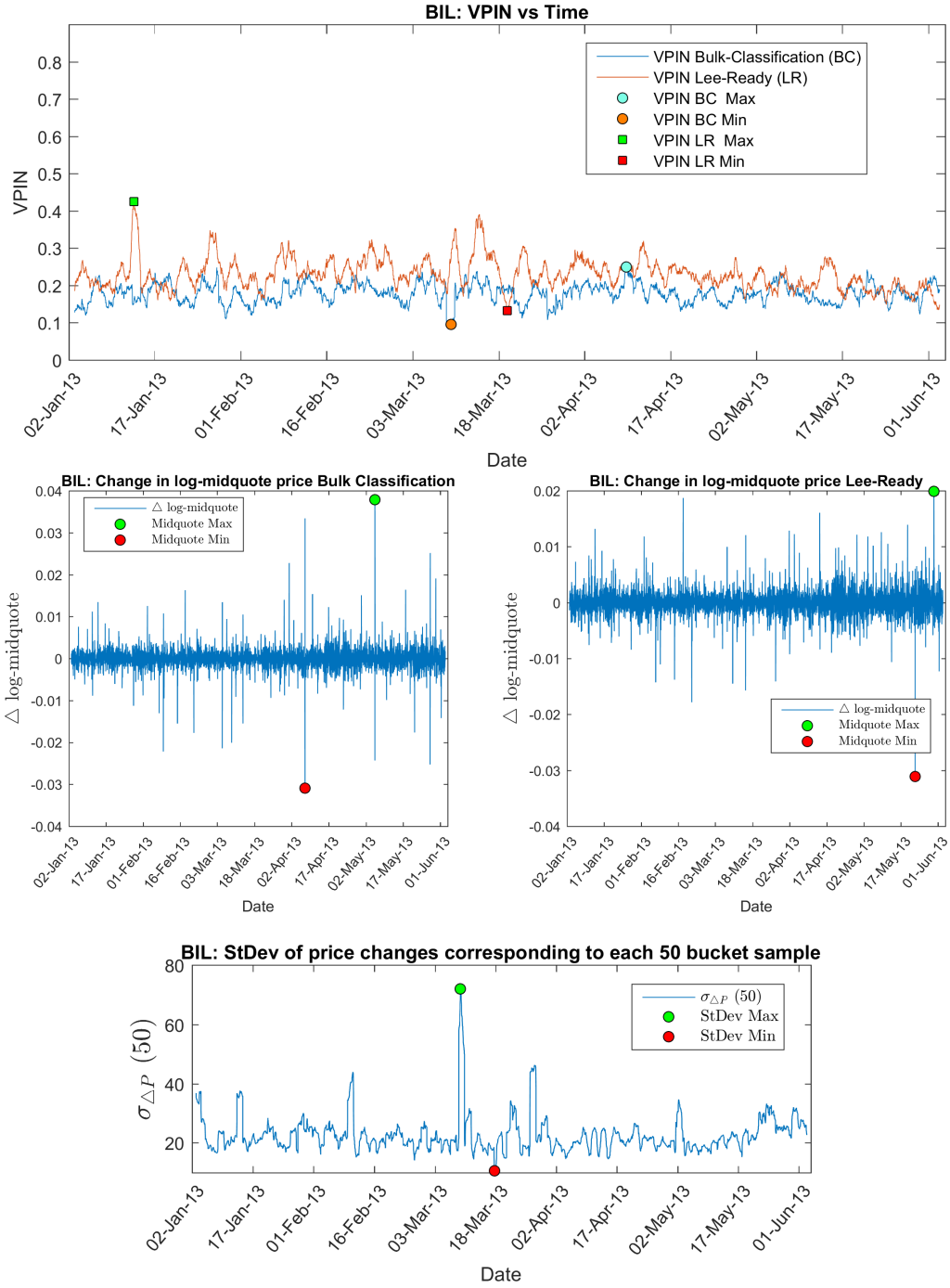


Figure 5: BHP Billiton PLC (BIL) for the period 1^{st} January 2013 until 3^{rd} June 2013: The top pane shows VPIN vs calendar time for the Bulk Classification and Lee-Ready approach, the middle two plots show the corresponding change in log-midquotes and the last plot shows the standard deviation of price changes ($\sigma_{\triangle P}$) over each 50 bucket sample.

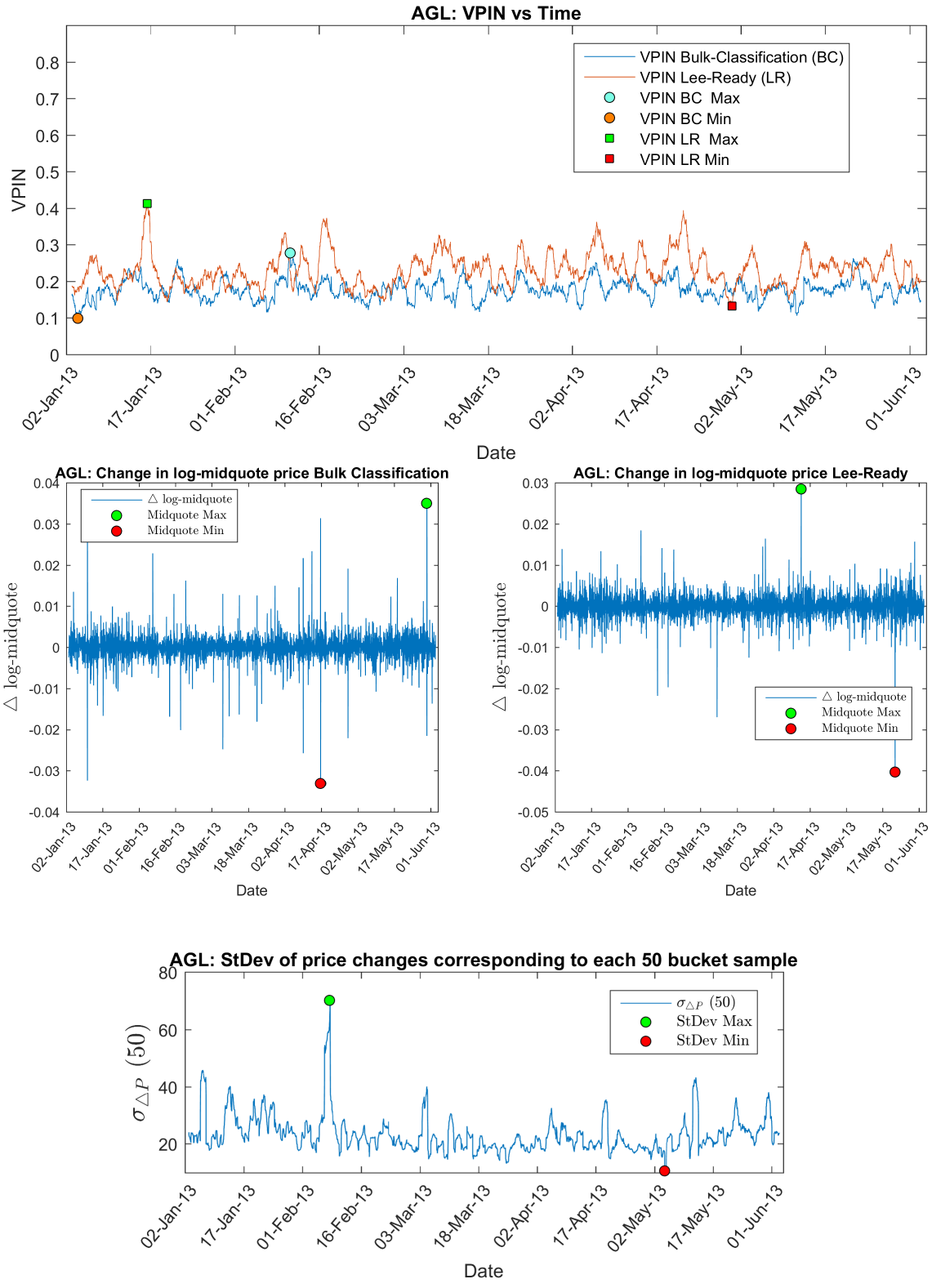


Figure 6: Anglo American PLC (AGL) for the period 1^{st} January 2013 until 3^{rd} June 2013: The top pane shows VPIN vs calendar time for the Bulk Classification and Lee-Ready approach, the middle two plots show the corresponding change in log-midquotes and the last plot shows the standard deviation of price changes ($\sigma_{\triangle P}$) over each 50 bucket sample.

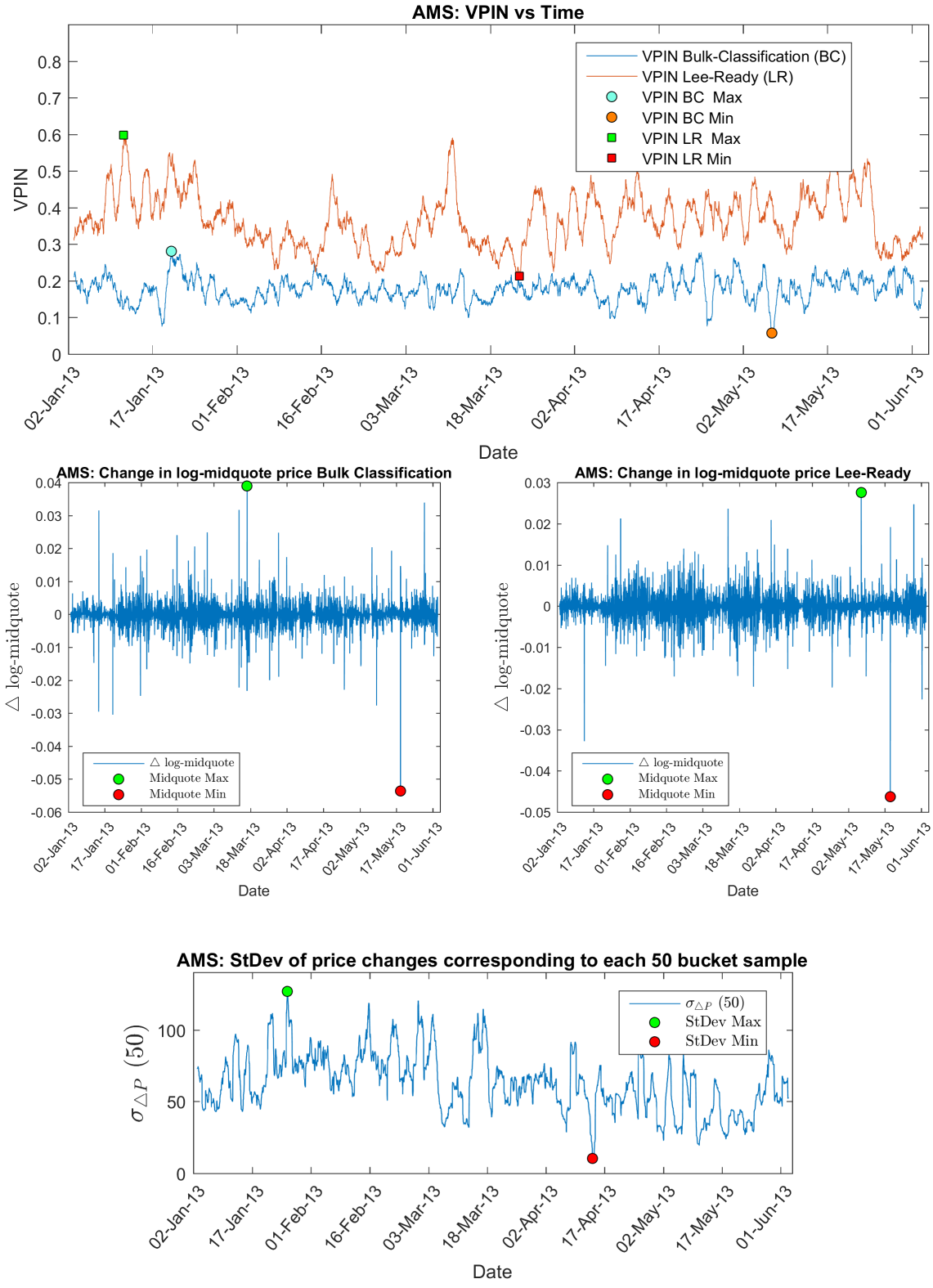


Figure 7: Anglo American Platinum Ltd (AMS) for the period 1^{st} January 2013 until 3^{rd} June 2013: The top pane shows VPIN vs calendar time for the Bulk Classification and Lee-Ready approach, the middle two plots show the corresponding change in log-midquotes and the last plot shows the standard deviation of price changes ($\sigma_{\triangle P}$) over each 50 bucket sample.

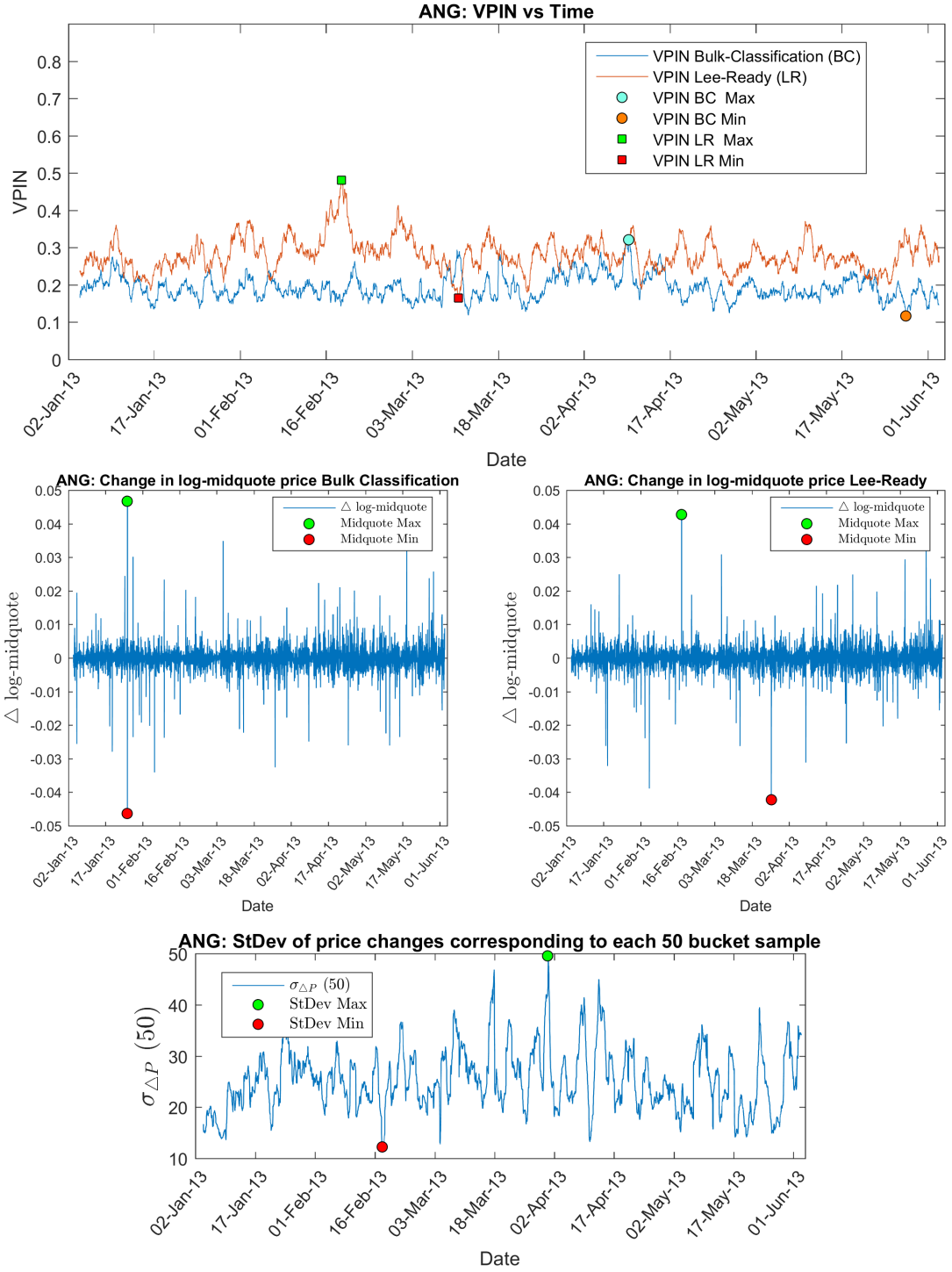


Figure 8: AngloGold Ashanti Ltd (ANG) for the period 1^{st} January 2013 until 3^{rd} June 2013: The top pane shows VPIN vs calendar time for the Bulk Classification and Lee-Ready approach, the middle two plots show the corresponding change in log-midquotes and the last plot shows the standard deviation of price changes ($\sigma_{\triangle P}$) over each 50 bucket sample.

A similar observation to that of Discovery Ltd, we see that just after the 16^{th} February the Lee-Ready (LR) VPIN reaches its maximum and the correspondong change in the log-midquote plot for LR also reaches its maximum at the same date. In addition to this, $\sigma_{\triangle P}$ reaches its minimum around the same date.

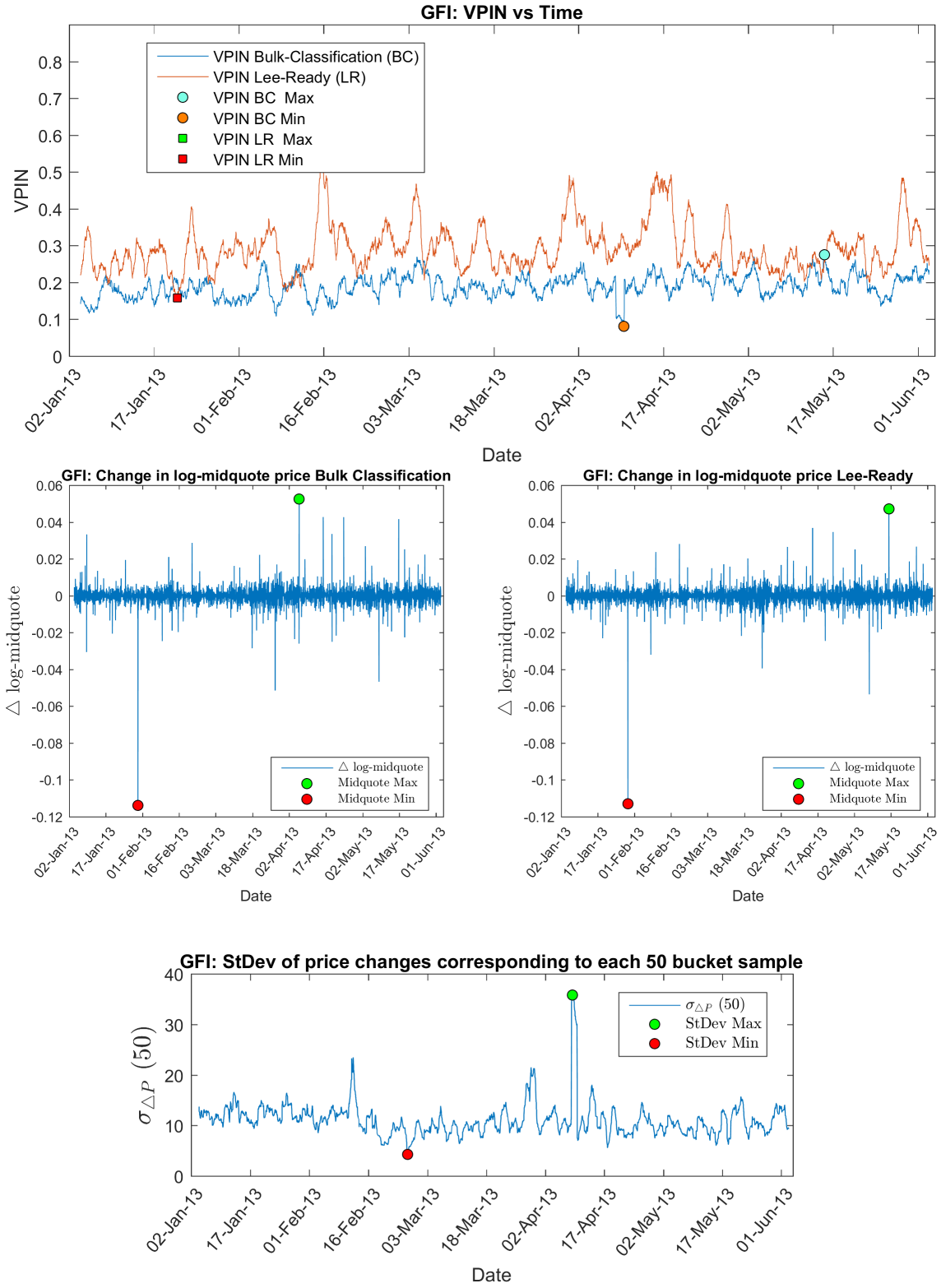


Figure 9: Gold Fields Ltd (GFI) for the period 1^{st} January 2013 until 3^{rd} June 2013: The top pane shows VPIN vs calendar time for the Bulk Classification and Lee-Ready approach, the middle two plots show the corresponding change in log-midquotes and the last plot shows the standard deviation of price changes ($\sigma_{\triangle P}$) over each 50 bucket sample.

