University of California, Berkeley

Capstone Project

Price Impact - Report

Cyril Tamraz

Friday, May 4, 2018

Contents

List of Figures

List of Tables

1 Data & Tools

We start our report by providing a full statistical description of the data we used, and compare each relevant piece of information to the corresponding one in the data that *Rama Cont* [2] used for testing various models. We also give a basic description of the tool `LOBSTER` [1] which we used in order to generate our 5-level limit order book at order times that we have used throughout our analyses.

1.1 LOBSTER

The first important tool we have been using is `LOBSTER`, an efficient limit order book reconstructor. *"*`LOBSTER` *generates a 'message' and an 'orderbook' file for each active trading day of a selected ticker. The 'orderbook' file contains the evolution of the limit order book up to the requested number of levels. The 'message' file contains indicators for the type of event causing an update of the limit order book in the requested price range. All events are timestamped to seconds after midnight, with decimal precision of at least milliseconds and up to nanoseconds depending on the requested period."* [1].
We modified this code in order to have an appropriate limit order book to work on. Our limit order book consists of the 5-level lob at each point in time (i.e. update time); that is, we have access to the 5 best bids, as well as the 5 best asks, along with the corresponding sizes. In short, this lob looks like the below table 1.1 (where $\mathbf{AP}_i$, $\mathbf{AS}_i$, $\mathbf{BP}_i$, $\mathbf{BS}_i$ represent respectively the i-th, $i \in \{1, ..., 5\}$, ask price, ask size, bid price, bid price):

Time	$\mathbf{AP}_1$	$\mathbf{AS}_1$	$\mathbf{BP}_1$	$\mathbf{BS}_1$	...	$\mathbf{AP}_5$	$\mathbf{AS}_5$	$\mathbf{BP}_5$	$\mathbf{BS}_5$
34200.017460	2239500	100	2231800	100	...	2244000	547	2226200	100

Table 1.1: Our Limit-Order-Book

1.2 Data Organisation

We have focused on the AMAZON data (TAQ) during the week of 2012-06-21 (5 days). Each spreadsheet contains the succession of events (i.e. orders placed for buying/selling AMAZON stocks), and each event consists of the following information:

1. **Time:** seconds after midnight of the submitted order, with decimal precision of at least milliseconds and up to nanoseconds.
2. **Type:** type of submitted order, $\in \{0, ..., 7\}$ i.e. $\{a, ..., g\}$, for:
    (a) Submission of a new limit order.
    (b) Cancellation (Partial deletion of a limit order).
    (c) Deletion (Total deletion of a limit order).
    (d) Execution of a visible limit order.
    (e) Execution of a hidden limit order.
3. **Order ID:** unique order reference number of submitted order (assigned in order flow).
4. **Size:** number of shares desired to be traded in the submitted order.
5. **Price:** dollar price times 10000 of the submitted order (i.e., a stock price of \$91.14 is given by 911400). That is, the tick size is of 1 cent.
6. **Direction:** -1 for bid (buy) submitted order, $+1$ for ask (sell) submitted order.

1.3 Data Description

Here we give all the relevant information/ full description of the aforementioned data, and we show that this TAQ data looks pretty much like the data used for testing and validation in Rama Cont paper [2]. We also include our code to generate these descriptions in appendix (A).

1.3.1 Market Orders

Here we give a description of the buy/ sell market orders. We provide a plot of all these orders throughout the day in figures 1.1 and 1.2, and we provide the relevant information about them in table 1.2.

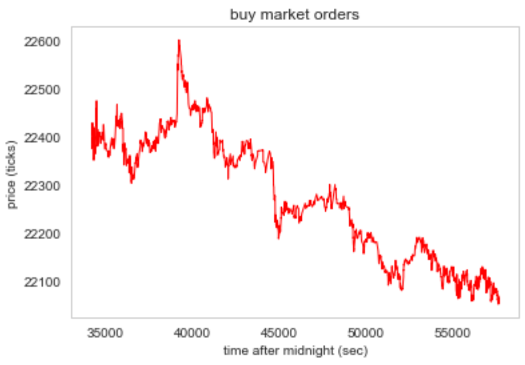


Figure 1.1: Buy Market Orders

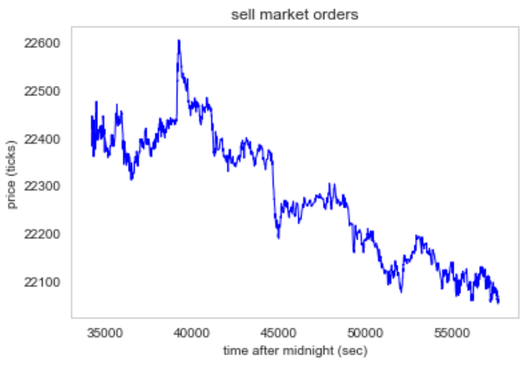


Figure 1.2: Sell Market Orders

	Buy	Sell
Average inter-time between two order	4.244 sec	3.964 sec
Average price of orders	22243.381 ticks	22266.129 ticks
Standard deviation of the price of orders	146.498 ticks	154.697 ticks

Table 1.2: Market Orders Relevant Information

We also test the inter-arrival times against exponential, normal, and uniform distributions of corresponding means and variances, and show the qq-plots in figures 1.3, 1.4, 1.5, 1.6, 1.7, 1.8.

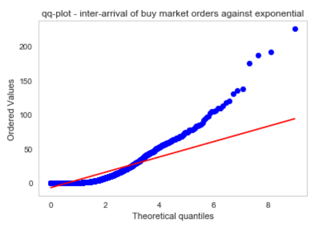


Figure 1.3: qq-plot Buy Market Inter-arrival VS Exponential

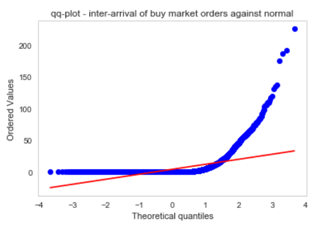


Figure 1.4: qq-plot Buy Market Inter-arrival VS Normal

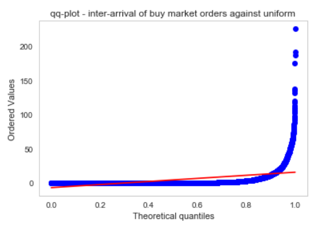


Figure 1.5: qq-plot Buy Market Inter-arrival VS Uniform

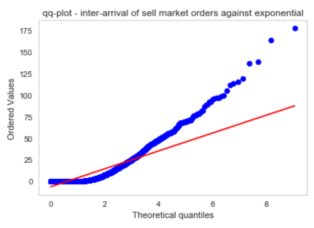


Figure 1.6: qq-plot Sell Market Inter-arrival VS Exponential

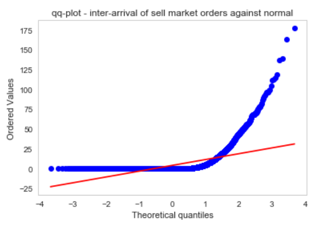


Figure 1.7: qq-plot Sell Market Inter-arrival VS Normal

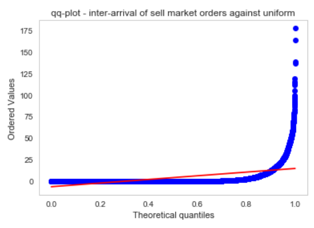


Figure 1.8: qq-plot Sell Market Inter-arrival VS Uniform

1.3.2 Limit Orders

Here we give a description of the buy/ sell limit orders. We provide a plot of all these orders throughout the day in figures 1.9 and 1.10, and we provide the relevant information about them in table 1.3.

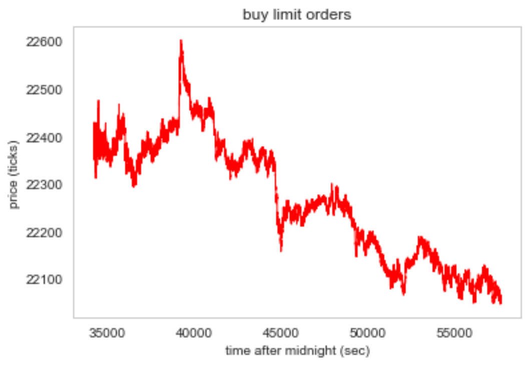


Figure 1.9: Buy Limit Orders

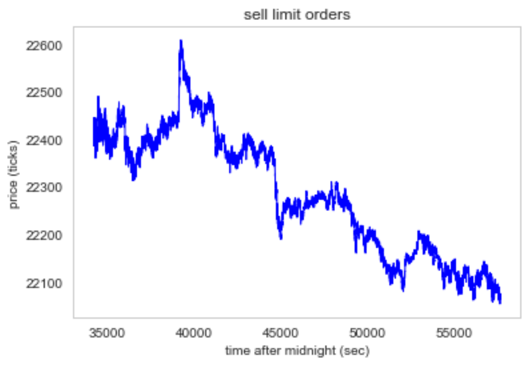


Figure 1.10: Sell Limit Orders

	Buy	**Sell**
Average inter-time between two order	0.623 sec	0.587 sec
Average price of orders	22270.801 ticks	22295.626 ticks
Standard deviation of the price of orders	132.238 ticks	139.970 ticks

Table 1.3: Limit Orders Relevant Information

We also test the inter-arrival times against exponential, normal, and uniform distributions of corresponding means and variances, and show the qq-plots in figures 1.11, 1.12, 1.13, 1.14, 1.15, 1.16.

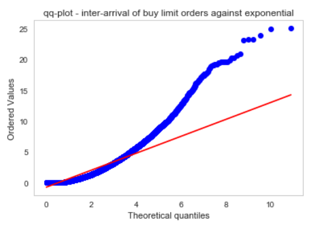


Figure 1.11: qq-plot Buy Limit Inter-arrival VS Exponential

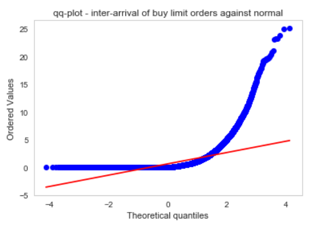


Figure 1.12: qq-plot Buy Limit Inter-arrival VS Normal

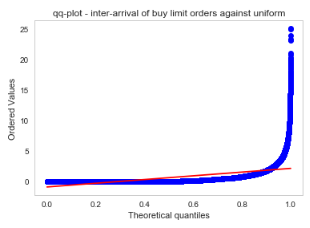


Figure 1.13: qq-plot Buy Limit Inter-arrival VS Uniform

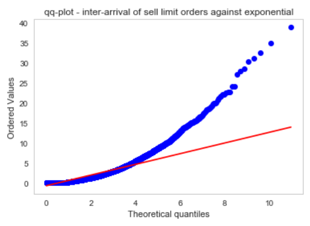


Figure 1.14: qq-plot Sell Limit Inter-arrival VS Exponential

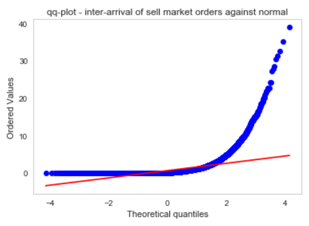


Figure 1.15: qq-plot Sell Limit Inter-arrival VS Normal

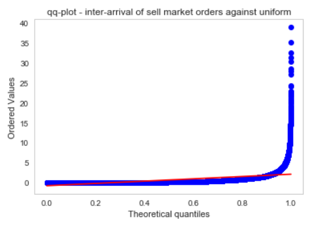


Figure 1.16: qq-plot Sell Limit Inter-arrival VS Uniform

1.3.3 Cancelled Orders

Here we give a description of the buy/ sell limit orders. We provide a plot of all these orders throughout the day in figures 1.17 and 1.18, and we provide the relevant information about them in table 1.4.

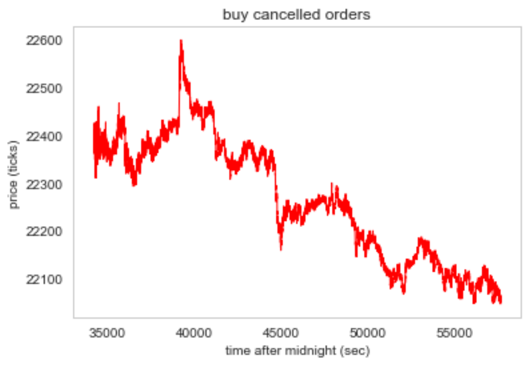


Figure 1.17: Buy Cancelled Orders

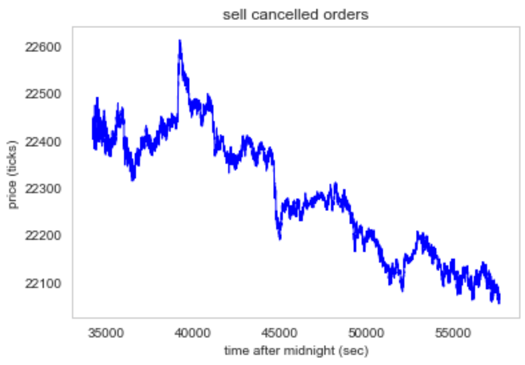


Figure 1.18: Sell Cancelled Orders

	Buy	Sell
Average inter-time between two order	0.698 sec	0.696 sec
Average price of orders	22272.232 ticks	22297.130 ticks
Standard deviation of the price of orders	131.275 ticks	138.641 ticks

Table 1.4: Cancelled Orders Relevant Information

We also test the inter-arrival times against exponential, normal, and uniform distributions of corresponding means and variances, and show the qq-plots in figures 1.19, 1.20, 1.21, 1.22, 1.23, 1.24.

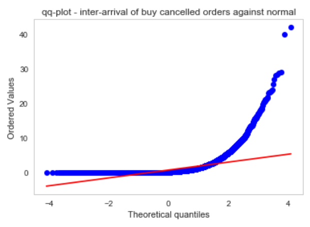


Figure 1.19: qq-plot Buy Cancelled Inter-arrival VS Exponential

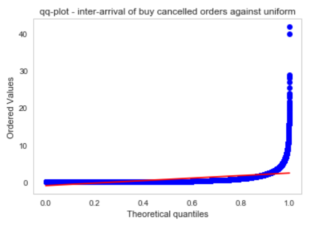


Figure 1.20: qq-plot Buy Cancelled Inter-arrival VS Normal

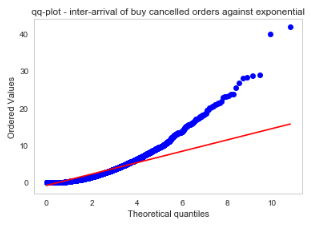


Figure 1.21: qq-plot Buy Cancelled Inter-arrival VS Uniform

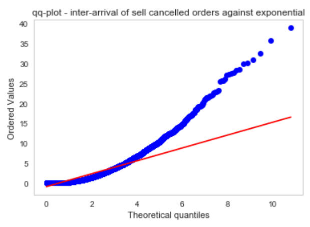


Figure 1.22: qq-plot Sell Cancelled Inter-arrival VS Exponential

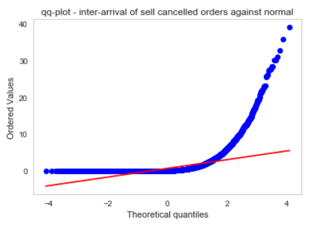


Figure 1.23: qq-plot Sell Cancelled Inter-arrival VS Normal

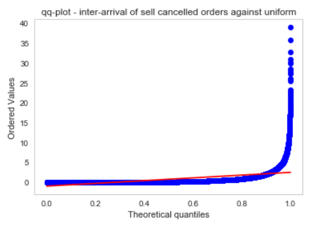


Figure 1.24: qq-plot Sell Cancelled Inter-arrival VS Uniform

1.3.4 Compare & Contrast: Market VS Limit VS Cancelled

Inter-arrivals we observe that all of the market, limit, and cancelled orders come "most likely" in a uniform fashion, as the qq-plots of their inter-arrival correspond to that of a uniform distribution for most quantiles. That is, they do not necessarily follow a Poisson distribution, as the qq-plots of their inter-arrival less correspond to that of an exponential distribution. For the case of the exponential distribution, we observed that our limit orders inter-arrival time diverge "exponentially" more for higher quantiles. We show in figure 1.25 that this result is the exact same as for the data used in Rama Cont paper [2].

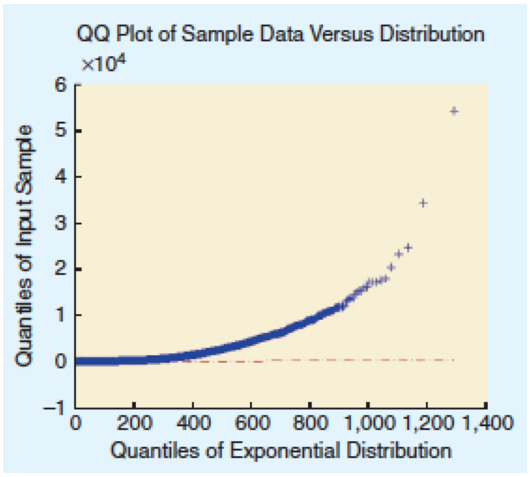


Figure 1.25: qq-plot Limit Inter-arrival VS Exponential (Rama Cont paper [2])

Mean & Standard Deviation we observe that, in all cases, buy orders have both less mean and less standard deviation in price than sell order, which is of course expected. Another interesting observation is the fact that both buy and sell cancelled orders are on average less in price than the corresponding market orders. This can be a sign that sellers are less *willing* to provide liquidity than buyers are willing to acquire shares. It would be interesting to further model these phenomena, but this is not the aim of this report.

1.3.5 Limit Order Book

Here we give a description of our limit order book. We first provide the numbers for diverse signals, and compare them with the corresponding signals for some data used in the paper, in table 1.5. We also show how the bid-ask spread varies with time in figure 1.26, and show how a "typical" 5-level limit order book looks like, i.e. the limit order book consisting in average bid/ask price i with associated average bid/ask size i, for $i \in \{1, ..., 5\}$, in figure 1.27.

