"A Hidden Markov Model Approach to Estimating VPIN"

August 31, 2015

Contents

1
Introduction

1.1 Problem Background

In the world of high frequency trading, market makers constitute the majority of the players. Their role in the markets comprises of providing liquidity to position takers by making passive orders. Instead of taking directional bets on financial asset, these market makers instead attempt to profit by making tiny margins on a huge number of trades. As passive providers of liquidy they often are subject to the risk of adverse selection. Adverse selection is often defined as the "natural tendency for passive orders to fill quickly when they should fill slowly, and slowly (or not at all) when they should fill quickly" (Jeria & Sofianos 2008). Order flow is considered toxic when informed traders take advantage of uninformed traders, and these market makers are adversely selected often without even realising it.

The most notable proxy for information assymetry (and hence order toxicity) is the Probability of Information-Based Trading (PIN) developed by Easley et al. (1996). David Easley, López De Prado, et al. (2012) then re-examined PIN, in order to update it for a high-frequency context, and hence developed Volume-Synchronised PIN (VPIN). The critical difference between PIN and VPIN is in the methodology employed to categorise trades as either buys or sells; where PIN uses intervals based off clock-time, VPIN instead uses intervals based off volume-time. At their essence however both are proxy measures of trade imbalances.

Historically PIN has been used as a metric to measure the extent of order flow toxicity. If order flow becomes too toxic, market makers are forced out of the market. As they withdraw, liquidity disappears, which increases even more the concentration of toxic flow in the overall volume, which then triggers a feedback mechanism that forces even more market makers out. This cascading effect has caused liquidity-induced crashes in the past, the 2010 Flash Crash being one (major) example of it. One hour before the flash crash, order flow toxicity was

at historically high levels relative to recent history. ELO claim that using the VPIN metric, this crash could have been predicted one hour before it actually happened.

This metric has been proposed as a new standard for market makers (along with financial regulators) as it enables them to set appropriate bid-ask prices such that the spread compensates them for the probability of dealing with an insider. ELO state that this is not to be a replacement for the CBOE Volatility Index (VIX) but rather as a complement to it.

VPIN's theory is consistent with the anecdotal evidence reported by a joint SEC-CFTC study on the events of May 6, 2010 (SEC 2010). Given the relevance of these findings, the SEC requested an independent study to be carried out by the Lawrence Berkeley National Laboratory. This Government laboratory concluded:

> This [VPIN] is the strongest early warning signal known to us at this time.
>
> (Bethel et al. 2011)

The goal of this paper is to validate the predictive power of the VPIN metric, and to explore the relationship between PIN and VPIN by way of a Hidden Markov Model (HMM) to estimate model parameters and decoded hidden market states.

1.2 Model Background

First we explain the background to the PIN model and its relationship with VPIN. Traditionally PIN is explained by way of a sequential trade model. This is as follows:

Denote a security's price as S. Its present value is S_1. Once a certain amount of new information has been incorporated into the price, S will be either S_B (bad news) or S_G (good news). There is a probability α that new information will arrive within the time-frame of the analysis, and a probability δ that the news will be bad (i.e., $1 - \delta$ that the news will be good). Traders are classified into two types: So-called noise, or liquidity traders, are those with no information based reason to trade. Their arrivals are measured as a Poisson process with rate ϵ. Information-based traders on the other hand are those that are trading based off some private information that has not been priced into the asset. They are also modelled as a Poisson process, with arrival rate μ.

If an information event arrives and it is a 'good news' event, both uninformed and informed traders buy, while only uninformed traders sell. Conversely if a 'bad news' information event occurs only uninformed traders buy, while both informed and uninformed traders sell.

If no information event occurs then only uninformed traders are in the market. The model is visually explained in the decision tree in Figure 1.

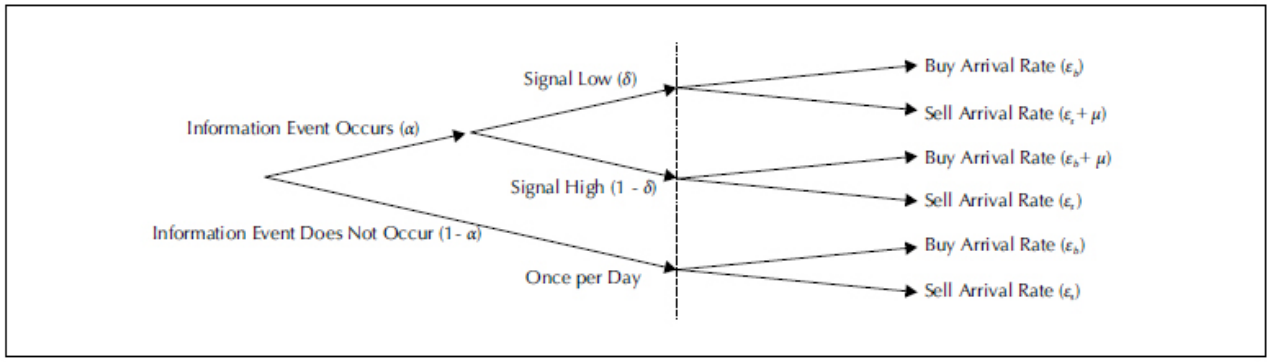


Figure 1 *Tree Diagram of the Trading Process*

Source: Adapted from Easley, Hvidkjaer, and O'Hara (2002).

The original PIN model requires the estimation of four non-observable parameters, namely α, δ, μ, and *epsilon*. This was originally done numerically via Maximum likelihood on the observed number of buys and sells, through the fitting of a mixture of three Poisson distributions. VPIN can in some regards be considered a high-frequency estimate of PIN which takes into account the time-varying nature of the data. Instead of calculating liquidity based off clock-time, VPIN uses what is called volume-time, whereby trades are placed into equally sized buckets each with a uniform amount of volume. The intuition behind this is that in a high-frequency environment, these volume bars will be spaced out widely when there is little information-based trading, and packed tightly when there is a higher amount. The idea is that the more relevant is a piece of information, the more volume it will attract. Within each of these bars the number of trades that are buys or sells are inferred, and then VPIN is calculated as the degree of imbalance between buys and sells. If we have $\tau = 1, 2, ...n$ volume buckets, within each bucket trades are classified as buys V_τ^B, and sells V_τ^S where $V = V_\tau^B + V_\tau^S$ for each τ. Using this approach, the parameters can be estimated anlytically instead of numerically.

A crucial part of this calculation involves the classification of trades into buys and sells, as traditional order-book data sources do not include the trade direction. Rather than using the Tick-rule, Lee & Ready (1991) or other trade classification techniques, ELO propose a new volume classification method called Bulk Volume Classification[1]. This departs from standard trade classification schemes in two ways: First, volume is classified in bulk, and second this methodology probabilistically classifies part of a bar's volume as buy, and the remainder

[1] The algorithm to compute the VPIN metric is in Appendix A

as sell. Empirical studies have shown Bulk Volume Classification to be more accurate than the Tick-rule, despite of not requiring level-1 tick data (only bars) (David Easley, Lopez de Prado, et al. 2012). Within a volume bucket, the amount of volume classified as buy is:

$$V_\tau^B = \sum_{i=t(\tau-1)+1}^{t(\tau)} V_i Z\left(\frac{S_i - S_{i-1}}{\sigma_{\Delta S}}\right) \tag{1.1}$$

where $t(\tau)$ is the index of the last (volume or time) bar included in bucket τ, V_τ^B is the buy volume (traded against the Ask), V_i is the total volume per bucket, Z is the Standard Normal Distribution, and $\sigma_{\Delta S}$ is the standard deviation of price changes betwenen (volume or time) bars. Because all buckets contain the same amount of volume V,

$$V_\tau^S = \sum_{i=t(\tau-1)+1}^{t(\tau)} V_i \left(1 - Z\left(\frac{S_i - S_{i-1}}{\sigma_{\Delta S}}\right)\right) = V - V_\tau^B \tag{1.2}$$

The dynamics of buys and sells are driven by the sequential trade model parameters mentioned previously (α, δ, μ and ϵ). The expected arrival rate of informed trade becomes: $E[V_\tau^B - V_\tau^S] = \alpha\mu(2\delta - 1)$ and the absolute expected values for sufficiently large μ is $E[|V_\tau^B - V_\tau^S|] \approx \alpha\mu$.

The total expected arrival rate is:

$$\frac{1}{n}\sum_{\tau=1}^{n}(V_\tau^B + V_\tau^S) = V =$$

$$\underbrace{\alpha(1-\delta)(\epsilon+\mu+\epsilon)}_{\text{Volume from good news}} + \underbrace{\alpha\delta(\mu+\epsilon+\epsilon)}_{\text{Volume from bad news}} + \underbrace{(1-\alpha)(\epsilon+\epsilon)}_{\text{Volume from no news}} = \alpha\mu + 2\epsilon$$

VPIN is then calculated as:

$$VPIN = \frac{\alpha\mu}{\alpha\mu + 2\epsilon} = \frac{\alpha\mu}{V} = \frac{\sum_{\tau=1}^{n}(V_\tau^S - V_\tau^B)}{(2\delta-1)nV} \approx \frac{\sum_{\tau=1}^{n}|V_\tau^S - V_\tau^B|}{nV}$$

To simulate the volume bars across the course of a hypothetical trading day I generate data according to the following procedure: Generate a Bernouilli random variable with parameter α to determine whether an information event has ocurred within this volume bar. Then generate a second Bernouilli R.V. with parameter δ to determine whether the event was a 'good' or a 'bad' information event. According to the sequential trade model we then simulate the number of buys and sells for the volume bar using Possion R.V.s. This is repeated for the total number of expected volume bars for the trading

days, then this hypothetical day can be simulated a large number of times in order to calculate an expected VPIN value. The code to generate Monte Carlo simulations of the volume bars has been outlined in Appendix B.

This approach is not without its detractors however, and a long-running dispute has been simmering for the past several years with counterclaims attacking the viability of both the bulk-classification methodology (Andersen & Bondarenko 2012) and the predictive power of VPIN itself(Andersen & Bondarenko 2014). Partly due to this controversy we believe that this is a topic worth investigating further.

We shall follow the approach postulated by Yin & Zhao (2014) in which the time bars are classified as buys and sells, and then analysed as a bivariate Poisson HMM. After fitting an appropriate HMM we shall be able to conduct a decoding exercise to find the most likely hidden market state at any given point in time. The purpose of this is then to find some relationship with the corresponding volume bars and their subsequent VPIN values.

We conduct this analysis on both simulated data and empirical data. The data we have acquired are the raw tick-by-tick trade and quote data from the NYSE for the S&P 500 SPDR ETF for the day of the Flash Crash, 6 May 2010.

2
HMM Model Derivation

2.1 Independent Mixture Model

This section will outline the derivation of the HMM for univariate Poisson observed data. We will use a toy example in order to illustrate the derivation.

An independent mixture model consists of m component distributions with probability functions p_i for $i \in 1, ..., m$ and a "mixing distribution". The mixing is performed by a discrete random variable C:

$$C = \begin{cases} 1 & \text{with probability } \delta_1 \\ \vdots & \vdots \\ i & \text{with probability } \delta_i \\ \vdots & \vdots \\ m & \text{with probability } \delta_m = 1 - \sum_{i=1}^{m-1} \delta_i \end{cases}$$

Thus $Pr(C = i) = \delta_i$ must obey $0 < \delta_i < 1$ and that $\sum_{i=1}^{m} \delta_i = 1$

For a discrete random variable X described by a mixture model consisting of m components, it holds that:

$$p(X) = \sum_{i=1}^{m} \delta_i p_i(X) \implies PR(X = x) = \sum_{i=1}^{m} Pr(X = x|C = i)$$

To estimate the parameters θ of the mixture distribution via ML estimation we maximize the combined likelihood of the components:

$$L(\theta_1, ..., \theta_m, \delta_1, ..., \delta_m | x_1, ..., x_n) = \prod_{j=1}^{n} \sum_{i=1}^{m} \delta_i p_i(x_j, \theta_i)$$

For the example where $m = 2$ and the component distributions are Poisson with parameters λ_1 and λ_2 the likelihood equation is as follows:

$$L(\lambda_1, \lambda_2, \delta_1, \delta_2 | x_1, ..., x_n) = \prod_{j=1}^{n} \left(\delta_1 \frac{\lambda_1^{x_j} e^{-\lambda_1}}{x_j!} + \delta_2 \frac{\lambda_2^{x_j} e^{-\lambda_2}}{x_j!} \right)$$

In R this could be implemented naively as follows:

```
lambdas = c(3, 15)
deltas = c(0.3, 0.7)
cdf = cumsum(deltas)
unifvec = runif(100)
d1 = rpois(sum(unifvec < cdf[1]), lambdas[1])
d2 = rpois(sum(unifvec >= cdf[1]), lambdas[2])
x = c(d1, d2)
```

This code generates a series of x values according to a mixture of two Poisson distributions with $\lambda_1 = 3, \lambda_2 = 15, \delta_1 = 0.3$ and $\delta_2 = 0.7$

We can see from both the following plots of the time series and the kernel density estimates, the data appears to conform to two separate distributions, or states, as it were:

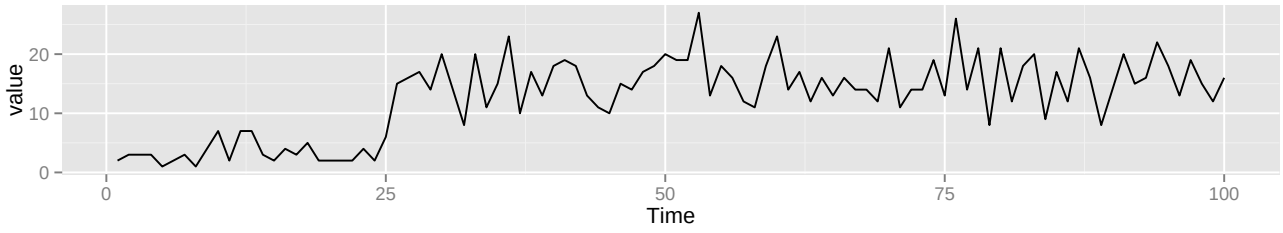


Figure 2.1: Time series of simulated Poisson mixture

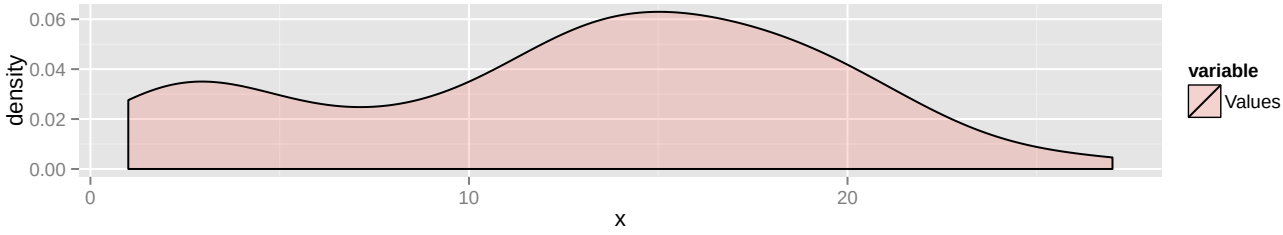


Figure 2.2: Kernel density estimate of simulated Poisson mixture (bandwidth=30, ker- nel=gaussian)

The kernel density estimates are generated with the Gaussian kernel and a bandwidth of 2.34. The multimodal appearance eof the plot clearly shows that the data are overdispersed compared to a single Poisson, and instead appear to come from a mixture of two Poissons.