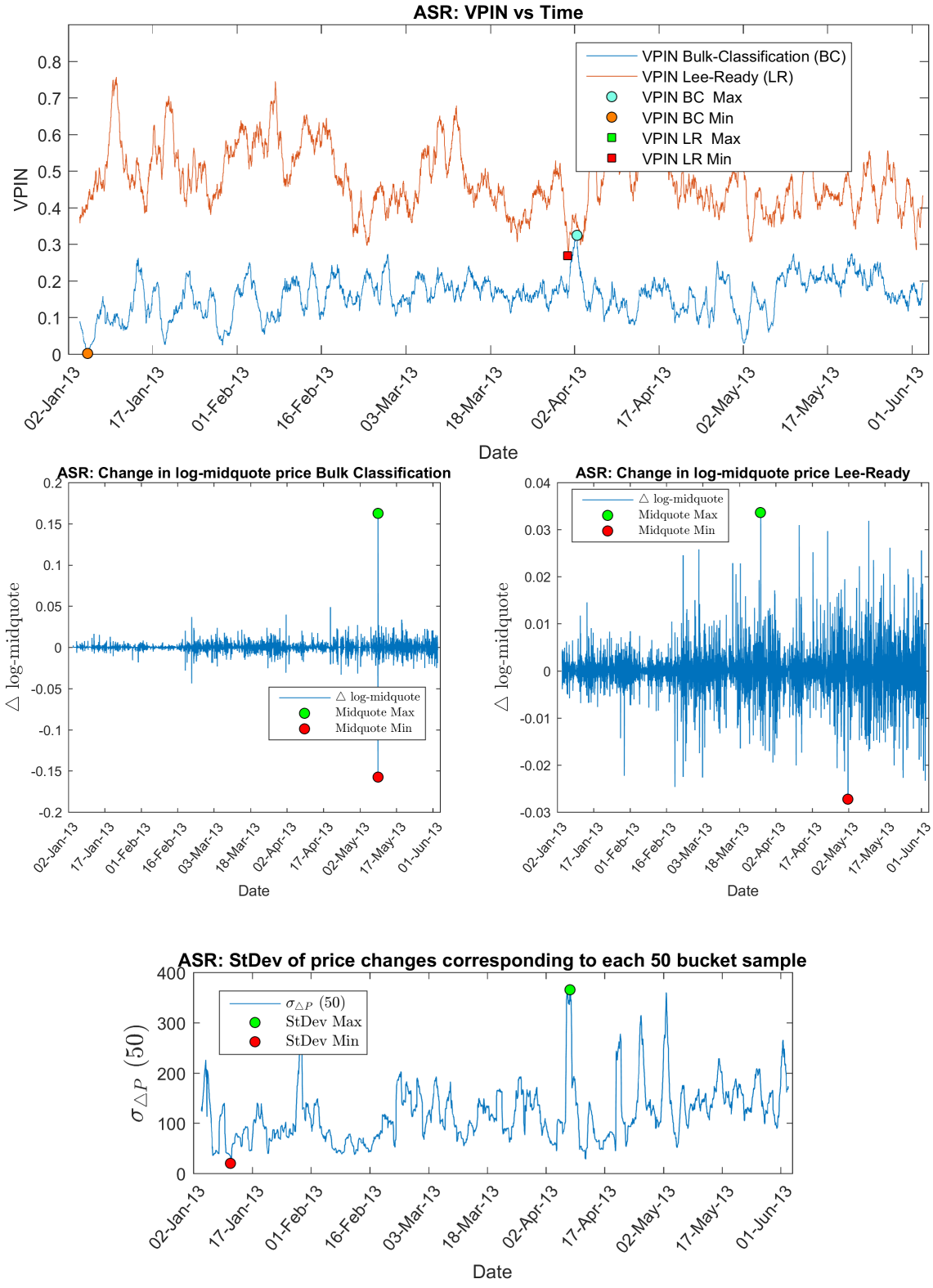


Figure 10: Assore Ltd (ASR) for the period 1^{st} January 2013 until 3^{rd} June 2013: The top pane shows VPIN vs calendar time for the Bulk Classification and Lee-Ready approach, the middle two plots show the corresponding change in log-midquotes and the last plot shows the standard deviation of price changes ($\sigma_{\triangle P}$) over each 50 bucket sample.

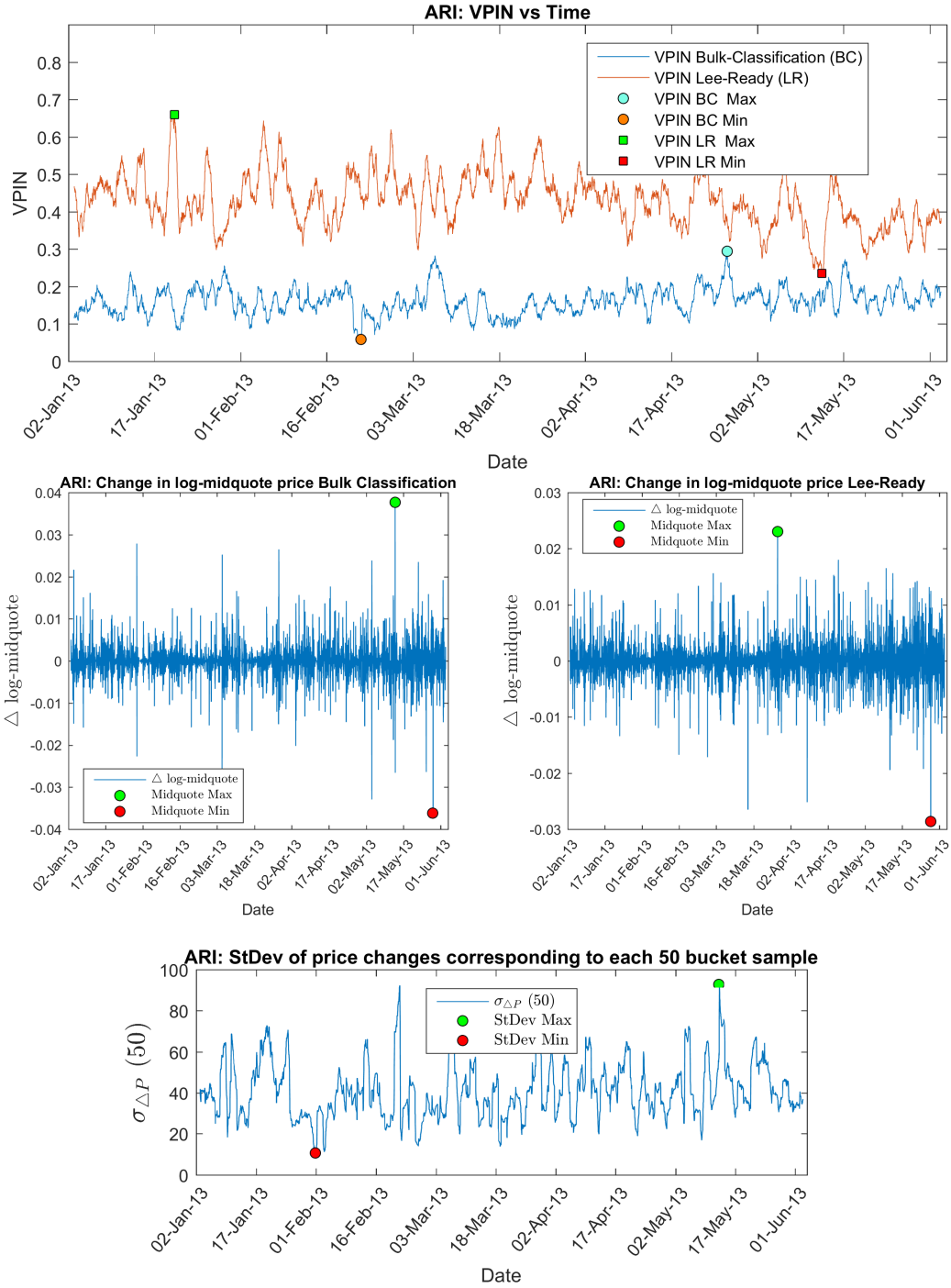


Figure 11: African Rainbow Minerals Ltd (ARI) for the period 1^{st} January 2013 until 3^{rd} June 2013: The top pane shows VPIN vs calendar time for the Bulk Classification and Lee-Ready approach, the middle two plots show the corresponding change in log-midquotes and the last plot shows the standard deviation of price changes ($\sigma_{\triangle P}$) over each 50 bucket sample.

It is clear that for the two stocks with the lowest market capitalisations in the Resources sector (ASR and ARI), the Lee-Ready VPIN's are very erratic and there are also large deviations between the overall levels of VPIN for the LR and the BC approach. This is in contrast to the much smaller deviations between LR and BC, and less erratic VPIN's as given by the two stocks highest market capitalisations (BIL and AGL).

5.3 Industrials (JSE-INDI)

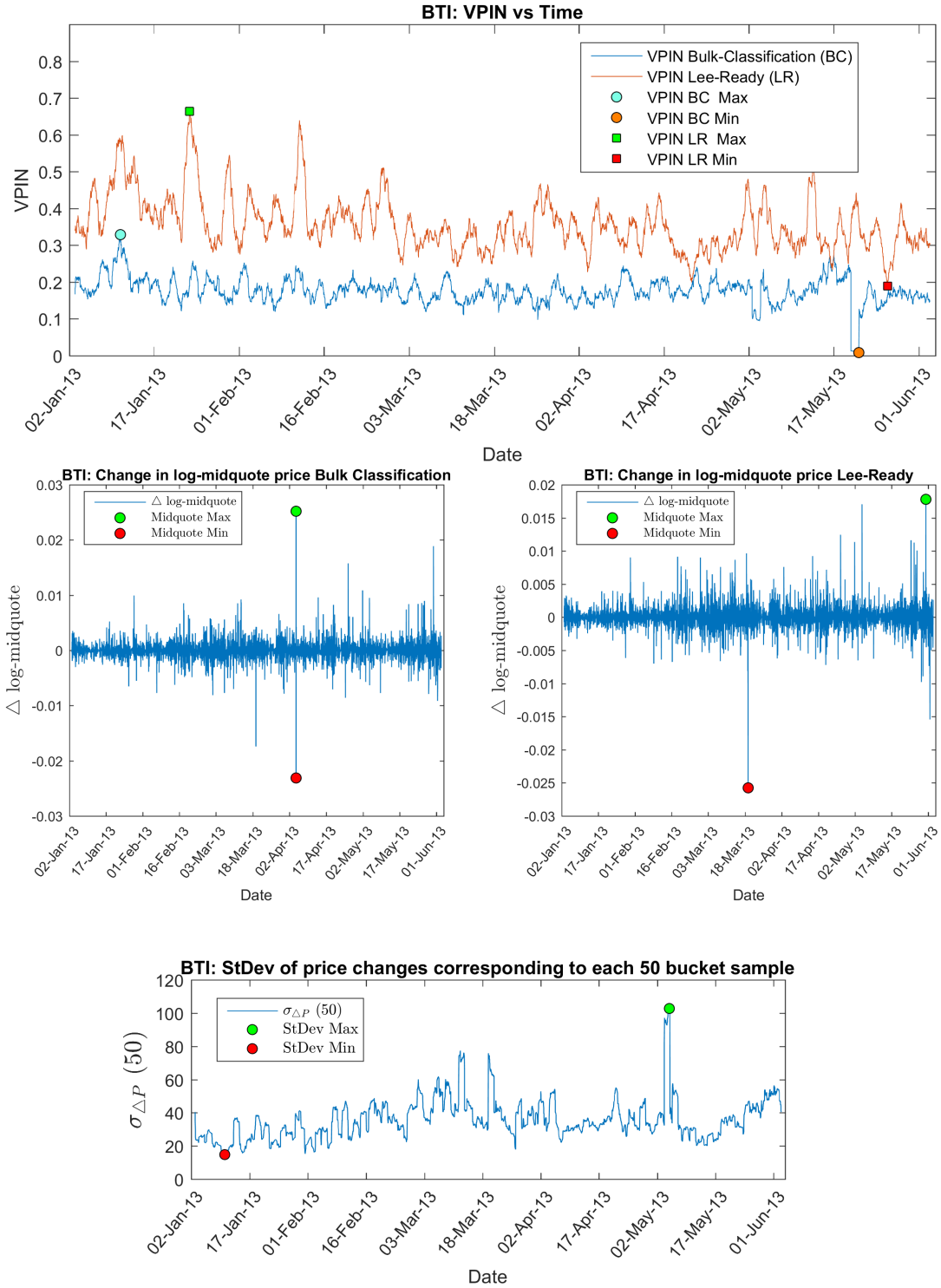


Figure 12: British American Tobacco (BTI) for the period 1^{st} January 2013 until 3^{rd} June 2013: The top pane shows VPIN vs calendar time for the Bulk Classification and Lee-Ready approach, the middle two plots show the corresponding change in log-midquotes and the last plot shows the standard deviation of price changes ($\sigma_{\triangle P}$) over each 50 bucket sample.

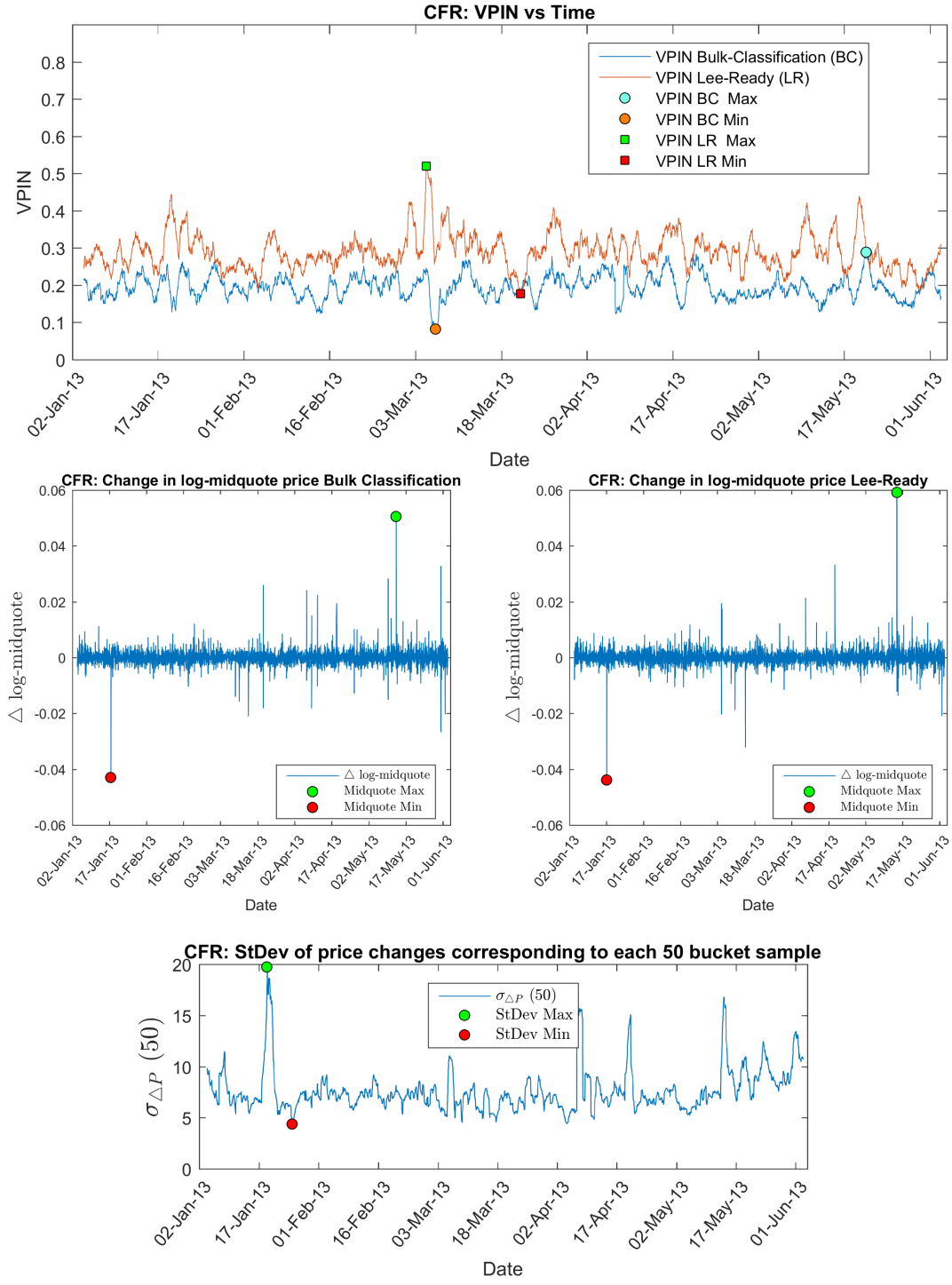


Figure 13: Compagnie Financire Richemont (CFR) for the period 1^{st} January 2013 until 3^{rd} June 2013: The top pane shows VPIN vs calendar time for the Bulk Classification and Lee-Ready approach, the middle two plots show the corresponding change in log-midquotes and the last plot shows the standard deviation of price changes ($\sigma_{\triangle P}$) over each 50 bucket sample.

In the VPIN plot for Compagnie Financire Richemont, it is interesting to observe that in a few cases, the LR VPIN has a peak when the BC VPIN has a trough, and vice versa. This can clearly be observed just after the 17^{th} of January and again around the 3^{rd} of March.

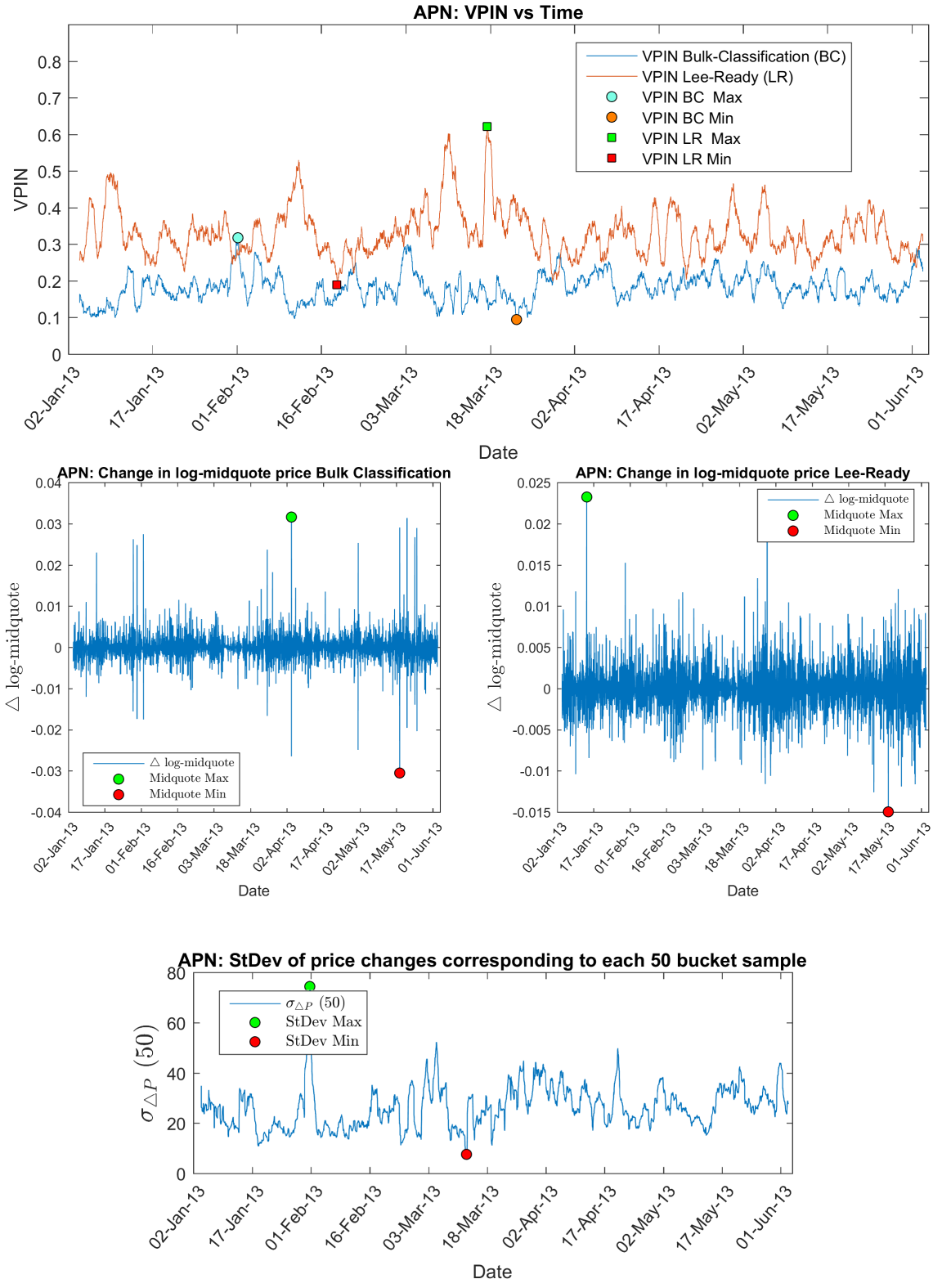


Figure 14: Aspen Pharmacare Holdings Ltd (APN) for the period 1^{st} January 2013 until 3^{rd} June 2013: The top pane shows VPIN vs calendar time for the Bulk Classification and Lee-Ready approach, the middle two plots show the corresponding change in log-midquotes and the last plot shows the standard deviation of price changes ($\sigma_{\triangle P}$) over each 50 bucket sample.