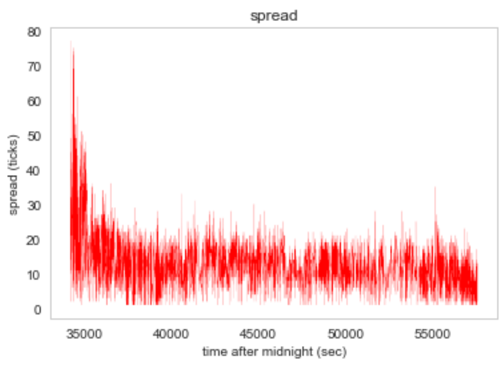


Figure 1.26: Spread Variation

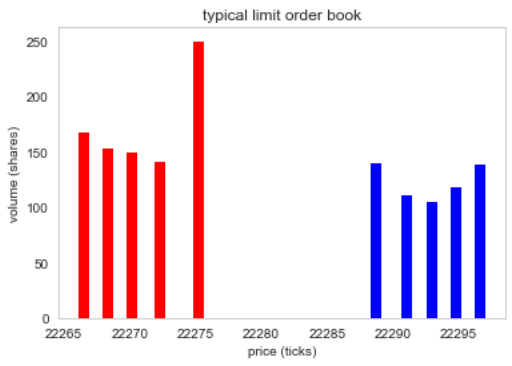


Figure 1.27: "Typical" Limit Order Book

	AMAZON	**Citigroup**
Average number of orders per sec	277	_
Average number of orders per 10 sec	2,769	4,469
Price changes in 1 day	27,557	12,499
Max change in 1 day	33.0 ticks	_
Min change in 1 day	0.0 ticks	_
Mean change in 1 day	0.263 ticks	_
Std change in 1 day	9.286 ticks	_
Price changes (between $\Delta = 1$ sec) in 1 day	10046 ticks	_
Max change (between $\Delta = 1$ sec) in 1 day	36.0 ticks	_
Min change (between $\Delta = 1$ sec) in 1 day	0.0 ticks	_
Mean change (between $\Delta = 1$ sec) in 1 day	0.919 ticks	_
Std change (between $\Delta = 1$ sec) in 1 day	20.982 ticks	_
Price changes (between $\Delta = 10$ sec) in 1 day	334 ticks	_
Max change (between $\Delta = 10$ sec) in 1 day	45.5 ticks	_
Min change (between $\Delta = 10$ sec) in 1 day	0.0 ticks	_
Mean change (between $\Delta = 10$ sec) in 1 day	3.760 ticks	_
Std change (between $\Delta = 10$ sec) in 1 day	45.000 ticks	_
Average spread	13.483 ticks	_
Standard deviation of spread	6.218 ticks	_
Average limit order book depth	146.937	_
Std limit order book depth	19.178	_

Table 1.5: Relevant Signals for Different Data

2 Introduction

This report on price impact modeling aims to model the difference in price across time periods $\Delta \in \{1, 10\}$ seconds in terms of **order events only**. That is, **we do not try to use other information** (such as price values during Δ, or the change in spread), **as these assume knowledge of some information that can be used directly to compute the difference in price**. We first explore the model introduced by *Rama Cont* [2] on the linear relation between the change in price and the order flow imbalance OFI, along with the underlying relation with the market depth. We then expand on these results by formulating models taking takes further factors into account, like for instance the total flow TF, the trade imbalance TI, the trade volume VOL, and other **event-related** information. Throughout this report, we will provide a thorough insight on all the methods we used (why do they mathematically make sense, why are they relevant in the context), as well as the high-level procedure we designed for each of them (with the associated Python code included in the appendix), and more importantly the significance of these results on the improvement/ refinement of the models. In this section, we formulate our main problem of interest - price impact modeling - while showing its relevance and dependence upon basic factors. We then we provide an outline on what follows.

2.1 An Intuition on Price Impact

We define and give some intuition on our main quantity of interest: OFI (as we will later see, most of our other features statistically reduce in some way to this quantity).

Definition 2.1. *Order Flow Imbalance (OFI)*
The OFI is a ***measure of excess demand over supply****. That is, the OFI measures, for a given time period Δ, the relative increase of orders in the buy-side of the market to the orders in the sell-side of the market. It can be expressed as:*

$$OFI(t,t{+}\Delta) = (L^b(t,t{+}\Delta){+}C^b(t,t{+}\Delta){+}M^s(t,t{+}\Delta)){-}(L^s(t,t{+}\Delta){+}C^s(t,t{+}\Delta){+}M^b(t,t{+}\Delta)) \tag{2.1}$$

where:

$$\begin{cases} L^b(t,t+\Delta) & \textit{number of limit orders placed on the buy-side of the market, during } t, t+\Delta \\ C^b(t,t+\Delta) & \textit{number of orders cancelled from the buy-side of the market, during } t, t+\Delta \\ M^s(t,t+\Delta) & \textit{number of market orders sold to the buy-side of the market, during } t, t+\Delta \\ L^s(t,t+\Delta) & \textit{number of limit orders placed on the sell-side of the market, during } t, t+\Delta \\ C^s(t,t+\Delta) & \textit{number of orders cancelled from the sell-side of the market, during } t, t+\Delta \\ M^b(t,t+\Delta) & \textit{number of market orders bought from the sell-side of the market, during } t, t+\Delta \end{cases}$$

From the above, we observe that the increase in demand and supply during $t, t+\Delta$ may respectively be represented by:

$$ID(t,t+\Delta) = L^b(t,t+\Delta) + C^b(t,t+\Delta) + M^s(t,t+\Delta) \tag{2.2}$$

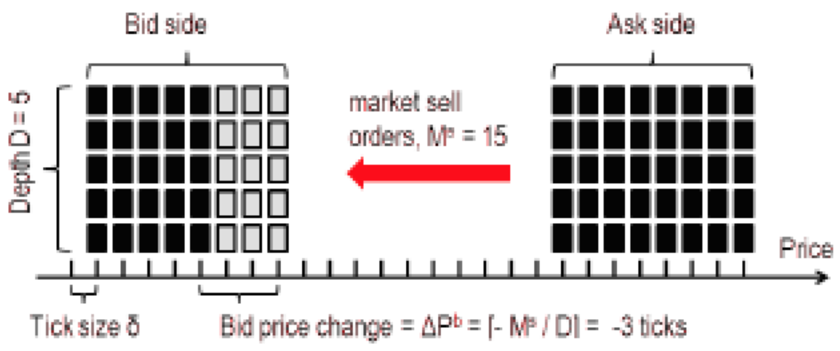
$$IS(t,t+\Delta) = L^s(t,t+\Delta) + C^s(t,t+\Delta) + M^b(t,t+\Delta) \tag{2.3}$$

so that definition 2.1 becomes:

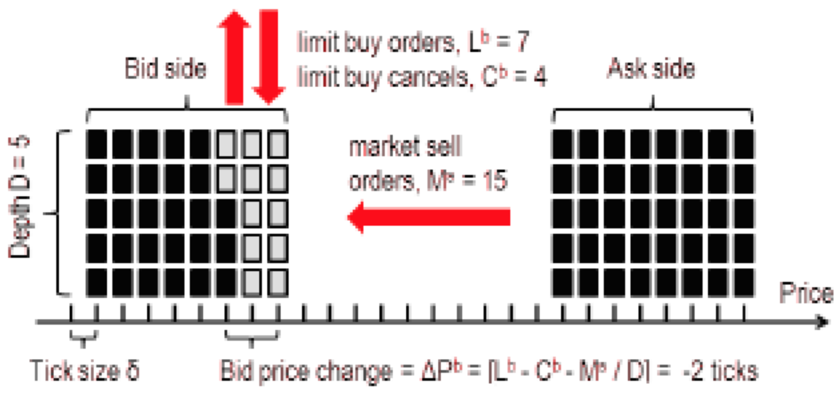
$$OFI(t,t+\Delta) = ID(t,t+\Delta) - IS(t,t+\Delta) \tag{2.4}$$

which clearly shows that the OFI is a measure of excess demand over supply. We thus *expect* that the price movements would be correlated with that of the OFI. For an intuition, consider the below scenario (figures taken from [2]):

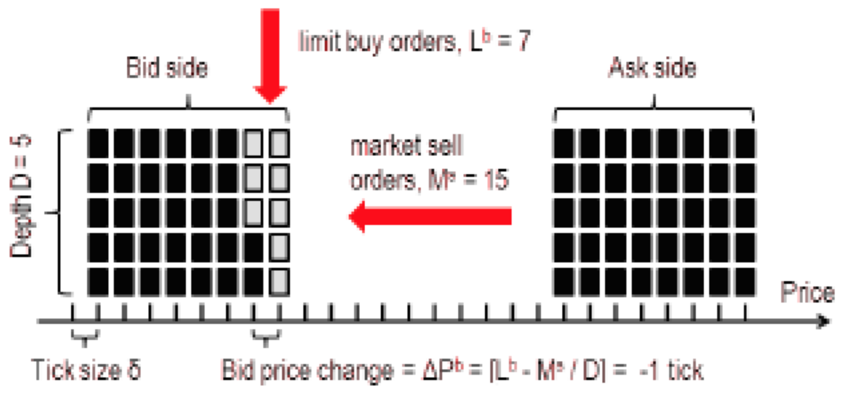
- Selling 15 orders to the bid-side participants reduces the demand by a similar number. Below, the depth is assumed to be constant = 5, so that $\frac{15}{5} = 3$ levels are removed from the demand. As a consequence, the price diminishes by $\frac{3}{2} = 1.5$ ticks.



- Ordering 7 orders from the bid-side participants, while cancelling 4 orders from the bid-side (other) participants, result in $7 - 4 = 3$ orders added to the demand, which roughly accounts for $\frac{3}{5} = 0.6$ levels removed. Since a full-level was **not** removed, the price remains unchanged.


limit buy orders, $L^b = 7$
limit buy cancels, $C^b = 4$
Bid side
Ask side
Depth D = 5
market sell
orders, $M^s = 15$
Price
Tick size δ
Bid price change = ΔP^b = [L^b - C^b - M^s / D] = -2 ticks

- Ordering 4 new orders from the bid-side participants results in 4 new orders added to the demand. With the previous number added, this account now for a level, so that the price has now increased by $\frac{1}{2} = 0.5$ ticks.


limit buy orders, $L^b = 7$
Bid side
Ask side
Depth D = 5
market sell
orders, $M^s = 15$
Price
Tick size δ
Bid price change = ΔP^b = [L^b - M^s / D] = -1 tick

2.2 Problem Statement

As was mentioned, we are interested in this report to model the price change across time periods $\Delta \in \{1, 10\}$ seconds in terms of **order events only**. Thus, we formulate our problem as follows:

Problem 2.2. *if $x_1, x_2, ..., x_l$ are l order-dependent factors, and $\Delta S(t, t + \Delta)$ is the price difference between $t, t + \Delta$, we are interested in modeling the behavior of this price difference in terms of the different factors. That is:*

$$\Delta S(t, t + \Delta) = f(x_1, x_2, ..., x_n) \tag{2.5}$$

where f is some function to estimate.

We will for additional reference create models taking factors that are not only order-dependent, such as the previous trade sign (was an increase or decrease), the initial spread, or other price-related factors.

2.3 Report outline

In section 3, we start with the basic linear model for price impact, where we replicate the results of Rama Cont [2], while coming up with additional tests/ estimation methods of our own in order to confirm/ show the relevance of the provided models/ results. In order to do so, we perform some linear and polynomial regressions using OLS, and show that the underlying relation between price change and OFI is indeed most likely linear. We also show that the assumption of the noise on the linear model is somehow validated, by testing a residual vector against a Gaussian distribution. We also show the underlying relation between the previous regression coefficient and the average market depth at regression time.

In section 4, we extend the previous model by taking into account **all** order book events. We will show that the OFI is the most important feature, by using a regression decision tree and showing how the splits are being done and what are the corresponding implications.

In section 5, we consider a generalized model that takes not only order book events, but also historic price related information. We will use universal approximation techniques in order to come up with the most appropriate model. For example, we will use polynomial regression in order to obtain a representative polynomial of the price impact, random forest in order to have a measure of the factors' importance, and neural nets in order to have a convoluted model of the price impact (and a highest predictive accuracy).

3 Linear Price Impact - OFI Model

In this section, we focus on the first model introduced in [2], namely:

$$\frac{1}{\delta}\Delta S(t, t+\Delta) = \beta_i \text{OFI}(t, t+\Delta) + \alpha_i \tag{3.1}$$

where $\delta = 100$ is the tick size (to have the prices in cents), and:

$$\begin{cases} \beta_i & \text{denotes the slope, constant throughout } i \\ \alpha_i & \text{denotes the } y\text{-intercept, constant throughout } i \end{cases}$$

The relevance of Δ and i is as follows:

- The price differences and OFI are taken over intervals of $\Delta \in \{1, 10\}$ seconds. Therefore, we may *estimate* the regression coefficients above by using pairs of samples across Δ.
- As Rama Cont conveys, the regression coefficient β_i is constant during 30 minutes, and is inversely proportional to the average market depth during the same period. Thus, for each period i of 30 minutes throughout the day, we should have a different estimate of β_i, and a different average depth AV_i corresponding to it, both of which are inversely related by a constant factor of c. That is, the *real model* coefficient is c (unchanged throughout the day), so that model 3.1 can more accurately be formulated as:

$$\frac{1}{\delta}\Delta S_k(t, t+\Delta) = c\text{OFI}_k + \epsilon_k \tag{3.2}$$

This means that, for every period of 30 min, we can samples Δ seconds apart, and use these to estimate the regression coefficients β_i and α_i. Thus, we get a sample of regression coefficients, along with the average market depth, for different periods of 30 min, and we use these to get our constant c. We provide below a detailed description of the methodology we used in order to acquire the relevant data and perform the mentioned estimations.

3.1 General Methodology to Estimate Coefficients

Note that the full code for this is included in appendix (A). We will here provide a high-level description of our procedure. First, we subsample our data into m consecutive samples of 30 min, call them $S_1, ..., S_m$. For each sample $S_i, 1 \le i \le m$, we consider a time difference of $\Delta = 1$ (or 10) sec, and we subsample S_i as the n_i samples $S_{i,0}, S_{i,\Delta}, ..., S_{i,n_i\Delta}$ that correspond to all Δ-seconds time subsamples of the 30 minutes sample S_i. For each subsample $S_{i,t\Delta}$ (with $t = 0, ..., n_i$), we do the following:

1. we compute $y_t = \frac{1}{\delta}\Delta S(t, t+\Delta)$ using our LOBSTER-generated limit-order-book, where we compute $S(t)$ and $S(t+\Delta)$ by taking the mid-quote price at the respective times (like in the paper).
2. we compute $x_t = OFI(t, t+\Delta)$, using the three structures of the limit orders, cancelled orders, and market orders, in order to count the parameters $L^b(t, t+\Delta), L^s(t, t+\Delta), M^b(t, t+\Delta), M^s(t, t+\Delta), C^b(t, t+\Delta), C^s(t, t+\Delta)$.

Thus, for every S_i (with $i = 0, ..., m$), we get a sample of pairs $(\{x_t, y_t\}_{t=1}^{n_i})_i$, along with the average depth AV_i. We then, for each $i = 1, ..., m$, get our estimated $\{\alpha_i, \beta_i\}_{i=1}^m$ (using 4-cross fold validation, so as to report the validation accuracy on unseen data). We finally estimate c using the pairs $\{\text{AV}_i, \beta_i\}_{i=1}^m$. A pseudo-code of this methodology is included below:

Algorithm 1 OFI Model Methodology

```
**Sample** our data into $m$ samples of 30 minutes: $S_1, ..., S_m$
**Initialize** our samples of regression coefficients $\beta = []$
**Initialize** our samples of average market depths $\text{AD} = []$

**for** $i \in 1, ..., m$:
    **Subsample** $S_i$ into the pairs $\{x_t, y_t\}_{t=0}^{n_i}$
    **Estimate** $\alpha_i, \beta_i$ using the pairs $\{x_t, y_t\}_{t=0}^{n_i}$ **using 4-cross fold validation.**
    **Compute** $\text{AD}_i$
    **Append** $\beta_i, \text{AD}_i$ in $\beta, \text{AD}$ respectively

**Measures** we get measures of $\text{RMSE}_{avg}, R^2_{avg}$ over the $m$ estimations above

**Estimate** $c$ using the pairs $\{\text{AD}_i, \beta_i\}_{i=1}^{m}$ **using 4-cross fold validation.**
```

3.2 Estimating the β_i's

We start by estimating the regression parameters of 3.1 for each sample $S_i, i \in \{1, ..., m\}$ individually. In order to do so, we reformulate 3.1 as:

$$\frac{1}{\delta}\Delta S(t, t+\Delta) = \beta_i \text{OFI}(t, t+\Delta) + \alpha_i + \epsilon(t, t+\Delta) \tag{3.3}$$

where $\epsilon(t, t+\Delta)$ is some white-noise term. Under a further assumption that $\epsilon(t, t+\Delta)$ is, at each point, Gaussian $\sim N(0, \sigma^2)$, we solve a Maximum Likelihood Estimator (MLE) problem on the pair $\{x_t, y_t\}_{t=0}^{n_i}$ to estimate β_i, which is equivalent to minimizing the OLS loss function:

$$\textbf{OLS} : \min_{\beta_i, \alpha_i} \sum_{t=0}^{n_i} (y_t - \beta_i x_t - \alpha_i)^2 \tag{3.4}$$

3.2.1 Estimation Results

We provide in table 3.1 the results of our estimations for different cases.

		$\Delta = 1$ second	$\Delta = 10$ seconds
Average regression coefficient	$\hat{\beta}_i^{avg}$	0.00426	0.00561
Min regression coefficient	$\hat{\beta}_i^{min}$	0.00315	0.00439
Max regression coefficient	$\hat{\beta}_i^{max}$	0.00512	0.00702
Average depth-adjusted regression coefficient	$\hat{c}_i^{avg}$	0.311	0.410
Min depth-adjusted regression coefficient	$\hat{c}_i^{min}$	0.229	0.320
Max depth-adjusted regression coefficient	$\hat{c}_i^{max}$	0.371	0.512
Average validation RMSE	RMSE_i^{avg}	0.585	3.668
Min validation RMSE	RMSE_i^{min}	0.357	1.538
Max validation RMSE	RMSE_i^{max}	1.143	5.292
Average validation R^2	$(R^2)_i^{avg}$	0.515	0.545
Min validation R^2	$(R^2)_i^{min}$	0.278	−0.141
Max validation R^2	$(R^2)_i^{max}$	0.625	0.799
Average training RMSE	RMSE_i^{avg}	0.560	3.070
Min training RMSE	RMSE_i^{min}	0.346	1.140
Max training RMSE	RMSE_i^{max}	1.101	4.840
Average training R^2	$(R^2)_i^{avg}$	0.578	0.767
Min training R^2	$(R^2)_i^{min}$	0.378	0.570
Max training R^2	$(R^2)_i^{max}$	0.680	0.855

Table 3.1: OLS Results on OFI Model

We observe the following:

1. For $\Delta = 1$ second, the average R^2 on unseen data (i.e. data used to validate our model) is of 0.515, which roughly means that the model fails to explain less than 50% of price variations on average.

2. For $\Delta = 10$ seconds, we get a similar R^2, but have a wider range $[-0.131, 0.799]$. The reason for that is the fact that less samples $\{x_t, y_t\}_{t=0}^{n_i}$ get produced for each $i = 1, ..., m$, so that validation on unseen data differs relatively more than the one for $\Delta = 1$ second.

3. For $\Delta = 1$ second, the depth-adjusted regression coefficient ranges in $[0.229, 0.371]$ with an average of 0.311. We can see that these deviations are not too large, and that the results are acceptable, as Rama Cont [2] conveys that this value should be in $[0.1, 1]$.

4. For $\Delta = 10$ seconds, the depth-adjusted regression coefficient ranges in $[0.320, 0.512]$ with an average of 0.410, which is a slight bigger deviation than for $\Delta = 1$ second. But again, these results are acceptable.

We also provide in figures 3.17 and 3.2 and idea of how does the OFI model look like for $\Delta = 1$ second and $\Delta = 10$ seconds respectively.