Before minimising the negative log-likelihood, the final step is that we must apply a transformation of the parameters in order to fulfill the requirements that $\sum_i \delta_i = 1$, $\delta_i > 0$ and $\lambda_i > 0$ for all i. This is achieved by log-transforming λ and logit transforming δ.

$$\begin{aligned} \text{Log-transform} \quad & \eta_i = log\lambda_i, i = 1, ..., m \\ \text{Logit-transform} \quad & \tau_i = log\left(\frac{\delta_1}{1-\delta_2}\right) \end{aligned} \tag{2.1}$$

In R:

```
logit <- function(delta2) log(delta2/(1 - delta2))

eta = log(lambdas)
tau = logit(deltas[2])
```

After the maximisation has occured the original parameters can be recovered by the inverse transformations:

$$\begin{aligned} \text{exp-transform} \quad & \lambda_i = e^{\eta_i}, i = 1, ..., m \\ \text{inverse logit} \quad & \delta_2 = \frac{e^{\tau}}{1+e^{\tau}} \end{aligned} \tag{2.2}$$

and additionally $\delta_1 = 1 - \delta_2$

```
invlogit <- function(tau) exp(tau)/(1 + exp(tau))
lambdas = exp(eta)
delta2 = invlogit(tau)
delta1 = 1 - delta2
```

Now we have all the building blocks to maximise the negative log-likelihood for a two-state mixture distribution

```
f <- function(theta, data) {
    eta <- theta[1:2]
    tau <- theta[3]
    lambdas = exp(eta)
    delta2 = invlogit(tau)
    delta1 = 1 - delta2

    L = delta1 * dpois(data, lambdas[1]) + delta2 *
        dpois(data, lambdas[2])
    -sum(log(L))
}
```

Use the 25% and 75% quantiles of the data as a guess for lambdas:

```
quantiles <- quantile(x)
lambdaGuess = c(quantiles[2], quantiles[4])
```

We can now optimise using the R `nlm` package:

```
deltaGuess = 0.1
theta <- c(log(lambdaGuess), logit(deltaGuess))
res = nlm(f, theta, data = x)
```

And back transform the results:

```
lambda1Hat = exp(res$estimate[1])
lambda2Hat = exp(res$estimate[2])
delta2Hat = invlogit(res$estimate[3])
delta1Hat = 1 - delta2Hat
```

Which provides us with the following estimated values:

$$\boldsymbol{\lambda} = \begin{pmatrix} 3 \\ 15 \end{pmatrix} \qquad \hat{\boldsymbol{\lambda}} = \begin{pmatrix} 3.191 \\ 15.683 \end{pmatrix}$$

$$\boldsymbol{\delta} = \begin{pmatrix} 0.3 \\ 0.7 \end{pmatrix} \qquad \hat{\boldsymbol{\delta}} = \begin{pmatrix} 0.241 \\ 0.759 \end{pmatrix}$$

The mixture distribution parameters appear to be estimated satisfactorily when compared with the true values from the simulated data.

The next step is to introduce into the model a way of capturing the transition behaviour between the states over time.

2.2 Markov switching process

The independent mixture model is used to cater for the overdispersion noticed in some time series data. The second feature often found is that they exhibit serial dependece between the observations. This dependence is identified via the autocorrelation function of the data. To cater for this we model the dependence between the mixtures by introducing a Markov process. With this building block we will have completely defined the hidden Markov model.

If we define $\{C_t : t = 1, 2, \dots T\}$ as the unobserved hidden states, and $\{X_t : t = 1, 2, \dots T\}$ as the observed data points, and $X^{(t)}$ denotes the sequence of all R.V.s X_i for $i = 1 \dots t$, we can say a hidden Markov is a dependent mixture where:

$$Pr(C_t = i|C^{(t-1)}) = Pr(C_t = i|C_{t-1}), t = 2, 3, \dots$$
$$Pr(X_t = x|X^{(t-1)}, C^{(t)}) = Pr(X_t = x|C_t), t \in \mathbb{N}$$

As is illustrated in Figure 2.3, this is saying that when C_t is known the distribution of X_t only depends on C_t

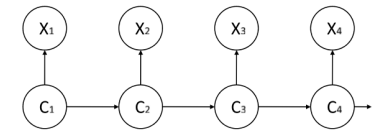


Figure 2.3: Hidden state directed graph

For discrete distributions we can define the m-state dependent distribution function as:

$$p_i(X) = Pr(X_t = x|C_t = i), i, 2, \dots, m \tag{2.3}$$

Which is simply the conditional probability of observing x at time t when C is in state i.

Additionally we can define the unconditional probability of being in state i at time t as:

$$u_i(t) = Pr(C_t = i), t = 1, \ldots, T \tag{2.4}$$

We will often want to calculate the marginal distribution of X. This can be achieved by summing over the states and using the hidden state (2.4) and state-dependent (2.3) distributions:

$$\begin{aligned} Pr(X_t = x) &= \sum_{i=1}^{m} Pr(C_t = i) Pr(X_t = x | C_t = i) \\ &= \sum_{i=1}^{m} u_i(t) p_i(x) \\ &= \begin{pmatrix} u_1(t), \ldots, u_m(t) \end{pmatrix} \begin{pmatrix} p_1(x) & & 0 \\ & \ddots & \\ 0 & & p_m(x) \end{pmatrix} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \\ &= \mathbf{u}(t) \mathbf{P}(x) \mathbf{1}' \end{aligned} \tag{2.5}$$

$\mathbf{u}(t)$ is the distribution of the hidden states, and $\mathbf{P}(x)$ is the state-dependent distribution of the observations. $\mathbf{u}(1)$ is defined as the initial distribution of the Markov chain.

The $m \times m$ diagonal matrix of conditional probabilities $\mathbf{P}(x)$ defined as:

$$\mathbf{P}(x) = \begin{pmatrix} p_1(x) & & 0 \\ & \ddots & \\ 0 & & p_m(x) \end{pmatrix}$$

Where for example: $\boldsymbol{\lambda} = \begin{pmatrix} 3 \\ 15 \end{pmatrix}$

$$\mathbf{P}(1) = \begin{pmatrix} \frac{e^{-3}3^1}{1!} & 0 \\ 0 & \frac{e^{-15}15^1}{1!} \end{pmatrix} = \begin{pmatrix} 0.149 & 0 \\ 0 & 4.589 \times 10^{-6} \end{pmatrix}$$

This can be implemented easily in R as:

```
PMat <- function(x, lambda) diag(dpois(x, lambda))

PMat(1, lambdas)
```

```
##       [,1]     [,2]
## [1,] 0.149 0.00e+00
## [2,] 0.000 4.59e-06
```

Next we introduce the transition probability matrix $\mathbf{\Gamma}(t)$ which is the $m \times m$ matrix of transition probabilities at state t:

$$\mathbf{\Gamma}(1) = \begin{pmatrix} \gamma_{11} & \cdots & \gamma_{1m} \\ \vdots & \ddots & \vdots \\ \gamma_{m1} & \cdots & \gamma_{mm} \end{pmatrix}$$

$$\gamma_{ij}(t) = Pr(C_{t+1} = j | C_t = i)$$

Because the Markov chain satisfies the Chapman-Kolmogorov equations (Zucchini & MacDonald 2009, p16) we can say that $\mathbf{u}(t) = \mathbf{u}(1)\mathbf{\Gamma}^{t-1}$ (where $\mathbf{\Gamma}$ is shorthand for $\mathbf{\Gamma}(1)$) and hence it follows that:

$$Pr(X_t = x) = \mathbf{u}(1)\mathbf{\Gamma}^{t-1}\mathbf{P}(x)\mathbf{1}' \tag{2.6}$$

If the Markov chain can be shown to be stationary with stationary distribution $\boldsymbol{\delta}$, because $\boldsymbol{\delta}\mathbf{\Gamma}^{t-1} = \boldsymbol{\delta}$ for all t we get:

$$Pr(X_t = x) = \boldsymbol{\delta}\mathbf{P}(x)\mathbf{1}' \tag{2.7}$$

It can be shown that if the stationary distribution exists then $\boldsymbol{\delta}$ can be found using the following calculation (proved in Zucchini & MacDonald 2009, p19):

$$\boldsymbol{\delta}(\mathbf{I}_m - \mathbf{\Gamma} + \mathbf{U}) = \mathbf{1} \tag{2.8}$$

Where $\mathbf{I}_m$ is an $m \times m$ identity matrix and $\mathbf{U}$ is an $m \times m$ matrix of ones. This can be implemented simply in R as follows:

```
statdist <- function(gamma) {
    m = dim(gamma)[1]
    matrix(1, 1, m) %*% solve(diag(1, m) - gamma +
        matrix(1, m, m))
}
```

To illustrate this concept, we can extend the example we used previously this time introducing a hypothetical transition matrix:

```
gamma = matrix(c(0.9, 0.1, 0.4, 0.6), nrow = 2,
    byrow = TRUE)
```

$$\boldsymbol{\lambda} = \begin{pmatrix} 3 \\ 15 \end{pmatrix} \qquad \mathbf{\Gamma} = \begin{pmatrix} 0.9 & 0.1 \\ 0.4 & 0.6 \end{pmatrix}$$

We can now calculate the stationary distribution $\boldsymbol{\delta}$ using equation (2.8):

```
delta = statdist(gamma)
```

$$\boldsymbol{\delta} = \begin{pmatrix} 0.8 \\ 0.2 \end{pmatrix}$$

With which we can now calculate the marginal distribution (2.7), for example for $P(X = 1)$:

```
x = 1
sum(delta %*% PMat(x, lambdas))
```

```
## [1] 0.119
```

2.3 Calculating the Likelihood

The first step to calculate the likelihood function is to extend the univariate marginal distribution (2.6) to a bivariate distribution. Zucchini & MacDonald (2009) show that (2.6) can be extended so that:

$$Pr(X_t = v, X_{t+k} = w) = \mathbf{u}(t)\mathbf{P}(v)\boldsymbol{\Gamma}^k\mathbf{P}(w)\mathbf{1}'$$

While the equivalent for a stationary chain reduces to:

$$Pr(X_t = v, X_{t+k} = w) = \boldsymbol{\delta}\mathbf{P}(v)\boldsymbol{\Gamma}^k\mathbf{P}(w)\mathbf{1}'$$

This then can be generalised to a T-th order distribution and hence the likelihood for a set of observations $\mathbf{X}^{(T)} = (X_1, X_2, \ldots, X_T)$ given the parameters of an HMM is:

$$L_T = \boldsymbol{\delta}\mathbf{P}(x_1)\boldsymbol{\Gamma}\mathbf{P}(x_2)\ldots\boldsymbol{\Gamma}\mathbf{P}(x_T)\mathbf{1}' \tag{2.9}$$

A recursive algorithm can be defined to calculate the likelihood, first by defining the forward probability vector:

$$\boldsymbol{\alpha_t} = \boldsymbol{\delta}\mathbf{P}(x_1)\boldsymbol{\Gamma}\mathbf{P}(x_2)\ldots\boldsymbol{\Gamma}\mathbf{P}(x_T) \tag{2.10}$$

From which the likelihood can now be recursively calculated:

$$\begin{aligned} \boldsymbol{\alpha_1} &= \boldsymbol{\delta}\mathbf{P}(x_1) \\ \boldsymbol{\alpha_t} &= \boldsymbol{\alpha_{t-1}}\boldsymbol{\Gamma}\mathbf{P}(x_t) \qquad for \quad t = 2, 3, \ldots, T \\ L_T &= \boldsymbol{\alpha_T}\mathbf{1}' \end{aligned}$$

If $\boldsymbol{\delta}$ provided is in fact the stationary distribution of the Markov chain, then the recursive scheme becomes:

$$\begin{aligned} \boldsymbol{\alpha_0} &= \boldsymbol{\delta} \\ \boldsymbol{\alpha_t} &= \boldsymbol{\alpha_{t-1}}\boldsymbol{\Gamma}\mathbf{P}(x_t) \qquad for \quad t = 1, 2, \ldots, T \\ L_T &= \boldsymbol{\alpha_T}\mathbf{1}' \end{aligned}$$

With R implementation:

```
pois.hmm.alpha <- function(x, gamma, delta = NULL) {
    n = length(x)
    alpha = vector(mode = "list", length = n)
    alpha0 = statdist(gamma)

    for (t in 1:n) {
        if (t == 1) {
            prevAlpha = alpha0
        } else {
            prevAlpha = alpha[[t - 1]]
        }
        alpha[[t]] = prevAlpha %*% gamma %*% PMat(x[t],
            lambdas)
    }
    return(alpha[[n]])  #return the last itertion
}
```

The problem with this scheme however is that as t increases, $\boldsymbol{\alpha_t}$ rapidly diminishes as it is the product of a large number of probabilities, and when implemented computationally, will be rounded to zero (a problem known as underflow). We can show that the calculation works for small t however needs to be re-worked for any non-trivial time series.