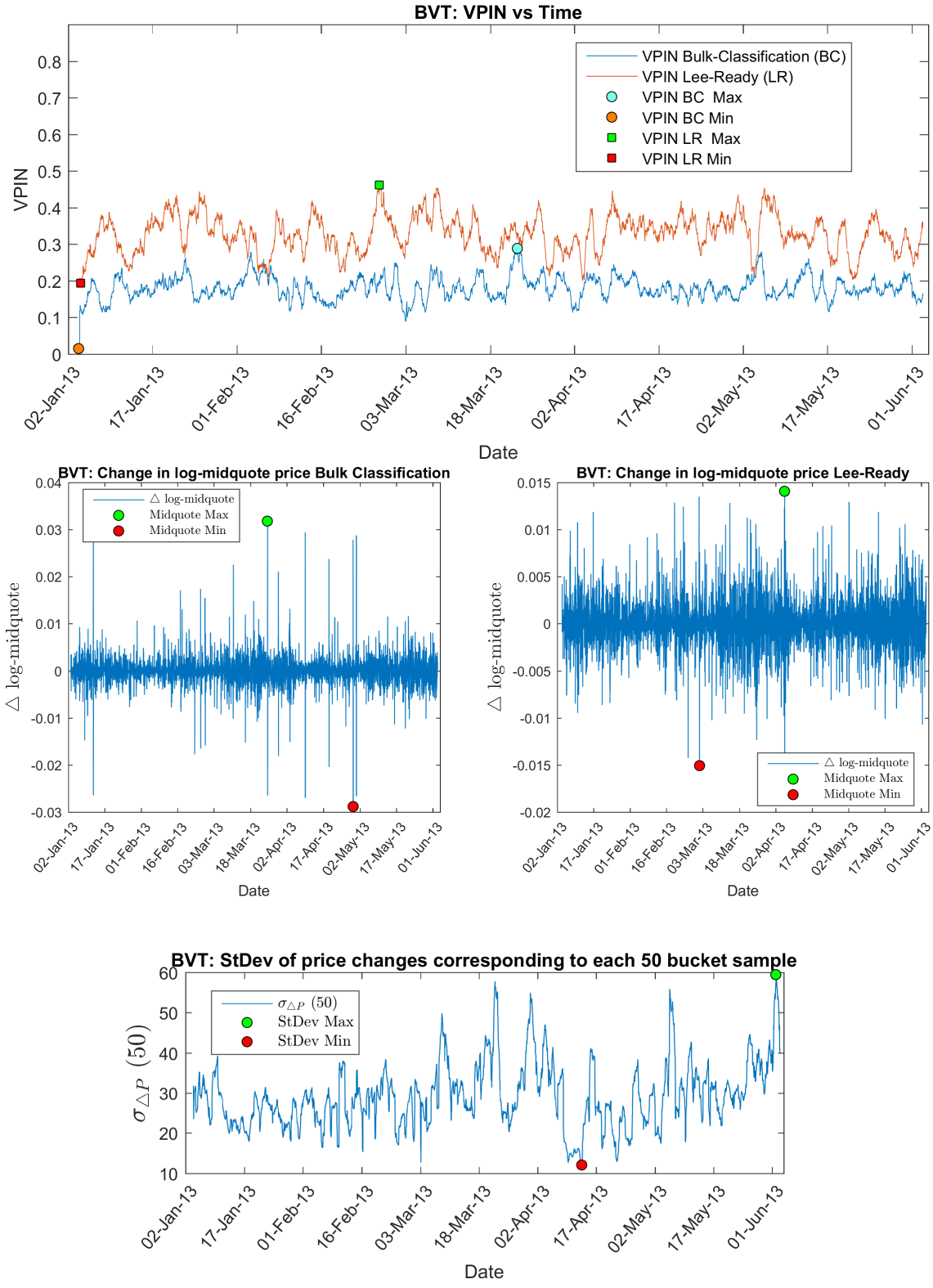


Figure 15: Bidvest Group Ltd (BVT) for the period 1^{st} January 2013 until 3^{rd} June 2013: The top pane shows VPIN vs calendar time for the Bulk Classification and Lee-Ready approach, the middle two plots show the corresponding change in log-midquotes and the last plot shows the standard deviation of price changes ($\sigma_{\triangle P}$) over each 50 bucket sample.

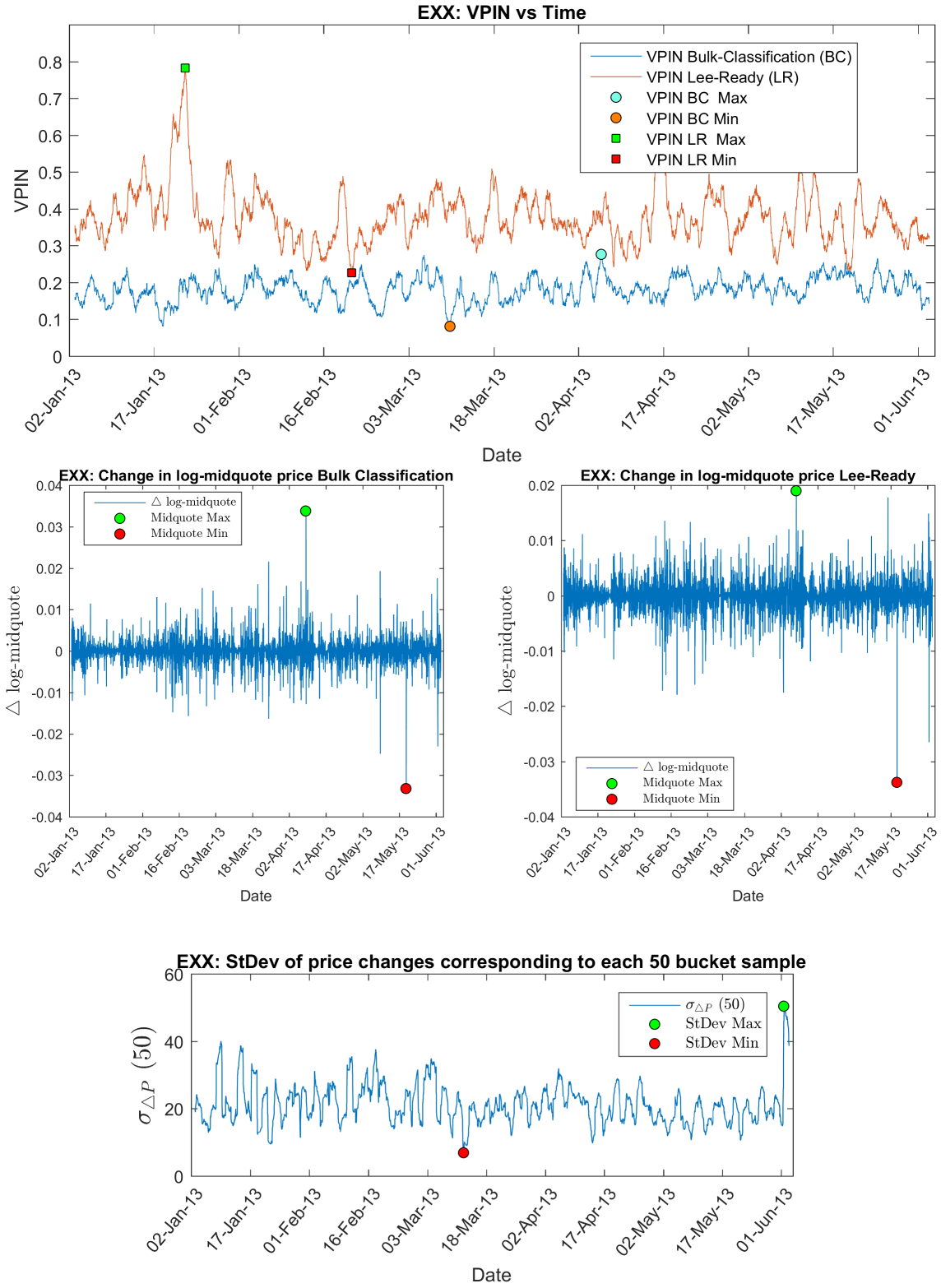


Figure 16: Exxaro Resources Ltd (EXX) for the period 1^{st} January 2013 until 3^{rd} June 2013: The top pane shows VPIN vs calendar time for the Bulk Classification and Lee-Ready approach, the middle two plots show the corresponding change in log-midquotes and the last plot shows the standard deviation of price changes ($\sigma_{\triangle P}$) over each 50 bucket sample.

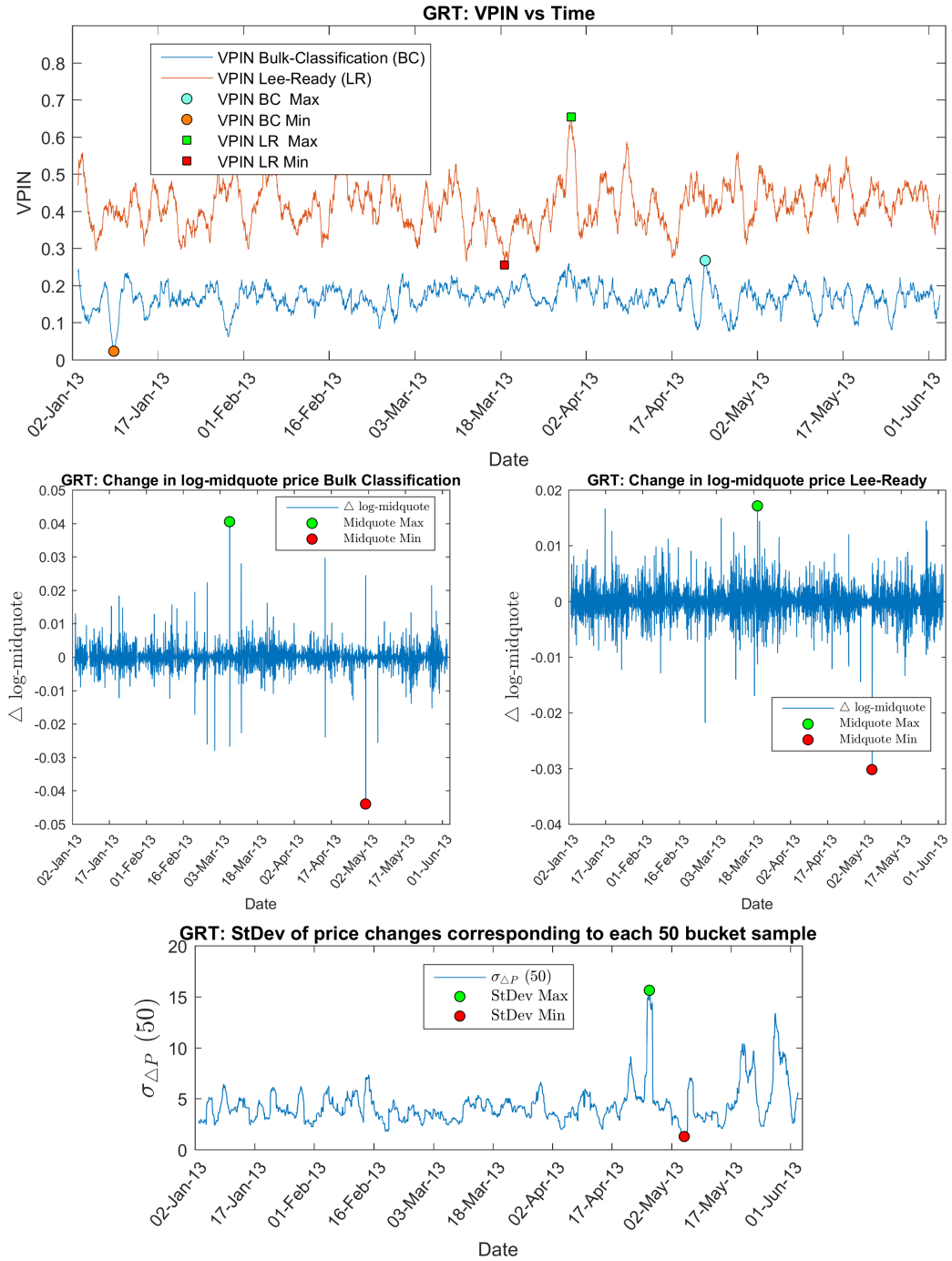


Figure 17: Growthpoint Properties Ltd (GRT) for the period 1^{st} January 2013 until 3^{rd} June 2013: The top pane shows VPIN vs calendar time for the Bulk Classification and Lee-Ready approach, the middle two plots show the corresponding change in log-midquotes and the last plot shows the standard deviation of price changes ($\sigma_{\triangle P}$) over each 50 bucket sample.

Again we see correspondence between the LR VPIN and the corresponding change in the log-midquote. The $\triangle$ log-midquote reaches its maximum shortly after the 18^{th} of March and LR VPIN reaches its minimum. This suggests that informed traders may have slowed their buying/selling rates as they anticipated large increases in the price of the stock which in turn produces smaller order imbalances and hence lower VPIN.

5.4 Statistics of Stocks

In table 17 below, the stocks considered in this project are arranged in descending order with regard to their market capitalisations. For each stock, the table shows the market capitalisation and the Volume Bucket Size (VBS). In addition to this, the mean VPIN and the standard deviation of VPIN is illustrated for both the Lee-Ready and the Bulk Classification approach.

			Bulk Classification		Lee-Ready	
Company (RIC[1])	Market Cap[2]	VBS	VPIN mean	VPIN StDev	VPIN mean	VPIN StDev
British American Tobacco (BTI)	R1.11tr	12417	0.1766	0.0355	0.3576	0.0727
BHP Billiton PLC (BIL)	R604.78bn	46298	0.1789	0.0263	0.2302	0.0411
Compagnie Financire Richemont (CFR)	R470.48bn	110863	0.1947	0.0319	0.2903	0.0480
Anglo American PLC (AGL)	R318.90bn	54200	0.1767	0.0276	0.2306	0.0450
FirstRand Ltd (FSR)	R153.63bn	156921	0.1834	0.0277	0.3144	0.0537
ABSA Group Ltd (ASA)	R103.06bn	23197	0.1764	0.0321	0.3267	0.0691
Aspen Pharmacare Holdings Ltd (APN)	R88.22bn	12411	0.1820	0.0384	0.3309	0.0659
Anglo American Platinum Ltd (AMS)	R85.68bn	5927	0.1776	0.0338	0.3669	0.0733
Bidvest Group Ltd (BVT)	R78.87bn	12781	0.1799	0.0338	0.3280	0.0504
AngloGold Ashanti Ltd (ANG)	R65.22bn	24148	0.1925	0.0314	0.2807	0.0437
Exxaro Resources Ltd (EXX)	R54.88bn	12714	0.1819	0.0334	0.3789	0.0752
Growthpoint Properties Ltd (GRT)	R50.50bn	46920	0.1666	0.0337	0.4168	0.0604
Discovery Ltd (DSY)	R47.73bn	17724	0.1766	0.0344	0.3820	0.0673
Gold Fields Ltd (GFI)	R45.54bn	50448	0.1892	0.0309	0.2927	0.0642
Assore Ltd (ASR)	R45.83bn	1547	0.1591	0.0519	0.4815	0.0938
African Rainbow Minerals Ltd (ARI)	R36.55bn	5563	0.1619	0.0361	0.4308	0.0704

Table 17: Market capitalisation, Volume Bucket Size (VBS), and mean and standard deviation of VPIN for both the Bulk Classification and Lee-Ready approach for various companies listed on the JSE 40

[1]Reuter's Intrument Code
[2]Market Capitalisations as at June 2013- supplied by Courtney Capital [9]

6 Conclusion

It is evident that there is no clear resemblance between the levels of VPIN resulting from Lee-Ready and that given by Bulk-Classification. The graphs do not fluctuate in the same directions over time and in some cases the Lee-Ready VPIN plot has a trough when the Bulk Classification VPIN plot has a peak and vice versa. This is due to the fact that in the majority of cases, VPIN is computed at different times for each of the approaches because of the aggregating into time bars required by the Bulk Classification approach.

Another interesting observation is that the stocks with higher market capitalisation on average have lower mean VPIN which is especially clear in LR approach and can be seen in table 17. The overall levels of VPIN deviate from each other at an increasing rate as the market capitalisation decreases. The plots also clearly show that infrequently stocks with lower market capitalisation have a much higher VPIN for Lee-Ready than for Bulk Classification and that the overall level for the Bulk Classification VPIN slowly decreases to levels of around 0.1-0.2 in many of the lower market capitalisation stocks. This indicates that Bulk Classification performs poorly for such stocks, which makes sense since Bulk Classification is built and better suited for high-frequency trading environments where stocks are traded in large volumes at frequent intensities. This observation suggests that the Lee-Ready approach may be better suited for calculating VPIN for such stocks.

In almost every stock, there is a clear and obvious relationship between the standard deviation of price changes over each sample ($\sigma_{\triangle P}$) and the level of VPIN on the same date. This is obvious since $\sigma_{\triangle P}$ is implicit in the VPIN formula (eq. (11)) since is is used in the calculation of the buy and sell volumes (eq. (7) and eq. (8)) of each bucket and hence the calculation of the OI in the VPIN formula. In addition to this, if one looks close enough, there are many instances in which there are correlations between the level of VPIN and the change in log-midquote on the same dates. Thus, VPIN is able to indicate the fluctuations of the changes in the log-midquotes over time although doesn't necessarily give an early indication of the possible fluctuations in the log-midquote.

Hence, the author's argument that VPIN is a high-frequency estimate of flow toxicity is clear in this study and we propose that in cases where the stock is infrequently traded and has a small market capitalisation, using Lee-Ready may serve as an alternative to calculating flow toxicity for such stocks.

7 References

[1] D. Easley, M. Lòpez de Prado,M. O'Hara, Flow toxicity and liquidity in a highv frequency world. Review of Financial Studies, February, 2012. frequency world. Review of Financial Studies, February, 2012.

[2] D. Easley, N. Kiefer, M. O'Hara, J. Paperman. Liquidity, Information, and Infrequently Traded Stocks. Journal of Finance, September, 1996.

[3] D. Abada, J. Yagüeb. From PIN to VPIN: An introduction to order flow toxicity. The Spanish Review of Financial Economics, 2012.

[4] D. Easley, R. Engle, M. O'Hara, L. Wu. Time-Varying Arrival Rates of Informed and Uninformed Trades. Journal of Financial Econometrics, 2008.

[5] W. Wei, D. Gerace, A. Frino. Informed Trading, Flow Toxicity and the Impact on Intraday Trading Factors. Australasian Accounting, Business and Finance Journal, 7(2), 2013.

[6] E. Theissen. A Test of the Accuracy of the Lee / Ready Trade Classification Algorithm, February 2000.

[7] C. M. C. Lee and M. J. Ready, Inferring Trade Direction from Intraday Data. Journal of Finance, vol .46, pp. 733-746, 1991.

[8] T. G. Andersen and O. Bondarenko. Assessing Measures of Order Flow Toxicity via Perfect Trade Classification. November 2013.