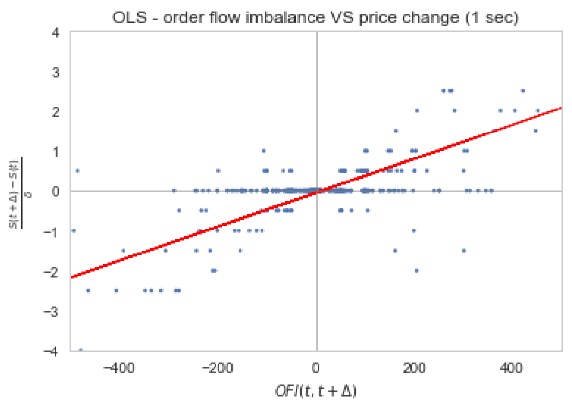


Figure 3.1: OFI Model - $\Delta = 1$ second

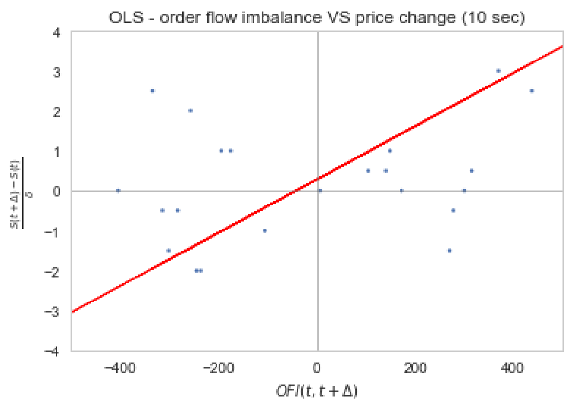


Figure 3.2: OFI Model - $\Delta = 10$ seconds

3.2.2 Testing Residuals

Recall that the main assumption to perform OLS on the basis of an MLE problem is that $\epsilon(t, t + \Delta)$ is Gaussian $\sim N(0, \sigma^2)$. We provide in figures 3.3 and 3.4 a qq-plot of the residual vector $\epsilon(t, t + \Delta) = \hat{y}_t - y_t$ in figures for $\Delta = 1$ second and $\Delta = 10$ seconds respectively.

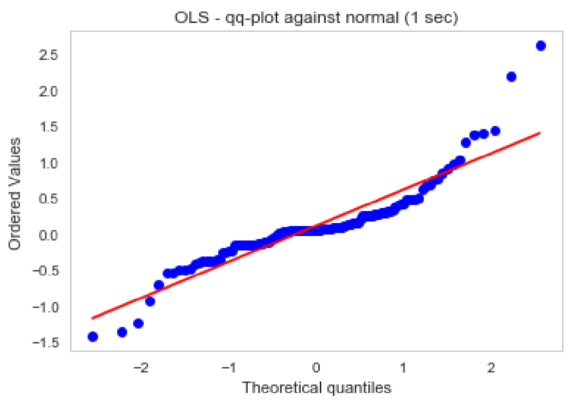


Figure 3.3: OFI Model - Residuals VS Normal - $\Delta = 1$ sec

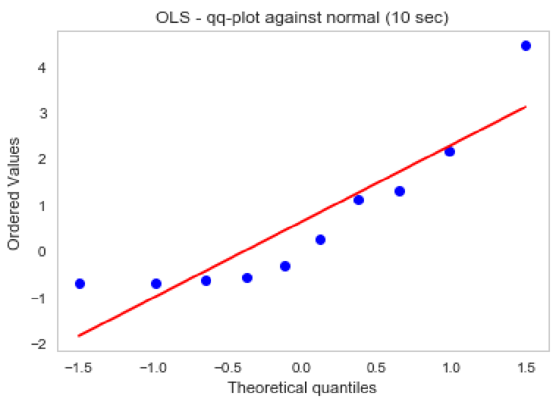


Figure 3.4: OFI Model - Residuals VS Normal - $\Delta = 10$ sec

3.2.3 Validating the Linearity of the Relation ΔS – OFI

We validate Rama Cont's [2] hypothesis that the ΔS – OFI relation is mostly linear, in the sense that adding quadratic terms or higher order terms does not explain significantly more variations of the price change. In order to do so, we augment our feature set to monomials of x_t, so that our *augmented* pairs now consist of $\{(x_t, x_t^2, ..., x_t^d), y_t\}_{t=0}^{n_i}$, where d is a hyperparameter. Our aim is to transform model 3.3 to the below:

$$\frac{1}{\delta}\Delta S(t, t+\Delta) = \sum_{j=1}^{d} \beta_{i,d}\text{OFI}(t, t+\Delta)^d + \alpha_i + \epsilon(t, t+\Delta) \tag{3.5}$$

and to discover which model with associated d is the most explanatory of our data. Thus, we repeat our previous estimations (by using $\{(x_t, x_t^2, ..., x_t^d), y_t\}_{t=0}^{n_i}$ now instead of just $\{x_t, y_t\}_{t=0}^{n_i}$) for multiple values of d. We plot as an example what would a degree 4 and 9 polynomial model would look like in figures 3.5, 3.7 for $\Delta = 1$ second, and in figures 3.6, 3.8 for $\Delta = 10$ seconds.

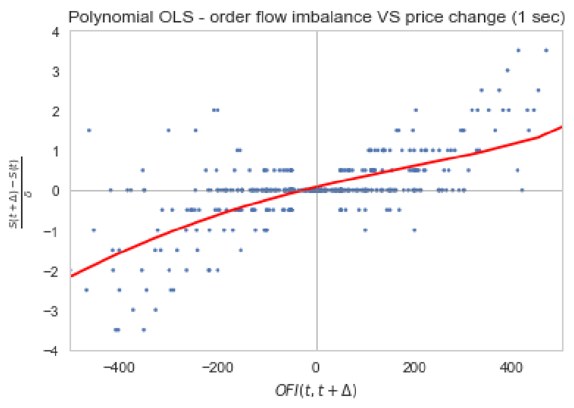


Figure 3.5: Poly OFI Model - $d = 4$ - $\Delta = 1$ sec

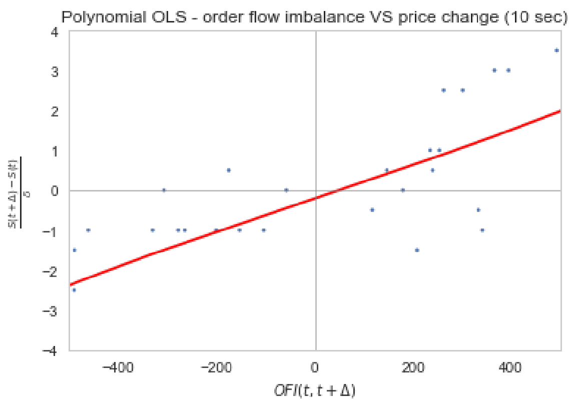


Figure 3.6: Poly OFI Model - $d = 4$ - $\Delta = 10$ sec

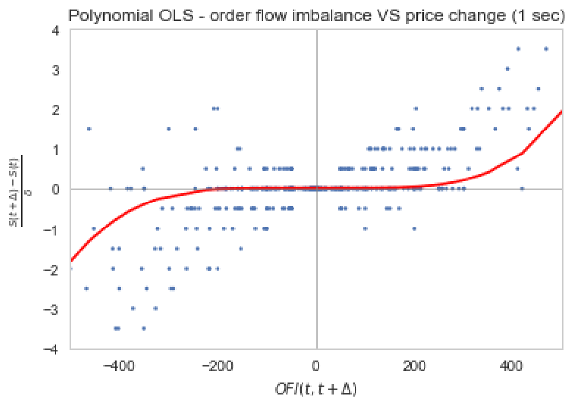


Figure 3.7: Poly OFI Model - $d = 9$ - $\Delta = 1$ sec

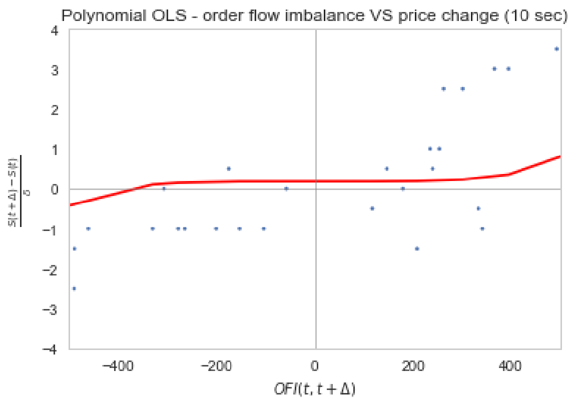


Figure 3.8: Poly OFI Model - $d = 9$ - $\Delta = 10$ sec

We now do some hyperparameter tuning on d, where we solve multiple OLS on our polynomial features in order to determine the best d. We plot both the training and validation variation of RMSE and R^2 in terms of d in figures 3.9, 3.11, 3.13, 3.15 for $\Delta = 1$ second, and in figures 3.10, 3.12, 3.14, 3.16 for $\Delta = 10$ seconds.

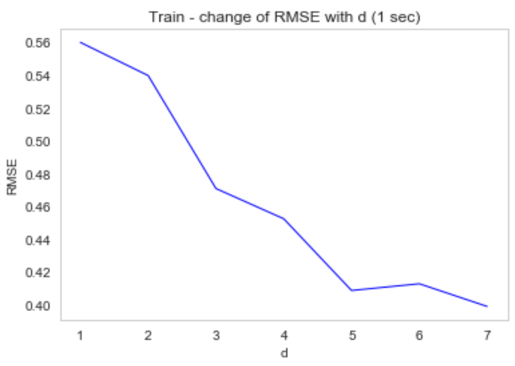


Figure 3.9: Training - RMSE VS d - $\Delta = 1$ sec

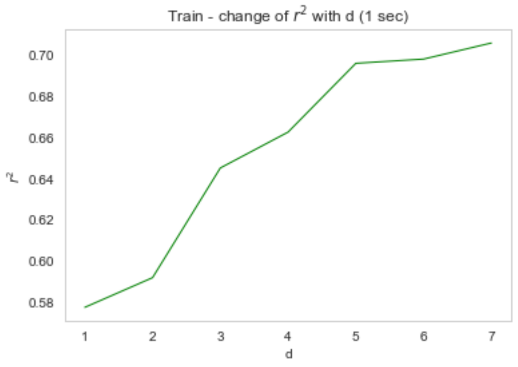


Figure 3.10: Training - R^2 VS d - $\Delta = 1$ sec

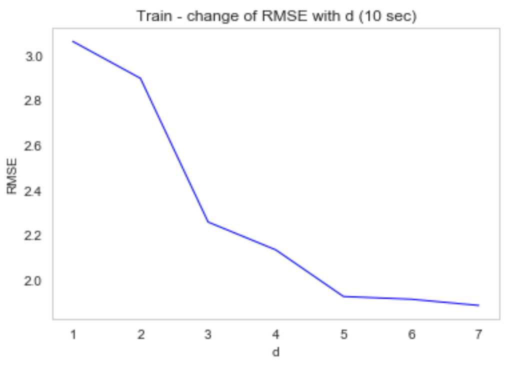


Figure 3.11: Training - RMSE VS d - $\Delta = 10$ sec

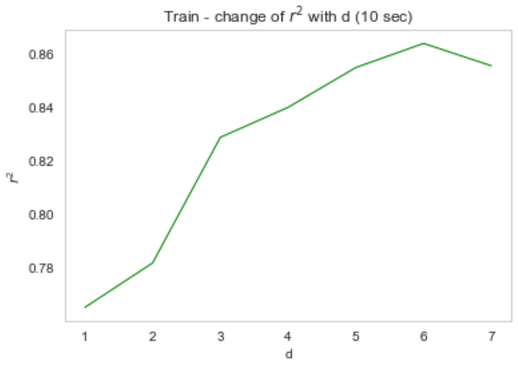


Figure 3.12: Training - R^2 VS d - $\Delta = 10$ sec

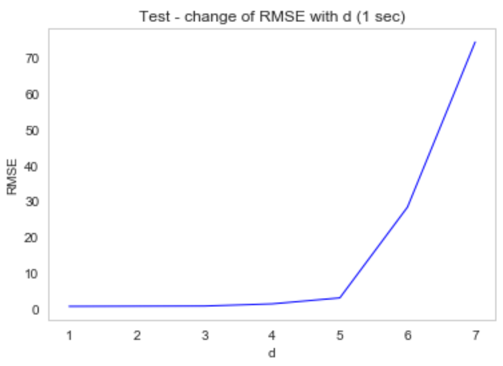


Figure 3.13: Validating - RMSE VS d - $\Delta = 1$ sec

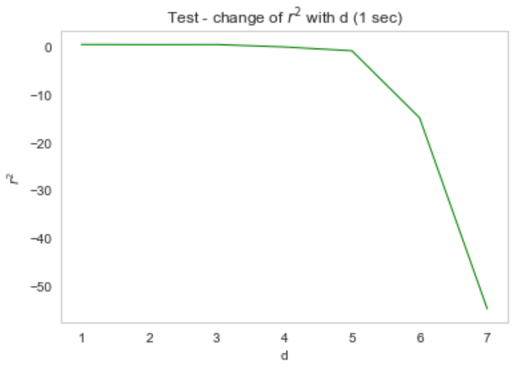


Figure 3.14: Validating - R^2 VS d - $\Delta = 1$ sec

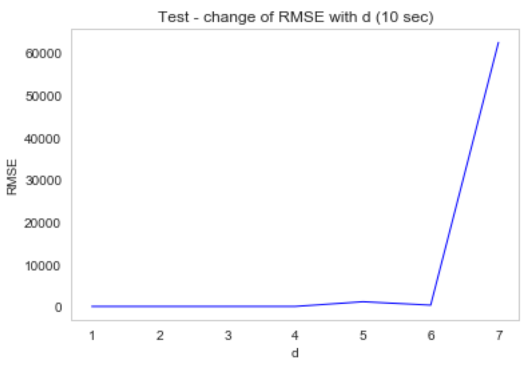


Figure 3.15: Validating - RMSE VS d - $\Delta = 10$ sec

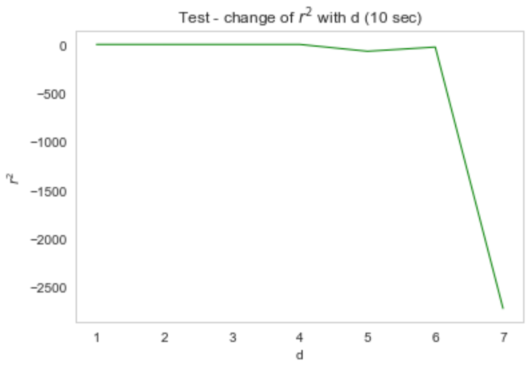


Figure 3.16: Validating - R^2 VS d - $\Delta = 10$ sec

We observe (as expected) that increasing our degree does improve our model **on training examples**. This is expected, as a higher d permits us to *hit* more points in our training set; thus, training data only permits to measure bias. From our validation plots, we observe that, in both cases of $\Delta = 1$ second and $\Delta = 10$ seconds, a degree of 5 is *thresholding* in the sense that all degrees above it start overfitting noise. Moreover, there is no improvement from $d = 1$ to $d = 5$, which means that true underlying model is somewhere in the range of $d \in \{1, ..., 5\}$, with $d = 1$ being an optimal choice as it reduces the model complexity **without** underperforming relative to the other valid degrees. This validates the hypothesis of linearity.

3.2.4 Comparison with Rama Cont's [2] Results

As a further note, we show how our results compare with Rama Cont's [2] results; although both plots are on different data, this at least gives an idea of why do our results make sense. We provide a plot comparison in figures 3.17 and 3.18.

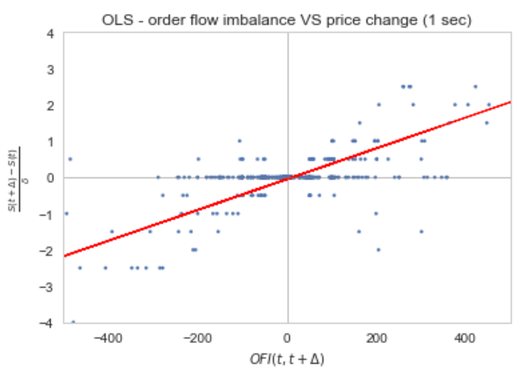


Figure 3.17: Our OFI Model - $\Delta = 1$ second

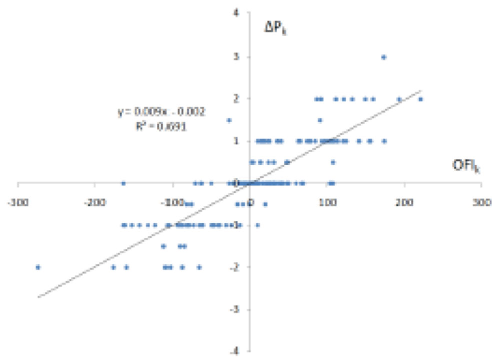


Figure 3.18: Rama Cont's [2] OFI Model - $\Delta = 1$ second

and a table showing how the regression parameters and results compare in table 3.2.

		Our ($\Delta = 1$ second)	Rama Cont's [2] ($\Delta = 1$ second)
Average regression coefficient	$\hat{\beta}_i^{avg}$	0.00426	0.002
Average training R^2	$(R^2)_i^{avg}$	0.578	0.691

Table 3.2: Our Results VS Rama Cont's [2] Results

3.3 Estimating c

We use our previous estimates of β_i, along with the average depths AD_i, in order to have an idea of how does these two relate. We start by providing an idea of how do both evolve throughout the day in figure 3.19.

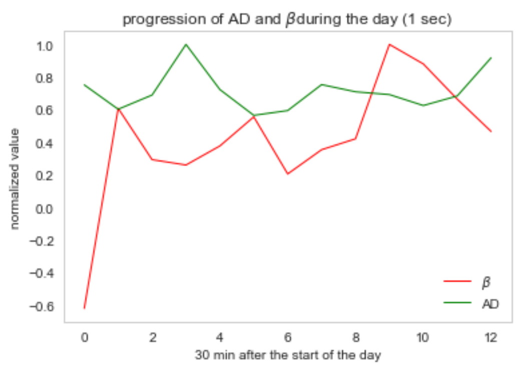


Figure 3.19: β_i's and AD_i's Throughout the Day

We observe that there is a clear inverse relation between both, and our aim here is to model it. We start by formulating the associated model provided by Rama Cont's [2] below:

$$\beta_i = \frac{c}{\mathrm{AD}_i^{\lambda}} + \nu_i \tag{3.6}$$

where ν_i is a white noise term, and c, λ are the parameters to estimate. This model indeed bases itself on the above intuition that the β_i's and the AD_i's are inversely related by a factor of c, but adds a further assumption that this relation is not inversely linear, as it adds a power of λ to AD_i. Although, the aim is to show that this power is close to 1, so as to have an inverse linear relationship. In order to estimate λ, we use our samples $\{\beta_i, AD_i\}_{i=1}^{m}$ in order to perform a log-OLS to estimate λ on the below equation:

$$\log \beta_i = \alpha_{M,i} - \lambda \log AD_i + \epsilon_{L,i} \tag{3.7}$$

3.3.1 Estimation Results & Comparison with Rama Cont's [2] Results

Note that our results are obtained from 24 hours of data, Rama Cont's [2] results are obtained from 173 hours of data. Thus, our data has *less points*, and so our estimations are *less complete*. Although, this at least gives an idea of why do our results make sense. We provide a plot comparison in figures 3.20 and 3.21.