As an example we can calculate $Pr(X_1 = 0, X_2 = 2, X_3 = 1)$, using the previously defined $\mathbf{\Gamma}$ and $\boldsymbol{\lambda}$ parameters:

```
x = c(0, 2, 1)
alpha = pois.hmm.alpha(x, gamma)
prob = sum(alpha)
prob
```

```
## [1] 0.00108
```

Durbin et al. (1998, p.99) present a method of scaling the computations to mitigate this issue of underflow. In summary they propose the following algorithm (where $\mathbf{\Gamma}$ and $\mathbf{P}(x_t)$ are $m \times m$ matrices, $\mathbf{v}$ and $\boldsymbol{\phi_t}$ are vectors of length m, u is a scalar, and l is the scalar which accumulates the log-likelihood).

set $\phi_0 \leftarrow \boldsymbol{\delta}$ and $l \leftarrow 0$
for t = 1,2, ..., T **do**
  $\mathbf{v} \leftarrow \boldsymbol{\phi_t}\mathbf{\Gamma}\mathbf{P}(x_t)$
  $u \leftarrow \mathbf{v}\mathbf{v}'$
  $l \leftarrow l + \log u$
  $\boldsymbol{\phi_t} \leftarrow \mathbf{v}/u$
end for

return l

In addition to this scaling solution, we also introduce two parameter transformations in a similar vein to what we did for the independent mixture models. Because it has been shown that the performance of constrained optimizers such as R's `constrOptim` can degrade when the parameter optimums lie close to their boundaries of the parameter space, we instead apply the following transformations to $\mathbf{\Gamma}$ and $\boldsymbol{\lambda}$

- As shown in (2.1) and (2.2), because the parameters λ_i for a Poisson HMM are limited to be non-negative, we apply the log transformation to get the working parameters η_i, and then exponentiate them to obtain the natural parameters again.
- Because the rows of the $\mathbf{\Gamma}$ matrix must all sum to one and the individual γ_{ij} must all be non-negative, we apply the following transformation.[1]

[1] For details see Zucchini & MacDonald (2009, pp.48–49)

$$\text{Natural to working } \tau_{ij} = \log\left(\frac{\gamma_{ij}}{1 - \sum_{k:k\neq i}\gamma_{ik}}\right) \qquad i = 1,\ldots,m, j = 2,\ldots,m$$

$$\text{Working to natural } \gamma_{ij} = \frac{\rho_{ij}}{1 + \sum_{k:k\neq i} exp(\tau_{ik})} \qquad i,j = 1,\ldots,m$$

$$\text{where } \rho_{ij} = \begin{cases} exp(\tau_{ij}) & \text{for } i \neq j \\ 1 & \text{for } i = j \end{cases}$$

2.4 Estimation via the EM Algorithm

As an alternative to direct numerical maximisation, often times the Expectation-Maximisation meta-algorithm is employed. EM is known as a meta-algorithm because rather than prescribing an explicit set of steps, it provides a framework that needs to be customised depending on the context. In the context of HMM's it can be useful due to the fact that the likelihood is guaranteed to increase for each iteration, and also that derivatives are not required (e.g. an optimiser such as `nlm` in R is not required). On the downside however, more implementation effort is often required, and convergence can sometimes be slow to achieve.

2.4.1 EM in General

EM in general is an iterative scheme to find maximum likelihood estimates of parameters when some of the data are missing. Therefore by treating the hidden states of an HMM as those missing data, we can calculate what is known as the complete data log likelihood. The

CDLL is the log likelihood of the parameters based on the observed and missing data

The iterative scheme is as follows:

- Choose the starting values of the parameters to be estimated
- E-Step: Compute the conditional expectations of those functions of the missing data that appear in the complete-data log-likelihood.
- M-step: Maximisation of the log-likelihood with respect to the set of parameters to be estimated (the missing data are substituted by their conditional expectation).
- Assess convergence (with respect to some criterion) and repeat the E and M-steps until convergence is reached.

2.4.2 EM for Hidden Markov Models

When applied in the context of an HMM, the CDLL is the log-likelihood of the observations $\mathbf{x}^{(T)}$ and the missing data $\mathbf{c}^{(T)}$ which is given by:

$$\begin{aligned}\log\Big(Pr(\mathbf{x}^{(T)},\mathbf{c}^{(T)})\Big) &= \log\left(\delta_{c_1}\prod_{t=2}^{T}\gamma_{c_{t-1},c_t}\prod_{t=1}^{T}p_{c_t}(x_t)\right)\\ &= \log\,\delta_{c_1}+\sum_{t=2}^{T}\log\,\gamma_{c_{t-1},c_t}+\sum_{t=1}^{T}\log\,p_{c_t}(x_t)\end{aligned}$$

And then by introducing the following indicator variables:

$$\begin{aligned}u_j(t)=1 &\iff c_t=j\\ v_{jk}(t)=1 &\iff c_{t-1}=j \text{ and } c_t=k\end{aligned}$$

We get:

$$\log\Big(Pr(\mathbf{x}^{(T)},\mathbf{c}^{(T)})\Big)=\underbrace{\sum_{j=1}^{m}u_j(1)\log\,\delta_j}_{\text{term 1}}+\underbrace{\sum_{j=1}^{m}\sum_{k=1}^{m}\left(\sum_{t=2}^{T}v_{jk}(t)\right)\log\,\gamma_{jk}}_{\text{term 2}}+\underbrace{\sum_{j=1}^{m}\sum_{t=1}^{T}u_j(t)\log\,p_j(x_t)}_{\text{term 3}} \tag{2.11}$$

This shows how with the use of the indicator variables $u_j(t)$ and $v_{jk}(t)$ the CDLL can partitioned into three terms that can be optimized separately. The first term depends only on the initial distribution, the second term depends only on the transition matrix, and the third term depends only on the parameters related to the state dependent distributions (Poisson, in our continuing example).

2.4.2.1 E Step

The conditional expectations of the functions of the missing data $\mathbf{c}^{(T)}$ given the observations $\mathbf{x}^{(T)}$ must be computed, and hence the quantities $v_{jk}(t)$ and $u_j(t)$ would be replaced by them:

$$\hat{u}_j(t) = Pr(C_t = j|\mathbf{x}^{(T)}) \tag{2.12}$$

$$\hat{v}_{jk}(t) = Pr(C_{t-1} = j, C_t = k|\mathbf{x}^{(T)}) \tag{2.13}$$

In calculating $Pr(C_t = j|\mathbf{x}^{(T)})$ we can use the previously calculated values L_T (2.9) and $\boldsymbol{\alpha_t}$ (2.10) along with a newly introduced backward probability vector $\boldsymbol{\beta_t}$ which is defined as follows:

$$\begin{aligned}\boldsymbol{\beta_t}' &= \mathbf{\Gamma P}(x_{t+1})\mathbf{\Gamma P}(x_{t+2})\dots\mathbf{\Gamma P}(x_T)\mathbf{1}' \\ &= \left(\prod_{s=t+1}^{T}\mathbf{\Gamma P}(x_s)\right)\mathbf{1}'\end{aligned} \tag{2.14}$$

The reason for the name "backward probabilities" is that a recursion backward in time is used to calculate $\boldsymbol{\beta_t}$

We now state the following propositions:[2]

[2] Proved in Zucchini & MacDonald (2009, pp.60–63)

$$\alpha_t(j) = Pr(\mathbf{X}^{(t)} = \mathbf{x}^{(t)}, C_t = j)$$

$$\beta_t(i) = Pr(\mathbf{X}_{t+1}^{T} = \mathbf{x}^{(t)}|C_t = i)$$

$$\begin{aligned}\alpha_t(i)\beta_t(i) &= Pr(\mathbf{X}^{(T)} = \mathbf{x}^{(T)}, C_t = i) \\ \text{and consequently } \boldsymbol{\alpha_t}\boldsymbol{\beta_t}' &= Pr(\mathbf{X}^{(T)} = \mathbf{x}^{(T)}) = L_T\end{aligned} \tag{2.15}$$

And now finally:

$$\begin{aligned}&\text{For } t = 1,\dots,T, \\ Pr(C_t = j|\mathbf{X}^{(T)} = \mathbf{x}^{(T)}) &= \alpha_t(j)\beta_t(j)/L_T \\ &\text{For } t = 2,\dots,T, \\ Pr(C_{t-1} = j, C_t = k|\mathbf{X}^{(T)} = \mathbf{x}^{(T)}) &= \alpha_{t-1}(j)\gamma_{jk}p_k(x_t)\beta_t(k)/L_T\end{aligned}$$

With these steps in place we are now in a position to compute the conditional expectations (2.12) and (2.13) which replace $u_j(t)$ and $v_{jk}(t)$ respectively in the CDDL (2.11):

$$\hat{u}_j(t) = Pr(C_t = j|\mathbf{x}^{(T)}) = \alpha_t(j)\beta_t(j)/L_T \tag{2.16}$$

$$\hat{v}_{jk}(t) = Pr(C_{t-1} = j, C_t = k|\mathbf{x}^{(T)}) = \alpha_{t-1}(j)\gamma_{jk}p_k(x_t)\beta_t(k)/L_T \tag{2.17}$$

2.4.2.2 M Step

After the E step is complete, we now follow the M Step by maximising the CDDL (2.11) with respect to the three sets of parameters of interest, $\boldsymbol{\delta}$, $\boldsymbol{\Gamma}$ and the parameters of the state-dependent distributions, namely λ_1,...,λ_m.

We replace $p_j(x_t)$ in the CDDL with $e^{-\lambda_j}\lambda_j^x/x!$

Term to maximise	Parameter	Solution
$\sum_{j=1}^{m} \hat{u}_j(1)\log \delta_j$	$\boldsymbol{\delta}$	$\delta_j = \frac{\hat{u}_j(1)}{\sum_{j=1}^{m} \hat{u}_j(1)} = \hat{u}_j(1)$
$\sum_{j=1}^{m}\sum_{k=1}^{m}\left(\sum_{t=2}^{T} \hat{v}_{jk}(t)\right)\log \gamma_{jk}$	$\boldsymbol{\Gamma}$	$\gamma_{jk} = \frac{\sum_{t=2}^{T} \hat{v}_{jk}(t)}{\sum_{k=1}^{m}\sum_{t=2}^{T} \hat{v}_{jk}(t)}$
$\sum_{j=1}^{m}\sum_{t=1}^{T} \hat{u}_j(t)\log e^{-\lambda_j}\lambda_j^x/x!$	$\lambda_1, \ldots, \lambda_m$	$0 = \sum_{t=1}^{T} \hat{u}_j(t)(-1 + x_t/\lambda_j)$
		that is, by $\hat{\lambda}_j = \frac{\sum_{t=1}^{T} \hat{u}_j(t)x_t}{\sum_{t=1}^{T} \hat{u}_j(t)}$

3
HMM Model Applied to VPIN

3.1 Calculating the Likelihood

Now that the mathematical foundations of the HMM have been laid down, we are ready to extend the model for the problem that this paper is examining. As explained in the introduction, the data we are attempting to fit an HMM with is the number of buy and sell orders per volume bucket. That is, for each volume bucket t we will have a bivariate observable trading process $\{X_t \equiv (B_t, S_t) : t = 1, 2, \dots T\}$. B_t and S_t are defined as the buyer and seller-initiated orders. We can further define the unobserved hidden state stochastic process $\{C_t \equiv (C_{b;t}, C_{s;t}) : t = 1, 2, \dots T\}$.

The probability of observing b_t buy orders and s_t sell orders is driven by the fact that they arrive according to independent Poisson process, and as such, conditional on the state at a particular volume bucket t is (i, j) we can state:

$$\begin{aligned} p_{i,j}(X) &= Pr(X_t = x | C_{b;t} = i, C_{s;t} = j) \\ &= p_i(b_t) p_j(s_t) \\ &= e^{-\lambda_{b;i}} \frac{(\lambda_{b;i})^{b_t}}{b_t!} \times e^{-\lambda_{s;j}} \frac{(\lambda_{s;j})^{s_t}}{s_t!} \end{aligned} \tag{3.1}$$

This implies that any state (i, j) that the market is in for the duration of the volume bucket is represented by a pair of distribution parameters $\lambda_{b;i}$ and $\lambda_{s;j}$.

The $mn \times mn$ diagonal matrix of conditional probabilities $P(x)$ is defined as:

$$\mathbf{P}(x) = \begin{pmatrix} p_1(b_t)p_1(s_t) & & 0 \\ & \ddots & \\ 0 & & p_m(b_t)p_n(s_t) \end{pmatrix}$$

We can now extend the unconditional state distribution defined in

(2.4) as follows:

$$u_{i,j}(t) = Pr(C_{b;t} = i, C_{s;t} = j), t = 1, \ldots, T$$

Or in matrix format:

$$u_{i,j}(t) = \Big(u_{1,1}(t), \ldots, u_{1,n}(t), \ldots u_{m,1}(t), \ldots u_{m,n}(t)\Big)$$

The marginal distribution of X extends logically from (2.5), showing the unconditional probability of observing $x_t = (b_t, s_t)$ within volume bucket t:

$$\begin{aligned}
Pr(X_t = x) &= \sum_{i=1}^{m}\sum_{j=1}^{n} Pr(C_{b;t} = i, C_{s;t} = j) Pr(X_t = x | C_{b;t} = i, C_{s;t} = j) \\
&= \sum_{i=1}^{m}\sum_{j=1}^{n} u_{i,j}(t) p_{i,j}(x) \\
&= \Big(u_{1,1}(t), \ldots, u_{1,n}(t), \ldots u_{m,1}(t), \ldots u_{m,n}(t)\Big) \begin{pmatrix} p_1(b_t)p_1(s_t) & & 0 \\ & \ddots & \\ 0 & & p_m(b_t)p_n(s_t) \end{pmatrix} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \\
&= \mathbf{u}(t)\mathbf{P}(x)\mathbf{1}'
\end{aligned}$$

The transition probability matrix $\mathbf{\Gamma}(t)$ which is the $mn \times mn$ matrix of transition probabilities at state t is defined as:

$$\mathbf{\Gamma}(1) = \begin{pmatrix}
\gamma_{1,1;1,1} & \gamma_{1,1;1,2} & \cdots & \gamma_{1,1;m,n-1} & \gamma_{1,1;m,n} \\
\gamma_{1,2;1,1} & \gamma_{1,2;1,2} & & \gamma_{1,2;m,n-1} & \gamma_{1,2;m,n} \\
\vdots & & \ddots & \vdots & \\
\gamma_{m,n-1;1,1} & \gamma_{m,n-1;1,2} & & \gamma_{m,n-1;m,n-1} & \gamma_{m,n-1;m,n} \\
\gamma_{m,n;1,1} & \gamma_{m,n;1,2} & \cdots & \gamma_{m,n;m,n-1} & \gamma_{m,n;m,n}
\end{pmatrix}$$

Where $\gamma_{i,j;k,l}(t) = Pr(C_{b;t+1} = k, C_{s;t+1} = l | C_{b;t} = i, C_{s;t} = j)$ is defined as the probability of volume bucket $t + 1$ being in the state (k, l) given that at bucket t was in state (i, j).