[9] Courtney Capital (June 2013), "JSE Top 40 shares", Retrieved from http://www.courtneycapital.co.za/jse-top-40-shares/

8 Appendices

A Data Processing

A.1 Data Processing Script: Bulk Classification

Below is the script which is attached to the Submission Script to be executed on the AMF High Performance Computing cluster. The script processes the data in order to implement the Bulk Classification approach to calculate VPIN.

```
%% AMF Research Project: Computing VPIN: Bulk Classification vs. Lee-Ready
%% data_processing_script_BC_715798
%
% Author: N.J. Murphy
%
% Person Version:
% AMF-Project-NJM-715798_NMurphy_10-06-2016_bulk-class006ab_data-processing-↩
   script-BC
%
% 1 Problem Specification:  This script aims to replicate the calclations and ↩
   analysis from
%   Easley et al[1] for stocks listed in the top 40 of the Johannesburg Stock ↩
   Exchange(JSE).
%
% 2 Data Specification: 6 months worth of order book data for 16 stocks listed
%   on the JSE Top 40. The specified period is from 01 January 2013 to 03 June
%   2013.
%   In order to run and publish this script, 2 stocks were relabelled in order
%   to loop over a 2 week period and 2 stocks from the test data was used.
%
% 3 Configuration Control:
%       userpath/MATLAB/AMF
%       userpath/MATLAB/AMF/AMF_Project_NJM_715798
%       userpath/MATLAB/AMF/AMF_Project_NJM_715798/Scripts
%       userpath/MATLAB/AMF/AMF_Project_NJM_715798/Functions
%       userpath/MATLAB/AMF/AMF_Project_NJM_715798/Data
%       userpath/MATLAB/AMF/AMF_Project_NJM_715798/html
%
% 4 Version Control: No current version control
%
% 5 References:
% [1] D.Easley, M.Lopez de Prado, M.O'Hara. Flow toxicity and liquidity
%     in a highv frequency world. Review of Financial Studies, February, 2012.
% [2] D.Easley, N.Kiefer, M.O'Hara, J.Paperman. Liquidity, Information, and
%     Infrequently Traded Stocks. Journal of Finance, September, 1996.
% [3] D.Abada, J.Yagueb. From PIN to VPIN: An introduction to order flow toxicity↩
   .
%     The Spanish Review of Financial Economics, 2012.
% [4] W.C.Wei, D.Gerace, A.Frino. Informed Trading, Flow Toxicity and the Impact
%     on Intraday Trading Factors. Australasian Accounting, Business and Finance
%     Journal, 7(2), 2013.
%
% 6 Current Situation: data_processing_script_BC_715798
% 7 Future Situation: no future situation
%
% Uses: This script will be the final data processing script for
% implementing the Bulk Classification algorithm to calculate VPIN
```

```
%% 1. Data Description

% 1  RIC              - Reuters instrument code.
%                       A Reuters instrument code, or RIC, is a ticker
%                       like code used by Thomson Reuters to identify
%                       financial instruments and indices.
%
% 2   DateL           - Date of transaction.
% 3   TimeL           - Time of transaction.
% 4   DateTimeL       - Date and time of transaction.
% 5   Type            - Type of transaction: * Auction
%                                            * Quote
%                                            * Trade
% 6   Price           - Price of transaction for auction and trade
% 7   Volume          - Volume of tramsaction for auction and trade
% 8   MarketVWAP      - Market Volumes Weighted average prices
%                       = sum(shares bought * share price)/(Total shares bought)
% 9   L1BidPrice      - The price a buyer is willing to pay
% 10  L1BidSize       - The size of order a buyer wishes to achieve
% 11  L1AskPrice      - The price the seller is willing to offer
% 12  L1AskSize       - The size of order the seller wishes to achieve
%


%% 2. Data Cases
%1) Trade entry with zero volume
% {'GRTJ.J','01-NOV-2013','09:00:09.315846','2013-11-01T09:00:09.315Z','Trade↩
   ',2550,0,0}

%2) Trades occurring after 17:00
%{'AGLJ.J','07-NOV-2013','17:00:05.746125','2013-11-07T17:00:05.746Z','Trade↩
   ',25283,363506,25602.5000000000,9765625,0,0,0;
% 'AGLJ.J','07-NOV-2013','17:00:05.746125','2013-11-07T17:00:05.746Z','Trade↩
   ',25659,0,0,0,0,0,0;
% 'AGLJ.J','07-NOV-2013','17:00:30.967130','2013-11-07T17:00:30.967Z','Quote↩
   ',0,0,0,0,0,25416,20205;
% 'AGLJ.J','07-NOV-2013','17:06:14.197755','2013-11-07T17:06:14.197Z','Trade↩
   ',25283,39430,0,0,0,0,0;
% 'AGLJ.J','07-NOV-2013','17:10:12.394170','2013-11-07T17:10:12.394Z','Trade↩
   ',25602,8000,0,0,0,0,0;
% 'AGLJ.J','07-NOV-2013','17:26:44.908276','2013-11-07T17:26:44.908Z','Trade↩
   ',25485,10460,0,0,0,0,0;
% 'AGLJ.J','07-NOV-2013','17:26:45.448335','2013-11-07T17:26:45.448Z','Trade↩
   ',25548,20358,0,0,0,0,0}

%{'GRTJ.J','06-NOV-2013','16:49:19.253191',2483,429;
% 'GRTJ.J','06-NOV-2013','17:00:02.488567',2480,599771;
% 'GRTJ.J','06-NOV-2013','17:24:55.045941',2477,76385}

%3) No trade occurs over consecutive time bars- thus no price shift nor volume ↩
   will be
%entered into these time bars
%{'GRTJ.J','01-NOV-2013','11:33:57.686807',2533,1008;'GRTJ.J','01-NOV↩
   -2013','11:37:11.160757',2533,1399}

%4) Empty time bars where no trade occurs(rows 2 and 3 below)
%{'01-NOV-2013','10:16:00.000','10:17:00.000',[2x5 cell' char(10) '],[];
%'01-NOV-2013','10:17:00.000','10:18:00.000',[],[];'01-NOV↩
   -2013','10:18:00.000','10:19:00.000',[],[];
%'01-NOV-2013','10:19:00.000','10:20:00.000',[],[];'01-NOV↩
   -2013','10:20:00.000','10:21:00.000',[],[];
%'01-NOV-2013','10:21:00.000','10:22:00.000',[3x5 cell' char(10) '],[]}


%% 3. Clear Workspace
% This section prepares the workspace for the implementation of the script.
close all;      % close any open figures
```

```
clc;            % clear command window
format long g;  % formating for output on comand window
format compact; % formating for output on comand window


%% 4. Path Setup
% Set the project paths
[projectPath] = pwd;
% Add the project path so that the script sees the necessary functions
addpath(projectPath);
userpathstr = userpath;
userpathstr = userpathstr(~ismember(userpathstr,';'));
%projectpath = 'AMF/AMF_FinalProject_NJM_715798';   %path for personal computer
projectpath = 'AMF_FinalProject_NJM_715798';   %path for AMF machine
addpath(fullfile(userpathstr,projectpath,'Functions'));
addpath(fullfile(userpathstr,projectpath,'Scripts'));
addpath(fullfile(userpathstr,projectpath,'html'));

% # Filepath to input data
% Path to where the data are stored
%[datafilepath] = 'C:\Users\Nicholas\Documents\MATLAB\AMF\↩
   AMF_FinalProject_NJM_715798\Data\Transactions';  %path to data on personal ↩
   computer
%[datafilepath] = '\\AMF\Data\Transactions\JSE';   %path to CSAM cluster data
[datafilepath] = 'Z:\Data\Transactions\JSE'; % This is the path for data on ↩
   AMFcluster

%%%%%%%%%%%%%%%%%%%%%%%%
%% 5. Process the Data %%
% Trade entries are sorted into buckets of equal size, V, which is
% specified to be the average daily volume over the entire period. The number
% of buckets is chosen to 50. Once all 50 buckets are complete, the VPIN metric
% is computed. If the last trade needed to complete a bucket is for a
% size greater than required, the excess size is given to the next bucket.
% The VPIN is then computed after each bucket is filled. This implies,
% for example, if bucket 51 is filled, we will drop the first bucket and
% computed the VPIN for buckets 2-51. Thus VPIN is calculated in volume-time
% which is argued to be necessary in the high frequency world. Perform steps to
% remove extraneous data and data falling outside the continuous trading session.

noofstocks = 8; %specify number of stocks to be analysed and processed
VPINstore = cell(1,noofstocks);  %initialise a cell array to store the timenars ↩
   for each stock for the entire period
VBS = zeros(1,noofstocks);  %initiate vector to store VBS for each stock
RICstore = cell(1,noofstocks);  %initialise array to store RIC's
outliers = zeros(1,noofstocks);   %initialise vector to store number of outliers

for n = 1:noofstocks  % Loop over stocks
 %%%%%%%%%%%%%%%%%%
 %% 6. Load Data %%
 % The data from the matfile for the specified dates and specified stock is
 % loaded on each iteration of the loop over stocks. The createdate.m function
 % is used to generate the required dates from these matfiles.

  % Step 1: Generate dates using createdates.m function
  %[startDate] = '01-Jan-2013';  %start date for cluster data
  %[endDate] = '03-Jun-2013';  %end date for cluster data
  [startDate] = '01-Nov-2013';  %start date for test data
  [endDate] = '07-Nov-2013';  %end date for test data
  [TradingDayStart]   = []; % Defaults to 08:30:00
  [TradingDayEnd]     = []; % Defaults to 17:30:00
  [Type]              = []; % Defaults to week
  [s,e] = createdates(startDate,endDate,TradingDayStart,TradingDayEnd,Type); %↩
     call createdates function to generate dates

  % Step 2: Predefine variable to store matfile names for each stock
  matfilename = cell(length(e),1);   %create a cell array to store the names of ↩
     the matfiles
```

```
storedata = cell(length(e),1); %create cell array to store trades for all ↩
   stocks for entire period
storequotes = cell(length(e),1); %create cell array to store trades for all ↩
   stocks for entire period

for k = 1:length(e)  %loop over weeks
   % Step 3: Pre-define path to the required folder
  folders = dir(datafilepath);  %directory of the datafilepath
  folderpath = cell(noofstocks,1);  %create a cell array to store the paths for↩
      each of the stock folders

  % Step 4: construct the matfile name for the required stock and week
  % and extract the data from that matfile
  weekStart = datestr(s{k,1},'ddmmmyyyy_HHMM');
  weekEnd = datestr(e{k,1},'ddmmmyyyy_HHMM');
  folderpath{n} = strcat(datafilepath,'\',folders(2+n).name);  %path to ↩
     specific stock folder
  [RIC] = folderpath{n}(end-5:end);  %extract stock RIC
  matfilename{k} = strcat(RIC,'-Transactions-',weekStart,'-to-',weekEnd,'.mat')↩
     ;  %retrieve name of required matfile for each week as we iterate over the↩
      weeks loop

  % Step 5: Load the required order book data
  [data] = load(fullfile(datafilepath,RIC,matfilename{k})); %load the matfile

  % Step 6: Extract only data
  data = data.data;

  % Step 7: Extract only trades
  trades = data(strncmpi(data(:,5),'Trade',1),:);  %extract only trade data

  % Step 8: Remove all unneccessary data- columns 8 to 12,rows
  % with zero trade volume and rows 4 and 5 which contain information we
  % will not need.
  trades(:,8:12) = [];
  trades(cell2mat(trades(:,7))==0,:) = [];
  trades(:,4:5) = [];

  % Step 9: store trade data for the nth stock for the kth week
  storedata{k,1} = trades;


%-------------------------------------------------------------------------
%%%%%%%%%%%%%%%%
%% Quote Data %%
% Step 1: Create a new data set which contain all quote entries from ↩
   compactdata and
% get extraction indices to extract daily data from this dataset.
  quotedata = data(strncmpi(data(:,5),'Quote',1),:);  %all quote entries

  [~,uniqueday] = unique(datenum(quotedata(:,2)),'first'); %find the indices ↩
     for each day
  [daystart] = uniqueday(1:end);  %find the indices for the start of each day
  [dayend] = [uniqueday(2:end)-1;size(quotedata,1)]; %find the indices for the ↩
     end of each day
   uniqueday =[];
  % Step 2: Drop down all bid/ask prices and volumes into cells which have zero
  % bid/ask prices and volumes and then calculate the midquote for each
  % quote entry
  quotedata(:,13) = {0}; %create column
  for i = 1:length(dayend)   %loop over days
     for j = (daystart(i)+1):dayend(i)  %loop from beginning of day to end of ↩
        day
       if cell2mat(quotedata(j,9:10)) == 0
          quotedata(j,9:10) = quotedata(j-1,9:10);  %find the previous best ↩
             bid price and volume if the cells contain zeros
       elseif cell2mat(quotedata(j,11:12)) == 0
          quotedata(j,11:12) = quotedata(j-1,11:12);  %find the previous best ↩
```

```
                ask price and volume if the cells contain zeros
            end
            if quotedata{j,9} ~= 0 && quotedata{j,11} ~= 0
              quotedata{j,13} = (quotedata{j,9} + quotedata{j,11})/2;  %Calculate ↩
                 the mid-quotes
            end
        end
    end
    clearvars daystart dayend data
   storequotes{k,1} = quotedata;
end

 %--------------------------------------------------------------------------
 %%%%%%%%%%%%%%%%%%%
 %% 9. Trade data %%
 % Extract only the trade data for entire period for
 % the nth stock.

quotedata = vertcat(storequotes{1:end,1});  %all quote entries
tradedata = vertcat(storedata{1:end,1});  %extract trades for nth stock for ↩
   entire period
clearvars trades storedata storequotes

% Step 2: Get the date-time of the trades
[yr,mt,dy,~,~,~] = datevec(tradedata(:,2)); %find year, month and day
[~,~,~,hr,mn,sc] = datevec(tradedata(:,3));  %find hour, minute and second
[tradedatetime] = datenum(yr,mt,dy,hr,mn,sc);  %create vector of dates and ↩
   times

% Step 3: Remove all events between before 09h00 and after 17h00
[removalindextime] = (hr<9 | hr>=17| (hr==16&mn>50)); %find indices of such ↩
   events
tradedatetime(removalindextime) = [];  %remove entries from date vector
tradedata(removalindextime,:) = [];  %remove entries from data

% Step 4: Remove unnecessary variables
clearvars removalindextime yr mt dy hr mn sc tradedatetime

% Step 5: Remove trades which have a volume greater than 5 standard
% deviations form the mean
nooftradesbefore = length(tradedata);  %calculate the number of trade entries ↩
   before removing outliers
tradedata(mean(cell2mat(tradedata(:,5))) + 5*std(cell2mat(tradedata(:,5)))< ↩
   cell2mat(tradedata(:,5)),:) = []; %remove all rows with a volume greater ↩
   than 5 standard
% deviations form the mean
nooftradesafter = length(tradedata); %calculate the number of trade entries ↩