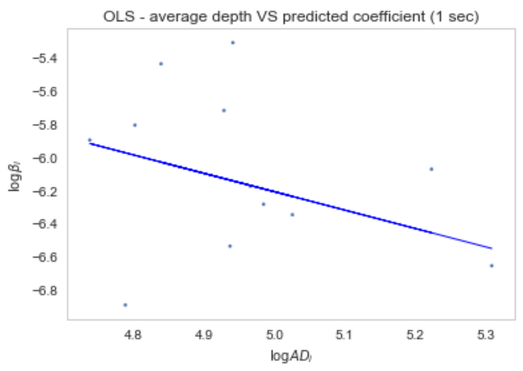


Figure 3.20: Our λ Model - $\Delta = 1$ second

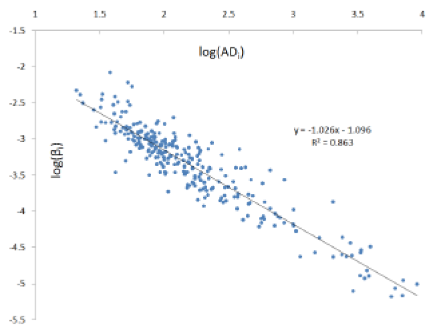


Figure 3.21: Rama Cont's [2] λ Model - $\Delta = 1$ second

and a table showing how the regression parameters and results compare in table 3.3.

		Our ($\Delta = 1$ second)	Rama Cont's [2] ($\Delta = 1$ second)
Average regression coefficient	$\hat{\lambda}^{avg}$	1.137	1.026
Average training R^2	$(R^2)^{avg}$	0.191	0.863

Table 3.3: Our Results VS Rama Cont's [2] Results

We observe that our estimated λ is, as expected, close to 1, which validates the fact that the underlying β_i VS AD_i model is *most likely* inverse-linear.

3.3.2 Using Ridge Regression to Refine our Range of c

By the fact that our data does not contain many points to have a good estimate of c, we decided to perform a ridge regression with an *a-priori* on what we believe is the *true* parameter $c \approx 0.311$ (value we got from our previous average depth-adjusted regression coefficient $\hat{c}_i^{avg} = \hat{\beta}_i^{avg} \times AD_i$). That is, we solve the below problem:

$$\textbf{RIDGE} : \min_c \sum_{t=1}^{n_i} (y_t - cx_t)^2 + \frac{1}{\sigma^2}(c - 0.311)^2 \tag{3.8}$$

where $\lambda = \frac{1}{\sigma^2}$ is our regularizer hyperparameter. Thus, solving the RIDGE problem 3.8 will provide us with the optimal λ, or in other terms the variance at which c falls away from its mean of $c_\mu \approx 0.311$. We plot our obtained results in figure 3.22.

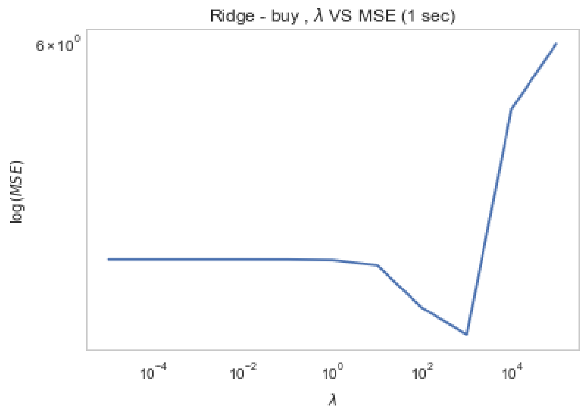


Figure 3.22: RIDGE - RMSE VS λ - $\Delta = 1$ sec

We observe that $\lambda^* = 1000$, so that $\sigma^* = \frac{1}{\sqrt{1000}} = 0.032$. This permits us to refine our c-range, as shown in table 3.4.

		Adjusted ($\Delta = 1$ second)	RIDGE-Refined ($\Delta = 1$ second)
Average depth-adjusted regression coefficient	$\hat{c}_i^{avg}$	0.311	0.311
Min depth-adjusted regression coefficient	$\hat{c}_i^{min}$	0.229	0.279
Max depth-adjusted regression coefficient	$\hat{c}_i^{max}$	0.371	0.343

Table 3.4: OLS Results on OFI Model

This estimate shows that c is in fact more *constant* than we previously thought (by just adjusting it by the depth), since ridge regression indicated a low variance of c away from its mean of $c_\mu \approx 0.311$.

3.4 Takeaways from OFI Model

To sum up the OFI model, we've seen the following:

1. The linear OFI model is *informative* enough, as shown in the scores display.
2. A higher-order model **does not** inform more on the price impact due to OFI only.
3. The regression coefficient β_i is AD$_i$ dependent, with a relation that is most inverse-linear.
4. The factor c of this relation should *always* be constant; in our case, it falls about $c_\mu \approx 0.311$ (where Rama Cont [2] indicates that it should fall in the $[0.1, 1]$ range in every market), with a variance of about $\sigma = 0.032$ (which accounts for roughly 10% of the c_μ, validating the fact that c should be, or is more-or-less, constant).

4 Price Impact - Order Events Model

In this section, we extend our previous model by taking into account **all** the possible order book events. We start by providing additional definitions of interest:

Definition 4.1. *Total Flow (TF)*
The TF is a measure of total activity. That is, the TF measures, for a given time period Δ, how active were the participants in both sides of the market. It can be expressed as:

$$TF(t,t+\Delta) = (L^b(t,t+\Delta)+C^b(t,t+\Delta)+M^s(t,t+\Delta))+(L^s(t,t+\Delta)+C^s(t,t+\Delta)+M^b(t,t+\Delta)) \tag{4.1}$$

Definition 4.2. *Trade Inflow (TI)*
The TI is a ***measure of excess inflow trades over outflow trades****. That is, the TI measures, for a given time period Δ, the relative increase of traded orders in the buy-side of the market to the traded orders in the sell-side of the market. It can be expressed as:*

$$TI(t,t+\Delta) = M^s(t,t+\Delta) - M^b(t,t+\Delta) \tag{4.2}$$

Definition 4.3. *Volume (VOL)*
The VOL is a ***measure of trade activity****. That is, the TF measures, for a given time period Δ, the total number of trades that occurred in both the buy-side and the sell-side of the market. It can be expressed as:*

$$VOL(t,t+\Delta) = M^s(t,t+\Delta) + M^b(t,t+\Delta) \tag{4.3}$$

Interestingly, Rama Cont [2] conveys that the OFI accounts for the other factors. That is, the OFI *covers* the information provided by TI, while VOL is *indirectly* related to the change of price. We will show in this section that OFI is indeed the most relevant factor, and that other factors may be additionally helpful in some minor cases.

4.1 Linear Model

We start by providing a quick extension of the OFI model 3.1 by adding the above factors. That is, we formulate the below linear model:

$$\frac{1}{\delta}\Delta S(t,t+\Delta) = \beta_i \text{OFI}(t,t+\Delta)+\gamma_i \text{TF}(t,t+\Delta)+\eta_i \text{TI}(t,t+\Delta)+\iota_i \text{VOL}(t,t+\Delta)+\epsilon(t,t+\Delta) \tag{4.4}$$

By proceeding similarly as previously, we perform an OLS in order to estimate the coefficients $\beta_i, \gamma_i, \eta_i, \iota_i$, and we report the results in table 4.1.

		OFI $\Delta = 1$	Model $\Delta = 10$	Extended $\Delta = 1$	Model $\Delta = 10$
Average validation RMSE	RMSE_i^{avg}	0.585	3.668	0.603	5.087
Min validation RMSE	RMSE_i^{min}	0.357	1.538	0.377	1.473
Max validation RMSE	RMSE_i^{max}	1.143	5.292	1.123	11.388
Average validation R^2	$(R^2)_i^{avg}$	0.515	0.545	0.526	0.471
Min validation R^2	$(R^2)_i^{min}$	0.278	-0.141	0.305	-0.261
Max validation R^2	$(R^2)_i^{max}$	0.625	0.799	0.647	0.785

Table 4.1: OLS Results on Order Events Model

We observe the following:

1. For $\Delta = 1$ second, there is a slight increase in R^2 score on validation (unseen) data.

2. For $\Delta = 10$ seconds, there is a slight decrease in R^2 score on validation (unseen) data. The reason for that will be explained in the next part.

3. For both Δ's, there is a slight increase in RMSE, which is normal as we increase the number of factors).

4. In any case, the increase is R^2 is not substantial (and we even have a decrease in R^2 for $\Delta = 10$ seconds. This gives a primal intuition on the fact that taking into account these additional factors **do not** help much.

Note on LASSO Regression. LASSO sets the coefficients η_i and ι_i to zero, and gives a very low value of γ_i. Although, we will not provide the LASSO report here as we deal with very few features. Instead, we will provide a thorough LASSO analysis on the next model, when we will use extra factors and expand our feature matrix to polynomial features.

4.2 Regression Decision Tree Modeling

By the fact that the above linear model does not provide much extra information, we decided to get an idea on the cases where the other factors are relevant through regression decision trees, where we use the `gini` impurity, and the two hyperparameters consisting of the max depth size D and min bucket size S. We get an optimal pair of $D^* = 4$, $S^* = 50$, and report the results in table 4.2.

		$\Delta = 1$ second	$\Delta = 10$ seconds
Average validation RMSE	RMSE_i^{avg}	0.63	4.45
Average validation R^2	$(R^2)_i^{avg}$	0.57	0.71

Table 4.2: Regression Decision Tree Results on OFI Model

In comparison with OLS or LASSO, this model performs better, and has the advantage of providing details that the other methods don't. Before explaining these, we plot the decision tree for $\Delta = 1$ second in figure 4.1, and for $\Delta = 10$ seconds in figure 4.2.

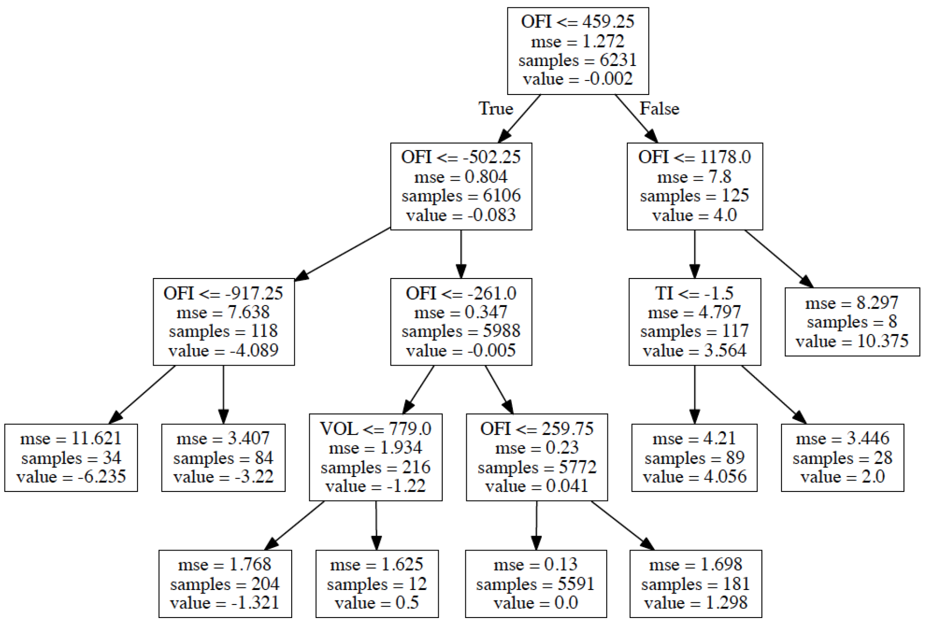


Figure 4.1: Regression Decision Tree - $\Delta = 1$ second

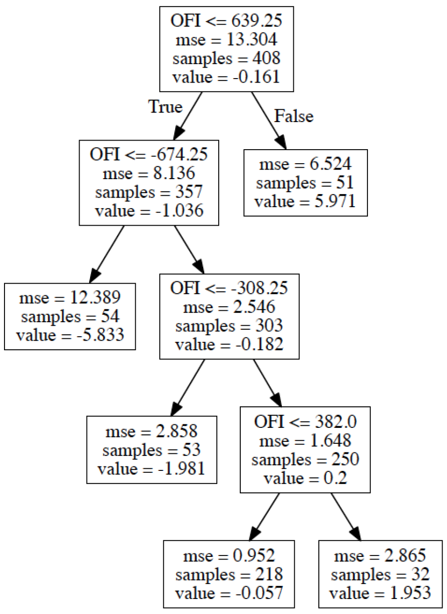


Figure 4.2: Regression Decision Tree - $\Delta = 10$ seconds

We observe the following:

1. For $\Delta = 1$ second, the Order Flow Imbalance (OFI) is the **most** important factor, as it has most splits at most levels of the trees. We observe that the Trade Inflow (TI) is informative in the cases of a moderately high OFI ($\in (569.25, 1178]$) to indicate a large increase in price in the case it is low enough (< -1.5). Moreover, the Trade Volume (VOL) actually determines the direction of the price change in the case where the OFI is moderately small ($\in (-502.25, -261)$).

2. For $\Delta = 10$ seconds, the Order Flow Imbalance (OFI) is the **only** important factor. A plausible reason for that would be that, over larger time periods, the number of market trades becomes even smaller relative the the total activity, so that the OFI covers for most (all) of the relevant information.

5 Using Universal Approximators to Identify Additional Factors

In this section, we formulate a general model for price impact, taking into account the two different kind of factors below:

1. Order event factors:
   (a) **OFI**: Order Flow Imbalance (previously defined).
   (b) **TF**: Total Flow (previously defined).
   (c) **TI**: Trade Inflow (previously defined).
   (d) **VOL**: Trade Volume (previously defined).
   (e) **pi**: ratio of trades over all activity.
   (f) **mu**: mean of trade sizes.
2. Historical price factors:
   (a) **SIGN**: sign of previous price change:
   (b) **SIGN_RAT**: average of all previous SIGN:
   (c) **VAL**: value of previous price change:
   (d) **VAL_RAT**: average of all previous VAL:
   (e) **IS**: initial spread:

We start by providing an intuition on how our factors data *separates* using principal component analysis, and the directions with most variances (which may permit us to reduce the dimensionality of our problem). We then use our first class of universal approximators - polynomials - in order to formulate a generalized model for price impact. We then use regression tree based methods in order to get some more intuition on the relevant factors. Lastly, we use regression neural networks in order to have our *strongest* model in terms of predictive power.

5.0.1 Defining the New Formulas

$$\text{pi} = \frac{\text{VOL}}{\text{TF}} \tag{5.1}$$

$$\mu = \frac{\text{average size of buy trades} + \text{average size of sell trades}}{2} \tag{5.2}$$

$$\text{SIGN}(t, t+\Delta) = \begin{cases} +1 & \Delta S(t-\Delta, t) > 0 \\ -1 & \text{otherwise} \end{cases} \tag{5.3}$$

$$\text{SIGN\_RAT} = \overline{\text{SIGN}} \tag{5.4}$$

$$\text{VAL}(t, t+\Delta) = \Delta S(t-\Delta, t) \tag{5.5}$$

$$\text{VAL\_RAT} = \overline{\text{VAL}} \tag{5.6}$$

$$\text{IS} = \text{ask}_L^t - \text{bid}_H^t \tag{5.7}$$

$$\tag{5.8}$$

5.1 Dimensionality Reduction Using Principal Component Analysis

We start by gaining some intuition by performing a PCA on our factors data. The aim of this is to show that our data is *separable* in the sense that the direction with highest variance shows a clear fluctuation of our factor data points. We provide in figures 5.1 and 5.2 the .

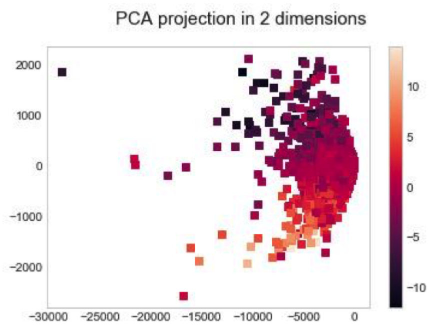


Figure 5.1: PCA Projection on Direction with Most Variance - $\Delta = 1$ second

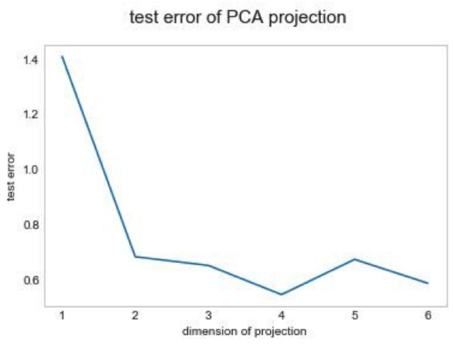


Figure 5.2: OLS Performance w.r.t. Number of Taken Directions - $\Delta = 1$ second

It appears that PCA projection is meaningful: when looking at the scatterplot of the 2-dimension PCA projection, the color gradient indicates that differentiated values of the output tend to belong to different areas. Qualitatively, this means that PCA dimensionality reduction has the potential to improve prediction. When performing cross-validation, i.e. looking for the dimension reduction that yields the best test error (MSE) for linear regression after PCA, it appears that dimensionality reduction only marginally improves the performance, at least for linear regression. Indeed for dimensions 2 to 6, the test error only marginally changes, with the best result obtained for 4 dimensions. So if it were very advantageous for some other reason to reduce dimension using PCA (for example to reduce computation time), it would be relevant to use dimension 4. In fact, the first two dimensions *eat up* more than 90% of the variance in our factors matrix.

5.2 Polynomial Regression Based Methods

For convenience, we will write the 11 factors above as $x_j, j \in \{1, 2, ..., 11\}$. The aim of this subsection is to model the price impact as a combination of (interacted monomials of) the above factors. That is, we wish to determine the most relevant polynomial p_d of x_j of degree d - a hyperparameter - such that:

$$\begin{aligned}\frac{1}{\delta}\Delta S(t, t+\Delta) &= p_d(x_1, x_2, ..., x_{11}) + \epsilon(t, t+\Delta) \\ &= \sum_{i=1}^{d} \sum_{\sum_{j=1}^{11} i_j = 11} c_{i_1, i_2, ..., i_{11}} \prod_{j=1}^{11} x_j^{i_j} + \epsilon(t, t+\Delta) \qquad (5.9)\end{aligned}$$

We first provide the motivation of this while providing the associated methodology, then show how to use sparsity-promoting techniques in order to reduce our *augmented factor space* to the significant factors, and write the finalized model.

5.2.1 Motivation for Polynomial Regression

Here we answer why are polynomial regressions relevant to our framework. We know from Taylor's theorem that polynomials are universal approximators, so that it is in fact possible to *approximate* any function f (with any margin of error e) using a polynomial of degree d

high-enough. That is, regardless of the *true* underlying model of price change relative to our above-defined factors:

$$\frac{1}{\delta}\Delta S(t, t+\Delta) = f(x_1, x_2, ..., x_{11}) + \epsilon(t, t+\Delta) \tag{5.10}$$

we can find a polynomial p_d of degree d high enough such that, for every margin of error $e > 0$:

$$|f(x_1, x_2, ..., x_{11}) - p_d(x_1, x_2, ..., x_{11})| < e, \forall x_1, x_2, ..., x_{11} \tag{5.11}$$

provided of course that f is differentiable as many times as the degree d of the approximate polynomial. Thus, performing a polynomial regression would, with enough precision, provide us with a *generalized* model for the price impact w.r.t. the above factors. However, we do know that there must be some fixed underlying model, and that performing a high-degree polynomial regression would certainly decrease the *training* error by decreasing the bias, but would increase the *validation* error by introducing some variance due to noise overfitting. Therefore, we require again validating our results with **unseen** data in order to come up with the optimal degree d^*.

5.2.2 Identifying the Optimal Degree

We now do some hyperparameter tuning on d, where we solve multiple linear regressions on our polynomial features in order to determine the best d. We plot validation variation of RMSE and R^2 in terms of d in figures 5.3, 5.4 respectively.