The likelihood L_T for the bivariate independent Poisson HMM is then formulated in exactly the same way as for the basic model (2.9) by calculating recursively using the forward probabilities $\boldsymbol{\alpha_t}$:

$$L_T = \boldsymbol{\delta}\mathbf{P}(x_1)\mathbf{\Gamma}\mathbf{P}(x_2)\ldots\mathbf{\Gamma}\mathbf{P}(x_T)\mathbf{1}'$$

3.2 Estimation via EM

By examining the original CDDL equation we introduced in (2.11), we can start to extend this for the bivariate Poisson case in order to estimate VPIN:

$$\log\Big(Pr(\mathbf{x}^{(T)},\mathbf{c}^{(T)})\Big)=\sum_{j=1}^{m}u_j(1)\log\,\delta_j+\sum_{j=1}^{m}\sum_{k=1}^{m}\left(\sum_{t=2}^{T}v_{jk}(t)\right)\log\,\gamma_{jk}+\sum_{j=1}^{m}\sum_{t=1}^{T}u_j(t)\log\,p_j(x_t)$$

First we extend the original indicator variables :

Confusingly, @zucchini re-use the notation u_j which has already been used to denote the unconditional state distribution

$$\begin{aligned}u_{i,j}(t)=1 &\iff C_{b;t}=i \text{ and } C_{s;t}=j\\ v_{i,j;k,l}(t)=1 &\iff C_{b;t-1}=i\ ,\ C_{s;t-1}=i\ ,\ C_{b;t}=k\ ,\ \text{and } C_{s;t}=l\end{aligned}$$

To give us the following CDDL:

$$\log\Big(Pr(\mathbf{x}^{(T)},\mathbf{c}^{(T)})\Big)=\sum_{i=1}^{m}\sum_{j=1}^{n}u_{i,j}(1)\log\,\delta_{i,j}+\sum_{i,k=1}^{m}\sum_{j,l=1}^{n}\left(\sum_{t=2}^{T}v_{i,j;k,l}(t)\right)\log\,\gamma_{i,j;k,l}(t)+\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{t=1}^{T}u_{i,j}(t)\log\,p_{i,j}(x_t)\tag{3.2}$$

3.2.1 E Step

Again, following the univariate examples (2.16) and (2.17) we find that the conditional expectations of the missing data given the observations are as follows:

$$\begin{aligned}\hat{u}_{i,j}(t)&=Pr(C_{b;t}=i,C_{s;t}=j|\mathbf{x}^{(T)})\\&=\alpha_t(i,j)\beta_t(i,j)/L_T\end{aligned}$$

$$\begin{aligned}\hat{v}_{i,j;k,l}(t)&=Pr(C_{b;t-1}=i,C_{s;t-1}=j,C_{b;t}=k,C_{s;t}=l|\mathbf{x}^{(T)})\\&=\alpha_{t-1}(i,j)\gamma_{i,j;k,l}p_{k,l}(x_t)\beta_t(k,l)/L_T\end{aligned}$$

3.2.2 M Step

On completing the E step is complete, we now maximise the CDDL (3.2) with respect to the four sets of parameters, $\boldsymbol{\delta}$, $\boldsymbol{\Gamma}$, $\lambda_{b;i}$ and $\lambda_{s;j}$. The state dependent distribution $p_{i,j}(x_t)$ as stated here (3.1) is the independent bivariate Poisson distribution: $e^{-\lambda_{b;i}}\frac{(\lambda_{b;i})^{b_t}}{b_t!}e^{-\lambda_{s;j}}\frac{(\lambda_{s;j})^{s_t}}{s_t!}$

For proof of this maximisation see Appendix 3

Term to maximise	Parameter	Solution
$\sum_{i=1}^{m}\sum_{j=1}^{n} u_{i,j}(1)\log\,\delta_{i,j}$	$\boldsymbol{\delta}$	$\delta_{i,j} = \frac{\hat{u}_{i,j}(1)}{\sum_{i=1}^{m}\sum_{j=1}^{n}\hat{u}_{i,j}(1)}$
$\sum_{i,k=1}^{m}\sum_{j,l=1}^{n}\left(\sum_{t=2}^{T} v_{i,j;k,l}(t)\right)\log\,\gamma_{i,j;k,l}(t)$	$\boldsymbol{\Gamma}$	$\gamma_{i,j;k,l} = \frac{\sum_{t=2}^{T}\hat{v}_{i,j;k,l}(t)}{\sum_{k'=1}^{m}\sum_{l'=1}^{n}\sum_{t=2}^{T}\hat{v}_{i,j;k',l'}(t)}$
$\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{t=1}^{T} u_{i,j}(t)\log\,p_{i,j}(x_t)$	$\lambda_{b;i}$	$\hat{\lambda}_{b;i} = \frac{\sum_{j=1}^{n}\sum_{t=1}^{T}\hat{u}_{i,j}(t)b_t}{\sum_{j=1}^{n}\sum_{t=1}^{T}\hat{u}_{i,j}(t)}$
	$\lambda_{s;j}$	$\hat{\lambda}_{s;j} = \frac{\sum_{i=1}^{m}\sum_{t=1}^{T}\hat{u}_{i,j}(t)s_t}{\sum_{i=1}^{m}\sum_{t=1}^{T}\hat{u}_{i,j}(t)}$

We repeat the above iterative scheme a large number of times until we meet our convergence criteria. In this instance we repeat until the marginal improvement in the CDDL is less than 10^{-6}.

The R code for this algorithm can be found in appendix TODO

The problem remains that without suitable starting values for the initial distribution $\boldsymbol{\delta}$, and the transition matrix $\boldsymbol{\Gamma}$, the EM algorithm may converge on local not global optima. Therefore a strategy for mitigating this problem is required. In the simulation study we outline our approach to solving this problem by way of estimation of trading motives via K-Means clustering

3.3 Decoding

A result that shall be used later on in this work is related to identifying the most likely state at a given t given the history of the observed values This process is known as decoding.

As shown in (2.15) the joint distribution was stated as:

$$Pr(\mathbf{X}^{(T)} = \mathbf{x}^{(T)}, C_t = i) = \alpha_t(i)\beta_t(i)$$

And hence, using Bayes rule, we can deduct the conditional distribution of C_t given the observations as:

$$\begin{aligned} Pr(C_t = i|\mathbf{X}^{(T)} = \mathbf{x}^{(T)}) &= \frac{Pr(\mathbf{X}^{(T)} = \mathbf{x}^{(T)}, C_t = i)}{Pr(\mathbf{X}^{(T)} = \mathbf{x}^{(T)})} \\ &= \frac{\alpha_t(i)\beta_t(i)}{L_T} \end{aligned} \tag{3.3}$$

4
A Simulation Analysis

4.1 Simulate data according to model

We employ a Monte Carlo algorithm to simulate order flow based off the known parameters: α (the probability of information event), μ (the arrival rate of informed traders) and ϵ (the arrival rate of uninformed traders). Once we have the simulated order flow we can use the VPIN estimation procedure to give an estimation of the VPIN metric.

1. Set Monte Carlo Parameters:
   (a) Number of Simulations: S
   (b) VPIN: (V, n)
   (c) PIN: (α, μ)
2. Set $\epsilon = \frac{V - \alpha\mu}{2}$, $s = 0$, $j = 0$
3. $j = j + 1$
4. Draw three random numbers from a $U(0, 1)$ distribution: u_1, u_2, u_3

In our R implementation of the VPIN simulation we do not in fact employ u_3 as the parameter to the inverse CDF of the Poisson distribution as R already comes with an efficient way to simulate random Poisson variables, namely the function 'rpois'

5. If $u_1 < \alpha$
   (a) If $u_2 < \delta$
      i. $V_j^B = F^{-1}(u_3, \epsilon)$
      ii. $V_j^S = F^{-1}(u_3, \mu + \epsilon)$
   (b) If $u_2 \geq \delta$
      i. $V_j^B = F^{-1}(u_3, \mu + \epsilon)$
      ii. $V_j^S = F^{-1}(u_3, \epsilon)$
6. If $u_1 < \alpha$
   (a) $V_j^B = F^{-1}(u_3, \epsilon)$

(b) $V_j^S = V_j^B$

7. If $j = n$

   (a) $j = 0$

   (b) $s = s + 1$

   (c) $VPIN_s = \frac{\sum_{j=1}^{n} |V_j^S - V_j^B|}{\sum_{j=1}^{n} (V_j^S - V_j^B)}$

8. If $s < S$, loop to Step 3

9. Compute results:

   (a) $E[VPIN] = \frac{1}{S} \sum_{s=1}^{S} VPIN_s$

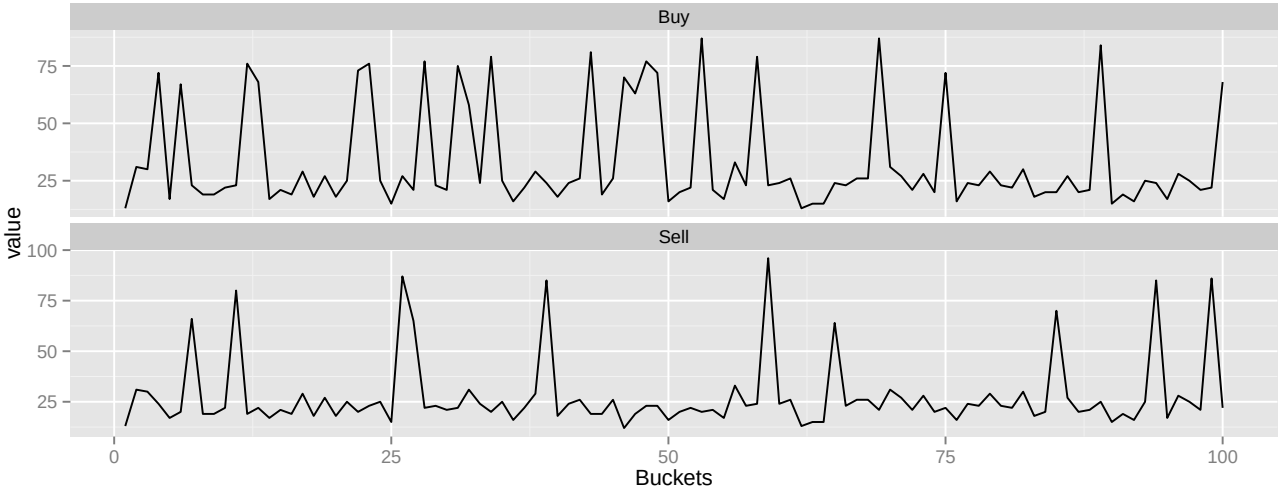
   (b) $V[VPIN] = \frac{1}{S-1} \sum_{s=1}^{S} VPIN_s^2 - \frac{1}{S(S-1)} (\sum_{s=1}^{S} VPIN_s)^2$

To illustrate the analysis of the simulated data, we shall examine the steps for one hypothetical trading day, using the following parameters:

- $\alpha = 0.28$
- $\delta = 0.33$
- $\mu = 55$
- $\epsilon = 22.3$

4.2 Exploratory analysis of data

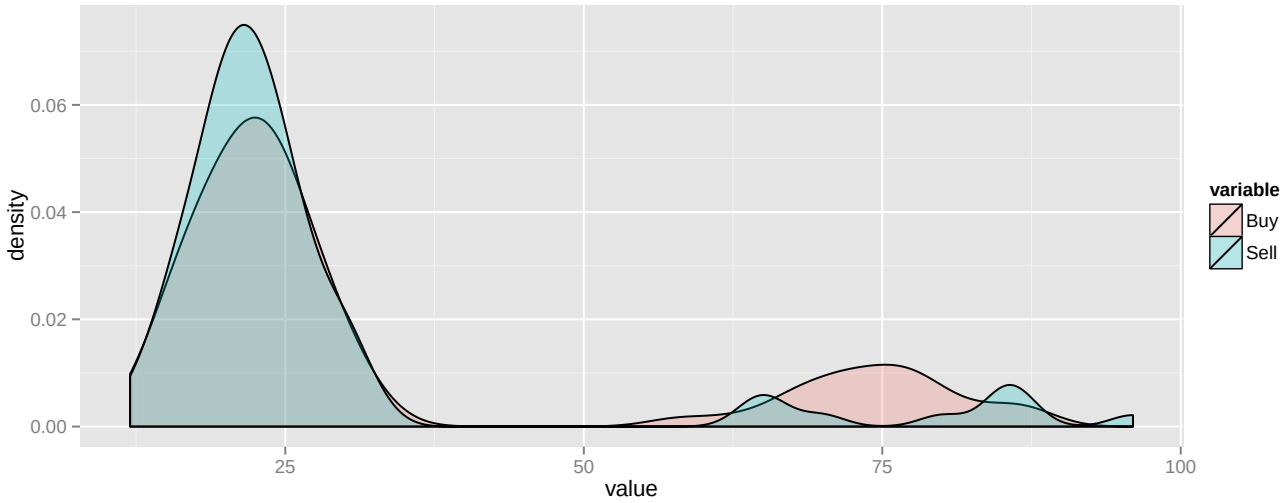
By first running some elementary visualisations against the simulated data we can gain an intuitive insight into the patterns that the data exhibit.



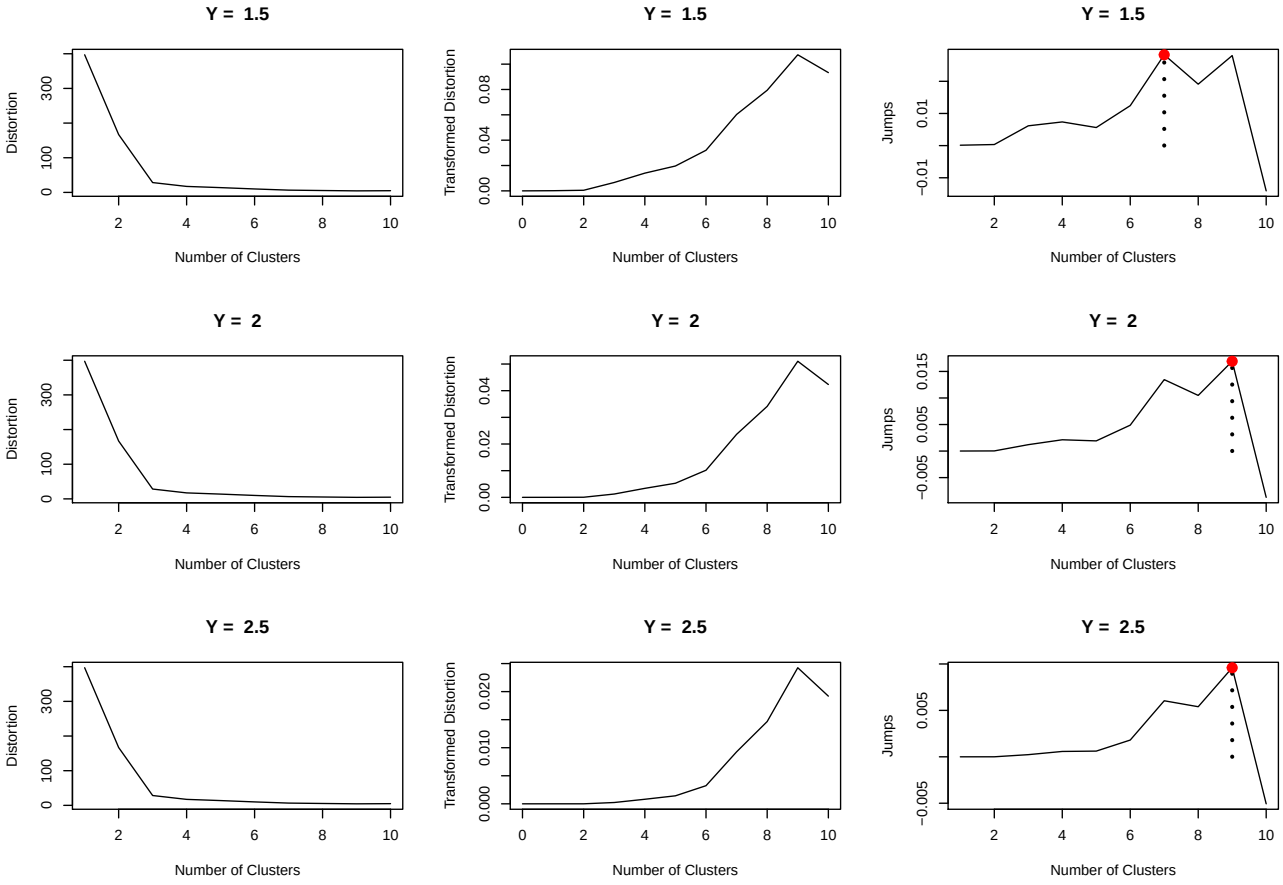
A simple time series plot of buys and sells appears to show longer periods of relative inactivity which are then interspersed by periods of

higher trading activity. This leads us to believe that the data are generated by a mixture of distributions which are regulated by some sort of mixing parameter. As we know in this situation the distributions are indeed driven by α and δ.