   after removing outliers
noofoutliers = nooftradesbefore-nooftradesafter;  %number of ouliers removed

%--------------------------------------------------------------------------
%%%%%%%%%%%%%%%%%%%
%% 10. VBS and ADV %%
% Calculate volume bucket size(VBS) based on average daily
% volume(ADV) over the entire period. We will choose a sample of 50 buckets per↩
    calculation of
% VPIN as specified by the authors. The VBS will then be the ADV divided
% by the number of buckets (50).

% Step 1: find indices for unique days, start of days and end of days
[~,uniquedaytr] = unique(datenum(tradedata(:,2)),'first');  %find the indices ↩
   for each day
[daystarttr] = uniquedaytr(1:end);  %find the indices for the start of each day
[dayendtr] = [uniquedaytr(2:end)-1;size(tradedata,1)]; %find the indices for ↩
   the end of each day

% Step 2: Calculate average daily volume for the entire period being
% considered
```

```
dailyvol = zeros(length(daystarttr),1); %initialise daily volume for the nth ↩
   stock

for i = 1:length(daystarttr) %loop over days
  dailyvol(i,1) = sum(cell2mat(tradedata(daystarttr(i):dayendtr(i),5))); %↩
     compute total daily volume
end
avedailyvol = sum(dailyvol(:,1))/length(daystarttr); %compute average daily ↩
   volume for the nth stock

% Step 3: Compute the bucket size and define the number of buckets
noofbuckets = 50; %define the number of buckets used to calulate the VBS
VBS(1,n) = round(avedailyvol/noofbuckets,0); %define VBS to be one fiftieth of↩
    average daily volume

%-------------------------------------------------------------------------
%%%%%%%%%%%%%%%%%%
%% 10. Time Bars %%
% aggregates trades into 1 minute time bars, computes
% the price change between the beginning and the end of each of
% the time bars and computes the total volume of all the trades that lie in
% each time bar. The first time bar will be at 9:00 when the continuous trading↩
    starts and
% the final time bar will end at 16:50 when the continuous trading ends. Thus ↩
   there
% are 7 hours 50 minutes in the trading day with regard to the JSE so there ↩
   will be
% 60*8-10 = 470 time bars each day.


% Step 1: Create an empty cell array to store the aggregated trades into 1 ↩
   minute time bars
% and find the date vectors of all the trades. Generate the time bars for
% a single day from 9:00 until 16:50 which is when closing auctions take place.
formatIn = 'HH:MM:SS.FFF'; %format of time needed for various functions
[yrs,mts,dys,~,~,~] = datevec(tradedata(:,2)); %find year, month and day
[~,~,~,hrs,mns,scs] = datevec(tradedata(:,3)); %find hour, minute and second
[tradedatetimes] = datenum(yrs,mts,dys,hrs,mns,scs);
nooftimebars = 60*8-9; %number of time bars per day
timebars = cell(nooftimebars*length(uniquedaytr),5); %nooftimebars*length(↩
   uniqueday)v%create array to store time bars and corresponding data
times = cellstr(datestr(9/24:1/1440:17/24,formatIn)); %create a character ↩
   string of 1 minute time bars- fill in beginning of time-bar
times = times(1:end-9,1); %remove the last 10 minutes of the day
times(1:(end-1),2) = times(2:end,1); %fill in end of time bars

% Step 2: Fill in the dates and times of each time-bar over the entire
% period into the empty cell array
ind = 0; %create a temporary index variable
for i = 1:length(uniquedaytr) %loop over days
  for j = 1:nooftimebars+1 %loop from beginning to end of day
    timebars{j+(i-1)*nooftimebars,1} = tradedata{j+uniquedaytr(i),2}; %fill in ↩
       dates of time bar
    timebars(j+(i-1)*nooftimebars,2:3) = times(j,1:2); %fill in times of ↩
       timeimebars bar
  end
  ind = ind + nooftimebars + 1; %use index to call times for each day
end
[yrbar,mtbar,dybar,~,~,~] = datevec(timebars(1:nooftimebars*length(uniquedaytr)↩
   ,1)); %find year, month and day for dates in time bars
[~,~,~,hrbar,mnbar,scbar] = datevec(timebars(1:nooftimebars*length(uniquedaytr)↩
   ,2)); %find hour, minute and second for dates in time bars
[tradedatetimebar] = datenum(yrbar,mtbar,dybar,hrbar,mnbar,scbar); %find serial↩
    number for dates and times in time-bars

% Step 3: Find the indices of the trades which lie in each of the time
% bars and sort the trades into the corresponding time bars
for i = 1:length(uniquedaytr) %loop over days
```

```
    for j = 1:nooftimebars-1  %loop from beginning to end of day
     [idx,~] = find(tradedatetimes(:,1) >= tradedatetimebar(j+(i-1)*nooftimebars
        ) & tradedatetimes(:,1) < tradedatetimebar(j+1+(i-1)*nooftimebars));  %
        find indices of trades that occur in current time bar
     timebars{j+(i-1)*nooftimebars,4} = tradedata(idx(1:end),:);  %capture all
        the trades which lie in the jth time bar bar
   end
end

% Step 4: Remove empty time bars in which no trades will occur
timebars(any(cellfun(@isempty,timebars(:,4)),2),:) = [];
clearvars tradedatetimebar dayendtr daystarttr uniquedaytr yrbar dybar mtbar
   scbar hrbar...
    scs mns hrs mts yrs mnbar tradedatetimes times

%--------------------------------------------------------------------------
%%%%%%%%%%%%%%%%%%%%%%%%
%% 10. Price Shifts %%%
% Compute the price shifts between consecutive
% bars by findong the change in price between the last price in the time bar
% in question and the last price in the most recent time bar.

[~,uniquedaybar] = unique(datenum(timebars(:,1)),'first');  %find the indices
   for each day in the time bars
[daystartbar] = uniquedaybar(1:end);  %find the indices for the start of each
   day
[dayendbar] = [uniquedaybar(2:end)-1;size(timebars,1)]; %find the indices for
   the end of each day

% Step 1: Loop over each day to avoid comparing price shifts between days
% as it may drastically skew results
for i = 1:length(uniquedaybar) %loop over time bars
 timebars{uniquedaybar(i),5} = 0;  %set price shift of first time bar to zero
    as there is no previous data to compute a price change
 timebars{uniquedaybar(i),6} = sum(cell2mat(timebars{uniquedaybar(i),4}(:,5)));
     %compute the total volume in 1st time bar of each day
  for j = daystartbar(i):dayendbar(i)-1  %loop from beginning to end of day
  timebars{j+1,5} = cell2mat(timebars{j+1,4}(end,4)) - cell2mat(timebars{j,4}(
     end,4));  %compute difference between last price in current time bar and
     last price shift in previous bar
  timebars{j+1,6} = sum(cell2mat(timebars{j+1,4}(:,5)));  %compute the total
     volume in (j+1)th time bar of each day
  end
end
timebars(:,4) = [];  %remove column 4 as this information will not be necessary
    for proceeding steps
clearvars dayendbar daystartbar uniquedaybar dys

%%%%%%%%%%%%%%%%%%%%%%%%
%% 11. Volume Buckets %%%
% Iterate through the 1 minute time bars and sum up the volumes of
% of the bars until the VBS threshold is reached. Assign the given set of
% time bars current bucket index. If the last time bar in the given set of
% time bars which make up the volume of the tau^th bucket is of volume greater
   than
% that required to fill the volume bucket, the last time bar is broke up into
% 2 parts where one will fill the most recent bucket and the excess volume will
    be given
% to the next bucket. The same associated price change is used for the time bar
% with excess volume.

% Step 1: initiate variables needed for the volume bcuketing
bucketind = 0; %initialise index for finding the index of time bars for which a
    bucket will be completed
tau = 1;  %initialise index for storing the time bar index at each completion
   of a bucket

% Step 2: Set up a loop over entire sample of data in order to find
```

```
% indices for time bars where buckets will strart and end. The
% excess volume in a given bucket is transferred to the next bucket along
% with the corresponding price change
i =0 ;
while i < length(timebars)
  if sum(cell2mat(timebars((bucketind(tau)+1):i,5))) >= VBS(1,n) %find index ↩
     for time bar where a bucket will be completed
    excess = sum(cell2mat(timebars((bucketind(tau)+1):i,5))) - sum(cell2mat(↩
       timebars((bucketind(tau)+1):(i-1),5))) - VBS(1,n) + sum(cell2mat(↩
       timebars((bucketind(tau)+1):(i-1),5))); %calculate the excess volume in ↩
       the current bucketind which will be given to next bucket
    timebars{i,5} = VBS(1,n) - sum(cell2mat(timebars((bucketind(tau)+1):(i-1)↩
       ,5))); %complete the current volume bucket with the required amount of ↩
       volume
    timebars((i+1):(end+1),:) = timebars((i):(end),:); %create an extra time ↩
       bar with the same times as the i-th time bar since extra volume must be ↩
       carried over to next bucket
    timebars{i+1,5} = excess; %how much volume the next bucket has received ↩
       from previous bucket
    timebars{i+1,4} = timebars{i,4}; %the price change associated with the ↩
       volume inherited by the next bucket
    timebars((bucketind(tau)+1):i,6) = {tau}; %the bucket index of the time ↩
       bars
    tau = tau + 1; %add one to bucket counter
    bucketind(tau) = i; %store the index of the time bar for which the bucket ↩
       will be filled
  end
i = i + 1; %add one to the while loop counter
end

%-------------------------------------------------------------------------
%%%%%%%%%%%%%%
%% 12. VPIN %%
% Iterate over the time bar data and as soon as the threshold of 50 buckets
% is reached and compute the VPIN metric.
% For each row in the sample, use the given probabilistic method to classify ↩
   buyer
% and seller initiated volume for the time bar. The buyer volume is
% calculated using the value of the normal distribution evaluated over the
% price change of the given bar standardised by the standard deviation of
% the expanded sample price changes. The seller volume is just the complement ↩
   of the
% probability measure for buyers. We then multiply the volume of the time bar
% by the buyer and seller probability measures to get the buyer and seller
% volume for each time bar. Extract the midquote at the time VPIN was
% printed.

[~,uniquebucket] = unique(cell2mat(timebars(:,6)),'first'); %find the total ↩
   number of buckets in the time bar data
[bucketstart] = uniquebucket(1:end); %find the indices for the start of each ↩
   day
[bucketend] = [uniquebucket(2:end)-1;find(cell2mat(timebars(:,6)),1,'last')];
totalbuckets = unique(cell2mat(timebars(:,6)),'first'); %total number of ↩
   buckets in the sample for the n-th stock
noofvpins = length(totalbuckets) - 50 + 1; %calculate the number of times VPIN↩
    will be updated
VPIN = cell(noofvpins,5); %initialise a vector to store dates, times and VPIN ↩
   metric
samplesize = 50; %define the number of buckets used in order to update VPIN

% Compute serial dates for quote data
[yrq,mtq,dyq,~,~,~] = datevec(quotedata(:,2)); %find year, month and day
[~,~,~,hrq,mnq,scq] = datevec(quotedata(:,3)); %find hour, minute and second
[tradedatetimequotes] = datenum(yrq,mtq,dyq,hrq,mnq,scq); %create vector of ↩
   dates and times
clearvars uniquebucket hrq dyq mnq mtq scq yrq totalbuckets bucketind

for i = 1:noofvpins %loop over the number of updates of VPIN
```

```
    if i == 1  %check if we are computing the buy and sell volume for the 1st ↩
       trade of the sample for which we don't have any prior information to ↩
       compute
      ind = 1;
    elseif i>1
      ind = 0;
    end
    % Step 2: compute the standard deviation of the current sample
    deltap = cell2mat(timebars((bucketstart(i)+ind):bucketend(i+samplesize-1),4))↩
       ; %vector of price shifts
    sigmadp = std(deltap);  %compute the standard deviation for the price shifts

    for j = (bucketstart(i)+ind):bucketend(i+samplesize-1)  %loop form 1st bucket↩
       in VPIN calculation to the last bucket in the calculation
      timebars{j,7} = timebars{j,5}*normcdf((timebars{j,4} - timebars{j-1,4})/↩
         sigmadp);  %compute the buy volume for the jth time bar
      timebars{j,8} = timebars{j,5}*(1-normcdf((timebars{j,4} - timebars{j-1,4})/↩
         sigmadp));  %compute the sell volume for the jth time bar
    end

    for j = i:(i+samplesize-1)  %loop over the current sample of buckets which we↩
       are computing VPIN for
      timebars{bucketend(j),9} = abs(sum(cell2mat(timebars(bucketstart(j):↩
         bucketend(j),8))) - sum(cell2mat(timebars(bucketstart(j):bucketend(j),7)↩
         ))); %compute order imbalance for each bucket in the i-th calculation of↩
          VPIN
    end
    VPIN{i,3} = sum(cell2mat(timebars((bucketstart(i)):bucketend(i+samplesize-1)↩
       ,9)))/(samplesize*VBS(1,n));  %compute VPIN for the i-th update of the ↩
       metric
    VPIN{i,2} = timebars(bucketend(i+samplesize-1),2); %enter in the time for the↩
       corresponding VPIN calculation
    VPIN{i,1} = timebars(bucketend(i+samplesize-1),1);  %enter in the date of the↩
       corresponding VPIN calculation

    % Find mid-quote when VPIN is printed
    % Step 1: Get the date-time of current VPIN
    [yrvpin,mtvpin,dyvpin,~,~,~] = datevec(VPIN{i,1}); %find year, month and day
    [~,~,~,hrvpin,mnvpin,scvpin] = datevec(VPIN{i,2});  %find hour, minute and ↩
       second
    tradedatetimeVPIN = datenum(yrvpin,mtvpin,dyvpin,hrvpin,mnvpin,scvpin);  %↩
       create vector of dates and times

    VPIN{i,4} = cell2mat(quotedata(find(tradedatetimequotes(:)<=tradedatetimeVPIN↩
       ,1,'last'),13)); %find the index of the most recent quote which occurred ↩
       before VPIN was updates

    % Compute the standard deviation for the current VPIN
    VPIN{i,5} = sigmadp;
  end

  % Store results
  VPINstore{1,n} = VPIN;
  RICstore{1,n} = tradedata{1,1};

  clearvars bucketend bucketstart tradedata timebars tradedatetimequotes...
    tradedatetimeVPIN VPIN sigmadp deltap yrvpin mtvpin dyvpin scvpin ...
    mnvpin hrvpin quotedata
end
```