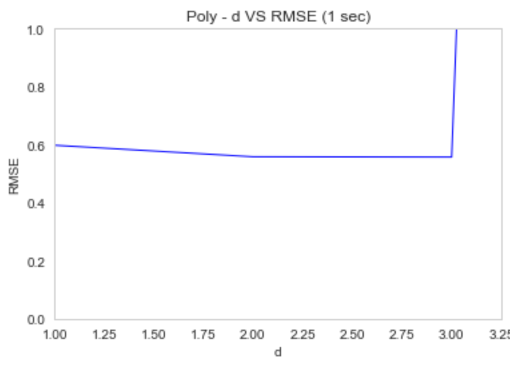


Figure 5.3: Poly - d VS RMSE - $\Delta = 1$ second

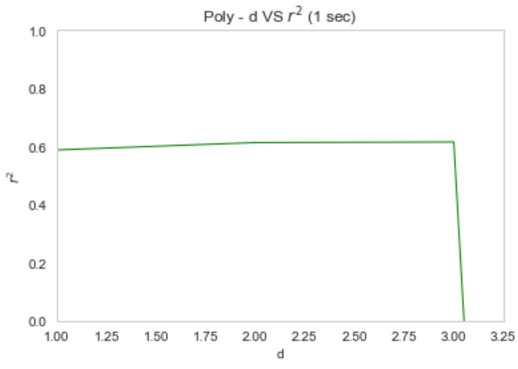


Figure 5.4: Poly - d VS R^2 - $\Delta = 1$ second

We observe a slight increase in model performance from $d = 1$ to $d = 2$, then no change from $d = 2$ to $d = 3$. Any model with higher degree than $d = 3$ performs very mad. We thus choose $d^* = 2$.

5.2.3 Polynomial LASSO Regression

After having identified above our optimal degree $d^* = 2$, we use the fact that LASSO promotes sparsity in order to get the most relevant factors of the $d^* = 2$-augmented 11, i.e. a total of $\binom{11+2}{2} = 78$. That is, we perform a LASSO regression - which will set to 0 the *useless* coefficients - by minimizing the above loss function over the coefficients $c_{i_1,i_2,...,i_{11}}$ in 5.9:

$$\textbf{LASSO} : \min_{\vec{c}} \sum_{t=1}^{n_i} (y_t - p_2(x_{1,t}, x_{2,t}, ..., x_{11,t}))^2 + \lambda ||\vec{c}||_1 \tag{5.12}$$

We provide an overview of the model performance against the regularizer parameter $\lambda = \frac{1}{\alpha^2}$ in figure 5.5.

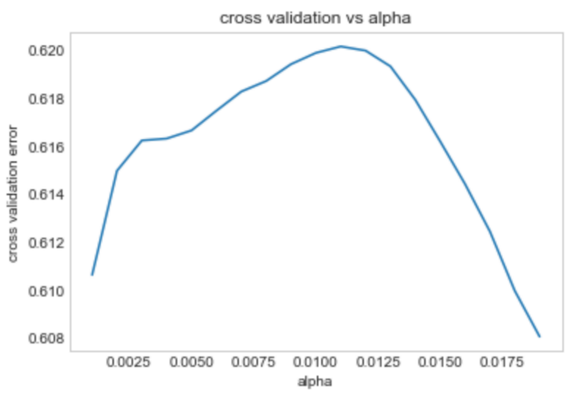


Figure 5.5: LASSO - RMSE VS α - $\Delta = 1$ sec

We also provide the non-zero coefficients of the predicted (most relevant) factors, along with the results, for the optimal cross-validation chosen $\alpha^* = 0.011$ in table 5.1.

Coefficient of:	**Value**
IS^2	$-7.973e-09$
OFI	$1.369e-01$
OFI $\times$ IS	$6.227e-07$
OFI $\times$ SIGN	$5.353e-05$
OFI $\times$ TF	$4.059e-07$
OFI $\times$ TI	$1.158e-06$
OFI $\times$ VAL	$7.326e-07$
OFI $\times$ VOL	$-3.007e-07$
OFI $\times$ mu	$1.536e-05$
OFI^2	$8.184e-07$
SIGN $\times$ IS	$1.918e-05$
TF	$-1.746e-04$
TF $\times$ IS	$4.463e-08$
TF $\times$ SIGN	$-5.108e-05$
TF $\times$ TI	$-7.839e-08$
TF $\times$ VAL	$3.837e-05$
TF $\times$ VOL	$-1.811e-07$
TF $\times$ mu	$2.208e-06$
TF $\times$ pi	$5.759e-05$
TI	$-2.298e-03$
TI $\times$ IS	$-1.251e-07$
TI $\times$ VAL	$9.271e-05$
TI $\times$ VOL	$9.782e-07$
TI $\times$ mu	$1.698e-05$
TI $\times$ pi	$2.563e-04$
TI^2	$3.697e-07$
VAL $\times$ IS	$-1.161e-05$
VOL	$7,995e-05$
VOL $\times$ IS	$5.416e-07$
VOL $\times$ SIGN	$-3.354e-05$
VOL $\times$ VAL	$-1.582e-04$
VOL $\times$ mu	$-1.074e-05$
VOL^2	$7.995e-05$
mu $\times$ IS	$-1.672e-06$
mu^2	$4.151e-06$

		$\Delta = 1$ second
Average validation RMSE	RMSE_i^{avg}	0.67
Average validation R^2	$(R^2)_i^{avg}$	0.62

Table 5.1: LASSO - Predicted Coefficients and Results - $\Delta = 1$ sec

5.2.4 Polynomial Elastic Net Regression

We now perform a polynomial elastic net on our polynomial model, which combines RIDGE and LASSO regularizers. The advantage of this method is:

1. It has the sparsity-promoting advantage of LASSO.
2. It has the *uniform radial expansion* of the predicted coefficients over its variance range property of RIDGE.

In short, it should have both the advantages of LASSO and RIDGE, and provide in some way the best combination of **performance** and **sparsity** of the prediction. It consists in minimizing the below loss function:

$$\textbf{ELASTIC NET} : \min_{\vec{c}} \sum_{t=1}^{n_i} (y_t - p_2(x_{1,t}, x_{2,t}, ..., x_{11,t}))^2 + \lambda_1 ||\vec{c}||_1 + \lambda_2 ||\vec{c}||_2 \tag{5.13}$$

We present the contour plots of the regularizer in figure 5.6.

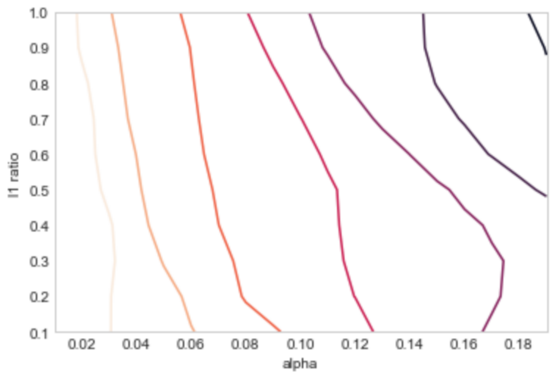


Figure 5.6: ELASTIC NET - Contours of λ VS α - $\Delta = 1$ sec

However, get get a regularizer ratio of 1, which means that LASSO took over RIDGE in our ELASTIC NET regression. This shows that LASSO regression was the best performing sub-model, as many factors were actually **irrelevant** in predicting price variations.

5.2.5 Final Polynomial Model

We now synthesis our whole polynomial analysis, by formulating (thanks to LASSO reduction) our finalized model below (by disregarding in figure 5.6 coefficients that are smaller than $1e-5$):

$$\begin{aligned} \frac{1}{\delta}\Delta S(t, t+\Delta) \approx 0.001 \times \text{OFI} - 0.0002 \times \text{TF} - 0.002 \times \text{TI} \\ - 0.0002 \times \text{TI} \times \text{pi} - 0.0001 \times \text{VOL} \times \text{VAL} \end{aligned} \tag{5.14}$$

Although, other factors should also be taken into consideration. For example, we expect other some joint factors to have importance in some cases as well. The power of this final model is that, as previously mentioned, is a **universal approximator** of the true underlying model of the price impact w.r.t. the factors we chose.

5.3 Regression Tree Based Methods

The aim of this subsection is to use regression tree based methods in order to get some insights on the most relevant factors w.r.t. the change of price.

5.3.1 Random Forest

We first tried Random Forest on our initial-non augmented data. A word on the hyperparameters we chose to grid over. For our tree we chose minimum samples on each leaf (equivalent to minimum bucket) = 5 and minimum samples split = 10. We chose to iterate on maximum tree depth and maximum features chosen as well as the number of trees that are build. The criterion we chose was MSE and we did not include minimum impurity gain. The best combination of hyperparameters thus were (number of features, maximum depth, number of trees) $= (5, 8, 500)$. We provide our results in table 5.2.

		$\Delta = 1$ second
Average validation RMSE	RMSE_i^{avg}	0.703
Average validation R^2	$(R^2)_i^{avg}$	0.574

Table 5.2: Random Forest - Results - $\Delta = 1$ sec

Unfortunately, this model performed less good than random forest, and so is less representative. We anyways provide the features of importance we obtained from Ada-Boost in the figure 5.8.

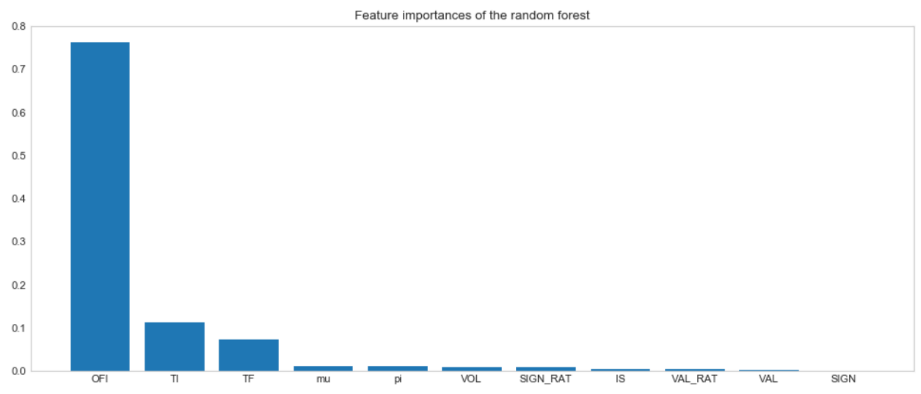


Figure 5.7: Random Forest - Most Relevant Features - $\Delta = 1$ sec

We observe as expected that OFI is the most significant factor, followed by TI and TF. The other factors do not seem to have much importance on their own (but as seen previously, do have when *combined* with others).

5.3.2 Ada-Boost

A Background Note. the algorithm for Boosting trees evolved from the application of boosting methods to regression trees. The concept of Boosting is essentially computing a sequence of simple weak models(high bias) where each model is built for the prediction residuals of the preceding model. Thus, in adaboost regression, we build a small regression tree, and then we sequentially build trees on the residuals of the proceeding models. In that way, the final model is a weighted combination of models, which in our case is shallow trees. Hence we can see that the adaboost algorithm is an expansion of the regression tree algorithm and it represents a very general and powerful machine learning algorithm.

Using Ada-Boost. next, we tried Ada-Boost. Regarding the hyperparameter tuning for AdaBoost we have the following: first, we have to choose your base estimator, which is going to be a regression tree for us, and thus we have to chose all the parameters of the tree. Then, we have to choose the number of trees that we sequentially are going to train, the loss function that is going to define the weights and the learning rate.. We set the parameters of tree to be the predefined besides the maximum depth. Also, we set the parameters of the boosting algorithm as the default ones except the shrinkage and the number of trees to be boosted. Then we performed grid search over the shrinkage parameter the number of trees to be boosted and the maximum depth. We found that the optimal combination of parameters is (max_depth, n_estimators, shrinkage) $= (4, 500, 0.11)$. We provide our results in table 5.3.

		$\Delta = 1$ second
Average validation RMSE	RMSE_i^{avg}	0.649
Average validation R^2	$(R^2)_i^{avg}$	0.645

Table 5.3: Random Forest - Results - $\Delta = 1$ sec

We observe that this model is the best so far in terms of not explaining the least variance of price variations (has the highest R^2. Moreover, The features of importance we obtained from the random forest are given in the figure 5.7.

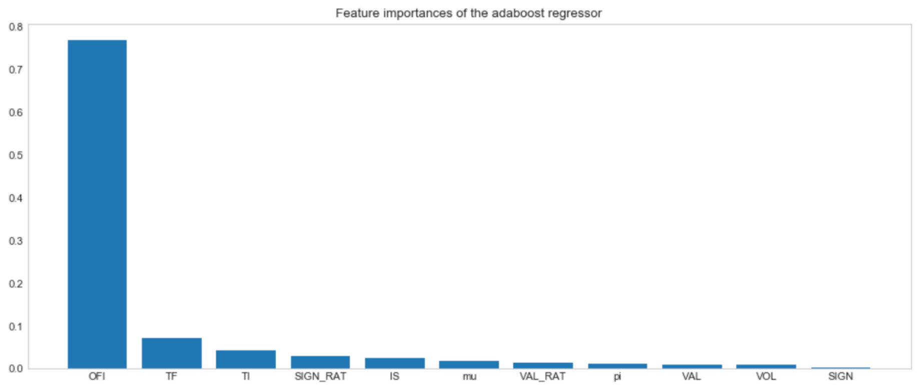


Figure 5.8: Ada-Boost - Most Relevant Features - $\Delta = 1$ sec

Interestingly, it shows that the SIGN_RAT factor is more important than previously told.

5.4 Regression Neural Networks Based Methods

Our last approach was to try neural networks. **Note that we did not have time to achieve it perfectly**, but at least we provide with the ideas we came up with and the reason for them.

5.4.1 Motivation

The motivation behind using neural nets is the fact that, similarly to polynomials, they are *universal approximators*, and so should approximate with **any** margin of error by sufficiently tuning hyperparameters the true underlying model of price impact w.r.t. our factors. We start by formulating the model we try to get below:

$$\frac{1}{\delta}\Delta S(t, t+\Delta) = \sigma_L(\mathbf{W}_L\sigma_{L-1}(...\sigma_2(\mathbf{W}_2\sigma_1(\mathbf{W}_1\mathbf{x}))...)) \tag{5.15}$$

Which shows the *universal approximation* property, as it appears in figure 5.9.

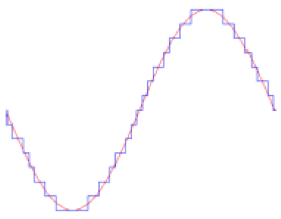


Figure 5.9: Neural Nets as Universal Approximators

5.4.2 Methodology & Results

We created a feedforward dense neural network, were we applied dropout after each layer including the input one. Then we performed grid search over the dropout retention and we iterated over and the number of neurons in each layer performing cross validation. We looked for dropout in range of $(0.2, 0.5)$ and number of neurons per layer from 100 to 300. We present the results of our neural net training in table 5.4.

		$\Delta = 1$ second
Average validation RMSE	RMSE_i^{avg}	0.731
Average validation R^2	$(R^2)_i^{avg}$	0.539
Best dropout		0.2
Best number of neurons per layer		100

Table 5.4: Neural Nets - Results - $\Delta = 1$ sec

We give an idea of how our neural net looks like in figure 5.10.

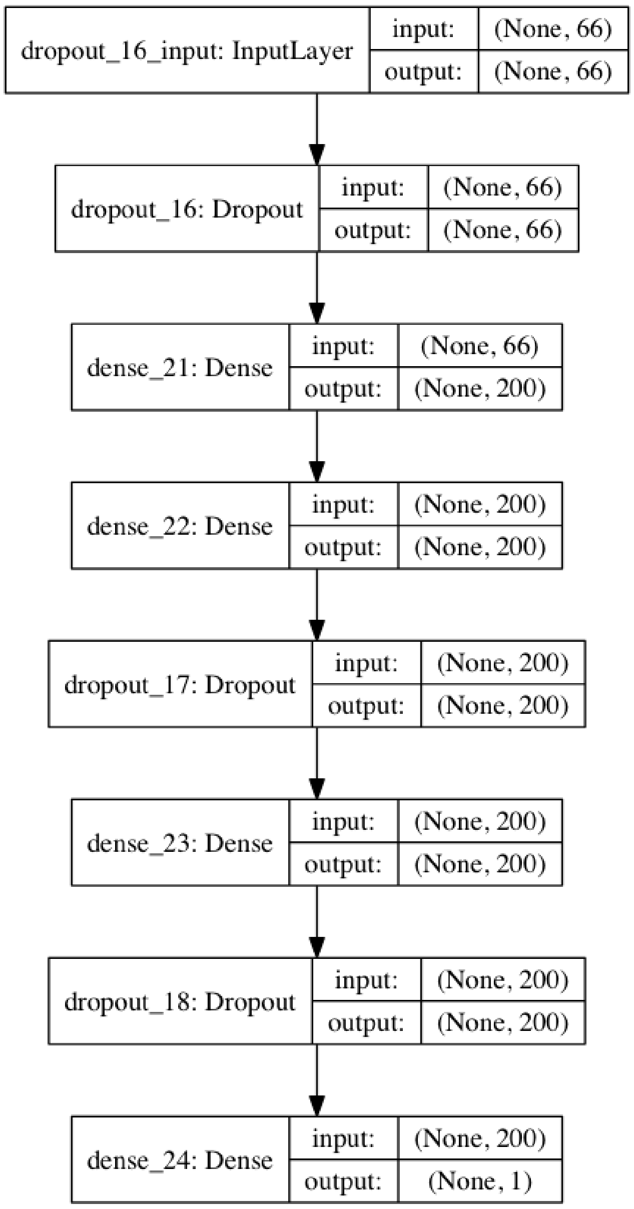


Figure 5.10: Our Neural Nets Visualisation

6 Conclusion

We sum up our work as a whole on price impact modeling. We have done the following:

1. **validated** the results of Rama Cont [2], while coming up with additional tests/ estimation methods of our own in order to confirm/ show the relevance of the provided models/ results. We perform some linear and polynomial regressions using OLS, and showed that the underlying relation between price change and OFI is indeed most likely linear. We also provided a clear view on the underlying relation between the previous regression coefficient and the average market depth at regression time.

2. **extended** the previous model by taking into account **all** order book events. However, we showed that the OFI is the most important feature, by using a regression decision tree and showing how the splits are being done and what are the corresponding implications, so that this small extension is not more informative, i.e. **most** (almost all) of the information provided by order book events actually come from OFI.

3. **generalized** a model that takes not only order book events, but also historic price related information. We used very powerful tools - universal approximation techniques - in order to come up with the most appropriate model. The use of polynomial regression permitted us to obtain a good representative polynomial of the price impact, while the use of random forest provided us with a measure of the factors' importance. We also tried another class of universal approximators, neural nets, in order to have a convoluted model of the price impact. Although incomplete, this last model can be worked out to have great results and give a good model of price impact.

As a promising further research, one may look at our proposed framework for neural nets, or dig deeper into other kind of factors that could affect the price impact (non-[order book event], non-[historic price related]).

References

[1] LOBSTER — academic data [Online].
Available: `http://LOBSTER.wiwi.hu-berlin.de` .

[2] Rama Cont, Arseniy Kukanov and Sasha Stoikov, *The price impact of order book events.* arXiv:1011.6402v3 [q-fin.TR] 13 Apr 2011.