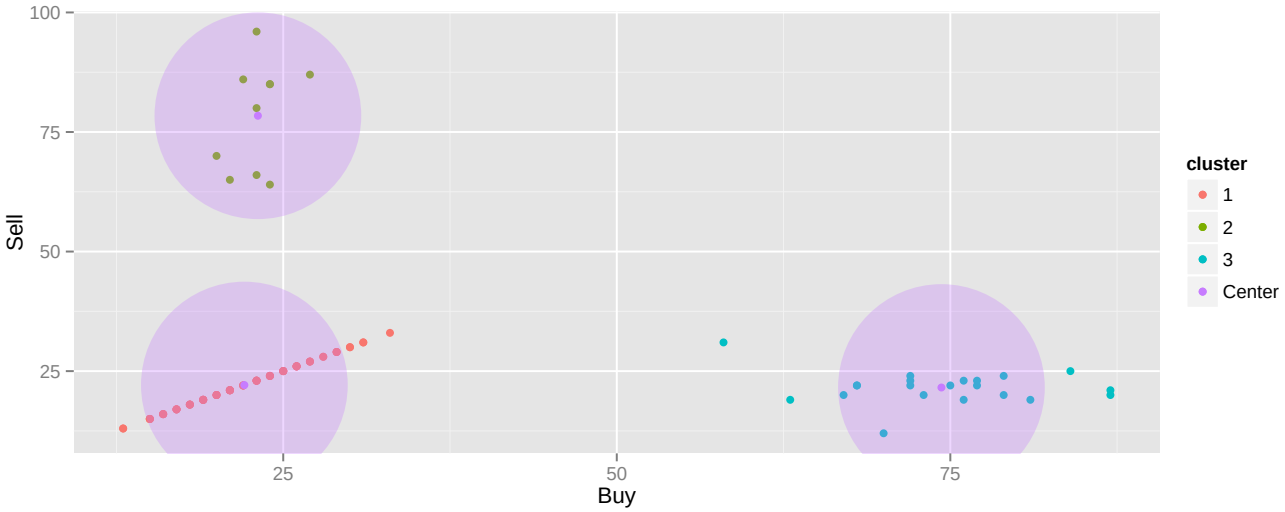
To get a sense of the relative magnitudes of these component distributions, we fit kernel density estimates against the data (kernel=gaussian, bandwidths=2.67 and 1.87 for buys and sells, respectively). Firstly we can see that both of the series are clearly overdispersed compared to a single Poisson distribution. Secondly we notice that both buys and sells appear to have two or three modes each. Again this conforms precisely with our knowledge of the underlying data generating process.



Following the Jump methodology of Sugar & Gareth (2003) we run the k-means clustering algorithm against the trade imbalance data for $K = 1 \ldots 10$. Sugar & Gareth (2003) provide a rigourous theoretical justification for the ability of the Jump method to calculate the optimum number of clusters. The following output shows that there is an elbow in the distortion levels when the cluster size is three. Again this matches with our knowledge of the underlying data generating process whereby we are either in a) a no-news state b) a news state with private information, or c) a news state without private information.



With the optimal number of clusters identified we can now proceed with the k-means analysis on the data and thereby visualise the clusters:



	Buy	Sell	Cluster Size
1	22.09	22.09	69.00
2	23.10	78.40	10.00
3	74.33	21.57	21.00

We can now interpret the cluster centroids in order to find appropriate starting values of ϵ, μ, $\lambda_{b;i}$ and $\lambda_{s;j}$. It is unrealistically simple given our small simulated data set, but the intuition is rather obvious. We know a priori that there are three states:

1. No news event
2. Good news event
3. Bad news event

For each state i we know there is a particular $\lambda_{b;i}$ and $\lambda_{s;i}$

$$\lambda_b = \begin{pmatrix} \epsilon_b \\ \epsilon_b + \mu_b \\ \epsilon_b \end{pmatrix} \qquad \lambda_s = \begin{pmatrix} \epsilon_s \\ \epsilon_s \\ \epsilon_s + \mu_s \end{pmatrix}$$

We infer that because most trading days (or buckets) do not have private information, we can deduce that the cluster with the greatest size is the one without any private information. From which we can further deduce that the true value ϵ the arrival rate of the uninformed trader must be close to that cluster's centroid. Based off this justification we identify the following ϵ values :

$$\epsilon_b = 22.087 \qquad \epsilon_s = 22.087$$

Next we infer that the cluster with the largest Buy centroid indicating a large order imbalance is the one with private information after a good news event. Hence:

$$\begin{aligned} \lambda_{b;2} = \epsilon_b + \mu_b \qquad \Rightarrow \qquad \mu_b &= \lambda_{b;2} - \epsilon_b \\ &= 74.333 - 22.087 \\ &= 52.246 \end{aligned}$$

Similarly we infer that the cluster with the largest Sell centroid indicating a large order imbalance is the one with private information after a bad news event. Hence:

$$\begin{aligned} \lambda_{s;2} = \epsilon_s + \mu_s \qquad \Rightarrow \qquad \mu_s &= \lambda_{s;2} - \epsilon_s \\ &= 78.4 - 22.087 \\ &= 56.313 \end{aligned}$$

Hence we can formulate the initial state parameter estimates:

$$\lambda_b = \begin{pmatrix} \epsilon_b \\ \epsilon_b + \mu_b \\ \epsilon_b \end{pmatrix} = \begin{pmatrix} 22.087 \\ 74.333 \\ 22.087 \end{pmatrix} \qquad \lambda_s = \begin{pmatrix} \epsilon_s \\ \epsilon_s \\ \epsilon_s + \mu_s \end{pmatrix} = \begin{pmatrix} 22.087 \\ 22.087 \\ 78.4 \end{pmatrix}$$

4.3 Fit HMM to data given inital values

A standard approach for model selection in the HMM literature is via the use of information criteria such as Akaike or Bayesian Information Criteria (AIC and BIC respectively)Zucchini & MacDonald (2009, pp.89–92). The theory of these criteria state that the model with the lowest number will be the one that most closely fits data, taking into account the theory of parsimony.

These criteria are stated as:

$$\begin{aligned} \text{AIC} &= -2 \log L + 2p \\ \text{BIC} &= -2 \log L + p \log T \end{aligned}$$

Where L is the likelihood, p is the number of parameters and T is the number of observations.

We generate AIC and BIC values for a variety of different states m and make a model selection based off these calculation.

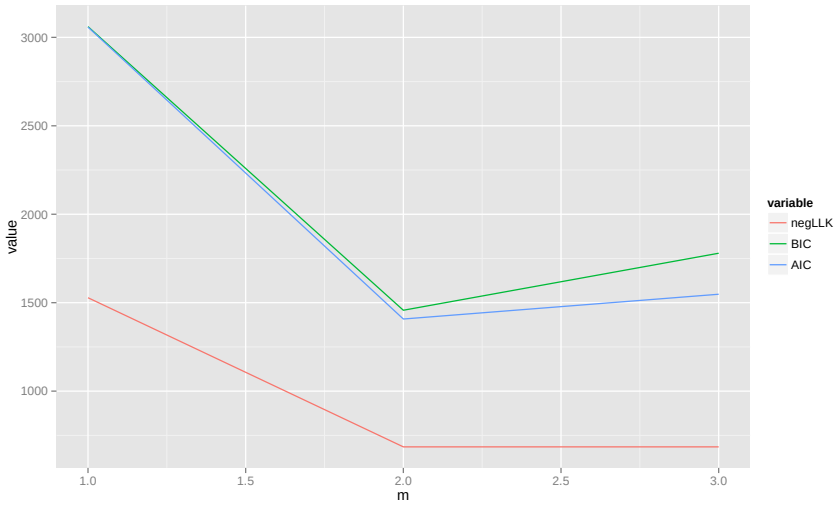


Figure 4.1: Diagostics plot for numbers of states.

In the above plot we show AIC and BIC values (along with negative log-likelihood for interest) for $m = 1, 2, 3$. We can see that the two and three state models appear to be dramatically lower that the one state model. Even though the three state model has a slightly larger AIC and BIC, for the purposes of interpretation we choose $m = 3$ and present the fitted model parameters as follows:

	Lambda.Buy.Hat	Lambda.Sell.Hat
1	22.22	21.97
2	74.33	21.97
3	22.22	78.40

Table 4.1: Lambda Buy and Sell: Fitted arrival rates of informed and uninformed traders

Which as can be seen below, appear to be very close to the original parameters of the simulated data, thereby demonstrating to us the effectiveness of the HMM in estimating model parameters.[1]

[1] A more thorough analysis would involve bootstrapping these parameter estimates in order to generate confidence intervals.

$$\lambda_b = \begin{pmatrix} \epsilon_b \\ \epsilon_b + \mu_b \\ \epsilon_b \end{pmatrix} = \begin{pmatrix} 22.3 \\ 77.3 \\ 22.3 \end{pmatrix} \qquad \lambda_s = \begin{pmatrix} \epsilon_s \\ \epsilon_s \\ \epsilon_s + \mu_s \end{pmatrix} = \begin{pmatrix} 22.3 \\ 22.3 \\ 77.3 \end{pmatrix}$$

4.4 Local Decoding for Hidden State Identification

On identification of the 'best' model, use this fitted model to run analysis of simulated data to see if expected hidden state matches the predetermined hidden state for all $t = 1 \dots T$

Because of the mathematical setup of the bivariate Poisson HMM, we are required to perform a translation step between the true states and the calculated states, somewhat analogously to the process we performed as part of the likelihood calculation whereby we translated parameters from natural to working representations. To illustrate let us look at two hypothetical λ vectors for $m = 3$:

$$\lambda_b = \begin{pmatrix} \lambda_{b1} \\ \lambda_{b2} \\ \lambda_{b3} \end{pmatrix} \qquad \lambda_s = \begin{pmatrix} \lambda_{s1} \\ \lambda_{s2} \\ \lambda_{s3} \end{pmatrix}$$

When formulating our parameter matrices, to represent the states in one dimension, we generate the Cartesian product of these parameters representing them with a unique indexer:

	Lambda Buy	Lambda Sell
1	b1	s1
2	b2	s1
3	b3	s1
4	b1	s2
5	b2	s2
6	b3	s2
7	b1	s3
8	b2	s3
9	b3	s3

In order to interprete the decoded states calculated by the HMM our remaining task is to translate the estimated 'working' state index to the 'natural' state index. When generating our simulated data we apply arbitrary state labels to each of the generated obvservations whereby:

- 1: No news state
- 2: Bad news state
- 3: Good news state

With these labels applied, we can output the simulated true state values:

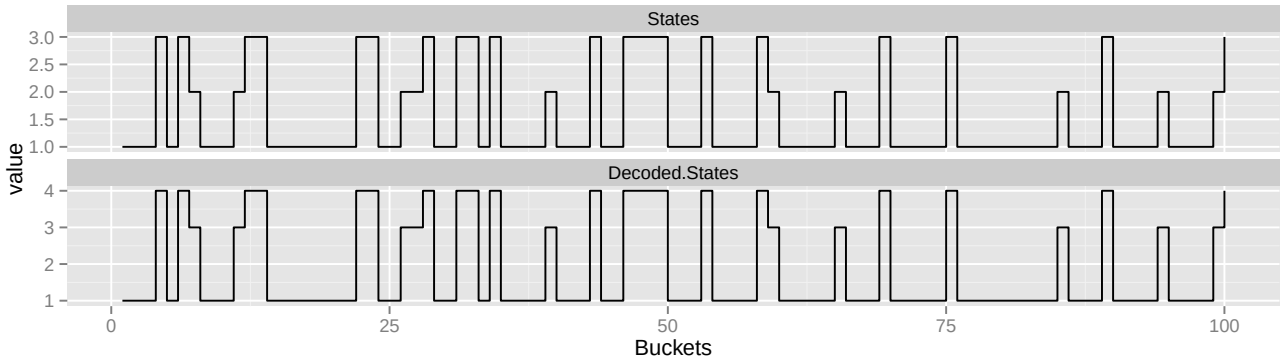


Figure 4.2: Untranslated decoded states

By examining the contrast between the untranslated decoded states and the true states we can easily interpret the following translation scheme:

- Working State 1 ⇒ Natural State 1
- Working State 4 ⇒ Natural State 3
- Working State 3 ⇒ Natural State 2

And after applying this translation to the decoded states, we see that the decoded states match the true states with 100% accuracy:

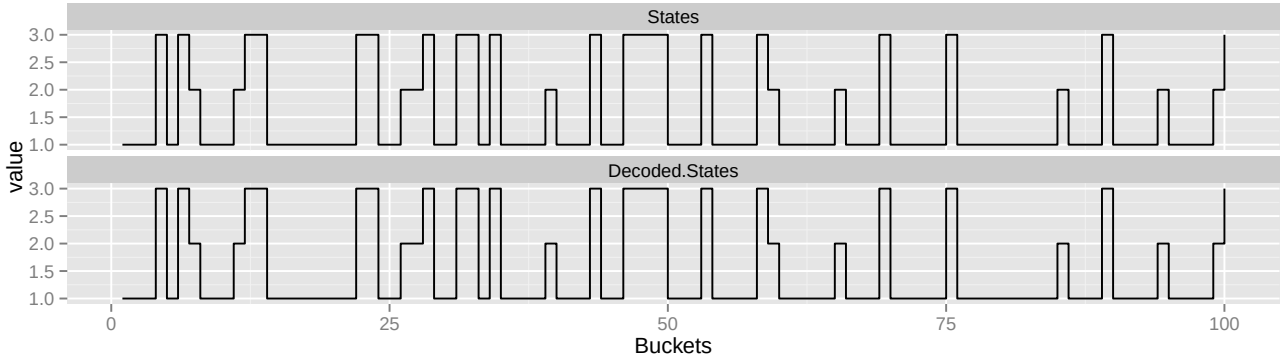


Figure 4.3: Translated decoded states

4.5 Analyse Relationship Between Hidden States and VPIN

The final step in our analysis is to make inferences about the rleationship between these decoded hidden states and the calculated VPIN metric. The hypothesis we are examining is whether there is a strong relationship between high VPIN values and private information-based trading. We challenge we face however is determining exactly what we are comparing.