A.2 Data Processing Script: Lee-Ready

Below is the script which is attached to the Submission Script to be executed on the AMF High Performance Computing cluster. The script processes the data in order to implement the Lee-Ready approach to calculate VPIN.

```
%% AMF Research Project: Computing VPIN: Bulk Classification vs. Lee-Ready
%% data_processing_script_LR_715798
%
% Author: N.J. Murphy
%
% Person Version:
% AMF-Project-NJM-715798_NMurphy_30-11-2016_L-R-005_data-processing-script-LR
%
% 1 Problem Specification:  This script aims to replicate the calclations and ↩
   analysis from
%   Easley et al[1] for stocks listed in the top 40 of the Johannesburg Stock ↩
   Exchange(JSE).
%
% 2 Data Specification: 6 months worth of order book data for 16 stocks listed
%   on the JSE Top 40. The specified period is from 01 January 2013 to 03 June
%   2013.
%   In order to run and publish this script, 2 stocks were ralabelled in order
%   to loop over a 2 week period and 2 stocks from the test data was
%   used.
%
% 3 Configuration Control:
%       userpath/MATLAB/AMF
%       userpath/MATLAB/AMF/AMF_Project_NJM_715798
%       userpath/MATLAB/AMF/AMF_Project_NJM_715798/Scripts
%       userpath/MATLAB/AMF/AMF_Project_NJM_715798/Functions
%       userpath/MATLAB/AMF/AMF_Project_NJM_715798/Data
%       userpath/MATLAB/AMF/AMF_Project_NJM_715798/html
%
% 4 Version Control: No current version control
%
% 5 References:
% [1] D.Easley, M.Lopez de Prado, M.O'Hara. Flow toxicity and liquidity
%     in a highv frequency world. Review of Financial Studies, February, 2012.
% [2] D.Easley, N.Kiefer, M.O'Hara, J.Paperman. Liquidity, Information, and
%     Infrequently Traded Stocks. Journal of Finance, September, 1996.
% [3] D.Abada, J.Yagueb. From PIN to VPIN: An introduction to order flow toxicity↩
   .
%     The Spanish Review of Financial Economics, 2012.
% [4] W.C.Wei, D.Gerace, A.Frino. Informed Trading, Flow Toxicity and the Impact
%     on Intraday Trading Factors. Australasian Accounting, Business and Finance
%     Journal, 7(2), 2013.
%
% 6 Current Situation: data_processing_script_LR_715798
% 7 Future Situation: no future situation
%
% Uses: This script will be the final data processing script for
% implementing the Lee-Ready algorithm to calculate VPIN


%% 1. Data Description
%
% 1  RIC               - Reuters instrument code.
%                            A Reuters instrument code, or RIC, is a ticker
%                            like code used by Thomson Reuters to identify
%                            financial instruments and indices.
%
% 2   DateL            - Date of transaction.
% 3   TimeL            - Time of transaction.
% 4   DateTimeL        - Date and time of transaction.
% 5   Type             - Type of transaction: * Auction
%                                             * Quote
%                                             * Trade
% 6   Price            - Price of transaction for auction and trade
% 7   Volume           - Volume of tramsaction for auction and trade
% 8   MarketVWAP       - Market Volumes Weighted average prices
%                        = sum(shares bought * share price)/(Total shares bought)
% 9   L1BidPrice       - The price a buyer is willing to pay
% 10  L1BidSize        - The size of order a buyer wishes to achieve
```

```
% 11  L1AskPrice       - The price the seller is willing to offer
% 12  L1AskSize        - The size of order the seller wishes to achieve
%


%% 2. Data Cases
% 1) Trade entry with zero volume
% {'GRTJ.J','01-NOV-2013','09:00:09.315846','2013-11-01T09:00:09.315Z','Trade↩
   ',2550,0,0}

% 2) Trades occurring after 17:00
% {'AGLJ.J','07-NOV-2013','17:00:05.746125','2013-11-07T17:00:05.746Z','Trade↩
   ',25283,363506,25602.5000000000,9765625,0,0,0;
% 'AGLJ.J','07-NOV-2013','17:00:05.746125','2013-11-07T17:00:05.746Z','Trade↩
   ',25659,0,0,0,0,0,0;
% 'AGLJ.J','07-NOV-2013','17:00:30.967130','2013-11-07T17:00:30.967Z','Quote↩
   ',0,0,0,0,0,25416,20205;
% 'AGLJ.J','07-NOV-2013','17:06:14.197755','2013-11-07T17:06:14.197Z','Trade↩
   ',25283,39430,0,0,0,0,0;
% 'AGLJ.J','07-NOV-2013','17:10:12.394170','2013-11-07T17:10:12.394Z','Trade↩
   ',25602,8000,0,0,0,0,0;
% 'AGLJ.J','07-NOV-2013','17:26:44.908276','2013-11-07T17:26:44.908Z','Trade↩
   ',25485,10460,0,0,0,0,0;
% 'AGLJ.J','07-NOV-2013','17:26:45.448335','2013-11-07T17:26:45.448Z','Trade↩
   ',25548,20358,0,0,0,0,0}

% {'GRTJ.J','06-NOV-2013','16:49:19.253191',2483,429;
% 'GRTJ.J','06-NOV-2013','17:00:02.488567',2480,599771;
% 'GRTJ.J','06-NOV-2013','17:24:55.045941',2477,76385}

% 3) No trade occurs over consecutive time bars- thus no price shift nor volume ↩
   will be
% entered into these time bars
% {'GRTJ.J','01-NOV-2013','11:33:57.686807',2533,1008;'GRTJ.J','01-NOV↩
   -2013','11:37:11.160757',2533,1399}

% 4) Empty time bars where no trade occurs(rows 2 and 3 below)
% {'01-NOV-2013','10:16:00.000','10:17:00.000',[2x5 cell' char(10) '],[];
% '01-NOV-2013','10:17:00.000','10:18:00.000',[],[];'01-NOV↩
   -2013','10:18:00.000','10:19:00.000',[],[];
% '01-NOV-2013','10:19:00.000','10:20:00.000',[],[];'01-NOV↩
   -2013','10:20:00.000','10:21:00.000',[],[];
% '01-NOV-2013','10:21:00.000','10:22:00.000',[3x5 cell' char(10) '],[]}


%% 3. Clear Workspace
% This section prepares the workspace for the implementation of the script.
close all;      % close any open figures
clc;            % clear command window
format long g;  % formating for output on comand window
format compact; % formating for output on comand window


%% 4. Path Setup
% Set the project paths
[projectPath] = pwd;
% Add the project path so that the script sees the necessary functions
addpath(projectPath);
userpathstr = userpath;
userpathstr = userpathstr(~ismember(userpathstr,';'));
%projectpath = 'AMF/AMF_FinalProject_NJM_715798';   %path for personal computer
projectpath = 'AMF_FinalProject_NJM_715798';   %path for AMF machine
addpath(fullfile(userpathstr,projectpath,'Functions'));
addpath(fullfile(userpathstr,projectpath,'Scripts'));
addpath(fullfile(userpathstr,projectpath,'html'));

% # Filepath to input data
% Path to where the data are stored
```

```
%[datafilepath] = 'C:\Users\Nicholas\Documents\MATLAB\AMF\↩
    AMF_FinalProject_NJM_715798\Data\Transactions';  %path to data on personal ↩
    computer
%[datafilepath] = '\\AMF\Data\Transactions\JSE';   %path to CSAM cluster data
[datafilepath] = 'Z:\Data\Transactions\JSE'; % This is the path for data on ↩
    AMFcluster

%%%%%%%%%%%%%%%%%%%%%%%%%
%% 5. Process the Data %%
% Trade entries are sorted into buckets of equal size, V, which is
% specified to be the average daily volume over the entire period. The number
% of buckets is chosen to 50. Once all 50 buckets are complete, the VPIN metric
% is computed. If the last trade needed to complete a bucket is for a
% size greater than required, the excess size is given to the next bucket.
% The VPIN is then computed after each bucket is filled. This implies,
% for example, if bucket 51 is filled, we will drop the first bucket and
% computed the VPIN for buckets 2-51. Thus VPIN is calculated in volume-time
% which is argued to be necessary in the high frequency world. Perform steps to
% remove extraneous data and data falling outside the continuous trading session.

noofstocks = 7; %specify number of stocks to be analysed and processed
VPINstore = cell(1,noofstocks);  %initialise a cell array to store the timenars ↩
    for each stock for the entire period
VBS = zeros(1,noofstocks);  %initiate vector to store VBS for each stock
RICstore = cell(1,noofstocks);  %initialise array to store RIC's
outliers = zeros(1,noofstocks);   %initialise vector to store number of outliers

for n = 1:noofstocks  % Loop over stocks
 %%%%%%%%%%%%%%%%%%
 %% 6. Load Data %%
 % The data from the matfile for the specified dates and specified stock is
 % loaded on each iteration of the loop over stocks. The createdate.m function
 % is used to generate the required dates from these matfiles.

  % Step 1: Generate dates using createdates.m function
  %[startDate] = '01-Jan-2013';  %start date for cluster data
  %[endDate] = '03-Jun-2013';  %end date for cluster data
  [startDate] = '01-Nov-2013';  %start date for test data
  [endDate] = '07-Nov-2013';  %end date for test data
  [TradingDayStart]    = []; % Defaults to 08:30:00
  [TradingDayEnd]      = []; % Defaults to 17:30:00
  [Type]               = []; % Defaults to week
  [s,e] = createdates(startDate,endDate,TradingDayStart,TradingDayEnd,Type); %↩
     call createdates function to generate dates

  % Step 2: Predefine variable to store matfile names for each stock
  matfilename = cell(length(e),1);   %create a cell array to store the names of ↩
     the matfiles
  storedata = cell(length(e),1); %create cell array to store trades for all ↩
     stocks for entire period
  storequotes = cell(length(e),1); %create cell array to store trades for all ↩
     stocks for entire period

  for k = 1:length(e)  %loop over weeks
     % Step 3: Pre-define path to the required folder
    folders = dir(datafilepath);  %directory of the datafilepath
    folderpath = cell(noofstocks,1);  %create a cell array to store the paths for↩
        each of the stock folders

    % Step 4: construct the matfile name for the required stock and week
    % and extract the data from that matfile
    weekStart = datestr(s{k,1},'ddmmmyyyy_HHMM');
    weekEnd = datestr(e{k,1},'ddmmmyyyy_HHMM');
    folderpath{n} = strcat(datafilepath,'\',folders(28+n).name);  %path to ↩
       specific stock folder
    [RIC] = folderpath{n}(end-5:end);  %extract stock RIC
    matfilename{k} = strcat(RIC,'-Transactions-',weekStart,'-to-',weekEnd,'.mat')↩
       ;  %retrieve name of required matfile for each week as we iterate over the↩
```

```
        weeks loop

    % Step 5: Load the required order book data
    [data] = load(fullfile(datafilepath,RIC,matfilename{k})); %load the matfile

    % Step 6: Extract only data
    data = data.data;

    %-----------------------------------------------------------------------
    %%%%%%%%%%%%%%%
    %% 7. Clean %%
    % Remove Auction data and get the date-time of all trades in order to extract
    % trade corresponding to the continuos trading session (9:00-16:50).
    % Also remove any trades which have a volume greater than 5 standard
    % deviations form the mean.

    % Step 1: Remove all rows which contain NAN's and zeros throughout. Also
    %remove all rows which have an auction price formation.
    data(any(strcmp(data(:,1:5),'')|2) | any(isnan(cell2mat(data(:,6:end)))|2),:)↩
        = [];
    data(strncmpi(data(:,5),'Auction',1),:) = [];

    % Step 2: Get the date-time of the trades
    [yr,mt,dy,~,~,~] = datevec(data(:,2)); %find year, month and day
    [~,~,~,hr,mn,sc] = datevec(data(:,3)); %find hour, minute and second
    [tradedatetime] = datenum(yr,mt,dy,hr,mn,sc); %create vector of dates and ↩
       times

    % Step 3: Remove all events between before 09h00 and after 17h00
    [removalindextime] = (hr<9 | hr>=17| (hr==16&mn>50)); %find indices of such ↩
       events
    tradedatetime(removalindextime) = []; %remove entries from date vector
    data(removalindextime,:) = []; %remove entries from data

    % Step 4: Remove unnecessary variables
    clearvars removalindextime yr mt dy hr mn sc tradedatetime

    % Step 5: Remove trades which have a volume greater than 3 standard
    % deviations form the mean
    nooftradesbefore = length(data); %calculate the number of trade entries ↩
       before removing outliers
    data(mean(cell2mat(data(:,7))) + 5*std(cell2mat(data(:,7)))< cell2mat(data↩
       (:,7)),:) = []; %remove all rows with a volume greater than 3 standard
    % deviations form the mean
    nooftradesafter = length(data); %calculate the number of trade entries after ↩
       removing outliers
    noofoutliers = nooftradesbefore-nooftradesafter; %number of ouliers removed

    %-----------------------------------------------------------------------
    %%%%%%%%%%%%%%%%%%
    %% 8. Lee-Ready %%
    % Implement the Lee-Ready algorithm to distinguish buyer-initiated from
    % seller-initiated trades. We first check if the transaction obeys the
    % quote rule and if that fails we then implement the tick rule. The quote
    % rule states: Transactions which occur at prices higher(lower) than the
    % midquote are classified as being buyer-initiated(buyer-initiated). If
    % the transaction does not obey the quote rule we move onto the tick
    % rule which states: Transactions which occur at a price that equals the
    % midquote but is higher(lower) than the previous transaction price are
    % classified as being buyer-initiated (seller-initiated).If there is no price
    % change but the previous tick change was up (down), then the trade is
    % classified as a buy (sell).

    % Step 1: Create a new data set which contain all quote entries from ↩
       compactdata and
    % get extraction indices to extract daily data from this dataset.
    quotedata = data(strncmpi(data(:,5),'Quote',1),:); %all quote entries
```

```
    [~,uniqueday] = unique(datenum(quotedata(:,2)),'first'); %find the indices ↩
       for each day
    [daystart] = uniqueday(1:end);  %find the indices for the start of each day
    [dayend] = [uniqueday(2:end)-1;size(quotedata,1)]; %find the indices for the ↩
       end of each day

    % Step 2: Drop down all bid/ask prices and volumes into cells which have zero
    % bid/ask prices and volumes and then calculate the midquote for each
    % quote entry
    quotedata(:,13) = {0}; %create column
    for i = 1:length(dayend)   %loop over days
       for j = (daystart(i)+1):dayend(i)  %loop from beginning of day to end of ↩
          day
          if cell2mat(quotedata(j,9:10)) == 0
             quotedata(j,9:10) = quotedata(j-1,9:10);  %find the previous best ↩
                bid price and volume if the cells contain zeros
          elseif cell2mat(quotedata(j,11:12)) == 0
             quotedata(j,11:12) = quotedata(j-1,11:12);  %find the previous best ↩
                ask price and volume if the cells contain zeros
          end
          if quotedata{j,9} ~= 0 && quotedata{j,11} ~= 0
            quotedata{j,13} = (quotedata{j,9} + quotedata{j,11})/2;  %Calculate ↩
               the mid-quotes
          end
       end
    end
    data(:,13) = {0}; %create column of zeros
    data(strncmpi(data(:,5),'Quote',1),:) = quotedata;  %Fill the quotes back ↩
       into the alldata set
    clearvars uniquedayq daystartq dayendq

    % Step 3: Get extraction indices to extract daily data from compactdata set.
    [~,uniqueday] = unique(datenum(data(:,2)),'first');  %find the indices for ↩
       each day
    [daystart] = uniqueday(1:end);
    [dayend] = [uniqueday(2:end)-1;size(data,1)];

    % Step 4: Find all rows in which a trade occurred as the first entry in the ↩
       data set on a particular day
    for i = 1:length(dayend)
      if strncmpi(data(daystart(i),5),'Trade',1) == 1
         j=0;
         while strncmpi(data(daystart(i)+j,5),'Trade',1) == 1
           data(daystart(i)+j,5) = {'Remove'};  %label all rows in which a trade ↩
              occurred as the first entry in the data
                                                          %set on a ↩
                                                             particular day ↩
                                                             with 'Remove' ↩
                                                             for removing ↩
                                                             later
           j = j+1;
         end
      end
    end

    data(strncmpi(data(:,5),'Remove',1),:) = [];

    % Step 5: Recalculate indices for daily data
    [~,uniqueday] = unique(datenum(data(:,2)),'first');  %recalculate the indices ↩
       for each day since transaction that were the first entry of the day were ↩
       removed
    [daystart] = uniqueday(1:end);  %find the indices for the start of each day
    [dayend] = [uniqueday(2:end)-1;size(data,1)];  %find the indices for the end ↩
       of each day

    % Step 6: Implement the Lee-Ready algorithm
    data(:,14) = {''};  %create column of zeros
    for i = 1:length(dayend)  %loop over the days
```

```
    for j = (daystart(i)+1):dayend(i)  %loop from the beginning to the end of ↩
       the day
      if strncmpi(data(j,5),'Trade',1) == 1   %check if a transaction has taken ↩
         place
         % Quote Rule
          if  strncmpi(data(j-1,5),'Quote',1) == 1 && isempty(cell2mat(data(j↩
             -1,13))) == 1  %row before the transaction is a quote and does ↩
             not have a midquote since it is at the beginning of the day

             if data{j,6} == data{j-1,9}
                data{j,14} = -1;  %since the bid was hit, it is a market sell
             else
                data{j,14} = 1;  %since the ask was hit, it is a market buy
             end
          elseif strncmpi(data(j-1,5),'Quote',1) == 1 && data{j,6} > data{j↩
             -1,13}  %the row before transaction is a quote and trade price is ↩
             greater than the midquote
              data{j,14} = 1;  %classify as buy
          elseif strncmpi(data(j-1,5),'Quote',1) == 1 && data{j,6} < data{j↩
             -1,13}  %the row before transaction is a quote and trade price is ↩
             less than the midquote

              data{j,14} = -1;  %classify as sell
         % Tick Rule
          elseif strncmpi(data(j-1,5),'Quote',1) == 1 && data{j,6} == data{j↩
             -1,13}  %the row before transaction is a quote and trade price is ↩
             equal to the midquote

             if cell2mat(data(find([data{daystart(i):j-1,6}],1,'last')+ ↩
                daystart(i)-1,6)) <...
                data{j,6}  %current trade is greater than the most recent ↩
                   trade
                data{j,14} = 1;  %classify as buy
             elseif cell2mat(data(find([data{daystart(i):j-1,6}],1,'last')+ ↩
                daystart(i)-1,6)) >...
                data{j,6}  %current trade is less than the most recent trade
                data{j,14} = -1;  %classify as sell
             elseif cell2mat(data(find([data{daystart(i):j-1,6}],1,'last')+ ↩
                daystart(i)-1,6)) ==...
                data{j,6}  %current trade is equal to the most recent trade
                data{j,14} = cell2mat(data(find([data{daystart(i):j-1,6}],1,'↩
                   last')+ daystart(i)-1,14));
             end
          elseif strncmpi(data(j-1,5),'Trade',1) == 1  %two trades happen in ↩
             succession
             if data{j,6} == data{j-1,6}  %first trade price is equal to the ↩
                second trade price
                data{j,14} = data{j-1,14};  %classify transaction the same as ↩
                   what the first one was classified
             elseif data{j,6} > data{j-1,6}  %first trade price is greater ↩
                than the second trade price
                 data{j,14} = 1;  %classify as buy
             elseif data{j,6} < data{j-1,6}  %first trade price is less than ↩
                the second trade price
                 data{j,14} = -1;  %classify as sell
             end
          end
      end
    end
  end

  %------------------------------------------------------------------------
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %% 9. Trade and Quote Data %%
  % Extract only the trade data for entire period for
  % the nth stock.

  % Step 1: Extract only trades
```

```
    trades = data(strncmpi(data(:,5),'Trade',1),:); %extract only trade data

    % Step 2: Remove all unneccessary data- columns 8 to 12,rows
    % with zero trade volume and rows 4 and 5 which contain information we
    % will not need.
    trades(:,8:13) = [];
    trades(cell2mat(trades(:,7))==0,:) = [];
    trades(:,4:5) = [];

    % Step 3: store data for the nth stock for the kth week
    storedata{k,1} = trades;
    storequotes{k,1} = quotedata;

end

quotedata = vertcat(storequotes{1:end,1}); %all quote entries
tradedata = vertcat(storedata{1:end,1}); %extract trades for nth stock for ↩
   entire period
clearvars trades storedata storequotes data

%--------------------------------------------------------------------------
%%%%%%%%%%%%%%%%%%%%
%% 10. VBS and ADV %%
% Calculate volume bucket size(VBS) based on average daily volume(ADV)over the ↩
   entire period.
% We will choose a sample of 50 buckets per calculation of
% VPIN as specified by the authors. The VBS will then be the ADV divided
% by the number of buckets(50).

% Step 1: Calculate indices for daily data
[~,uniqueday] = unique(datenum(tradedata(:,2)),'first'); %recalculate the ↩
   indices for each day since transaction that were the first entry of the day ↩
   were removed
[daystart] = uniqueday(1:end); %find the indices for the start of each day
[dayend] = [uniqueday(2:end)-1;size(tradedata,1)]; %find the indices for the ↩
   end of each day

dailyvol = zeros(length(daystart),1); %initialise daily volume for the nth ↩
   stock

for i = 1:length(daystart) %loop over days
  dailyvol(i,1) = sum(cell2mat(tradedata(daystart(i):dayend(i),5))); %compute ↩
     total daily volume
end
avedailyvol = sum(dailyvol(:,1))/length(daystart); %compute average daily ↩
   volume for the nth stock

% Step 2: Compute the bucket size and define the number of buckets
noofbuckets = 50; %define the number of buckets used to calulate the VBS
VBS(1,n) = round(avedailyvol/noofbuckets,0); %define VBS to be one fiftieth of↩
    average daily volume
clearvars dayend daystart uniqueday dailyvol

%--------------------------------------------------------------------------
%%%%%%%%%%%%%%%%%%%%%%%%
%% 11. Volume Buckets %%
% Iterate through the trades and sum up the volumes of
% of the entries until the VBS threshold is reached. Assign the given set of
% trades the current bucket index. If the last trade in the given set of
% trades which make up the volume of the tau-th bucket is of volume greater ↩
   than
% that required to fill the volume bucket, the last trade is broken up into
% 2 parts where one will fill the most recent bucket and the excess volume will↩
    be given
% to the next bucket.

% Step 1: initiate variables needed for the volume bcuketing
bucketind = 0; %initialise index for finding the index of time bars for which a↩
```

```
    bucket will be completed
tau = 1; %initialise index for storing the time bar index at each completion ↩
   of a bucket

% Step 2: Set up a loop over entire sample of data in order to find
% indices for time bars where buckets will strart and end. The
% excess volume in a given bucket is transferred to the next bucket along
% with the corresponding price change
i =0 ;
while i < length(tradedata)
  if sum(cell2mat(tradedata((bucketind(tau)+1):i,5))) >= VBS(1,n) %find index ↩
     for time bar where a bucket will be completed
    excess = sum(cell2mat(tradedata((bucketind(tau)+1):i,5))) - sum(cell2mat(↩
       tradedata((bucketind(tau)+1):(i-1),5))) - VBS(1,n) + sum(cell2mat(↩
       tradedata((bucketind(tau)+1):(i-1),5))); %calculate the excess volume in↩
        the current bucketind which will be given to next bucket
    tradedata{i,5} = VBS(1,n) - sum(cell2mat(tradedata((bucketind(tau)+1):(i-1)↩
       ,5))); %complete the current volume bucket with the required amount of ↩
       volume
    tradedata((i+1):(end+1),:) = tradedata((i):(end),:); %create an extra time↩
        bar with the same times as the i-th time bar since extra volume must be↩
        carried over to next bucket
    tradedata{i+1,5} = excess; %amount of volume the next bucket has received ↩
       from previous bucket
    tradedata((bucketind(tau)+1):i,7) = {tau}; %assign the trade a bucket ↩
       index
    tau = tau + 1; %add one to bucket counter
    bucketind(tau) = i; %store the index of the time bar for which the bucket ↩
       will be filled
  end
i = i + 1; %add one to the while loop counter
end

%-------------------------------------------------------------------------
%%%%%%%%%%%%%%
%% 12. VPIN %%
% Iterate over the trade bar data and as soon as the threshold of 50 buckets
% is reached and compute the VPIN metric.
% For each bucket, sum up the transaction buy and sell volumes and print the
% order imbalance at the time the last trade occurred. Compute VPIN and extract↩
    the
% midquote at the time VPIN was printed.

% Step 1: Gather necessary inputs for calculation
[~,uniquebucket] = unique(cell2mat(tradedata(:,7)),'first'); %find the total ↩
   number of buckets in the time bar data
[bucketstart] = uniquebucket(1:end); %find the indices for the start of each ↩
   day
[bucketend] = [uniquebucket(2:end)-1;find(cell2mat(tradedata(:,6)),1,'last')];
totalbuckets = unique(cell2mat(tradedata(:,7)),'first'); %total number of ↩
   buckets in the sample for the n-th stock
noofvpins = length(totalbuckets) - 50 + 1; %calculate the number of times VPIN↩
    will be updated
VPIN = cell(noofvpins,3); %initialise a vector to store dates, times and VPIN ↩
   metric
samplesize = 50; %define the number of buckets used in order to update VPIN

% Step 2: Compute serial dates for quote data
[yrq,mtq,dyq,~,~,~] = datevec(quotedata(:,2)); %find year, month and day
[~,~,~,hrq,mnq,scq] = datevec(quotedata(:,3)); %find hour, minute and second
[tradedatetimequotes] = datenum(yrq,mtq,dyq,hrq,mnq,scq); %create vector of ↩
   dates and times
clearvars uniquebucket hrq dyq mnq mtq scq yrq

for i = 1:noofvpins %loop over the number of updates of VPIN
  for j = (bucketstart(i)):bucketend(i+samplesize-1) %loop form 1st bucket in ↩
     VPIN calculation to the last bucket(50) in the calculation
    if tradedata{j,6} == 1
```

```
        tradedata{j,8} = tradedata{j,5};  %compute the buy volume for the jth ↩
            volume bucket
      elseif tradedata{j,6} == -1
        tradedata{j,9} = tradedata{j,5};  %compute the sell volume for the jth ↩
            volume bucket
      end
    end

    for j = i:(i+samplesize-1)  %loop over the current sample of buckets which we↩
        are computing VPIN for
      tradedata{bucketend(j),10} = abs(sum(cell2mat(tradedata(bucketstart(j):↩
          bucketend(j),9))) - sum(cell2mat(tradedata(bucketstart(j):bucketend(j)↩
          ,8)))); %compute Order Imbalance for each bucket in the i-th calculation↩
           of VPIN
    end
    VPIN{i,3} = sum(cell2mat(tradedata((bucketstart(i)):bucketend(i+samplesize-1)↩
        ,10)))/(samplesize*VBS(1,n));  %compute VPIN for the i-th update of the ↩
        metric
    VPIN{i,2} = tradedata(bucketend(i+samplesize-1),3); %enter in the time for ↩
        the corresponding VPIN calculation
    VPIN{i,1} = tradedata(bucketend(i+samplesize-1),2);  %enter in the date of ↩
        the corresponding VPIN calculation

    % Find mid-quote when VPIN is printed
    % Step 1: Get the date-time of current VPIN
    [yrvpin,mtvpin,dyvpin,~,~,~] = datevec(VPIN{i,1}); %find year, month and day
    [~,~,~,hrvpin,mnvpin,scvpin] = datevec(VPIN{i,2});  %find hour, minute and ↩
        second
    tradedatetimeVPIN = datenum(yrvpin,mtvpin,dyvpin,hrvpin,mnvpin,scvpin);  %↩
        create vector of dates and times

    VPIN{i,4} = cell2mat(quotedata(find(tradedatetimequotes(:)<=tradedatetimeVPIN↩
        ,1,'last'),13)); %find the index of the most recent quote which occurred ↩
        before VPIN was updates
  end

  % Store results
  VPINstore{1,n} = VPIN;  %store the timebars for the n-th stock
  RICstore{1,n} = tradedata{1,1};  %store the RIC for the n-th stock for ↩
      retrieval
  outliers(1,n) = noofoutliers;  %store the number of outliers for the nth stock

  clearvars tradedata tradedatetimequotes quotedata tradedatetimeVPIN...
      VPIN yrvpin dyvpin bucketend bucketstart
end
%=========================================================================
```

B AMF High Performance Computing Cluster

The two scripts below were used in order to execute the data processing scripts on the AMF MDCS cluster:

B.1 Submission Script

The data processing script was submitted to the MATLAB Job Sheduler by the Submission Script in order to execute the attached scripts on the required data.

```
%% MATLAB Job Scheduler Batch Submission Script File
% AUTHORS: M Harvey (michael.harvey@students.wits.ac.za)
% REVISED BY: D Platt (donovan.platt@students.wits.ac.za)
% DATE: 12-Sep-2016 13:49:00
% VERSION: mjs_client_submit_script_v001ba
%
% +-------------------------------------------------------------------- +
```

```
% THIS SCRIPT IS THE SCRIPT USED FOR SUBMITTING A SCRIPT FROM A REMOTE
% CLIENT SESSION FOR EXECUTION ON THE AMF HIGH PERFORMANCE COMPUTING
% CLUSTER. THE SCRIPT MAY CONTAIN PARFOR LOOPS.
% +-------------------------------------------------------------------+
%
% # PROBLEM SPECIFICATION: This script is to be run in the client MATLAB
% session on the remote client computer (i.e. on one of the computers in
% the AMF lab or on the remote client machine "amfentry.ms.wits.ac.za"
% which is accessed via Windows Remote Desktop). This script is used to
% submit a single job to the MATLAB Job Scheduler (MJS) to be run on the
% AMF high performance computing cluster: "AMFcluster". In particular this
% script uses the BATCH command to submit a script or function (i.e. a job)
% to the MJS to be run on worker(s) in the AMF cluster. This script
% itself must NOT be submitted to the cluster using the batch command.
%
% This script will specifically pertain to the process of submitting jobs
% to the AMF cluster using the batch command in the remote MATLAB
% client on "amfentry.ms.wits.ac.za". This is because the cluster profile
% has already been appropriately set up on "amfentry.ms.wits.ac.za" and it
% does not need to be configured by the user. For a cluster to be
% accessible from a client machine that cluster's profile must be loaded
% onto the client machine.
%
% The two important MATLAB functions that this script makes use of are:
%
% * PARCLUSTER: This command creates a cluster object in the client's
% MATLAB workspace. This cluster object is used to store any information
% that must be sent between the remote cluster and your MATLAB client.
% EG: [hpcBladeCluster] = parcluster('AMFcluster');
% * BATCH: This command begins an automated process that will connect to
% the AMF cluster, submit a job to MJS, and initialize the MATLAB
% Distributed Computing Server (MDCS).
%
% NOTE! THE SECTIONS IN THIS SCRIPT THAT REQUIRE USER INPUT ARE INDICATED
% BY:
%                       ************************
%                  --> *       USER INPUT       *
%                       ************************
%
% For ease of use all the required user inputs are specified in STEP 2 and
% STEP 3.
%
% # DATA SPECIFICATION: This particular script does not require any data.
%
% # CONFIGURATION CONTROL:
%
% EG: C:\Users\<USERNAME>\Documents\MATLAB\<PROJECT>\<SUBDIRECTORY>
%
%       ~\MATLAB\mjs
%       ~\MATLAB\mjs\scripts
%       ~\MATLAB\mjs\latex
%
% # VERSION CONTROL: This script was updated by Donovan Platt to be
% consistent with migration from CSAMMJS to AMFcluster.
%
% INPUTS
% After ensuring that this script file has the correct paths specified (see
% STEP 1 below) the user is required to specify the following inputs:
% # clusterUserName
% # myJobName
% # clusterProfileName
% # scriptName
% # poolSize
% # AdditionalPaths
% # AttachedFiles
% # emailMe
% # myWitsEmailAddress
% # myWitsUsername
```

```
% # myWitsPassword
%
% OUTPUTS
% A MAT-file with naming configuration control:
%
%   mjsJobID_<MYJOBID>_<MYJOBNAME>.mat
%
% that is saved in the same directory as this script and contains:
% # myJobID - the job's MJS job ID number
% # myJobName - the user specified job name
% # clusterProfileName - the cluster profile of the job
%--------------------------------------------------------------------------%



%% STEP 0: ENSURE THAT THIS SCRIPT RUNS IN ITS DIRECTORY
% Get info
tmp = matlab.desktop.editor.getActive;
% Set cd
cd(fileparts(tmp.Filename));
% Clear
clear tmp;
%--------------------------------------------------------------------------%



%% STEP 1: SET UP WORKSPACE
% The following commands prepare the client session of MATLAB by clearing
% all variables from the client session workspace, closes all
% open figures and clears the command window of any displayed output.
clear, close, clc;
%--------------------------------------------------------------------------%



%% STEP 2: SET UP PATHS
%
%                        ************************
%                        *       USER INPUT     *
%                        ************************
%
% This script must be placed in the folder that contains the script that
% will be submitted to the AMF cluster for remote processing. The paths
% that are specified here pertain to the project as it is stored on the
% CLIENT MACHINE. Note, the client machine is the machine used to submit a
% job to the remote cluster. Currently this would either be a computer in
% the AMF computer lab or "amfentry.ms.wits.ac.za".
%
% IMPORTANT: The paths specified here are extremely important because they
% will be used in the submission of your job to the remote AMF cluster.
% What we mean by this is that if you have some project structure with
% separate folders contain your scripts, functions, data then we must
% ensure that ALL WORKERS in the cluster have access to what is required.
% Specifically, we mean the following:
%
% SCRIPTS: The script file is what we submit with the batch command. This
% submission script (mjs_client_submit_scripts_v001ba) must be in the same
% folder as this script. Obviously this varies for each user; you may have
% NO file hierarchy, with all the required functions and data in a single
% project folder. The point is you need to be extremely aware of where all
% the components of your project are and what you must include in the job
% object for correct execution on the remote cluster. Regarding scripts,
% we class them as follows:
%
%   * INDEPENDENT: This is the trivial case. It contains everything
%   necessary to run correctly. I.e. it does not require any extra input
%   like data or reference any functions. This case is trivial because we
%   do not need to tell the cluster anything else when submitting the job;
```

```
%   we just need to tell it which script we require it to run.
%   * DEPENDENT: This case requires thought. It applies to projects with or
%   without a file hierarchy. If a script cannot run without some external
%   input (i.e. it must either reference some external data set or it makes
%   use of other scripts and functions) then it is classed as a DEPENDENT
%   script. In this situation we have to tell the cluster where EXACTLY it
%   finds ALL the required FUNCTIONS as well as ALL referenced DATA. To do
%   this we merely need to ATTACH ALL NECESSARY FILES when submitting the
%   job to the cluster. To do this we require all the paths to the
%   necessary inputs.
%
% Set up the configuration control of the PROJECT on the CLIENT machine.
%
% [projectPath]       = 'C:\Users\<USERNAME>\Documents\MATLAB\<USER>\<PROJECT>'
% [scriptsFilePath]   = 'C:\Users\<USERNAME>\Documents\MATLAB\<USER>\<PROJECT>\↩
   scripts'
% [functionsFilePath] = 'C:\Users\<USERNAME>\Documents\MATLAB\<USER>\<PROJECT>\↩
   functions'
% [dataFilePath]      = 'C:\Users\<USERNAME>\Documents\MATLAB\<USER>\<PROJECT>\↩
   data'
% [figFilePath]       = 'C:\Users\<USERNAME>\Documents\MATLAB\<USER>\<PROJECT>\↩
   figures'
%
% A note on getting the PWD (present working directory):
% PWD specifies the FULL FILE PATH to the CURRENT DIRECTORY. Hence if your
% scripts are in a separate scripts directory PWD will return:
% pwd = '~\MATLAB\<PROJECT>\scripts'
% Since this project has a directory structure we use the command FILEPARTS
% to get the path to the project directory. That is:
% fileparts(pwd) = '~\MATLAB\<PROJECT>'
% [projectPath] = fileparts(pwd);
[projectPath] = pwd;
% Add path
addpath(projectPath)
%-------------------------------------------------------------------------%



% + ------------------------------------------------------------------------ +
% IMPORTANT! WHEN RUNNING A SCRIPT ON THE REMOTE CLUSTER (THAT IS,
% SUBMITTING A JOB TO THE REMOTE CLUSTER) ANY FIGURES AND MAT FILES THAT
% ARE SAVED AT ANY POINT IN YOUR SCRIPT WILL NOT BE SAVED ON THE CLIENT
% MACHINE. WHEN YOU SUBMIT A JOB TO A REMOTE CLUSTER, THE JOB RUNS ON THE
% REMOTE CLUSTER AND ANY DATA OR FIGURES THAT ARE ITERATIVELY SAVED ARE
% SAVED IN A TEMPORARY JOB DATA STORAGE LOCATION ON THE CLUSTER. CURRENTLY
% THE AMF CLUSTER, AS ACCESSED VIA THE MATLAB JOB SCHEDULER, DOES NOT
% OFFER THE USER THE OPTION OF RETRIEVING ANY MAT FILES OR FIG FILES THAT
% ARE SAVED ON THE CLUSTER WHILE THE JOB RUNS. THIS MEANS THAT WHEN
% DEVELOPING A PROJECT THAT WILL BE RUN ON THE AMF REMOTE CLUSTER THE USER
% MUST BE VERY AWARE OF WHAT DATA MUST BE RETRIEVED AFTER THE JOB IS
% COMPLETED. WHEN A JOB IS DONE, THE RETRIEVAL SCRIPT WILL ONLY ALLOW THE
% USER TO RETRIEVE THE WORKSPACE OF THAT COMPLETED JOB. THIS MEANS THAT ALL
% THE OUTPUT THE USER WISHES TO RETRIEVE MUST BE IN THE WORKSPACE WHEN THE
% JOB IS COMPLETED. HENCE IN THE DEVELOPMENT STAGE THE USER SHOULD CHECK TO
% SEE THAT AFTER A SCRIPT IS RUN THE WORKSPACE ONLY CONTAINS THE DESIRED
% OUTPUTS. TO THIS END, THE USE OF THE CLEARVARS COMMAND AT THE END OF A
% SCRIPT IS STRONGLY RECOMMENDED.
% + ------------------------------------------------------------------------ +



%% STEP 3: USER INPUTS (SET UP THE PARAMETERS OF THE JOB SUBMISSION)
%
%                         *************************
%                         *       USER INPUT      *
%                         *************************
%
% In this section the user must define all the inputs and parameters of the
```

```
% job submission. These are the only inputs required and define how the job
% is be submitted to the MJS to be run on the (remote) cluster.

% # DEFINE YOUR AMF CLUSTER USERNAME
% Give your AMF MDCS cluster username. The configuration control for this
% is your first initial plus your surname, all in lower case
% [clusterUserName] = <INITIAL><SURNAME>
% EG: [clusterUserName] = 'jbrown';
[clusterUserName] = 'nmurphy';

% # DEFINE A JOB NAME
% Give your job a descriptive name with configuration control:
% [myJobName] = <INITIALS>_<PROJECTNAME>
% EG: [myJobName] = 'mh_parForTest';
[myJobName] = 'JanJunLR1_16';
%[myJobName] = 'JanJunBC1_16';

% # CHOOSE THE CLUSTER PROFILE NAME
% EG: [clusterProfileName] = 'local'
% EG: [clusterProfileName] = 'AMFcluster'
[clusterProfileName] = 'AMFcluster';

% # SCRIPT NAME
% Declare the name of the script without the extension (i.e. DO NOT put the
% '.m' in the script name.)
% [scriptName] = 'my_script_name';
[scriptName] = 'data_processing_script_LR_715798';
%[scriptName] = 'ata_processing_script_BC_715798';

% # POOL SIZE
% An integer specifying the number of workers to make into a parallel pool
% for the job IN ADDTION to the worker running the batch job itself. It is
% this pool that will be used for execution of statements such as parfor
% and spmd that are inside the batch code (i.e. the code submitted to the
% cluster). Because the pool requires N workers in ADDITION to the worker
% running the batch, there must be at least N+1 workers available on the
% cluster.
[poolSize] = 4;

% # ADDITIONAL PATHS
% Include all the additional files paths of your project as they appear on
% the client machine.
% For example:
% [AdditionalPaths] = {...
%     'C:\Users\<USERNAME>\Documents\MATLAB\<USER>\<PROJECT>';...
%     'C:\Users\<USERNAME>\Documents\MATLAB\<USER>\<PROJECT>\scripts';...
%     'C:\Users\<USERNAME>\Documents\MATLAB\<USER>\<PROJECT>\functions';...
%     'C:\Users\<USERNAME>\Documents\MATLAB\<USER>\<PROJECT>\data';...
%     };
%[AdditionalPaths] = {...
%    'C:\Users\amfentry\Desktop\dataparfortestproject';...
%    };
 [AdditionalPaths] = {...
    'C:\Users\base\Documents\MATLAB\AMF_FinalProject_NJM_715798\Scripts';
    'C:\Users\base\Documents\MATLAB\AMF_FinalProject_NJM_715798\Functions';
    };
% # ADDITIONAL FILES TO BE SUBMITTED ALONG WITH THE JOB
% If your script requires the use of any functions or scripts the paths to
% ALL of these components must be explicitly stated here.
% If your job also requires any SMALL input data sets these files should be
% attached here. NB! THESE MUST BE SMALL!!!
% For example:
% [AttachedFiles] = {...
%         'C:\Users\<USERNAME>\Documents\MATLAB\<USER>\<PROJECT>\scripts\↩
   myScriptA.m';...
%         'C:\Users\<USERNAME>\Documents\MATLAB\<USER>\<PROJECT>\scripts\↩
   myScriptB.m';...
%         'C:\Users\<USERNAME>\Documents\MATLAB\<USER>\<PROJECT>\functions\↩
```

```
    myFunctionA.m';...
%           'C:\Users\<USERNAME>\Documents\MATLAB\<USER>\<PROJECT>\functions\↩
    myFunctionB.m';...
%           'C:\Users\<USERNAME>\Documents\MATLAB\<USER>\<PROJECT>\data\my_MAT_file↩
    .mat';...
%           'C:\Users\<USERNAME>\Documents\MATLAB\<USER>\<PROJECT>\data\my_CSV_file↩
    .csv';...
%           'C:\Users\<USERNAME>\Documents\MATLAB\<USER>\<PROJECT>\data\my_TXT_file↩
    .csv';...
%           };
% If there are no extra components required set this input to empty:
% For example:
% [AttachedFiles] = {};
%[AttachedFiles] = {...
%     'C:\Users\amfentry\Desktop\dataparfortestproject\dataparfortestscript.m' ...
%     'C:\Users\amfentry\Desktop\dataparfortestproject\bardata.m'
%     };
[AttachedFiles] = {...
    %'C:\Users\base\Documents\MATLAB\AMF_FinalProject_NJM_715798\Scripts\↩
       data_processing_script_BC_715798.m';
     'C:\Users\base\Documents\MATLAB\AMF_FinalProject_NJM_715798\Scripts\↩
        data_processing_script_LR_715798.m';
    'C:\Users\base\Documents\MATLAB\AMF_FinalProject_NJM_715798\Functions\↩
       createdates.m';
    };
%--------------------------------------------------------------------------%



%% STEP 5: SET UP THE CLUSTER OBJECT
% Create a cluster object "hpcBladeCluster" using the appropriate cluster
% profile (the AMF MATLAB HPC Cluster uses 'AMFcluster').
% The "parcluster" command creates a cluster object in the client's MATLAB
% workspace. This cluster object is used to store any information that must
% be sent between the remote cluster and your MATLAB client.
% EG: [hpcBladeCluster] = parcluster('AMFcluster');
[hpcBladeCluster] = parcluster(clusterProfileName);
% Print details of job object to Command Window
fprintf('+ ------------------------------------------------------------ +\n\n')
fprintf('  CLUSTER OBJECT PROPERTIES:\n\n')
disp(hpcBladeCluster)
fprintf('+ ------------------------------------------------------------ +\n\n')
%--------------------------------------------------------------------------%



%% STEP 6: SUBMIT THE JOB
% This section submits the job to the MJS to be scheduled (i.e. placed in a
% queue) to be run on the chosen cluster. No user input is required here.
% The "batch" command begins an automated process that will connect to the
% AMF cluster, submit a job to MJS, and initialize the MATLAB Distributed
% Computing Server (MDCS).
jobObject = batch(...
    hpcBladeCluster,...
    scriptName,...
    'Matlabpool',poolSize,...
    'AdditionalPaths',AdditionalPaths,...
    'AttachedFiles',AttachedFiles,...
    'CurrentFolder','.',...
    'AutoAttachFiles',false,...
    'CaptureDiary',true);
% Print details of job object to Command Window
fprintf('  JOB OBJECT PROPERTIES:\n\n')
disp(jobObject)
fprintf('+ ------------------------------------------------------------ +\n')
%--------------------------------------------------------------------------%
```

```
%% STEP 7: WAIT UNTIL JOB IS QUEUED
% Wait for the job to be successfully queued by the MJS. This step is
% necessary because the user requires a MJS job ID for retrieval of the
% output of the job. Hence, when this script is run, the user will need to
% wait until the job is successfully queued by the MJS.
wait(jobObject,'queued')
% Print success of job queuing to Command Window
fprintf('\nYour job has been successfully submitted to\nthe MJS cluster using the↩
    ''%s'' profile.\n',hpcBladeCluster.Profile);
fprintf('\n+ ---------------------------------------------------------- +\n')
%--------------------------------------------------------------------------%



%% STEP 8: DISPLAY AND SAVE THE JOB ID
% Here we display and save the job ID of the job that this script has
% just submitted to the MJS for execution on the remote cluster. The job
% index (MYJOBID) is the index (or position) of the job in the
% MATLAB Distributed Computing Server (MDCS) and it is required by the user
% to access the output of the job upon completion via the
% mjs_client_retrieve_script_v001ba. It allows the user to log off of the
% client machine and go and do other things and then come back later and
% retrieve the output from the job which is stored on the head node in a
% temporary job storage location. The job ID is only available after the
% job has been successfully queued.

% Get the MATLAB job index
myJobID = hpcBladeCluster.Jobs(end).ID;
% Create folder
[~,~,~] = mkdir(fullfile('Z:\JobData',clusterUserName,myJobName));
% Print job ID to Command Window
fprintf('\nYour job has a MDCS job ID of %d.', myJobID);
fprintf(' Please\nuse this ID to retrieve your data from\nthe MDCS.\n');
fprintf('\n+ ---------------------------------------------------------- +\n')
% Save a MAT-file with the job ID as well as the (approximate) job
% submission time and date. The MAT-file will be saved in your PWD and will
% be deleted automatically once the job is retrieved from the MJS and the
% job's output has been saved. The naming configuration for this MAT-file
% is:
%
%   mjsJobID_<MYJOBID>_<MYJOBNAME>_<ddmmmyyyy_HHMM>.mat
%
% This matrix also contains the job name stored in the variable MYJOBNAME
% as well as the cluster profile name.
% Create this file name
myJobIDMATFileName = ['mjsJobID_',num2str(myJobID),'_',myJobName,'_',datestr(now,↩
    'ddmmmyyyy')];
% Save the job ID in a MAT-file using the name MYJOBIDMATFILENAME
save(fullfile(pwd,myJobIDMATFileName),'myJobID','myJobName','clusterProfileName')↩
    ;
%--------------------------------------------------------------------------%
```

B.2 Retrieval Script

The submitted scripts are later retrieved by running the Retrieval Script to retrieve the processed data.

```
%% MATLAB Job Scheduler Output Retrieval Script File
% AUTHORS: M Harvey (michael.harvey@students.wits.ac.za)
% REVISED BY: D Platt (donovan.platt@students.wits.ac.za
% DATE: 12-Sep-2016 13:49:00
% VERSION: mjs_client_submit_script_v001ba
%
% # PROBLEM SPECIFICATION: This script is to be run in the client MATLAB
```

```
% session on the remote client computer (i.e. on one of the computers in
% the AMF lab or on the remote client machine "amfentry.ms.wits.ac.za"
% which is accessed via Windows Remote Desktop). This script is used to
% retrieve the output of a single job from the AMF cluster. The script
% requires as input the MAT-file with the MJS job ID, generated by the job
% submission script (mjs_client_submit_script_v001ba). To be specific, this
% script should be kept in the same directory as the script (batch code)
% submitted to the MJS as well as the submission script. When this script
% is run two dialogue boxes will open. The first will ask the user to find
% the appropriate MAT-file containing the relevant job ID, which should be
% in the same directory as this script. The second will ask the user to
% save a MAT-file of the job's outputs. Once the job ID is loaded, the
% script checks with the MJS to see if the job has finished running. If it
% has not finished, the user is requested to try again later. If the
% job has finished running this script will automatically fetch the output
% of the job and a dialogue box will open up asking the user to save a MAT-
% file of the job's output. This dialogue box cannot be cancelled and
% forces the user to save the fetched output. This is because the job will
% be deleted from the MJS when running this script. It is extremely
% important to be aware that the job's output is literally whatever is left
% in the workspace of the job after it has run and hence, when developing
% the script the user should consciously choose what variables are left in
% the workspace after the script has run as these are the only ones that
% will be returned. It should be stressed that only the relevant outputs
% should be left in the workspace so as to prevent unnecessary usage of
% disk space on the MJS's remote job data storage location as well as for
% quicker transfers of only non-redundant outputs. Further, if the
% workspace of the job exceeds 2GB this script returns an error and the job
% will not be deleted from the MJS. This is a problem that we have not
% officially sorted out yet.
%
% # DATA SPECIFICATION: This script requires an input. Specifically, the
% output MAT-file of the script used to submit the job to the MJS (i.e. the
% output of mjs_client_submit_script_v001ba) is required as input as it
% contains the job ID of the particular job the user wishes to retrieve the
% output of. The input is specifically the MAT-file containing the job name
% (MYJOBNAME) and the MJS job ID (MYJOBID). When the user runs this script
% a dialogue box will open and ask the user to 'Please load the job
% retrieval MAT-file' which should be found in the same directory as this
% script. It has naming convention:
%
%   mjsJobID_<MYJOBID>_<MYJOBNAME>.mat
%
% Once this MAT-file is loaded, the job ID (MYJOBID) and the user specified
% job name (MYJOBNAME) are loaded into the current workspace and are used
% to retrieve information about the job from the MJS.
%
% # CONFIGURATION CONTROL:
%
% EG: C:\Users\<USERNAME>\Documents\MATLAB\<PROJECT>\<SUBDIRECTORY>
%
%       C:\Users\James Brown\Documents\MATLAB\mjs
%       C:\Users\James Brown\Documents\MATLAB\mjs\scripts
%       C:\Users\James Brown\Documents\MATLAB\mjs\latex
%
% # VERSION CONTROL: This script was updated by Donovan Platt to be
% consistent with migration from CSAMMJS to AMFcluster.
%-------------------------------------------------------------------------%



%% STEP 0: ENSURE THAT THIS SCRIPT RUNS IN ITS DIRECTORY
% Get info
tmp = matlab.desktop.editor.getActive;
% Set cd
cd(fileparts(tmp.Filename));
% Clear
clear tmp;
```

```
%--------------------------------------------------------------------------%



%% STEP 1: SET UP WORKSPACE
% The following commands prepare the client session of MATLAB by clearing
% all variables from the client session workspace, closes all
% open figures and clears the command window of any displayed output.
clear, close, clc;
%--------------------------------------------------------------------------%



%% STEP 2: LOAD THE JOB ID
% This section opens up a dialogue box asking the user to load the MAT-file
% that contains the job name (MYJOBNAME) and the MJS job ID (MYJOBID).
% Please see above in the section entitled "DATA SPECIFICATION".

% Load the MAT-file name into the workspace. This MAT-file is the output of
% the submit script and has naming convention:
%   mjsJobID_<MYJOBID>_<MYJOBNAME>.mat
[mjsJobIDMATFileName,mjsJobIDMATPathName,~] = uigetfile('*.mat','Please load the ↩
   job retrieval MAT-file','your_job_name');
% Load the MAT-file that contains the job name (MYJOBNAME) and the MJS job
% ID (MYJOBID)
[myJobData] = load(mjsJobIDMATFileName);
% Extract the MJS job ID and store in the variable MYJOBID
[myJobID] = myJobData.myJobID;
% Extract the user specified name of the job with naming convention
% [myJobName] = <USER_INITIALS>_<JOB_NAME>_<DDMMMYYYY_HHMM>
[myJobName] = myJobData.myJobName;
% Extract the name of the cluster profile
[clusterProfileName] = myJobData.clusterProfileName;
%--------------------------------------------------------------------------%



%% STEP 3: RECONNECT TO THE CLUSTER
% Reinitialize the cluster object using the same profile as used in the
% job submission script.
[hpcBladeCluster] = parcluster(clusterProfileName);
% Print details of job object to Command Window
fprintf('+ ---------------------------------------------------------- +\n\n')
fprintf('  CLUSTER OBJECT PROPERTIES:\n\n')
disp(hpcBladeCluster)
fprintf('+ ---------------------------------------------------------- +\n\n')
%--------------------------------------------------------------------------%



%% STEP 4: RETRIEVE THE JOB OBJECT CORRESPONDING TO YOUR JOB ID
% Get the job object of your job from the cluster
jobObject = findJob(hpcBladeCluster,'ID',myJobID);
% Print details of job object to Command Window
fprintf('  JOB OBJECT PROPERTIES:\n\n')
disp(jobObject)
fprintf('+ ---------------------------------------------------------- +\n')
%--------------------------------------------------------------------------%



%% STEP 5: CHECK THE STATUS OF THE JOB
% Check the job status on the cluster
myJobStatus = jobObject.State;
% Display job status to Command Window
fprintf('\nThe status of the job with ID %d that was submitted to the \n',myJobID↩
   );
fprintf(''''%s'' cluster on %s is: %s \n',hpcBladeCluster.Profile,jobObject.↩
```

```
    SubmitTime);
fprintf('\n\n  --> %s\n',upper(myJobStatus))
if strncmpi('finished',myJobStatus,3)
    fprintf('\nPlease SAVE a MAT-file of the job''s output in an appropriate\n');
    fprintf('location on the client machine using the dialogue box that has\n');
    fprintf('just popped up on the screen.\n');
else
    fprintf('\nThe requested job with job ID %d is currently still %s.\n',myJobID↩
        ,myJobStatus);
    fprintf('Please return at a later stage and try again.\n');
    return;
end
fprintf('\n+ ------------------------------------------------------------------ +\n');
%--------------------------------------------------------------------------%



%% STEP 6: RETRIEVE OUTPUT OF JOB AND THEN DELETE IT FROM THE MJS
% If the job is finished then we can retrieve the workspace of the job from
% the cluster and load it into the workspace of the client machine, We then
% save it to an appropriate location on the client machine.

% Retrieve the output of the requested job and store in the [1x1]
% structure JOBOUTPUT.
[myJobOutput] = fetchOutputs(jobObject);
% Expand the structure.
[myJobOutput] = myJobOutput{:};
% Get diary
diary(jobObject);
% Save the output of the job, as stored in the structure JOBOUTPUT, in
% a user specified location as a MAT-file. A user dialogue box will open
% asking the user to choose a desired file name and location for saving of
% the job's output.
% Begin a while loop that will force the user to save the output of the
% job. This is to ensure that the any output that was fetched from the
% cluster is saved with probability 1. After the save is successful the job
% is DELETED from the MJS and the MAT-file containing the job ID is also
% deleted from its location in the project directory.
[jobOutputFileName] = 0;
while (jobOutputFileName == 0)
    [jobOutputFileName,jobOutputPathName] = uiputfile('*.mat','Please save your ↩
      jobs output','ChooseAPrettyFileName');
end
save(fullfile(jobOutputPathName,jobOutputFileName),'myJobOutput','-v7.3');
% Delete the job from the cluster. This is a matter of good practice.
% Since ALL possible output from your job has just been saved by the
% user in a user chosen location with a user chosen name it is assumed
% that the job does not need to be accessed anymore and is therefore
% deleted from the cluster. This ensures that cluster storage resources
% are not wasted pointlessly by jobs that have already been retrieved.
% The onus falls on the user to ensure that they save the output of the
% job they have submitted.
delete(jobObject);
% Delete the MAT-file containing the job ID
delete(fullfile(mjsJobIDMATPathName,mjsJobIDMATFileName));
% Display
fprintf('\nThe requested job with job ID %d has been processed by the\n',myJobID)↩
    ;
fprintf('cluster and the output has been loaded into the client''s\n');
fprintf('workspace as the structure "jobOutput". You should have also\n');
fprintf('saved this structure as a MAT-file when you were prompted.\n\n');
fprintf('IMPORTANT! Please note that the requested job with job ID %d\n',myJobID)↩
    ;
fprintf('has now been DELETED from the cluster and is now irretrievable.\n');
fprintf('If you have not saved the structure "jobOutput" please do so\n');
fprintf('now as once this session?s workspace is cleared the job''s\n');
fprintf('output will be gone forever.\n');
fprintf('\n+ ------------------------------------------------------------------ +\n');
```

```
%--------------------------------------------------------------------------%
```