[3] Rama Cont, *Statistical Modeling of High-Frequency Financial Data.* IEEE Signal Processing Magazine, September 2001.

[4] Soroush Nasiriany, Garrett Thomas, William Wei Wang, Alex Yang, *A Comprehensive Guide to Machine Learning.* University of California, Berkeley, January 2018.

A Python Code for Data Description

A.1 Procedure to describe the buy/sell market orders

```
# here we work on the market buys
title = 'buy market orders'
plt.figure()
lines = plt.plot(market_b[0], market_b[4]/100)
plt.setp(lines, 'color', 'r', 'linewidth', 1.0)
plt.xlabel('time after midnight (sec)', fontsize=10)
plt.ylabel('price (ticks)', fontsize=10)
plt.title(title)
plt.show()

inter_t = np.zeros(market_b[0].size - 1)
for i in range(1,market_b[0].size):
    inter_t[i-1] = market_b[0].iloc[i] - market_b[0].iloc[i-1]

print('Average inter-time between two market buys = ', np.mean(inter_t),' sec')
print('Average price of market buys = ', np.mean(market_b[4]/100),' ticks')
print('Standard deviation price of market buys = ', np.sqrt(np.var(market_b[4]/100)),' ticks')

import pylab
import scipy.stats as stats
stats.probplot(inter_t, dist="expon", plot=pylab)
pylab.title('qq-plot - inter-arrival of buy market orders against exponential')
pylab.show()

import pylab
import scipy.stats as stats
stats.probplot(inter_t, dist="norm", plot=pylab)
pylab.title('qq-plot - inter-arrival of buy market orders against normal')
pylab.show()

import pylab
import scipy.stats as stats
stats.probplot(inter_t, dist="uniform", plot=pylab)
pylab.title('qq-plot - inter-arrival of buy market orders against uniform')
pylab.show()
```

```
# here we work on the market sells
title = 'sell market orders'
plt.figure()
lines = plt.plot(market_s[0], market_s[4]/100)
plt.setp(lines, 'color', 'b', 'linewidth', 1.0)
plt.xlabel('time after midnight (sec)', fontsize=10)
plt.ylabel('price (ticks)', fontsize=10)
plt.title(title)
plt.show()

inter_t = np.zeros(market_s[0].size - 1)
for i in range(1,market_s[0].size):
    inter_t[i-1] = market_s[0].iloc[i] - market_s[0].iloc[i-1]

print('Average inter-time between two market sells = ', np.mean(inter_t),' sec')
print('Average price of market sells = ', np.mean(market_s[4]/100),' ticks')
print('Standard deviation price of market sells = ', np.sqrt(np.var(market_s[4]/100)),' ticks')

import pylab
import scipy.stats as stats
stats.probplot(inter_t, dist="expon", plot=pylab)
pylab.title('qq-plot - inter-arrival of sell market orders against exponential')
pylab.show()

import pylab
import scipy.stats as stats
stats.probplot(inter_t, dist="norm", plot=pylab)
pylab.title('qq-plot - inter-arrival of sell market orders against normal')
pylab.show()

import pylab
import scipy.stats as stats
stats.probplot(inter_t, dist="uniform", plot=pylab)
pylab.title('qq-plot - inter-arrival of sell market orders against uniform')
pylab.show()
```

A.2 Procedure to describe the buy/sell limit orders

```
# here we work on the limit buys
title = 'buy limit orders'
plt.figure()
lines = plt.plot(limit_b[0], limit_b[4]/100)
plt.setp(lines, 'color', 'r', 'linewidth', 1.0)
plt.xlabel('time after midnight (sec)', fontsize=10)
plt.ylabel('price (ticks)', fontsize=10)
plt.title(title)
plt.show()

inter_t = np.zeros(limit_b[0].size - 1)
for i in range(1,limit_b[0].size):
    inter_t[i-1] = limit_b[0].iloc[i] - limit_b[0].iloc[i-1]

print('Average inter-time between two limit buys = ', np.mean(inter_t),' sec')
print('Average price of limit buys = ', np.mean(limit_b[4]/100),' ticks')
print('Standard deviation price of limit buys = ', np.sqrt(np.var(limit_b[4]/100)),' ticks')

import pylab
import scipy.stats as stats
stats.probplot(inter_t, dist="expon", plot=pylab)
pylab.title('qq-plot - inter-arrival of buy limit orders against exponential')
pylab.show()

import pylab
import scipy.stats as stats
stats.probplot(inter_t, dist="norm", plot=pylab)
pylab.title('qq-plot - inter-arrival of buy limit orders against normal')
pylab.show()

import pylab
import scipy.stats as stats
stats.probplot(inter_t, dist="uniform", plot=pylab)
pylab.title('qq-plot - inter-arrival of buy limit orders against uniform')
pylab.show()
```

```
# here we work on the limit sells
title = 'sell limit orders'
plt.figure()
lines = plt.plot(limit_s[0], limit_s[4]/100)
plt.setp(lines, 'color', 'b', 'linewidth', 1.0)
plt.xlabel('time after midnight (sec)', fontsize=10)
plt.ylabel('price (ticks)', fontsize=10)
plt.title(title)
plt.show()

inter_t = np.zeros(limit_s[0].size - 1)
for i in range(1,limit_s[0].size):
    inter_t[i-1] = limit_s[0].iloc[i] - limit_s[0].iloc[i-1]

print('Average inter-time between two limit sells = ', np.mean(inter_t),' sec')
print('Average price of limit sells = ', np.mean(limit_s[4]/100),' ticks')
print('Standard deviation price of limit sells = ', np.sqrt(np.var(limit_s[4]/100)),' ticks')

import pylab
import scipy.stats as stats
stats.probplot(inter_t, dist="expon", plot=pylab)
pylab.title('qq-plot - inter-arrival of sell limit orders against exponential')
pylab.show()

import pylab
import scipy.stats as stats
stats.probplot(inter_t, dist="norm", plot=pylab)
pylab.title('qq-plot - inter-arrival of sell market orders against normal')
pylab.show()

import pylab
import scipy.stats as stats
stats.probplot(inter_t, dist="uniform", plot=pylab)
pylab.title('qq-plot - inter-arrival of sell market orders against uniform')
pylab.show()
```

A.3 Procedure to describe the buy/sell cancelled orders

```
# here we work on the cancelled buys
title = 'buy cancelled orders'
plt.figure()
lines = plt.plot(cancelled_b[0], cancelled_b[4]/100)
plt.setp(lines, 'color', 'r', 'linewidth', 1.0)
plt.xlabel('time after midnight (sec)', fontsize=10)
plt.ylabel('price (ticks)', fontsize=10)
plt.title(title)
plt.show()

inter_t = np.zeros(cancelled_b[0].size - 1)
for i in range(1,cancelled_b[0].size):
    inter_t[i-1] = cancelled_b[0].iloc[i] - cancelled_b[0].iloc[i-1]

print('Average inter-time between two cancelled buys = ', np.mean(inter_t),' sec')
print('Average price of cancelled buys = ', np.mean(cancelled_b[4]/100),' ticks')
print('Standard deviation price of cancelled buys = ', np.sqrt(np.var(cancelled_b[4]/100)),' ticks')

import pylab
import scipy.stats as stats
stats.probplot(inter_t, dist="expon", plot=pylab)
pylab.title('qq-plot - inter-arrival of buy cancelled orders against exponential')
pylab.show()

import pylab
import scipy.stats as stats
stats.probplot(inter_t, dist="norm", plot=pylab)
pylab.title('qq-plot - inter-arrival of buy cancelled orders against normal')
pylab.show()

import pylab
import scipy.stats as stats
stats.probplot(inter_t, dist="uniform", plot=pylab)
pylab.title('qq-plot - inter-arrival of buy cancelled orders against uniform')
pylab.show()
```

```
# here we work on the cancelled sells
title = 'sell cancelled orders'
plt.figure()
lines = plt.plot(cancelled_s[0], cancelled_s[4]/100)
plt.setp(lines, 'color', 'b', 'linewidth', 1.0)
plt.xlabel('time after midnight (sec)', fontsize=10)
plt.ylabel('price (ticks)', fontsize=10)
plt.title(title)
plt.show()

inter_t = np.zeros(cancelled_s[0].size - 1)
for i in range(1,cancelled_s[0].size):
    inter_t[i-1] = cancelled_s[0].iloc[i] - cancelled_s[0].iloc[i-1]

print('Average inter-time between two cancelled sells = ', np.mean(inter_t),' sec')
print('Average price of cancelled sells = ', np.mean(cancelled_s[4]/100),' ticks')
print('Standard deviation price of cancelled sells = ', np.sqrt(np.var(cancelled_s[4]/100)),' ticks')

import pylab
import scipy.stats as stats
stats.probplot(inter_t, dist="expon", plot=pylab)
pylab.title('qq-plot - inter-arrival of sell cancelled orders against exponential')
pylab.show()


import pylab
import scipy.stats as stats
stats.probplot(inter_t, dist="norm", plot=pylab)
pylab.title('qq-plot - inter-arrival of sell cancelled orders against normal')
pylab.show()

import pylab
import scipy.stats as stats
stats.probplot(inter_t, dist="uniform", plot=pylab)
pylab.title('qq-plot - inter-arrival of sell cancelled orders against uniform')
pylab.show()
```

A.4 Procedure to describe the spread

```
# here we work on the spread
spreads = lob['Ask_Price_1'] - lob['Bid_Price_1']
title = 'spread'
plt.figure()
lines = plt.plot(lob['Time'], spreads/100)
plt.setp(lines, 'color', 'r', 'linewidth', 0.1)
plt.xlabel('time after midnight (sec)', fontsize=10)
plt.ylabel('spread (ticks)', fontsize=10)
plt.title(title)
plt.show()

print('Average spread = ',np.mean(spreads/100),' ticks')
print('Standatd deviation spread = ',np.sqrt(np.var(spreads/100)),' ticks')
```

A.5 Procedure to describe the limit order book statistics

```
# here we work on the number of signals/ sec (like in Rama Cont's paper)
start = lob['Time'].iloc[0]
end = lob['Time'].iloc[-1]
avg_sig = 0
itr = 1
while(start<end):
    lim_cur = limit_b[(limit_b[0] >= start) & (limit_b[0] <= start+1)]
    avg_sig = avg_sig + np.sum(lim_cur[3])
    lim_cur = limit_s[(limit_s[0] >= start) & (limit_s[0] <= start+1)]
    avg_sig = avg_sig + np.sum(lim_cur[3])
    start = start + 1
    itr = itr + 1
avg_sig = avg_sig/itr

print('Average number of orders per sec = ',avg_sig)
```

```
# here we work on the number of signals/ 10 sec (like in Rama Cont's paper)
start = lob['Time'].iloc[0]
end = lob['Time'].iloc[-1]
avg_sig = 0
itr = 1
while(start<end):
    lim_cur = limit_b[(limit_b[0] >= start) & (limit_b[0] <= start+10)]
    avg_sig = avg_sig + np.sum(lim_cur[3])
    lim_cur = limit_s[(limit_s[0] >= start) & (limit_s[0] <= start+10)]
    avg_sig = avg_sig + np.sum(lim_cur[3])
    start = start + 10
    itr = itr + 1
avg_sig = avg_sig/itr

print('Average number of orders per 10 sec = ',avg_sig)
```

```
# here we work on the changes of price
prices = np.array(0.5*(lob['Ask_Price_1'] + lob['Bid_Price_1']))
```

```
changes = np.zeros(prices.size -1)
num_changes = 0
for i in range(1,prices.size):
    changes[i-1] = prices[i] - prices[i-1]
    if(prices[i] != prices[i-1]):
        num_changes = num_changes + 1

print('Price changes in 1 day = ',num_changes)
print('Max change = ',np.max(np.abs(changes))/100)
print('Min change = ',np.min(np.abs(changes))/100)
print('Mean change = ',np.mean(np.abs(changes))/100)
print('Std change = ',np.sqrt(np.var(np.abs(changes))/100))
```

```
# here we work on the changes of price for 1 sec
start = lob['Time'].iloc[0]
end = lob['Time'].iloc[-1]
changes = []
while(start<end):
    lob_cur = lob[(lob['Time'] >= start) & (lob['Time'] <= start+1)]
    if(lob_cur.size==0):
        start = start + 1
        continue
    p1 = 0.5*(lob_cur['Ask_Price_1'].iloc[0] + lob_cur['Bid_Price_1'].iloc[0])
    p2 = 0.5*(lob_cur['Ask_Price_1'].iloc[-1] + lob_cur['Bid_Price_1'].iloc[-1])
    changes.append(p2-p1)
    start = start + 1
changes = np.array(changes)

print('Price changes (T=1sec) in 1 day = ',changes[changes==0].size)
print('Max change = ',np.max(np.abs(changes))/100)
print('Min change = ',np.min(np.abs(changes))/100)
print('Mean change = ',np.mean(np.abs(changes))/100)
print('Std change = ',np.sqrt(np.var(np.abs(changes))/100))
```

```
# here we work on the changes of price for 10 sec
start = lob['Time'].iloc[0] + 3600
end = lob['Time'].iloc[-1]
changes = []
while(start<end):
    lob_cur = lob[(lob['Time'] >= start) & (lob['Time'] <= start+10)]
    if(lob_cur.size==0):
        start = start + 10
        continue
    p1 = 0.5*(lob_cur['Ask_Price_1'].iloc[0] + lob_cur['Bid_Price_1'].iloc[0])
    p2 = 0.5*(lob_cur['Ask_Price_1'].iloc[-1] + lob_cur['Bid_Price_1'].iloc[-1])
    changes.append(p2-p1)
    start = start + 10
changes = np.array(changes)

print('Price changes (T=10sec) in 1 day = ',changes[changes==0].size)
print('Max change = ',np.max(np.abs(changes))/100)
print('Min change = ',np.min(np.abs(changes))/100)
print('Mean change = ',np.mean(np.abs(changes))/100)
print('Std change = ',np.sqrt(np.var(np.abs(changes))/100))
```

A.6 Procedure to describe the "typical" limit order book

```
# here we give an idea of an "average" order book
avg_b5 = lob['Bid_Price_5'].mean()
avg_b4 = lob['Bid_Price_4'].mean()
avg_b3 = lob['Bid_Price_3'].mean()
avg_b2 = lob['Bid_Price_2'].mean()
avg_b1 = lob['Bid_Price_1'].mean()
avg_a1 = lob['Ask_Price_1'].mean()
avg_a2 = lob['Ask_Price_2'].mean()
avg_a3 = lob['Ask_Price_3'].mean()
avg_a4 = lob['Ask_Price_4'].mean()
avg_a5 = lob['Ask_Price_5'].mean()

avg_b5s = lob['Bid_Size_5'].mean()
avg_b4s = lob['Bid_Size_4'].mean()
avg_b3s = lob['Bid_Size_3'].mean()
avg_b2s = lob['Bid_Size_2'].mean()
avg_b1s = lob['Bid_Size_1'].mean()
avg_a1s = lob['Ask_Size_1'].mean()
avg_a2s = lob['Ask_Size_2'].mean()
avg_a3s = lob['Ask_Size_3'].mean()
avg_a4s = lob['Ask_Size_4'].mean()
avg_a5s = lob['Ask_Size_5'].mean()

avg_b5v = lob['Bid_Size_5'].var()
avg_b4v = lob['Bid_Size_4'].var()
avg_b3v = lob['Bid_Size_3'].var()
avg_b2v = lob['Bid_Size_2'].var()
avg_b1v = lob['Bid_Size_1'].var()
avg_a1v = lob['Ask_Size_1'].var()
avg_a2v = lob['Ask_Size_2'].var()
avg_a3v = lob['Ask_Size_3'].var()
avg_a4v = lob['Ask_Size_4'].var()
avg_a5v = lob['Ask_Size_5'].var()

X = np.array([avg_b5,avg_b4,avg_b3,avg_b2,avg_b1,avg_a1,avg_a2,avg_a3,avg_a4,avg_a5])/100
Y = np.array([avg_b5s,avg_b4s,avg_b3s,avg_b2s,avg_b1s,avg_a1s,avg_a2s,avg_a3s,avg_a4s,avg_a5s])
Z = np.sqrt(np.sqrt(np.array([avg_b5v,avg_b4v,avg_b3v,avg_b2v,avg_b1v,avg_a1v,avg_a2v,avg_a3v,avg_a4v,avg_a5v])))
```

```
print('Average limit order book depth = ',np.mean(Y))
print('Standard Deviation limit order book depth = ',np.mean(Z))

title = 'typical limit order book'
plt.figure()
#lines = plt.plot(lob['Time'], spreads/100)
#plt.setp(lines, 'color', 'r', 'linewidth', 0.1)
plt.bar(X,Y, color=['r','r','r','r','r','b','b','b','b','b'])
plt.xlabel('price (ticks)', fontsize=10)
plt.ylabel('volume (shares)', fontsize=10)
plt.title(title)
plt.show()
```

B Python Code for OFI Model

B.1 Procedure to obtain the empirical parameters from the data

```python
# this function will return the feature and label vectors.
# in order to determine the linear price impact coefficient c
# it takes as input:
#    - the limit order book lob (previously built)
#    - the buy/sell limit orders limit_{b/s} (previously built)
#    - the buy/sell cancelled orders cancelled_{b/s} (previously built)
#    - the buy/sell market orders market_{b/s} (previously built)
#    - the desired time period D (in seconds)
# it returns as output:
#    - the samples S_X, S_y of all (y = price changes) and (X = OFI's) from start to end for 30 min
#    - the average depth (AVD) from start to end for 30 min
def get_features_labels(lob, limit_b, limit_s, cancelled_b, cancelled_s, market_b, market_s, D):
    S_X = []
    S_y = []
    AVD = []
    start = lob['Time'].iloc[0]
    end = lob['Time'].iloc[-1]
    sample = 0
    while(start < end):

        X = []
        y = []

        cur_end = start + 1800

        # get the average depth during this period
        AD = 0
        lob_cur = lob[(lob['Time'] >= start) & (lob['Time'] <= cur_end)]
        for i in range(5):
            AD = AD + lob_cur['Ask_Size_'+str(i+1)].mean()
            AD = AD + lob_cur['Bid_Size_'+str(i+1)].mean()
        AD = AD / 10

        # get the other samples during this period
        while(start<cur_end):

            # a- get the corresponding OFI(t,t+D)/D(t)
            # how? we just count the parameters L,C,M
            # and we "estimate" the average depth of a level i.e. number of shares by level in the lob

            # here we compute the parameters L,C,M
            # L:
            limit_cur_1 = limit_b[(limit_b[0] >= start) & (limit_b[0] <= start + D)]
            limit_cur_2 = limit_s[(limit_s[0] >= start) & (limit_s[0] <= start + D)]
            L_b = 0 # number of buy limit orders
            L_s = 0 # number of sell limit orders
            for idx,row in limit_cur_1.iterrows():
                L_b = L_b + row[3]
            for idx,row in limit_cur_2.iterrows():
                L_s = L_s + row[3]
            # C:
            cancelled_cur_1 = cancelled_b[(cancelled_b[0] >= start) & (cancelled_b[0] <= start + D)]
            cancelled_cur_2 = cancelled_s[(cancelled_s[0] >= start) & (cancelled_s[0] <= start + D)]
            C_b = 0 # number of buy cancelled orders
            C_s = 0 # number of sell cancelled orders
            for idx,row in cancelled_cur_1.iterrows():
                C_b = C_b + row[3]
            for idx,row in cancelled_cur_2.iterrows():
                C_s = C_s + row[3]
            # M:
            market_cur_1 = market_b[(market_b[0] >= start) & (market_b[0] <= start + D)]
            market_cur_2 = market_s[(market_s[0] >= start) & (market_s[0] <= start + D)]
            M_b = 0 # number of buy market orders
            M_s = 0 # number of sell market orders
            for idx,row in market_cur_1.iterrows():
                M_b = M_b + row[3]
            for idx,row in market_cur_2.iterrows():
                M_s = M_s + row[3]
            # here we check that the lob is non-empty during the time period
            lob1 = lob[(lob['Time'] >= start) & (lob['Time'] <= start + 1)]
            lob2 = lob[(lob['Time'] >= start + D) & (lob['Time'] <= start + D + 1)]
            if(lob1.size < 2 or lob2.size < 2):
                start = start + D
                continue


            # b- get the corresponding label (S(t+D)-S(t))/d
            Delta_S = (0.5*(lob2['Ask_Price_1'].iloc[0] + lob2['Bid_Price_1'].iloc[0])\
                    -0.5*(lob1['Ask_Price_1'].iloc[0] + lob1['Bid_Price_1'].iloc[0]))/100

            # here we filter some bad examples
            OFI = 0.5*(L_b-C_b-M_b) - 0.5*(L_s-C_s-M_s)
            if(Delta_S < min(-0.01*OFI,0.01*OFI) or Delta_S > max(-0.01*OFI,0.01*OFI) or OFI < -1500 or OFI > 1500):
                start = start + D
                continue

            # here we append our examples to our feature vector
            X.append(OFI)
            y.append(Delta_S)

            # next time period
            start = start + D

        X = np.array(X)
        X = X.reshape((X.shape[0],1))
        y = np.array(y)
        y = y.reshape((y.shape[0],1))

        if(X.size>0 and y.size>0):
            S_X.append(X)
            S_y.append(y)
            AVD.append(AD)

        # next sample
        print('Done with adding sample ',sample,' with size ',X.size,' = ',y.size)
        sample = sample+1
        start = cur_end

    S_X = np.array(S_X)
    S_y = np.array(S_y)
    AVD = np.array(AVD)

    return S_X, S_y, AVD
```