First we briefly repeat how VPIN is calculated. Trading activity is typically measured in time bars, that is a summary of the tick-by-tick activity within a particular time period, e.g. seconds, minutes, hours etc. As part of our simulation analysis up to this point we have created time series showing generated buys and sells aggregated by time bars. As explained in the introduction, due to the difficulties of classifying trading activity as either buy or sell in a high-frequency context, trade data is spliced up into volume bars and these bars are then probabilistically allocated as buys or sells by examining the price differentials within those volume bars. Once the probabilistic buy and sell amounts have been calculated per volume bar, the trading imbalances and hence the VPIN can be calculated. The missing piece of the analysis puzzle is hence how to associate the decoded hidden state of a time bar with the VPIN value of the volume bar. The following is our attempt to define that linkage.[2]

[2] The procedure is an extension of the algorithm found in Appendix 1.

TimeBar	1	2	3	4	5
Buy	1	2	2	3	11
Sell	2	4	5	7	3
State	1	2	3	2	3
Total	3	6	7	10	14

VolBar	1										2										3										4									
State	1	1	1	2	2	2	2	2	2	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3	3	3	3	3	3	3	3
State (Mode)	**2**										**3**										**2**										**3**									
Buy	1.7										2.4										6.2										11									
Sell	3.5										5.8										5.4										3									
Imbalance	1.8										3.4										0.8										8									
VPIN	**0.18**										**0.34**										**0.08**										**0.8**									

Table 4.5 demonstrates a simple example of the process by which we translate from time bars to volume bars. The first table shows a simulated realisation of five time bars. Each bar has a buy and sell value and a decoded state associated with that time bar. In our example we assume that the size of each volume bar is $V = 10$. We expand each time bar up into the total number of trades per bar e.g. timebar 1 is split into 3 entries and timebar 2 is split into 6 entries. We

continue this for all the time bars. Once all the time bars have been expanded out we group these entries up into volume bars all equally sized at V.[3]

[3] The R code used to implement this translation can be found in Appendix 2.

In the volume bar schematic, the relative sizes of the bars are all to scale and indicate that as time progresses (left to right) and trading activity increases, the number of volume bars required to represent the time bars increases. The amount of information in each bar remains constant and hence they are directly comparable with each other.

Each time bar that composes the volume bars has an associated time bar state. What we need to do is decide what is the most likely hidden state for the volume bar. To do this we determine the mode of all the states in the volume bar. For example volume bar 1, state 2 is the most frequent state therefore 2 becomes the volume bar state.

The next step is to calculate the probabilistic buy and sell amounts for each volume bar. Because the input to this translation process is in fact signed order flow data, we don't need to infer the trade directions based off the price differentials as in the original VPIN algorithm. Instead we simply take a weighted average of the constituent time bars' buy and sell values. For example because volume bar 1 is composed of 30% from time bar 1, 60% from time bar 2, and 10% from time bar 3, we evaluate volume bar 1's buy and sell amounts as follows:

$$\begin{aligned} V_{b1} &= 0.3 \times T_{b1} + 0.6 \times T_{b2} + 0.1 \times T_{b3} \\ &= 0.3 \times 1 + 0.6 \times 2 + 0.1 \times 2 \\ &= 1.7 \end{aligned}$$

$$\begin{aligned} V_{s1} &= 0.3 \times T_{s1} + 0.6 \times T_{s2} + 0.1 \times T_{s3} \\ &= 0.3 \times 2 + 0.6 \times 4 + 0.1 \times 5 \\ &= 3.5 \end{aligned}$$

We then take the absolute difference between these two buy and sell values and divide them by V to get that volume bar's VPIN value:

$$VPIN_\tau = \frac{|V_{b\tau} - V_{s\tau}|}{V}$$

A small sample of the generated volume bars from the simulated data is shown below:

	State	Buy	Sell	Imbalance	VPIN
4	1	66.8	52.5	14.3	0.238333333333333
5	3	82	43	39	0.65
6	3	72.0833333333333	44.1333333333333	27.95	0.465833333333333
7	1	47	47	0	0
8	1	49.8666666666667	49.8666666666667	0	0
9	1	51.2833333333333	51.2	0.0833333333333357	0.00138888888888893

A simple visual exploration of the difference in means between the identified states is shown below. Visually we can see that there is a large difference in the VPIN values between state 1, and states 2 and 3. We can interprete that as telling us the no-news event appears to be correlated with low values of VPIN, while the private information states tend to be associated with higher VPIN values.

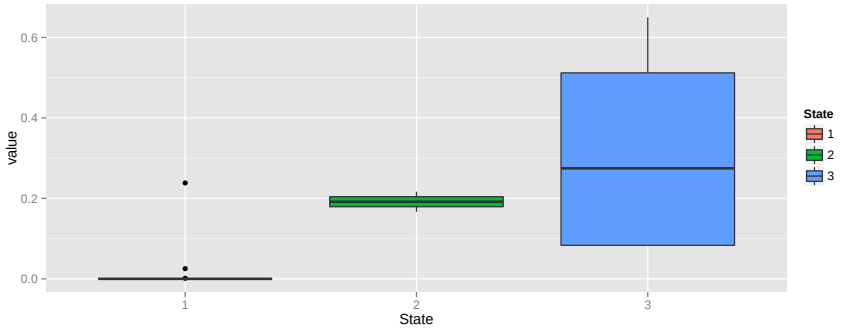


Figure 4.4: Box plots of VPIN values against decoded HMM states

To further test the strength of this relationship we conduct multiple linear regression of VPIN against State which give us the following results (we define state 1, 2 and 3 as dummy variables with state 1 set as the baseline):

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	0.0221	0.0405	0.55	0.5935
State2	0.1696	0.1071	1.58	0.1342
State3	0.2985	0.0810	3.69	0.0022

The results of the regression indicate to us that there does indeed appear to be a very strong ($p << 0.01$) indication that VPIN values associated with the no-news state are going to be significantly lower than VPIN values from the private information states.

5
Empirical Data Analysis

5.1 The Data

In order to validate the assumptions of the preceeding chapters the next step was to run an empirical data analysis. The data we chose to analyse was the SPDR S&P 500 exchange-traded fund (ticker symbol SPY) as this one one of the financial products that was most adversely affected in the May 6 2010 Flash Crash. The reason why we analyse SPY instead of the E-Mini S&P 500 futures contracts as per Easley et al. (2011) is due to the fact that the data we had available was NYSE TAQ, and the E-Mini data would only have been available from the Chicago Mercantile Exchange. It is our hope that SPY will be a suitable proxy for the E-Mini.

On downloading the data from the NYSE TAQ database for ticker SPY for the 6th of May 2010, we found we had 1,549,379 trades and 8,343,441 quotes in raw CSV format. For various reasons, raw trade and quote data contains numerous data errors (Brownlees & Gallo 2006). Therefore, the data is not suited for analysis right-away and data-cleaning is an essential step in dealing with tick-by-tick data. Trade cleanup involved removing zero prices, limiting the observations to a single exchange, filtering by sale condition, and merging any records with the same timestamp. This was then followed by quote cleanup procedure which was responsible for removing any errant values with abnormally large spreads and outliers where the mid-quote deviated by more than 25 median absolute deviations from a rolling centred median. Once the data were cleaned up the trades and quotes were matched up using the technique as outlined in Vergote (2005). Finally becuase the data from the exchange doesn't report the trade direction, we employed the Lee & Ready (1991) approach to inferring whether individual trades were buys or sells.

After this preprocessing, the consolidated trade and quote dataset now comprised of 791,173 entries, which by converting the data into one second time bars was then further compressed into 24,839 records,

a dataset size that is much more tractable for in-memory analysis in R. The data are displayed visually in the following plots:

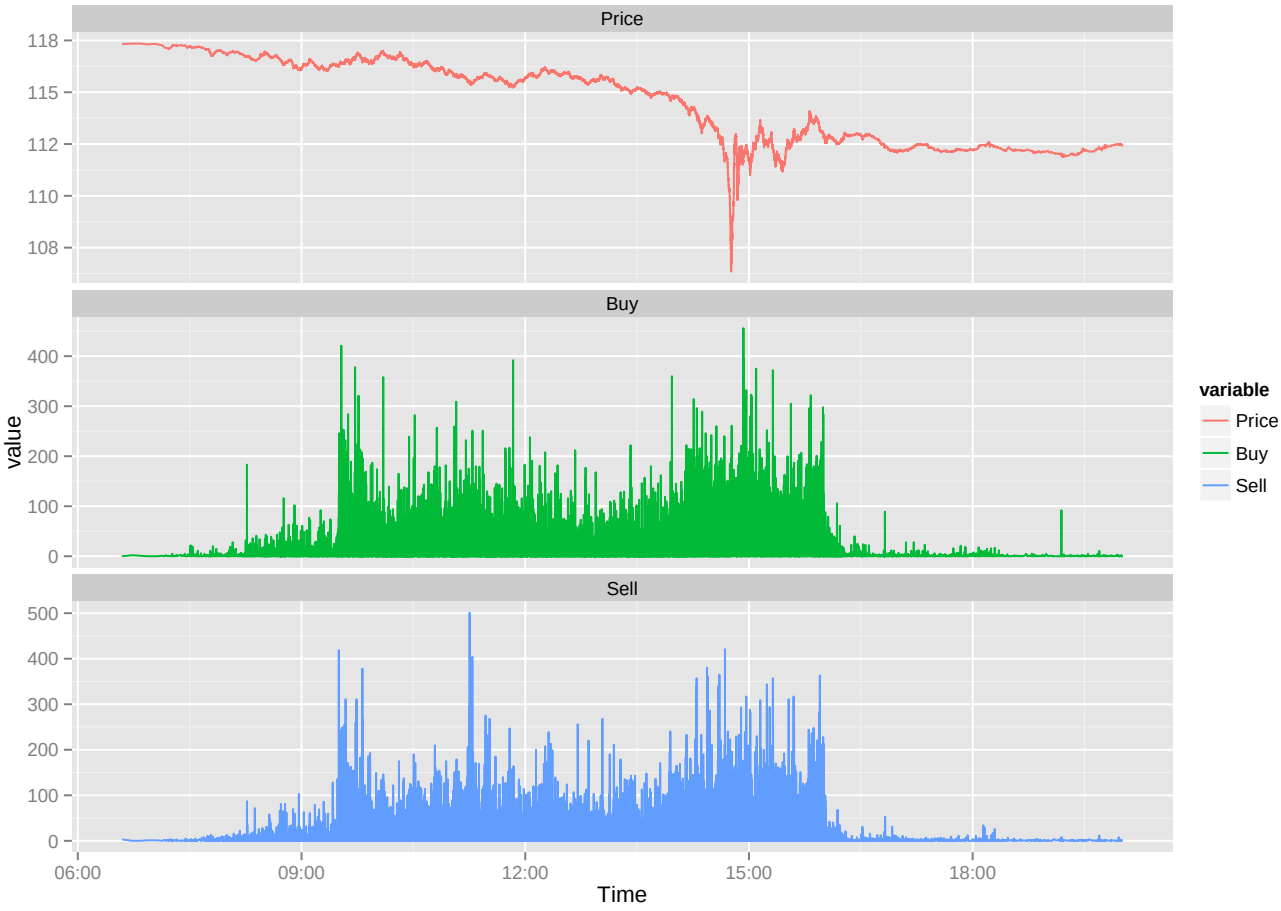


Figure 5.1: Time series of price and inferred buys and sells

5.2 Analysis Procedures

A visual examination of the data show the clear downward spike in price, matching the known events that were known to happen between 2:32 and the subsequent 36 minutes. However there does not appear to be any untoward pattern in the number of buys and sells, apart from a marginal uptick in activity after the crash, nothing that seems out of character when compared to the rest of the trading day.

The fitted density of the buys and sells however lead us to potential issues with the modelling approach we have undertaken. First of all neither of the distributions appear to match a simple Poisson mixture distribution. It is however a possibility that the data observations from the lower tail constitute other component distributions for some underlying reason so far unexplained in this work. The second issue is

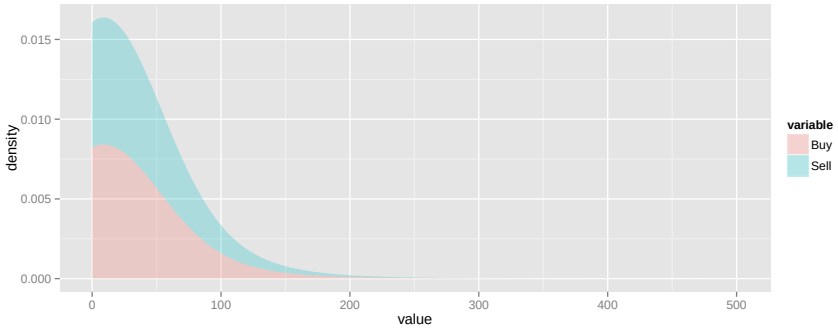


Figure 5.2: Fitted density of inferred buys and sells (bandwidth=30, kernel=gaussian)

that the multi-modal nature of our hypothesised model does not seem to apply with this empirical data.

As per the simulation study, we repeat the Jump analysis to find the optimum number of clusters. Unsurprisingly given our results up to this point, no clear number of clusters emerges, therefore we continue with the assumption that 3 is still the optimum number:

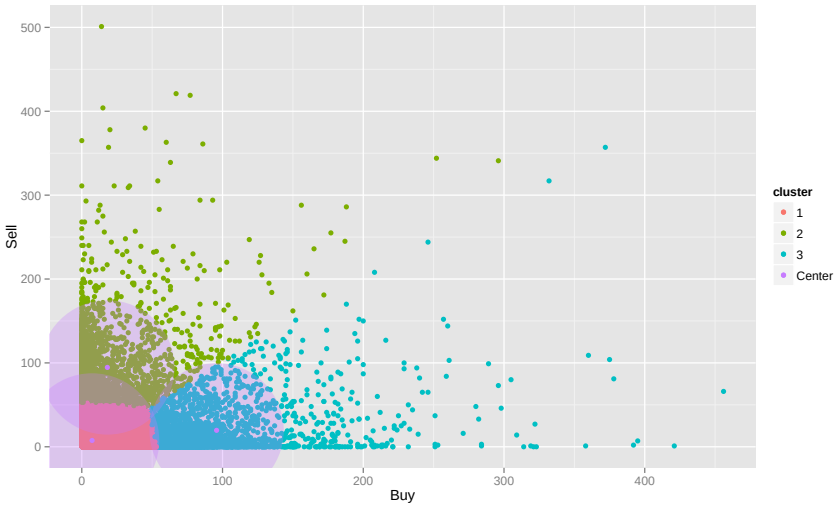


Figure 5.3: Inferred buys and sells with results of 3 fitted clusters from k-means procedure

Visualising the clusters to identify the centroids, again does not appear to be too helpful here as there are clearly no demarcated clusters in the dataset. The cluster centroids however are as follows:

	Buy	Sell	Cluster Size
1	7.31	7.52	20580.00
2	18.18	94.62	2136.00
3	95.77	19.61	2123.00

Table 5.1: Cluster centroids and sizes

Given the centroids of the three fitted clusters (no matter how uninterpretable they may be) we repeat the analysis we conducted as

part of the simulation study in order to identify inital starting values for the HMM EM estimation. Hence we can formulate the initial state parameter estimates:

$$\lambda_b = \begin{pmatrix} \epsilon_b \\ \epsilon_b + \mu_b \\ \epsilon_b \end{pmatrix} = \begin{pmatrix} 7.31 \\ 95.771 \\ 7.31 \end{pmatrix} \qquad \lambda_s = \begin{pmatrix} \epsilon_s \\ \epsilon_s \\ \epsilon_s + \mu_s \end{pmatrix} = \begin{pmatrix} 7.524 \\ 7.524 \\ 94.617 \end{pmatrix}$$

After running the EM estimation procedure for $m = 3$ we obtain the following fitted values:

	Lambda.Buy.Hat	Lambda.Sell.Hat
1	3.49	3.57
2	59.07	3.57
3	3.49	58.75

Table 5.2: Lambda Buy and Sell: Fitted arrival rates of informed and uninformed traders

On estimation of a fitted model, we are now able to calculate the decoded hidden states for our observed series of buys and sells. Plotting the states as a time series is not particularly illustrative, therefore we repeat the analysis we conducted in the simulation study and attempt to calculate volume bars from the time bars, calculate VPIN values for these volume bars and then regress these VPIN values against the decoded states.