C Data Science

Below is the data science script which loads the saved MAT-file which was retrieved from the cluster using the Retrieval Script. The script prepares the data for plotting and then plots the necessary graphs which are then saved to the local machine.

```
%% AMF Research Project: Computing VPIN: Bulk Classification vs. Lee-Ready
%% data_science_script_715798
%
% Author: N.J.Murphy
%
% Person Version:
% AMF-Project-NJM-715798_NMurphy_23-11-2016_data-science-001↩
   _data_science_script_715798
%
% 1 Problem Specification: This script aims to replicate the calclations and ↩
   analysis from
%   Easley et al[1] for stocks listed in the top 40 of the Johannesburg Stock ↩
   Exchange(JSE).
%
% 2 Data Specification: 6 months worth of order book data for 16 stocks listed
%   on the JSE Top 40. The specified period is from 01 January 2013 to 03
%   June 2013.
%
% 3 Configuration Control:
%       userpath/MATLAB/AMF
%       userpath/MATLAB/AMF/AMF_Project_NJM_715798
%       userpath/MATLAB/AMF/AMF_Project_NJM_715798/Scripts
%       userpath/MATLAB/AMF/AMF_Project_NJM_715798/Functions
%       userpath/MATLAB/AMF/AMF_Project_NJM_715798/Data
%       userpath/MATLAB/AMF/AMF_Project_NJM_715798/html
%
% 4 Version Control: No current version control
%
% 5 References:
% [1] D.Easley, M.Lpez de Prado, M.OHara. Flow toxicity and liquidity
%     in a highv frequency world. Review of Financial Studies, February, 2012.
% [2] D.Easley, N.Kiefer, M.OHara, J.Paperman. Liquidity, Information, and
%     Infrequently Traded Stocks. Journal of Finance, September, 1996.
% [3] D.Abada, J.Yageb. From PIN to VPIN: An introduction to order flow toxicity↩
   .
%     The Spanish Review of Financial Economics, 2012.
% [4] R.Engle, J.Lange. Predicting VNET: A Model of the Dynamics of Market
%     Depth, Journal of Financial Markets, 4: 113-142, 2001.
% [5] W.C.Wei, D.Gerace, A.Frino. Informed Trading, Flow Toxicity and the Impact
%     on Intraday Trading Factors. Australasian Accounting, Business and Finance
%     Journal, 7(2), 2013.
%
% 6 Current Situation: VBS_ADV001aa
% 7 Future Situation:
%
% Uses: This script will be used calculate the VPIN given the output of the
% data processing for the bulk classification algorithm but changes the
% calulcation of the price changes to not be expanded observations


%% 1. Data Description

% 1  RIC              - Reuters instrument code.
%                       A Reuters instrument code, or RIC, is a ticker
%                       like code used by Thomson Reuters to identify
%                       financial instruments and indices.
%
```

```
% 2   DateL            - Date of transaction.
% 3   TimeL            - Time of transaction.
% 4   DateTimeL        - Date and time of transaction.
% 5   Type             - Type of transaction: * Auction
%                                             * Quote
%                                             * Trade
% 6   Price            - Price of transaction for auction and trade
% 7   Volume           - Volume of tramsaction for auction and trade
% 8   MarketVWAP       - Market Volumes Weighted average prices
%                        = sum(shares bought * share price)/(Total shares bought)
% 9   L1BidPrice       - The price a buyer is willing to pay
% 10  L1BidSize        - The size of order a buyer wishes to achieve
% 11  L1AskPrice       - The price the seller is willing to offer
% 12  L1AskSize        - The size of order the seller wishes to achieve


%% 2. Data Cases
% 1) Trade entry with zero volume
% {'GRTJ.J','01-NOV-2013','09:00:09.315846','2013-11-01T09:00:09.315Z','Trade↩
   ',2550,0,0}

% 2) Trades occurring after 17:00
% {'AGLJ.J','07-NOV-2013','17:00:05.746125','2013-11-07T17:00:05.746Z','Trade↩
   ',25283,363506,25602.5000000000,9765625,0,0,0;
% 'AGLJ.J','07-NOV-2013','17:00:05.746125','2013-11-07T17:00:05.746Z','Trade↩
   ',25659,0,0,0,0,0,0;
% 'AGLJ.J','07-NOV-2013','17:00:30.967130','2013-11-07T17:00:30.967Z','Quote↩
   ',0,0,0,0,0,25416,20205;
% 'AGLJ.J','07-NOV-2013','17:06:14.197755','2013-11-07T17:06:14.197Z','Trade↩
   ',25283,39430,0,0,0,0,0;
% 'AGLJ.J','07-NOV-2013','17:10:12.394170','2013-11-07T17:10:12.394Z','Trade↩
   ',25602,8000,0,0,0,0,0;
% 'AGLJ.J','07-NOV-2013','17:26:44.908276','2013-11-07T17:26:44.908Z','Trade↩
   ',25485,10460,0,0,0,0,0;
% 'AGLJ.J','07-NOV-2013','17:26:45.448335','2013-11-07T17:26:45.448Z','Trade↩
   ',25548,20358,0,0,0,0,0}

% {'GRTJ.J','06-NOV-2013','16:49:19.253191',2483,429;
% 'GRTJ.J','06-NOV-2013','17:00:02.488567',2480,599771;
% 'GRTJ.J','06-NOV-2013','17:24:55.045941',2477,76385}

% 3) No trade occurs over consecutive time bars- thus no price shift nor volume ↩
   will be
% entered into these time bars
% {'GRTJ.J','01-NOV-2013','11:33:57.686807',2533,1008;'GRTJ.J','01-NOV↩
   -2013','11:37:11.160757',2533,1399}

% 4) Empty time bars where no trade occurs(rows 2 and 3 below)
% {'01-NOV-2013','10:16:00.000','10:17:00.000',[2x5 cell' char(10) '],[];
% '01-NOV-2013','10:17:00.000','10:18:00.000',[],[];'01-NOV↩
   -2013','10:18:00.000','10:19:00.000',[],[];
% '01-NOV-2013','10:19:00.000','10:20:00.000',[],[];'01-NOV↩
   -2013','10:20:00.000','10:21:00.000',[],[];
% '01-NOV-2013','10:21:00.000','10:22:00.000',[3x5 cell' char(10) '],[]}


%% 3. Clear Workspace
% This section prepares the workspace for the implementation of the script.
close all;      % close any open figures
clc;            % clear command window
format long g;  % formating for output on comand window
format compact; % formating for output on comand window


%% 4. Path Setup
% Set the project paths
[projectPath] = pwd;
```

```
% Add the project path so that the script sees the necessary functions
addpath(projectPath);
userpathstr = userpath;
userpathstr = userpathstr(~ismember(userpathstr,';'));
projectpath = 'AMF/AMF_FinalProject_NJM_715798';   %path for personal computer
addpath(fullfile(userpathstr,projectpath,'Functions'));
addpath(fullfile(userpathstr,projectpath,'Scripts'));
addpath(fullfile(userpathstr,projectpath,'html'));
addpath(fullfile(userpathstr,projectpath,'Data'));
figpath = 'C:\Users\Nicholas\Documents\Work\Varsity\AMF\Project\Semester 2\↩
   Proposal\Plots';

% Filepath to input data
[datafilepath] = 'C:\Users\Nicholas\Documents\MATLAB\AMF\↩
   AMF_FinalProject_NJM_715798\Data\Final';  %path to data on personal computer


%% 5. Load Data
% Step 1: Load the Lee-Ready data
[dataLR] = load(fullfile(datafilepath,'\JanJunLR1_12')); %load the matfile

% Step 2: Extract only data
dataLR = dataLR.myJobOutput;

% Step 3: Load the Bulk Classification data
[dataBC] = load(fullfile(datafilepath,'\JanJunBC1_17')); %load the matfile

% Step 4: Extract only data
dataBC = dataBC.myJobOutput;


%% 6. Calculate and plot VPIN

% Initialise variables
 noofstocks = dataLR.noofstocks; %specify number of stocks to be analysed and ↩
    processed
 stock_timebars = cell(1,noofstocks);  %initialise array to store all time bars ↩
    for each stock
 samplesize = 50;  %define the number of buckets used in order to update VPIN
 VBS_BC = zeros(1,noofstocks); %volume bucket size BC
 AveVPIN_BC = zeros(1,noofstocks); %average VPIN value for each stock
 VBS_LR = zeros(1,noofstocks); %volume bucket size LR
 AveVPIN_LR = zeros(1,noofstocks);
 plotnum = 0;
 VPINStDev_LR = zeros(1,noofstocks);
 VPINStDev_BC = zeros(1,noofstocks);

 for n = 1:noofstocks  % loop over stocks
  %extract the name for the nth stock
  stockname = dataLR.RICstore{1,n+lrstep};
  title_stock = strcat(stockname(1:3)); % create title

  % Exctract VPIN data for each method
  VPIN_LR = dataLR.VPINstore{1,n};
  VPIN_BC = dataBC.VPINstore{1,n};

  % Remove Nans
  VPIN_BC(any(isnan(cell2mat(VPIN_BC(:,3))),2),:) = [];
  VPIN_LR(any(isnan(cell2mat(VPIN_LR(:,3))),2),:) = [];

  % Compute log changes in mid price
  logchangesmidprice_BC = log(cell2mat(VPIN_BC(2:end,4)))-log(cell2mat(VPIN_BC(1:↩
     end-1,4)));
  logchangesmidprice_BC(any(logchangesmidprice_BC(:)==Inf | logchangesmidprice_BC↩
     (:)==-Inf,2),:) = [];
  % Compute log changes in mid price
  logchangesmidprice_LR = log(cell2mat(VPIN_LR(2:end,4)))-log(cell2mat(VPIN_LR(1:↩
     end-1,4)));
```

```
logchangesmidprice_LR(any(logchangesmidprice_LR(:)==Inf | logchangesmidprice_LR(:)==-Inf,2),:) = [];

% Standard deviation of VPIN
VPINStdDev_LR(n,1) = std(cell2mat(VPIN_LR(:,3)));
VPINStdDev_BC(n,1) = std(cell2mat(VPIN_BC(:,3)));

% Extract StDev of price changes
VPINstd = cell2mat(VPIN_BC(:,5));

% Compute average VPIN for n-th stock
AveVPIN_BC(n,1) = mean(cell2mat(VPIN_BC(:,3)));
AveVPIN_LR(n,1) = mean(cell2mat(VPIN_LR(:,3)));

% VPIN Skews
VPINskew_BC(n,1) = skewness(cell2mat(VPIN_BC(:,3)));
VPINskew_LR(n,1) = skewness(cell2mat(VPIN_LR(:,3)));

% Extract VBS's
VBS_LR(n,1) = dataLR.VBS(1,n);
VBS_BC(n,1) = dataBC.VBS(1,n);

% Compute data needed for x-axis
for i = 1:length(VPIN_LR)
   [yrlr(i),mtlr(i),dylr(i),~,~,~]= datevec(VPIN_LR{i,1});
end
Startdateplot = datenum(2013,01,02);
[d] = datenum(yrlr,mtlr,dylr);
xData_LR = linspace(d(1),d(end),length(VPIN_LR));
xDatamid_LR = linspace(d(1),d(end),length(logchangesmidprice_LR));
dateV_LR = Startdateplot:d(1,end);

for i = 1:length(VPIN_BC)
  [yrbc(i),mtbc(i),dybc(i),~,~,~] = datevec(VPIN_BC{i,1});
end
[d_BC] = datenum(yrbc,mtbc,dybc);
xData_BC = linspace(d_BC(1),d_BC(end),length(VPIN_BC));
xDatamid_BC = linspace(d_BC(1),d_BC(end),length(logchangesmidprice_BC));
xDatastd = linspace(d(1),d(end),length(VPINstd));
dateV_BC = Startdateplot:d_BC(1,end);

%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot VPIN versus dates %
%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(n+plotnum)
subplot(2,2,[1 2])
h1 = plot(xData_BC,cell2mat(VPIN_BC(:,3)),xData_LR ,cell2mat(VPIN_LR(:,3)));
hold on
hax1 = get(h1(1), 'Parent'); % axes handle
set(gca,'XTickLabelRotation',50)%,'xticklabel',datestr(xData_BC(1:200:end),'dd-mmm-yyyy'))
set(hax1(1), 'XTick', dateV_BC(1:15:end));
datetick('x','dd-mmm-yy','keepticks')
ylim([0,0.9])
xlim([Startdateplot (d(end)+1)])

% plot VPIN Peak
[Peak_LR, PeakIdx_LR] = findpeaks(cell2mat(VPIN_LR(:,3)));
[Peak_BC, PeakIdx_BC] = findpeaks(cell2mat(VPIN_BC(:,3)));
Peak_BC = max(Peak_BC);
Peak_LR = max(Peak_LR);
indexmax_BC = find(max(cell2mat(VPIN_BC(:,3))) == cell2mat(VPIN_BC(:,3)));
indexmax_LR = find(max(cell2mat(VPIN_LR(:,3))) == cell2mat(VPIN_LR(:,3)));
plot(xData_BC(indexmax_BC(1)),Peak_BC,'ko','MarkerSize',6,'MarkerFaceColor','[.49 1 .9]')

% plot minimum point
indexmin_BC = find(min(cell2mat(VPIN_BC(:,3))) == cell2mat(VPIN_BC(:,3)));
```

```
indexmin_LR = find(min(cell2mat(VPIN_LR(:,3))) == cell2mat(VPIN_LR(:,3)));
plot(xData_BC(indexmin_BC(1)),min(cell2mat(VPIN_BC(:,3))),'ko','MarkerSize',6,'↩
    MarkerFaceColor','[1 .5 0]') %
plot(xData_LR(indexmax_LR(1)),Peak_LR,'ks','MarkerSize',6,'MarkerFaceColor','g'↩
    )
plot(xData_LR(indexmin_LR(1)),min(cell2mat(VPIN_LR(:,3))),'ks','MarkerSize',6,'↩
    MarkerFaceColor','r') %

% title and labels
title1 = strcat(title_stock,{': '},{'VPIN vs Time'});
title(title1)
legend('VPIN Bulk-Classification (BC)','VPIN Lee-Ready (LR)','VPIN BC  Max','↩
    VPIN BC Min','VPIN LR  Max','VPIN LR Min','Location','Best')
xlabel('Date')
ylabel(hax1,'VPIN')
hold off

%%%%%%%%%%%%%%%%%%%
% plot miquote BC %
%%%%%%%%%%%%%%%%%%%
subplot(2,2,3)
h2 = plot(xDatamid_BC(1:end),logchangesmidprice_BC(:));
hold on
hax2 = get(h2, 'Parent'); % axes handle
set(gca,'XTickLabelRotation',50,'FontSize',8)
set(hax2, 'XTick', dateV_BC(1:15:end));
datetick('x','dd-mmm-yy','keepticks')

% plot maximum point
indexmaxmid_BC = find(max(logchangesmidprice_BC(:)) == logchangesmidprice_BC(:)↩
    );
plot(xDatamid_BC(indexmaxmid_BC(1)),max(logchangesmidprice_BC(:)),'ko','↩
    MarkerSize',6,'MarkerFaceColor','g') %
% plot minimum point
indexminmid_BC = find(min(logchangesmidprice_BC(:)) == logchangesmidprice_BC(:)↩
    );
plot(xDatamid_BC(indexminmid_BC(1)),min(logchangesmidprice_BC(:)),'ko','↩
    MarkerSize',6,'MarkerFaceColor','r')

% titles, labels and axes
xlim([Startdateplot (d(end)+1)])
title2 = strcat(title_stock,{': '},{'Change in log-midquote price Bulk ↩
    Classification'});
title(title2)
xlabel('Date')
ylabel(hax2(1),'$\triangle$ log-midquote','interpreter','latex','FontSize', 11)
h0 = legend('$\triangle$ log-midquote','Midquote Max','Midquote Min','Location'↩
    ,'Best');
set(h0,'Interpreter','latex')
hold off

%%%%%%%%%%%%%%%%%%%
% plot miquote LR %
%%%%%%%%%%%%%%%%%%%
subplot(2,2,4)
h2 = plot(xDatamid_LR(1:end),logchangesmidprice_LR(:));
hold on
hax2 = get(h2, 'Parent'); % axes handle
set(gca,'XTickLabelRotation',50,'FontSize',8)
set(hax2, 'XTick', dateV_BC(1:15:end));
datetick('x','dd-mmm-yy','keepticks')

% plot maximum point
indexmaxmid_LR = find(max(logchangesmidprice_LR(:)) == logchangesmidprice_LR(:)↩
    );
plot(xDatamid_LR(indexmaxmid_LR(1)),max(logchangesmidprice_LR(:)),'ko','↩
    MarkerSize',6,'MarkerFaceColor','g')
% plot minimum point
```

```
indexminmid_LR = find(min(logchangesmidprice_LR(:)) == logchangesmidprice_LR(:)↩
    );
plot(xDatamid_LR(indexminmid_LR(1)),min(logchangesmidprice_LR(:)),'ko','↩
    MarkerSize',6,'MarkerFaceColor','r')

% titles, labels and axes
xlim([Startdateplot (d(end)+1)])
title2 = strcat(title_stock,{': '},{'Change in log-midquote price Lee-Ready'});
title(title2)
xlabel('Date')
ylabel(hax2(1),'$\triangle$ log-midquote','interpreter','latex','FontSize', 11)
h2 = legend('$\triangle$ log-midquote','Midquote Max','Midquote Min','Location'↩
    ,'Best');
set(h2,'Interpreter','latex')
hold off

%%%%%%%%%%%%%%%
% save figure %
%%%%%%%%%%%%%%%
savetoname = char(strcat(figpath,'\',title_stock));
fig = gcf;
fig.PaperUnits = 'centimeters';
fig.PaperPosition = [0 0 23 20]; %proposal size
print(savetoname,'-dpng','-r300')

%%%%%%%%%%%%%%%%%%%%%
% plot StDev vs time %
%%%%%%%%%%%%%%%%%%%%%
figure(n+plotnum+1)
hl = plot(xDatastd,VPINstd);
hold on
hax3 = get(hl, 'Parent'); % axes handle
set(gca,'XTickLabelRotation',50)
set(hax3, 'XTick', dateV_BC(1:15:end));
datetick(hax3,'x', 'dd-mmm-yy', 'keepticks');

% plot maximum point
indexmaxstd = find(max(VPINstd) == VPINstd);
plot(xDatastd(indexmaxstd(1)),max(VPINstd),'ko','MarkerSize',6,'MarkerFaceColor↩
    ','g') %
% plot minimum point
indexminstd = find(min(VPINstd) == VPINstd);
plot(xDatastd(indexminstd(1)),min(VPINstd),'ko','MarkerSize',6,'MarkerFaceColor↩
    ','r') %

% titles, labels and axes
title3 = strcat(title_stock,{': '},{'StDev of price changes corresponding to ↩
    each 50 bucket sample'});
title(title3)
xlabel('Date')
ylabel('$\sigma_{\triangle P}$ (50)','interpreter','latex','FontSize', 15)
xlim([Startdateplot (d(end)+1)])
h = legend('$\sigma_{\triangle P}$ (50)','StDev Max','StDev Min','Location','↩
    Best');
set(h,'Interpreter','latex')
hold off

%%%%%%%%%%%%%%%
% save figure %
%%%%%%%%%%%%%%%
savetoname = char(strcat(figpath,'\',title_stock,'_StDev'));
fig = gcf;
fig.PaperUnits = 'centimeters';
fig.PaperPosition = [0 0 16 7];
print(savetoname,'-dpng','-r300')

plotnum = n+1;
```

```
  % Remove variables
  clearvars d dataV yr mt dy xData logchangesmidprice_BC x1 xtime = [];
end
```