B.2 Procedure to perform OLS and test residuals

```
# this function performs OLS on X,y.
# takes also as a parameter D (time diff), and to_plot (if we desire plots on the OLS)
def OLS(X,y,D,to_plot):

    # 1- OLS
    from sklearn import datasets, linear_model
    from sklearn.metrics import mean_squared_error, r2_score
    from sklearn.model_selection import KFold

    coefs = []
    rmses = []
    r2s = []
    rmses_train = []
    r2s_train = []
    kf = KFold(n_splits=4, random_state=None, shuffle=True)
    kf.get_n_splits(X)
    for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        regr = linear_model.LinearRegression()
        regr.fit(X_train, y_train)
        y_pred = regr.predict(X_test)
        coefs.append(regr.coef_)
        rmses.append(mean_squared_error(y_test, y_pred))
        r2s.append(r2_score(y_test, y_pred))
        y_pred_2 = regr.predict(X_train)
        rmses_train.append(mean_squared_error(y_train, y_pred_2))
        r2s_train.append(r2_score(y_train, y_pred_2))
    coefs = np.array(coefs)
    rmses = np.array(rmses)
    r2s = np.array(r2s)
    rmses_train = np.array(rmses_train)
    r2s_train = np.array(r2s_train)

    print('Coefficients: \n', np.mean(coefs))
    print("Mean squared error: %.2f"
              % np.mean(rmses))
    print('r^2: %.2f' % np.mean(r2s))
    print("(training) Mean squared error: %.2f"
              % np.mean(rmses_train))
    print('(training) r^2: %.2f' % np.mean(r2s_train))

    if(to_plot == 1):
        # 2- qq-plot of y_test - y_pred against normal distribution
        import pylab
        import scipy.stats as stats
        measurements = y_test[:,0] - y_pred[:,0]
        stats.probplot(measurements, dist="norm", plot=pylab)
        pylab.title('OLS - qq-plot against normal (' + str(D) + ' sec)')
        pylab.show()

        # 3- qq-plot of y_test - y_pred against uniform distribution
        measurements = y_test[:,0] - y_pred[:,0]
        stats.probplot(measurements, dist="uniform", plot=pylab)
        pylab.title('OLS - qq-plot against uniform(' + str(D) + ' sec)')
        pylab.show()

        # 4- post-OLS plot
        s = [5 for n in range(len(X))]
        title = 'OLS - order flow imbalance VS price change (' + str(D) + ' sec)'
        plt.figure()
        plt.scatter(X,y,s=s)
        lines = plt.plot(X_test, y_pred)
        plt.setp(lines, 'color', 'r', 'linewidth', 1.0)
        plt.xlabel(r'$OFI(t,t+\Delta)$', fontsize=10)
        plt.ylabel(r'$\frac{S(t+\Delta)-S(t)}{\delta}$', fontsize=10)
        plt.axhline(0, linewidth = 0.5, color='grey')
        plt.axvline(0, linewidth = 0.5, color='grey')
        plt.xlim((-500,500))
        plt.ylim((-4,4))

        plt.title(title)
        plt.show()

    return np.mean(coefs),np.mean(rmses),np.mean(r2s), np.mean(rmses_train),np.mean(r2s_train)
```

B.3 Procedure to compute the OLS relevant values

```
# here we get the average OLS info accross all the samples - D = 1 sec
sample = 0
betas = []
rmses = []
r2s = []
rmses_train = []
r2s_train = []
while(sample < len(S_X)):
    print('sample #',sample)
    beta,rmse,r2, rmse_train, r2_train = OLS(S_X[sample],S_y[sample],1,0)
    print()
    betas.append(beta)
    rmses.append(rmse)
    r2s.append(r2)
    rmses_train.append(rmse_train)
    r2s_train.append(r2_train)
    sample = sample + 1
betas = np.array(betas)
rmses = np.array(rmses)
r2s = np.array(r2s)
cs = betas*73
print('Average regression coefficient: beta_avg = ',np.mean(betas))
print('min regression coefficient: beta_min = ',np.min(betas))
print('max regression coefficient: beta_max = ',np.max(betas))
print('Average depth-adjusted regression coefficient: c_avg',np.mean(cs))
print('min depth-adjusted regression coefficient: c_min',np.min(cs))
print('max depth-adjusted regression coefficient: c_max',np.max(cs))
print('Average RMSE: RMSE_avg = ',np.mean(rmses))
print('min RMSE: RMSE_min = ',np.min(rmses))
print('max RMSE: RMSE_max = ',np.max(rmses))
print('Average r^2: r2_avg = ',np.mean(r2s))
print('min r^2: r2_min = ',np.min(r2s))
print('max r^2: r2_max = ',np.max(r2s))
print('(train) Average RMSE: RMSE_avg = ',np.mean(rmses_train))
print('(train) min RMSE: RMSE_min = ',np.min(rmses_train))
print('(train) max RMSE: RMSE_max = ',np.max(rmses_train))
print('(train) Average r^2: r2_avg = ',np.mean(r2s_train))
print('(train) min r^2: r2_min = ',np.min(r2s_train))
print('(train) max r^2: r2_max = ',np.max(r2s_train))
```

B.4 Procedure to perform polynomial OLS

```
# this function performs polynomial OLS on X,y with degree deg
# takes also as a parameter D (time diff), and to_plot (if we desire plots on the OLS)
def OLS_poly(X,y,deg,D,to_plot):

    # 1- augment X to deg
    X_poly = np.zeros((X.shape[0],deg+1))
    for i in range(len(X)):
        for j in range(deg+1):
            X_poly[i,j] = X[i]**j

    # 2- poly-OLS
    from sklearn import datasets, linear_model
    from sklearn.metrics import mean_squared_error, r2_score
    from sklearn.model_selection import KFold
    coefs = []
    rmses = []
    r2s = []
    rmses_train = []
    r2s_train = []
    kf = KFold(n_splits=4, random_state=None, shuffle=True)
    kf.get_n_splits(X_poly)
    for train_index, test_index in kf.split(X_poly):
        X_train, X_test, X_test_plot = X_poly[train_index], X_poly[test_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        regr = linear_model.LinearRegression()
        regr.fit(X_train, y_train)
        y_pred = regr.predict(X_test)
        coefs.append(regr.coef_)
        rmses.append(mean_squared_error(y_test, y_pred))
        r2s.append(r2_score(y_test, y_pred))
        y_pred_2 = regr.predict(X_train)
        rmses_train.append(mean_squared_error(y_train, y_pred_2))
        r2s_train.append(r2_score(y_train, y_pred_2))
    coefs = np.array(coefs)
    rmses = np.array(rmses)
    r2s = np.array(r2s)
    rmses_train = np.array(rmses_train)
    r2s_train = np.array(r2s_train)

    print('Coefficients: \n', np.mean(coefs))
    print("Mean squared error: %.2f"
          % np.mean(rmses))
    print('r^2: %.2f' % np.mean(r2s))
    print("(training) Mean squared error: %.2f"
          % np.mean(rmses_train))
    print('(training) r^2: %.2f' % np.mean(r2s_train))

    if(to_plot == 1):
        # 2- qq-plot of y_test - y_pred against normal distribution
        import pylab
        import scipy.stats as stats
        measurements = y_test[:,0] - y_pred[:,0]
        stats.probplot(measurements, dist="norm", plot=pylab)
        pylab.title('Poly - qq-plot against normal (' + str(D) + ' sec)')
        pylab.show()

        # 3- qq-plot of y_test - y_pred against uniform distribution
        measurements = y_test[:,0] - y_pred[:,0]
        stats.probplot(measurements, dist="uniform", plot=pylab)
        pylab.title('Poly - qq-plot against uniform (' + str(D) + ' sec)')
        pylab.show()

        # 4- post-OLS_poly plot
        s = [5 for n in range(len(X))]
        X_plot = [x for x,_ in sorted(zip(X_test_plot,y_pred))]
        y_plot = [y for _,y in sorted(zip(X_test_plot,y_pred))]
        title = 'Polynomial OLS - order flow imbalance VS price change (' + str(D) + ' sec)'
        plt.figure()
        plt.scatter(X,y,s=s)
        plt.plot(X_plot, y_plot, color='red')
        plt.xlabel(r'$OFI(t,t+\Delta)$', fontsize=10)
        plt.ylabel(r'$\frac{S(t+\Delta)-S(t)}{\delta}$', fontsize=10)
        plt.axhline(0, linewidth = 0.5, color='grey')
        plt.axvline(0, linewidth = 0.5, color='grey')
        plt.xlim((-500,500))
        plt.ylim((-4,4))
        plt.title(title)
        plt.show()

    return np.mean(coefs),np.mean(rmses),np.mean(r2s), np.mean(rmses_train),np.mean(r2s_train)
```

B.5 Procedure to choose the optimal hyperparam d for poly-OLS

```
# here we get the average OLS info accross all the samples - D = 1 sec
degs = [1,2,3,4,5,6,7]
rmses = []
r2s = []
rmses_train = []
r2s_train = []
for deg in degs:
    sample = 0
    c_avg = 0
    rmse_avg = 0
    r2_avg = 0
    rmse_train_avg = 0
    r2_train_avg = 0
    while(sample < S_X.size):
        c,rmse,r2,rmse_train,r2_train = OLS_poly(S_X[sample],S_y[sample],deg,1,0)
        c_avg = c_avg + c
        rmse_avg = rmse_avg + rmse
        r2_avg = r2_avg + r2
        rmse_train_avg = rmse_train_avg + rmse_train
        r2_train_avg = r2_train_avg + r2_train
        sample = sample + 1
    c_avg = c_avg / sample
    rmse_avg = rmse_avg / sample
    r2_avg = r2_avg / sample
    rmse_train_avg = rmse_train_avg / sample
    r2_train_avg = r2_train_avg / sample
    rmses.append(rmse_avg)
    r2s.append(r2_avg)
    rmses_train.append(rmse_train_avg)
    r2s_train.append(r2_train_avg)

title = 'Train - change of RMSE with d (' + str(1) + ' sec)'
plt.figure()
lines = plt.plot(degs, rmses_train)
plt.setp(lines, 'color', 'b', 'linewidth', 1.0)
plt.xlabel('d', fontsize=10)
plt.ylabel('RMSE', fontsize=10)
plt.title(title)
plt.show()

title = 'Train - change of ' + r'$r^2$'+ ' with d (' + str(1) + ' sec)'
plt.figure()
lines = plt.plot(degs, r2s_train)
plt.setp(lines, 'color', 'g', 'linewidth', 1.0)
plt.xlabel('d', fontsize=10)
plt.ylabel(r'$r^2$', fontsize=10)
plt.title(title)
plt.show()

title = 'Test - change of RMSE with d (' + str(1) + ' sec)'
plt.figure()
lines = plt.plot(degs, rmses)
plt.setp(lines, 'color', 'b', 'linewidth', 1.0)
plt.xlabel('d', fontsize=10)
plt.ylabel('RMSE', fontsize=10)
plt.title(title)
plt.show()

title = 'Test - change of ' + r'$r^2$'+ ' with d (' + str(1) + ' sec)'
plt.figure()
lines = plt.plot(degs, r2s)
plt.setp(lines, 'color', 'g', 'linewidth', 1.0)
plt.xlabel('d', fontsize=10)
plt.ylabel(r'$r^2$', fontsize=10)
plt.title(title)
plt.show()
```

B.6 Procedure to use OLS in order to estimate the β_i's

```
# this function performs OLS on X,y.
# takes also as a parameter buy_sell and D for the plots
def OLS(X,y,buy_sell,D,to_plot):
    # 1- OLS
    from sklearn import datasets, linear_model
    from sklearn.metrics import mean_squared_error, r2_score
    X_train = X[:int(3*len(X)/4)]
    X_test = X[-int(len(X)/4):]
    y_train = y[:int(3*len(X)/4)]
    y_test = y[-int(len(X)/4):]
    regr = linear_model.LinearRegression()
    regr.fit(X_train, y_train)
    y_pred = regr.predict(X_test)
    print('Coefficients: \n', regr.coef_)
    print("Mean squared error: %.2f"
          % mean_squared_error(y_test, y_pred))
    print('Variance score: %.2f' % r2_score(y_test, y_pred))

    # if to_plot is set to 1, plot the desired plots
    if(to_plot == 1):

        # 3- qq-plot of y_test - y_pred against normal distribution
        import pylab
        import scipy.stats as stats
        measurements = y_test[:,0] - y_pred[:,0]
        stats.probplot(measurements, dist="norm", plot=pylab)
        pylab.title('qq-plot against normal')
        pylab.show()

        # 4- qq-plot of y_test - y_pred against uniform distribution
        measurements = y_test[:,0] - y_pred[:,0]
        stats.probplot(measurements, dist="uniform", plot=pylab)
        pylab.title('qq-plot against uniform')
        pylab.show()

        # 5- post-OLS plot
        s = [5 for n in range(len(X))]
        title = 'OLS - ' + buy_sell + ' order flow imbalance VS price change (' + str(D) + ' sec)'
        plt.figure()
        plt.scatter(X,y,s=s)
        lines = plt.plot(X_test, y_pred)
        plt.setp(lines, 'color', 'r', 'linewidth', 1.0)
        plt.xlabel(r'$OFI(t,t+\Delta)$', fontsize=10)
        plt.ylabel(r'$\frac{S(t+D)-S(t)}{\delta}$', fontsize=10)
        plt.title(title)
        plt.show()

    return regr.coef_, mean_squared_error(y_test, y_pred)
```

B.7 Procedure to compute all the β_i's

```
# get the features and labels vectors for buy/sell for a sec
start = lob['Time'].iloc[0]
end = lob['Time'].iloc[-1]
X_buy_s = []
y_buy_s = []
beta_buy_s = []
AD_buy_s = []
rms_s = []
itr = 1
while(start<end):
    X_buy, y_buy, AD_buy = get_features_labels(lob, limit_b, cancelled_b, market_s, 0, 1, start, start+300)
    X_buy_s.append(X_buy)
    y_buy_s.append(y_buy)
    AD_buy_s.append(AD_buy)
    beta, rms = OLS(X_buy,y_buy,'buy',1,0)
    beta_buy_s.append(beta)
    rms_s.append(rms)
    print('done with iteration ',itr)
    print()
    itr = itr + 1
    start = start + 300
```

B.8 Procedure to choose the optimal hyperparam λ for ridge regression

```
# this function performs ridge regression on X,y.
# takes also as a parameter buy_sell and D for the plots
def RIDGE(X,y,buy_sell,D):

    # 1- RIDGE
    # we will try all a-prioris from 0.1 to 1 (as Rama-Cont), with variance 0.16^2 (--> lambda = 25)
    from sklearn import datasets, linear_model
    from sklearn.metrics import mean_squared_error, r2_score
    X_train = X[:int(len(X)/3)]
    X_test = X[-int(2*len(X)/3):]
    y_train = y[:int(len(X)/3)]
    y_test = y[-int(2*len(X)/3):]
    lambdas = [0.00001, 0.0001, 0.001,0.01,0.1,1, 10, 100, 1000, 10000, 100000]
    errs = []
    coeffs = []
    for _lambda in lambdas:
        regr = linear_model.Ridge(_lambda)
        regr.fit(X_train, y_train)
        y_pred = regr.predict(X_test)
        errs.append(mean_squared_error(y_test, y_pred))
        coeffs.append(regr.coef_)

    # 2- plot of the RIDGE validation error in terms of alpha
    s = [5 for n in range(len(X))]
    title = 'Ridge - ' + buy_sell + ' , ' + r'$\lambda$ VS MSE (' + str(D) + ' sec)'
    plt.figure()
    plt.loglog(lambdas, errs)
    plt.xlabel(r'$\lambda$', fontsize=10)
    plt.ylabel(r'$\log(MSE)$', fontsize=10)
    plt.title(title)
    plt.show()

    # 3- post-RIDGE on best performing coeff
    idx = np.argmin(errs)
    regr = linear_model.Ridge(lambdas[idx])
    regr.fit(X_train, y_train)
    print('Coefficients: \n', regr.coef_)
    print("Mean squared error: %.2f"
          % mean_squared_error(y_test, y_pred))
    print('Variance score: %.2f' % r2_score(y_test, y_pred))

    # 4- qq-plot of y_test - y_pred against normal distribution
    import pylab
    import scipy.stats as stats
    measurements = y_test[:,0] - y_pred[:,0]
    stats.probplot(measurements, dist="norm", plot=pylab)
    pylab.title('qq-plot against normal')
    pylab.show()

    # 5- qq-plot of y_test - y_pred against uniform distribution
    measurements = y_test[:,0] - y_pred[:,0]
    stats.probplot(measurements, dist="uniform", plot=pylab)
    pylab.title('qq-plot against uniform')
    pylab.show()

    # 6- post-RIDGE plot on best performing coeff
    y_pred = regr.predict(X_test)
    s = [5 for n in range(len(X))]
    title = 'Ridge - ' + buy_sell + ' order flow imbalance VS price change (' + str(D) + ' sec)'
    plt.figure()
    plt.scatter(X,y,s=s)
    lines = plt.plot(X_test, y_pred)
    plt.setp(lines, 'color', 'r', 'linewidth', 1.0)
    plt.xlabel(r'$\frac{OFI(t,t+\Delta)}{D(t)}$', fontsize=10)
    plt.ylabel(r'$\frac{S(t+D)-S(t)}{\delta}$', fontsize=10)
    plt.title(title)
    plt.show()
```