A small sample of the generated volume bars from the empirical data is shown below:

	State	Buy	Sell	Imbalance	VPIN
1	1	0.927272727272727	1.12727272727273	0.199999999999999	0.00363636363636363
2	1	4.10909090909091	1	3.10909090909091	0.0565289256198347
3	1	12.4545454545455	0.509090909090909	11.9454545454545	0.217190082644628
4	1	14.5090909090909	0.0909090909090909	14.4181818181818	0.262148760330578
5	1	3.61818181818182	1.23636363636364	2.38181818181818	0.0433057851239669
6	1	0.763636363636364	3.43636363636364	2.67272727272727	0.048595041322314

To show the difference in the distributions between the identified states we show a boxplot of the data below Visually we can see that there appears to be a difference in the VPIN values between state 1, and states 3, 4 and 6. We can interprete that as telling us the no-news event appears to be correlated with low values of VPIN, while the private information states tend to be associated with higher VPIN values.

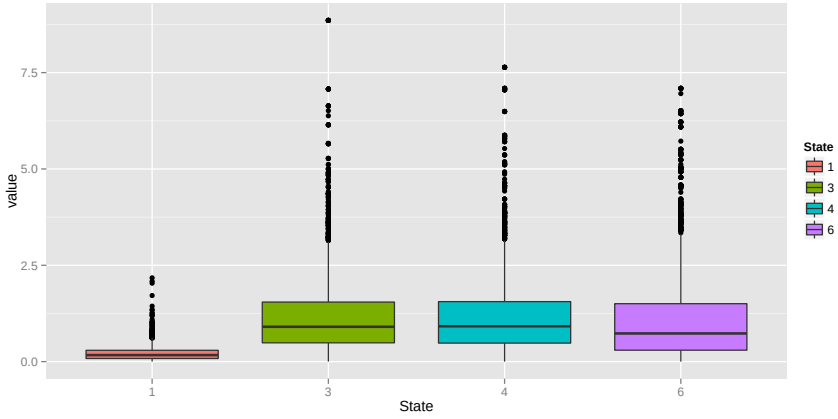


Figure 5.4: Box plots of VPIN values against decoded HMM states

To further test the strength of this relationship we conduct multiple linear regression of VPIN against State which give us the following results (we define state 1, 3, 4 and 6 as dummy variables with state 1 set as the baseline):

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	0.2228	0.0270	8.26	0.0000
State3	0.9541	0.0313	30.49	0.0000
State4	0.9721	0.0314	30.99	0.0000
State6	0.8746	0.0311	28.13	0.0000

Table 5.3: Results of multiple linear regression of VPIN State

The results of the regression indicate to us that there does indeed appear to be a very strong ($p << 0.01$) indication that VPIN values associated with the no-news state are going to be significantly lower than VPIN values from the private information states.

6
Conclusion and Further Work

7

Appendix 1: Algorithm to Compute the VPIN Metric

Data

1. Time series of transactions of a particular instrument (T_i, P_i, V_i)
   (a) T_i: Time of the trade.
   (b) P_i: Price at which securities were exchanged.
   (c) V_i: Volume exchanged
2. V: Volume size (determined by user of the formula)
3. n: Sample of volume buckets used in the estimation.

Result: Prepare Equal Volume Buckets

1. Sort transactions by time ascending: $T_{i+1} \geq T_i, \forall i$
2. Compute $\Delta P_i, \forall i$
3. Expand the number of observations by repeating each observation ΔP_i as many times as V_i. This generates a total of $I = \sum_i V_i$ observations ΔP_i.
4. Re-index ΔP_i observations, $i = 1, ..., I$
5. Initiate counter: $\tau = 0$
6. While $\tau V < I$
   (a) Add one unit to τ
   (b) $\forall i \in [(\tau - 1)V + 1, \tau V]$, split volume between buy or sell intitiated:
      i. Assign to Vb the number of observations classified as buy: $V_\tau^B = \sum_{i=t(\tau-1)+1}^{t(\tau)} V_i Z\left(\frac{S_i - S_{i-1}}{\sigma_{\Delta S}}\right)$
      ii. Assign to Vs the number of observations classified as sell: $V_\tau^S = \sum_{i=t(\tau-1)+1}^{t(\tau)} V_i \left(1 - Z\left(\frac{S_i - S_{i-1}}{\sigma_{\Delta S}}\right)\right) = V - V_\tau^B$
7. Set $L = \tau - 1$

8
Appendix 2: R Code Used in this Project

8.1 EM Estimation of Bivariate Poisson HMM Parameters

This code is adapted from that found in Zucchini & MacDonald (2009)
to support bivariate Poisson models

```
bi.pois.HMM.EM <- function(x, m_buy, m_sell, lambda_buy,
    lambda_sell, gamma, delta, maxiter = 1000,
    tol = 1e-06, ...) {
    n <- dim(x)[1]  # num of observations
    m <- m_buy * m_sell
    lambda_buy.next <- lambda_buy
    lambda_sell.next <- lambda_sell
    gamma.next <- gamma
    delta.next <- delta

    # initiliase a state index lookup table for
    # resolving (i,j) -> stateIndex
    stateEnv <- new.env()
    counter = 1
    for (i in 1:m_buy) {
        for (j in 1:m_sell) {
            stateEnv[[paste(i, j)]] = counter
            counter = counter + 1
        }
    }

    for (iter in 1:maxiter) {
        lallprobs <- log(PMatAll(x, lambda_buy,
            lambda_sell))
        fb <- bi.pois.HMM.lalphabeta(x, m, lambda_buy,
            lambda_sell, gamma, delta = delta)
        la <- fb$la
```

```
lb <- fb$lb
c <- max(la[, n])
llk <- c + log(sum(exp(la[, n] - c)))
for (j in 1:m) {
    for (k in 1:m) {
        gamma.next[j, k] <- gamma[j, k] *
          sum(exp(la[j, 1:(n - 1)] + lallprobs[2:n,
            k] + lb[k, 2:n] - llk))
    }
}
gamma.next <- gamma.next/apply(gamma.next,
    1, sum)
uhat <- function(j, t) {
    exp(la[j, t] + lb[j, t] - llk)
}
buy <- x[, 1]
sell <- x[, 2]
# calculate lambda_buy_i
for (i in 1:m_buy) {
    numerator <- 0
    denominator <- 0
    for (j in 1:m_sell) {
        ij = stateEnv[[paste(i, j)]]
        numerator <- numerator + sum(uhat(ij) *
          buy)
        denominator <- denominator + sum(uhat(ij))
    }

    lambda_buy.next[i] <- numerator/denominator
}
# calculate lambda_sell_j
for (j in 1:m_sell) {
    numerator <- 0
    denominator <- 0
    for (i in 1:m_buy) {
        ij = stateEnv[[paste(i, j)]]
        numerator <- numerator + sum(uhat(ij) *
          sell)
        denominator <- denominator + sum(uhat(ij))
    }

    lambda_sell.next[j] <- numerator/denominator
}
```

```
        delta.next <- exp(la[, 1] + lb[, 1] -
            llk)
        delta.next <- delta.next/sum(delta.next)

        crit <- sum(abs(lambda_buy - lambda_buy.next)) +
            sum(abs(lambda_sell - lambda_sell.next)) +
            sum(abs(gamma - gamma.next)) + sum(abs(delta -
            delta.next))

        print(paste("Iteration:", iter, " Crit:",
            crit, "LLK:", llk))

        if (is.na(crit)) {
            AIC <- NA
            BIC <- NA
            return(list(lambda_buy = lambda_buy,
                lambda_sell = lambda_sell, gamma = gamma,
                delta = delta, mllk = -llk, AIC = AIC,
                BIC = BIC))
        } else if (crit < tol) {
            np <- m * m + m - 1
            AIC <- -2 * (llk - np)
            BIC <- -2 * llk + np * log(n)
            return(list(lambda_buy = lambda_buy,
                lambda_sell = lambda_sell, gamma = gamma,
                delta = delta, mllk = -llk, AIC = AIC,
                BIC = BIC))
        }
        lambda_buy <- lambda_buy.next
        lambda_sell <- lambda_sell.next
        gamma <- gamma.next
        delta <- delta.next
    }
    print(paste("No convergence after", maxiter,
        "iterations"))
    NA
}
```

8.2 Calculation of Forward and Backward Probabilities

This code is adapted from that found in Zucchini & MacDonald (2009) to support bivariate Poisson models

```
bi.pois.HMM.lalphabeta <- function(x, m, lambda_buy,
    lambda_sell, gamma, delta = NULL) {
    if (is.null(delta))
        delta <- solve(t(diag(m) - gamma + 1),
            rep(1, m))
    n <- dim(x)[1]
    lalpha <- lbeta <- matrix(NA, m, n)
    allprobs <- PMatAll(x, lambda_buy, lambda_sell)
    foo <- delta * allprobs[1, ]
    sumfoo <- sum(foo)
    lscale <- log(sumfoo)
    foo <- foo/sumfoo
    lalpha[, 1] <- log(foo) + lscale
    for (i in 2:n) {
        foo <- foo %*% gamma * allprobs[i, ]
        sumfoo <- sum(foo)
        lscale <- lscale + log(sumfoo)
        foo <- foo/sumfoo
        lalpha[, i] <- log(foo) + lscale
    }
    lbeta[, n] <- rep(0, m)
    foo <- rep(1/m, m)
    lscale <- log(m)
    for (i in (n - 1):1) {
        foo <- gamma %*% (allprobs[i + 1, ] *
            foo)
        lbeta[, i] <- log(foo) + lscale
        sumfoo <- sum(foo)
        foo <- foo/sumfoo
        lscale <- lscale + log(sumfoo)
    }
    list(la = lalpha, lb = lbeta)
}
```

8.3 Monte-Carlo Simulation of Volume Buckets

```
generate.trades.sim <- function(n, alpha, delta,
    epsilon, mu) {
    Vbuy = numeric(n)  #buy volume buckets
    Vsell = numeric(n)  #sell volume buckets
    TrueStates = numeric(n)  #sell volume buckets
    j = 1
    while (j <= n) {
        u1 = runif(1)
        u2 = runif(1)
        if (u1 < alpha) {
            # we have an information event its a bad news
            # event only uninformed traders buy when
            # there's bad news
            if (u2 < delta) {
                Vbuy[j] = rpois(1, epsilon)
                # both informed and uninformed traders sell
                # when there's bad news
                Vsell[j] = rpois(1, mu + epsilon)
                TrueStates[j] = 2
            } else {
                # its a good news event both informed and
                # uninformed traders buy when there's good
                # news
                Vbuy[j] = rpois(1, mu + epsilon)
                # only uninformed traders sell when there's
                # good news
                Vsell[j] = rpois(1, epsilon)
                TrueStates[j] = 3
            }
        } else {
            # no information event uninformed traders buy
            # and sell in equal quantities
            Vbuy[j] = rpois(1, epsilon)
            Vsell[j] = Vbuy[j]
            TrueStates[j] = 1
        }
        j = j + 1
    }
    return(data.frame(Buckets = 1:n, Buy = Vbuy,
        Sell = Vsell, States = TrueStates))
}
```

8.4 Data Loading and Cleansing of NYSE TAQ Data

Following instructions found in the highfrequency package manual
http://highfrequency.herokuapp.com

```
library(zoo)
library(xts)
library(highfrequency)
library(Defaults)
library(TTR)
library(quantmod)
library(timeDate)

conversionRequired = FALSE
setwd("/Users/nick/Documents/RStudioProjects/Dissertation/code")
datasource = "/Users/nick/Documents/RStudioProjects/Dissertation/data/raw"
datadestination = "/Users/nick/Documents/RStudioProjects/Dissertation/data/xts"
datadestination_cleaned = "/Users/nick/Documents/RStudioProjects/Dissertation/data/xts_cleaned"

from = "2010-05-06"
to = "2010-05-06"

tickers = c("SPY")

tradeColNames <- c("SYMBOL", "DATE", "TIME", "PRICE",
    "SIZE", "G127", "CORR", "COND", "EX")

quoteColNames <- c("SYMBOL", "DATE", "TIME", "BID",
    "OFR", "BIDSIZ", "OFRSIZ", "MODE", "EX", "MMID")

if (conversionRequired) {
    # TODO: check the source of the G127 NA errors
    print("Beginning conversion")
    suppressWarnings(convert(from, to, datasource,
        datadestination, trades = TRUE, quotes = TRUE,
        ticker = tickers, dir = FALSE, extension = "csv",
        header = TRUE, tradecolnames = tradeColNames,
        quotecolnames = quoteColNames, format = "%Y%m%d %H:%M:%S"))
    print("Completed conversion")
}

# options('digits.secs'=3); #Shows
# milliseconds
print("Loading Trades")
tdata = TAQLoad(tickers = tickers, from = from,
```

```
    to = to, trades = T, quotes = F, datasource = datadestination)

print("Loading Quotes")
qdata = TAQLoad(tickers = tickers, from = from,
    to = to, trades = F, quotes = T, datasource = datadestination)

print("Cleaning up quotes")
qdata = quotesCleanup(qdataraw = qdata, exchanges = "T",
    report = FALSE, maxi = 25)

print("Cleaning up trades")
tdataAfterFinalCleanup = tradesCleanupFinal(qdata = qdata,
    tdata = tdata)

print("Matching trades and quotes")
tqdata = matchTradesQuotes(tdataAfterFinalCleanup,
    qdata)

print("Getting trade directions using Lee-Ready")
tradeDirection = getTradeDirection(tqdata)

core = coredata(tradeDirection)
n = length(tradeDirection)
buy = rep(0, n)
sell = rep(0, n)
buy[which(core[1:n] == 1)] = 1
sell[which(core[1:n] == -1)] = 1

print("Calculating time bars")
buys = zoo(buy, index(tqdata))
sells = zoo(sell, index(tqdata))
buys.ts = period.sum(buys, endpoints(buys, "seconds",
    1))
sells.ts = period.sum(sells, endpoints(sells,
    "seconds", 1))
buysSells.ts = merge.xts(buys.ts, sells.ts)
buysSells.ts = merge.xts(tqdata$PRICE, buysSells.ts,
    join = "inner")

plot.zoo(buysSells.ts)
saveRDS(buysSells.ts, "../data/SPY_BuysSells.RDS")
```

8.5 Technique to calculate VPIN from Time Bars

```
GenerateVolBars <- function(trades, n) {
    library(data.table)
    trades = data.frame(trades, Total = trades$Buy +
        trades$Sell)
    totalVol = sum(trades$Total)
    expanded = data.table(Timebar = integer(totalVol),
        TimeState = integer(totalVol), Volume = integer(totalVol),
        TimebarStart = integer(totalVol))
    startBlock = 1
    for (i in 1:n) {
        print(paste(i, "of", n, "trades processing"))
        repeats = trades[i, ]$Total
        state = trades[i, ]$States
        timeBucket = trades[i, ]$TimeBar
        endBlock = startBlock + repeats - 1
        expanded[startBlock:endBlock, `:=`(1,
            i)]
        expanded[startBlock:endBlock, `:=`(2,
            state)]
        expanded[startBlock:endBlock, `:=`(3,
            repeats)]
        expanded[startBlock:endBlock, `:=`(4,
            as.ITime(timeBucket))]  #just stores the integer repesentation of the seconds component of
        startBlock = endBlock + 1
    }

    noOfVolBars <- floor(totalVol/V)
    VolBars = data.frame(VolBar = rep(NA, noOfVolBars),
        State = rep(NA, noOfVolBars), Buy = rep(NA,
            noOfVolBars), Sell = rep(NA, noOfVolBars),
        Imbalance = rep(NA, noOfVolBars), VPIN = rep(NA,
            noOfVolBars), TimeStart = rep(as.POSIXct(NA,
            "", tz = "GMT"), noOfVolBars), stringsAsFactors = FALSE)
    for (i in 1:noOfVolBars) {
        print(paste(i, "of", noOfVolBars, "volbars processing"))
        end = i * V
        start = end - V + 1
        volData = expanded[start:end, ]

        mode = names(sort(-table(volData$TimeState)))[1]

        timeBars = unique(volData$Timebar)
```

```
        timeBarWeights = numeric()
        j = 1
        for (t in min(timeBars):max(timeBars)) {
            timeBarWeights[j] = sum(volData$Timebar ==
                t)/V
            j = j + 1
        }
        buy = trades[timeBars, ]$Buy %*% timeBarWeights
        sell = trades[timeBars, ]$Sell %*% timeBarWeights
        imbalance <- abs(buy - sell)

        VolBars[i, ] = c(i, mode, buy, sell, imbalance,
            imbalance/V, as.POSIXct(NA, "", tz = "GMT"))
        VolBars$TimeStart[i] = as.POSIXct(volData$TimebarStart[1],
            origin = "2010-05-06 00:00:00", tz = "GMT")
    }
    VolBars
}

# print(VolBars) plot.xts(zoo(VolBars$VPIN,
# VolBars$TimeStart)) data.df =
# data.frame(vpin = as.numeric(VolBars$VPIN),
# state=factor(VolBars$State),
# stringsAsFactors = FALSE) boxplot(vpin ~
# state, data = data.df, ylab = 'VPIN Value')
# model.lm = lm(vpin ~ state, data = data.df)
# summary(model.lm) model.aov = aov(vpin ~
# state, data = data.df) summary(model.aov)
```

8.6 Jump method to Identify Number of Clusters using K-Means

Thanks to Gareth M James for making this code available
http://www-bcf.usc.edu/~gareth/research/jump

```
jump <- function(data = NULL, K = 10, y = NULL,
    plotjumps = T, rand = 10, fits = NULL, B = 0,
    dist = NULL, trace = F) {
    if (!is.null(data)) {
        # Compute the kmeans fit to the data
        if (is.null(fits))
            fits <- kmeans.rndstart(data, K, rand)
        if (is.null(y))
            y <- dim(data)[2]/2
        n <- nrow(data)
        p <- ncol(data)
        # Compute the distortion associated with the
        # kmeans fit
        dist <- fits/(n * p)
    }
    # Call the compute.jump function to produce
    # plots and calculate maximum jump for each
    # value of Y
    jump.results <- compute.jump(dist, y, plotjumps)
    jump.results$fits <- fits
    # Implement bootstrap routine
    if (B > 0 & !is.null(data)) {
        n <- nrow(data)
        boot.results <- matrix(0, length(y), K)
        bootdist <- rep(0, K)
        for (b in 1:B) {
            if (trace)
                print(paste("Bootstrap Iteration ",
                  b))
            # Make bootstrap data
            bootdata <- data[sample(1:n, replace = T),
                ]
            # Get kmeans fit to the bootstrap data
            bootfits <- kmeans.rndstart(bootdata,
                K, rand)
            # Compute bootstrap distortion and maximum
            # jumps
            for (k in 1:K) bootdist[k] <- sum(bootfits[[k]]$within)/(n *
                p)
            bootmaxjumps <- compute.jump(bootdist,
```

```
                y, plotjumps = F, printresults = F)$maxjump
            for (j in 1:length(y)) boot.results[j,
                bootmaxjumps[j]] <- boot.results[j,
                bootmaxjumps[j]] + 1
        }
        # Calculate proportions of each number of
        # clusters chosen
        jump.results$boot.result <- round(boot.results/B,
            3)
        for (j in 1:length(y)) print(paste(jump.results$boot.result[j,
            jump.results$maxjump[j]] * 100, "% of bootstrap iterations corresponding to ",
            jump.results$maxjump[j], "clusters with Y=",
            y[j]))
    }
    jump.results
}

compute.jump <- function(dist, y, plotjumps = T,
    printresults = T) {
    K <- length(dist)
    numb.y <- length(y)
    numbclust <- rep(0, numb.y)
    transdist <- matrix(0, numb.y, K + 1)
    jumps <- matrix(0, numb.y, K)
    if (plotjumps)
        par(mfrow = c(numb.y, 3))
    for (i in 1:numb.y) {
        # Compute the transformed distortion
        transdist[i, ] <- c(0, dist^(-y[i]))
        # Compute the jumps in transformed distortion
        jumps[i, ] <- diff(transdist[i, ])
        # Compute the maximum jump
        numbclust[i] <- order(-jumps[i, ])[1]
        # Plot distortion, transformed distortion and
        # jumps
        if (plotjumps) {
            plot(1:K, dist, type = "l", xlab = "Number of Clusters",
                ylab = "Distortion", main = paste("Y = ",
                  y[i]))
            plot(0:K, transdist[i, ], type = "l",
                xlab = "Number of Clusters", ylab = "Transformed Distortion",
                main = paste("Y = ", y[i]))
            plot(1:K, jumps[i, ], type = "l",
                xlab = "Number of Clusters", ylab = "Jumps",
```

```
                main = paste("Y = ", y[i]))
            # Plot line and point to indicate maximum jump
            lines(rep(numbclust[i], 2), c(0, jumps[i,
                numbclust[i]]), lty = 3, lwd = 3)
            points(numbclust[i], jumps[i, numbclust[i]],
                col = 2, pch = 19, cex = 1.5)
        }
        # Report maximum jump
        if (printresults)
            print(paste("The maximum jump occurred at ",
                numbclust[i], "clusters with Y=",
                y[i]))
    }
    list(maxjump = numbclust, dist = dist, transdist = transdist[,
        -1], jumps = jumps)
}

kmeans.rndstart <- function(x, K, rand = 10) {
    fits <- sum((t(x) - apply(x, 2, mean))^2)
    iter.max <- 10
    # Run kmeans for 2 to K clusters
    for (k in 2:K) {
        Z = kmeans(x, k, nstart = rand)
        fits = c(fits, sum(Z$withinss))
    }
    fits
}

# testdata <-
# matrix(c(rnorm(2000),3+rnorm(2000)),byrow=T,ncol=4)
# temp <-
# jump(testdata,y=c(1.5,2,2.5),rand=10,trace=F)
```

9
Appendix 3: Proof of the Bivariate Poisson CDDL

As shown in (3.2) the CDDL of the bivariate Poisson EM problem is stated as:

$$\log\Big(Pr(\mathbf{x}^{(T)},\mathbf{c}^{(T)})\Big)=\sum_{i=1}^{m}\sum_{j=1}^{n}u_{i,j}(1)\log\,\delta_{i,j}+\sum_{i,k=1}^{m}\sum_{j,l=1}^{n}\left(\sum_{t=2}^{T}v_{i,j;k,l}(t)\right)\log\,\gamma_{i,j;k,l}(t)+\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{t=1}^{T}u_{i,j}(t)\log\,p_{i,j}(x_t)$$

The third term is the one we wish to maximise:

$$\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{t=1}^{T}u_{i,j}(t)\log\,p_{i,j}(x_t)$$

Where:

$$p_{i,j}(x_t)=e^{-\lambda_{b;i}}\frac{(\lambda_{b;i})^{b_t}}{b_t!}e^{-\lambda_{s;j}}\frac{(\lambda_{s;j})^{s_t}}{s_t!}$$

Which means we must maximise the following equation with respect to each of the $\lambda_{b;i}$ and $\lambda_{s;j}$

$$\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{t=1}^{T}u_{i,j}(t)\log\,e^{-\lambda_{b;i}}\frac{(\lambda_{b;i})^{b_t}}{b_t!}e^{-\lambda_{s;j}}\frac{(\lambda_{s;j})^{s_t}}{s_t!}$$

Firstly with respect to $\lambda_{b;i}$:

$$\frac{\partial}{\partial\lambda_{b;i}}\left(\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{t=1}^{T}u_{i,j}(t)\log\,e^{-\lambda_{b;i}}\frac{(\lambda_{b;i})^{b_t}}{b_t!}e^{-\lambda_{s;j}}\frac{(\lambda_{s;j})^{s_t}}{s_t!}\right)$$

$$=\frac{\partial}{\partial\lambda_{b;i}}\left(\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{t=1}^{T}u_{i,j}(t)\left\{-(\lambda_{b;i}+\lambda_{s;j})+b_t\log\,\lambda_{b;i}-b_t!+s_t\log\,\lambda_{s;i}-s_t!\right\}\right)$$

$$=\sum_{j=1}^{n}\sum_{t=1}^{T}u_{i,j}(t)\left\{-1+\frac{b_t}{\lambda_{b;i}}\right\}$$

Which we now solve for 0:

$$\sum_{j=1}^{n}\sum_{t=1}^{T} u_{i,j}(t) = \frac{1}{\lambda_{b;i}}\sum_{j=1}^{n}\sum_{t=1}^{T} u_{i,j}(t)b_t$$

Hence giving us an expression for $\hat{\lambda}_{b;i}$:

$$\hat{\lambda}_{b;i} = \frac{\sum_{j=1}^{n}\sum_{t=1}^{T} u_{i,j}(t)b_t}{\sum_{j=1}^{n}\sum_{t=1}^{T} u_{i,j}(t)}$$

Following the same steps for $\hat{\lambda}_{s;j}$ gives us:

$$\hat{\lambda}_{s;j} = \frac{\sum_{i=1}^{m}\sum_{t=1}^{T} u_{i,j}(t)s_t}{\sum_{i=1}^{m}\sum_{t=1}^{T} u_{i,j}(t)}$$

10
References

Andersen, T.G. & Bondarenko, O., 2014. Reflecting on the VPIN dispute. *Journal of Financial Markets*, 17(1), pp.53–64.

Andersen, T.G. & Bondarenko, O., 2012. VPIN and the flash crash. *Journal of Financial Markets*, 2013(May).

Bethel, W. et al., 2011. Federal Market Information Technology in the Post Flash Crash Era: Roles for Supercomputing. *SSRN Electronic Journal*. Available at: http://papers.ssrn.com/abstract=1939522.

Brownlees, C.T. & Gallo, G.M., 2006. Financial econometric analysis at ultra-high frequency: Data handling concerns. *Computational Statistics & Data Analysis*, 51(4), pp.2232–2245. Available at: http://www.sciencedirect.com/science/article/pii/S0167947306003458.

Durbin, R. et al., 1998. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*, Available at: http://eisc.univalle.edu.co/cursos/web/material/750068/1/6368030-Durbin-Et-Al-Biological-Sequence-Analysis-CUP-2002.pdf.

Easley, D. et al., 1996. Liquidity, information, and infrequently traded stocks. *Journal of Finance*, 51(4), pp.1405–1436. Available at: http://www.jstor.org/stable/2329399.

Easley, D., Lopez de Prado, M.M. & O'Hara, M., 2012. Bulk Classification of Trading Activity. *SSRN Electronic Journal*. Available at: http://papers.ssrn.com/abstract=1989555.

Easley, D., López De Prado, M.M. & O'Hara, M., 2012. Flow toxicity and liquidity in a high-frequency world. *Review of Financial Studies*, 25, pp.1457–1493.

Easley, D., Prado, M.D. & O'Hara, M., 2011. Flow toxicity and volatility in a high frequency world. *Johnson School Research Paper …*, p.39. Available at: http://business.nd.edu/uploadedFiles/Academic/_Centers/Study/_of/_Financial/_Regulation/pdf/_and/_documents/2011/_conf/_David/_Easley.pdf.

Jeria, D. & Sofianos, G., 2008. Passive orders and natural adverse

selection. *Street Smart*, (33).

Lee, C.M.C. & Ready, M.J., 1991. Inferring Trade Direction from Intraday Data. *The Journal of Finance*, 46(2), pp.733–746. Available at: http://dx.doi.org/10.1111/j.1540-6261.1991.tb02683.x.

SEC, S. of the C. and, 2010. Findings Regarding the Market Events of May 6, 2010. *Commission, US Securities and Exchange*, 20549(202), p.1ß104. Available at: https://www.sec.gov/news/studies/2010/marketevents-report.pdf.

Sugar, C. & Gareth, J., 2003. Finding the number of clusters in a data set : An information theoretic approach. *Journal of the American Statistical Association*, 98, pp.750–763.

Vergote, O., 2005. How to Match Trades and Quotes for NYSE Stocks ? *Discussions Paper Series - Center For Economic Studies -Kuleuven*, (March), pp.1–21.

Yin, X. & Zhao, J., 2014. A Hidden Markov Model Approach to Information-Based Trading: Theory and Applications. Available at: http://papers.ssrn.com/abstract=2412321.

Zucchini, W. & MacDonald, I.L., 2009. *Hidden Markov Models for Time Series: An Introduction Using R*, CRC Press. Available at: https://books.google.co.uk/books/about/Hidden/_Markov/_Models/_for/_Time/_Series.html?id=LDDzvCsdVs8C/&pgis=1.