B.9 Procedure to use OLS in order to estimate $\lambda$

```
# this function performs OLS on X,y.
# takes also as a parameter buy_sell and D (inner time interval) for the plots
def OLS_2(AD,B,buy_sell,D):
    # 1- OLS
    from sklearn import datasets, linear_model
    from sklearn.metrics import mean_squared_error, r2_score
    X_train = np.log(AD[:int(len(AD)/2)])
    X_test = np.log(AD[-int(len(AD)/2):])
    y_train = np.log(B[:int(len(AD)/2)])
    y_test = np.log(B[-int(len(AD)/2):])
    regr = linear_model.LinearRegression()
    regr.fit(X_train, y_train)
    y_pred = regr.predict(X_test)
    print('Coefficients: \n', regr.coef_)
    print("Mean squared error: %.2f"
          % mean_squared_error(y_test, y_pred))
    print('Variance score: %.2f' % r2_score(y_test, y_pred))

    # 3- qq-plot of y_test - y_pred against normal distribution
    import pylab
    import scipy.stats as stats
    measurements = y_test[:,0] - y_pred[:,0]
    stats.probplot(measurements, dist="norm", plot=pylab)
    pylab.title('qq-plot against normal')
    pylab.show()

    # 4- qq-plot of y_test - y_pred against uniform distribution
    measurements = y_test[:,0] - y_pred[:,0]
    stats.probplot(measurements, dist="uniform", plot=pylab)
    pylab.title('qq-plot against uniform')
    pylab.show()

    # 5- post-OLS plot
    s = [5 for n in range(len(AD))]
    title = 'OLS - ' + buy_sell + r'$\log(AD_i)$' + ' VS ' r'$\log(\beta_i)$' + ' (' + str(D) + ' sec)'
    plt.figure()
    plt.scatter(np.log(AD),np.log(B),s=s)
    lines = plt.plot(X_test, y_pred)
    plt.setp(lines, 'color', 'r', 'linewidth', 1.0)
    plt.xlabel(r'$\log(AD_i)$', fontsize=10)
    plt.ylabel(r'$\log(\beta_i)$', fontsize=10)
    plt.title(title)
    plt.show()
```

C Python Code for the Order Events Model

C.1 Procedure to obtain the empirical parameters from the data (revised)

```python
# this function will return the feature and label vectors.
# in order to determine the linear price impact coefficient c
# it takes as input:
#     - the limit order book lob (previously built)
#     - the buy/sell limit orders limit_{b/s} (previously built)
#     - the buy/sell cancelled orders cancelled_{b/s} (previously built)
#     - the buy/sell market orders market_{b/s} (previously built)
#     - the desired time period D (in seconds)
# it returns as output:
#     - the samples S_X, S_y of all (y = price changes) and (X = OFI's) from start to end for 30 min
def get_features_labels(lob, limit_b, limit_s, cancelled_b, cancelled_s, market_b, market_s, D):
    S_X = []
    S_y = []
    start = lob['Time'].iloc[0]
    end = lob['Time'].iloc[-1]
    sample = 0
    while(start < end):

        X = []
        y = []

        cur_end = start + 1800
        while(start<cur_end):

            x = []

            # a- get the corresponding OFI(t,t+D)
            # how? we just count the parameters L,C,M
            # and we "estimate" the average depth of a level i.e. number of shares by level in the lob

            # here we compute the parameters L,C,M
            # L:
            limit_cur_1 = limit_b[(limit_b[0] >= start) & (limit_b[0] <= start + D)]
            limit_cur_2 = limit_s[(limit_s[0] >= start) & (limit_s[0] <= start + D)]
            L_b = 0 # number of buy limit orders
            L_s = 0 # number of sell limit orders
            for idx,row in limit_cur_1.iterrows():
                L_b = L_b + row[3]
            for idx,row in limit_cur_2.iterrows():
                L_s = L_s + row[3]
            # C:
            cancelled_cur_1 = cancelled_b[(cancelled_b[0] >= start) & (cancelled_b[0] <= start + D)]
            cancelled_cur_2 = cancelled_s[(cancelled_s[0] >= start) & (cancelled_s[0] <= start + D)]
            C_b = 0 # number of buy cancelled orders
            C_s = 0 # number of sell cancelled orders
            for idx,row in cancelled_cur_1.iterrows():
                C_b = C_b + row[3]
            for idx,row in cancelled_cur_2.iterrows():
                C_s = C_s + row[3]
            # M:
            market_cur_1 = market_b[(market_b[0] >= start) & (market_b[0] <= start + D)]
            market_cur_2 = market_s[(market_s[0] >= start) & (market_s[0] <= start + D)]
            M_b = 0 # number of buy market orders
            M_s = 0 # number of sell market orders
            for idx,row in market_cur_1.iterrows():
                M_b = M_b + row[3]
            for idx,row in market_cur_2.iterrows():
                M_s = M_s + row[3]
            # here we check that the lob is non-empty during the time period
            lob1 = lob[(lob['Time'] >= start) & (lob['Time'] <= start + 1)]
            lob2 = lob[(lob['Time'] >= start + D) & (lob['Time'] <= start + D + 1)]
            if(lob1.size < 2 or lob2.size < 2):
                start = start + D
                continue


            # b- get the corresponding label (S(t+D)-S(t))/d
            Delta_S = (0.5*(lob2['Ask_Price_1'].iloc[0] + lob2['Bid_Price_1'].iloc[0])\
                      -0.5*(lob1['Ask_Price_1'].iloc[0] + lob1['Bid_Price_1'].iloc[0]))/100

            # here we filter some bad examples
            OFI = 0.5*(L_b-C_b-M_b) - 0.5*(L_s-C_s-M_s)
            if(Delta_S < min(-0.01*OFI,0.01*OFI) or Delta_S > max(-0.01*OFI,0.01*OFI) or OFI < -1500 or OFI > 1500):
                start = start + D
                continue


            # here we append all the our feature matrix
            x.append(OFI) # OFI
            x.append(L_b+C_b+M_s+L_s+C_s+M_b) # TF (Total Flow)
            x.append(M_b - M_s) # TI (Trade Flow Imbalance)
            x.append(M_b + M_s) # VOL (Trade volume)

            # here we append our examples to our set of samples
            X.append(x)
            y.append(Delta_S)

            # b- get the corresponding label (S(t+D)-S(t))/d
            Delta_S = (0.5*(lob2['Ask_Price_1'].iloc[0] + lob2['Bid_Price_1'].iloc[0])\
                      -0.5*(lob1['Ask_Price_1'].iloc[0] + lob1['Bid_Price_1'].iloc[0]))/100

            # c- next time period
            start = start + D
        X = np.array(X)
        y = np.array(y)
        y = y.reshape((y.shape[0],1))
        # shuffle them similarly
        from sklearn.utils import shuffle
        X, y = shuffle(X, y, random_state=0)

        if(X.size>0 and y.size>0):
            S_X.append(X)
            S_y.append(y)

        # next sample
        print('Done with adding sample ',sample,' with size ',X.size,' = ',y.size)
        sample = sample+1
        start = cur_end

    S_X = np.array(S_X)
    S_y = np.array(S_y)

    return S_X, S_y
```

C.2 Procedure to get perform Regression Decision Tree

```
# this function performs DT on X,y.
# takes also as a parameter D, and sample number and to_viz for the graph visualisation
def DT_reg(X,y,D,sample,to_viz):

    # 1- DT (regression)
    from sklearn import datasets, tree
    from sklearn.metrics import mean_squared_error, r2_score
    from sklearn.model_selection import KFold

    rmses = []
    r2s = []
    kf = KFold(n_splits=4)
    kf.get_n_splits(X)
    KFold(n_splits=2, random_state=None, shuffle=True)
    for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        regr = tree.DecisionTreeRegressor(max_depth = 4, min_samples_split = 100)
        regr.fit(X_train, y_train)
        y_pred = regr.predict(X_test)
        rmses.append(mean_squared_error(y_test, y_pred))
        r2s.append(r2_score(y_test, y_pred))
    rmses = np.array(rmses)
    r2s = np.array(r2s)

    print("Mean squared error: %.2f"
              % np.mean(rmses))
    print('r^2: %.2f' % np.mean(r2s))

    # 2- draw the tree
    if(to_viz):
        import io
        import pydot
        feature_names = ['OFI','TF','TI','VOL','1/ADB','1/ADS']
        feature_names = ['OFI','TF','TI','VOL']
        out = io.StringIO()
        tree.export_graphviz(
            regr, out_file=out, feature_names=feature_names)
        graph = pydot.graph_from_dot_data(out.getvalue())
        pydot.graph_from_dot_data(out.getvalue())[0].write_pdf("tree_"+str(D)+'s_'+str(sample)+".pdf")

    return np.mean(rmses,axis=0),np.mean(r2s,axis=0)
```

D Python Code for the Generalized Model

D.1 Procedure to obtain the empirical parameters from the data (revised - again)

```
# this function will return the feature and label vectors.
# in order to determine the linear price impact coefficient c
# it takes as input:
#    - the limit order book lob (previously built)
#    - the buy/sell limit orders limit_{b/s} (previously built)
#    - the buy/sell cancelled orders cancelled_{b/s} (previously built)
#    - the buy/sell market orders market_{b/s} (previously built)
#    - the desired time period D (in seconds)
# it returns as output:
#    - the samples S_X, S_y of all (y = price changes) and (X = OFI's) from start to end for 30 min
def get_features_labels(lob, limit_b, limit_s, cancelled_b, cancelled_s, market_b, market_s, D):
    import math
    S_X = []
    S_y = []
    start = lob['Time'].iloc[0]
    end = lob['Time'].iloc[-1]
    sample = 0
    while(start < end):

        X = []
        y = []

        cur_end = start + 1800
        while(start<cur_end):

            x = []

            # a- get the corresponding relevant order book events
            # how? we just count the parameters L,C,M
            # and we "estimate" the average depth of a level i.e. number of shares by level in the lob

            # here we compute the parameters L,C,M
            # L:
            limit_cur_1 = limit_b[(limit_b[0] >= start) & (limit_b[0] <= start + D)]
            limit_cur_2 = limit_s[(limit_s[0] >= start) & (limit_s[0] <= start + D)]
            L_b = 0 # number of buy limit orders
            L_s = 0 # number of sell limit orders
            for idx,row in limit_cur_1.iterrows():
                L_b = L_b + row[3]
            for idx,row in limit_cur_2.iterrows():
                L_s = L_s + row[3]
            # C:
            cancelled_cur_1 = cancelled_b[(cancelled_b[0] >= start) & (cancelled_b[0] <= start + D)]
            cancelled_cur_2 = cancelled_s[(cancelled_s[0] >= start) & (cancelled_s[0] <= start + D)]
            C_b = 0 # number of buy cancelled orders
            C_s = 0 # number of sell cancelled orders
            for idx,row in cancelled_cur_1.iterrows():
                C_b = C_b + row[3]
            for idx,row in cancelled_cur_2.iterrows():
                C_s = C_s + row[3]
            # M:
            market_cur_1 = market_b[(market_b[0] >= start) & (market_b[0] <= start + D)]
            market_cur_2 = market_s[(market_s[0] >= start) & (market_s[0] <= start + D)]
            M_b = 0 # number of buy market orders
            M_s = 0 # number of sell market orders
            for idx,row in market_cur_1.iterrows():
                M_b = M_b + row[3]
            for idx,row in market_cur_2.iterrows():
                M_s = M_s + row[3]
            # here we check that the lob is non-empty during the time period
            lob1 = lob[(lob['Time'] >= start) & (lob['Time'] <= start + 1)]
            lob2 = lob[(lob['Time'] >= start + D) & (lob['Time'] <= start + D + 1)]
            if(lob1.size < 2 or lob2.size < 2):
                start = start + D
                continue


            # b- get the corresponding label (S(t+D)-S(t))/d
            Delta_S = (0.5*(lob2['Ask_Price_1'].iloc[0] + lob2['Bid_Price_1'].iloc[0])\
                      -0.5*(lob1['Ask_Price_1'].iloc[0] + lob1['Bid_Price_1'].iloc[0]))/100

            # c- compute main quantities of interest (order events)
            OFI = (L_b-C_b-M_b) - (L_s-C_s-M_s)
            TF = L_b+C_b+M_b+L_s+C_s+M_s # total flow (total number of order book events)
            TI = M_b - M_s # trade flow imbalance
            VOL = M_b + M_s # volume
            pi = VOL/TF # ratio of trade over all order book events
            mu = 0
            if(len(market_cur_1) > 0):
                mu = mu + 0.5*market_cur_1[3].mean()
            if(len(market_cur_2) > 0):
                mu = mu + 0.5*market_cur_2[3].mean() # average trade size
            # d- other quantities of interest (price related)
            SIGN = 0 # previous trade sign
            if(len(y) > 1):
                if(y[-1] > y[-2]):
                    SIGN = 1
                elif(y[-1] < y[-2]):
                    SIGN = -1
            SIGN_RAT = 0 # ratio of previous trade signs
            if(len(X) > 1):
                SIGN_RAT = np.mean(np.array(X)[:,6])
            VAL = 0 # value of previous trade
            if(len(y) > 1):
                VAL = y[-1] - y[-2]
            VAL_RAT = 0 # ratio of previous trade values
            if(len(X) > 1):
                VAL_RAT = np.mean(np.array(X)[:,8])
            IS = lob1['Ask_Price_1'].iloc[0] - lob1['Bid_Price_1'].iloc[0] # initial spread


            # here we filter some bad examples
            OFI = 0.5*(L_b-C_b-M_b) - 0.5*(L_s-C_s-M_s)
            if(Delta_S < min(-0.01*OFI,0.01*OFI) or Delta_S > max(-0.01*OFI,0.01*OFI) or OFI < -1500 or OFI > 1500):
                start = start + D
                continue
            """
            rat = np.sqrt(VOL)/(sq_pi*sq_mu)
            if( (not math.isnan(rat)) and
               (Delta_S < min(-0.01*rat,0.01*rat) or Delta_S > max(-0.01*rat,0.01*rat) or rat < -1500 or rat > 1500)):
                start = start + D
                continue
            """

            # here we append all the our feature matrix
            x.append(OFI) # OFI
            x.append(TF) # TF (Total Flow)
            x.append(TI) # TI (Trade Flow Imbalance)
            x.append(VOL) # VOL (Trade volume)

            x.append(pi)
            x.append(mu)

            x.append(SIGN)
            x.append(SIGN_RAT)
            x.append(VAL)
            x.append(VAL_RAT)
            x.append(IS)

            # here we append our examples to our set of samples
            X.append(x)
            y.append(Delta_S)

            # b- get the corresponding label (S(t+D)-S(t))/d
            Delta_S = (0.5*(lob2['Ask_Price_1'].iloc[0] + lob2['Bid_Price_1'].iloc[0])\
                     -0.5*(lob1['Ask_Price_1'].iloc[0] + lob1['Bid_Price_1'].iloc[0]))/100

            # c- next time period
            start = start + D

        X = np.array(X)
        y = np.array(y)
        y = y.reshape((y.shape[0],1))
        # shuffle them similarly
        from sklearn.utils import shuffle
        X, y = shuffle(X, y, random_state=0)

        if(X.size>0 and y.size>0):
            S_X.append(X)
            S_y.append(y)

        # next sample
        print('Done with adding sample ',sample,' with size ',X.size,' = ',y.size)
        sample = sample+1
        start = cur_end

    S_X = np.array(S_X)
    S_y = np.array(S_y)

    return S_X, S_y
```

D.2 Procedure to perform PCA

```
###########################################
# Functions to compute PCA and projection
###########################################
def my_pca(X, k):
    '''
    compute PCA components
    X = data matrix (each row as a sample)
    k = chosen number of principal components
    '''
    n, d = X.shape
    assert(d>=k)
    _, _, Vh = np.linalg.svd(X)
    V = Vh.T
    return V[:, :k]

def pca_proj(X, k):
    '''
    compute projection of matrix X
    along its first k principal components
    '''
    P = my_pca(X, k)
    return X.dot(P)

####################################################
# Evaluates prediction perf using the reduced input
####################################################
def pca_proj_accuracy(X, y, k, method='linreg'):
    '''
    Fitting a k dimensional feature set obtained
    from PCA projection of X, versus y
    for regression
    '''

    # test-train split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)

    # pca projection
    P = my_pca(X_train, k)
    P = P.dot(P.T)
    pca_proj_X = X_train.dot(P)

    if method == 'linreg':
        # fit a linear model
        line = sklearn.linear_model.LinearRegression(fit_intercept=False)
        line.fit(pca_proj_X, y_train)

         # predict y
        y_pred=line.predict(X_test.dot(P))

    n_test = y_test.shape[0]
    mse = (1/n_test) * sum( (y_pred-y_test)**2 )


    # return the test error (MSE)
    return mse
```

D.3 Procedure to perform Dimensionality Reduction

```
#Cross-validation for PCA using linear regression

d = X.shape[1]
test_error = np.zeros(d)

y = np.array(y_as_list)

for k in range(1,d+1):

    #PCA projection
    test_error[k-1] = pca_proj_accuracy(X, y, k)

x = np.arange(1, d+1, 1)

fig1 = plt.figure()
fig1.suptitle('test error of PCA projection', fontsize=15)
plt.xlabel('dimension of projection')
plt.ylabel('test error')
plt.plot(x, test_error)
```

D.4 Procedure to perform Polynomial LASSO

```
print ('\nFiting lasso regression on the data')

poly_false = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)
X_scaled = preprocessing.scale(X)
X_poly = poly_false.fit_transform(X_scaled)
X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.3, random_state=0)

#Creating grid for search
grid = np.arange(0.001,0.02,0.001)
cross_val = np.zeros((grid.shape[0],2))

for index,l in enumerate(grid):

    lasso_reg = linear_model.Lasso(alpha=l, max_iter = 1000000 )
    scores = cross_val_score(lasso_reg, X_train, y_train, cv=5)
    avg_score = sum(scores)/len(scores)
    cross_val[index,0] = l
    cross_val[index,1] += avg_score

best_index = np.argmax(cross_val[:,1])
plt.plot(cross_val[:,0],cross_val[:,1])
plt.xlabel('alpha')
plt.ylabel('cross validation error')
plt.title('cross validation vs alpha')
plt.show()

print ('\n best alpha is :', cross_val[best_index,0], 'with average cross validation score : ',cross_val[best_index,1])
# Out of sample error
best_model = linear_model.Lasso(alpha= cross_val[best_index,0])
best_model.fit(X_train,y_train)


R_out = best_model.score(X_test,y_test)
print ('\nOut of sample R**2 for lasso regression with alpha = ',  cross_val[best_index,0], 'is: ', R_out)
#print (best_model.coef_)
zero_weight_features = pd.DataFrame(X_poly).columns[best_model.coef_==0]
print ('The features that were set to zero are :', list(zero_weight_features))
```

D.5 Procedure to perform Polynomial ELASTIC NET

```
print ('\nFiting Elastic Net regression on the data')

poly_false = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)
X_scaled = preprocessing.scale(X)
X_poly = poly_false.fit_transform(X_scaled)
X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.3, random_state=0)


l1_ratio_grid = np.arange(0.1,1.1,0.1)
a_grid = np.arange(0.01,0.2,0.01)
cross_score = np.zeros((l1_ratio_grid.shape[0],a_grid.shape[0]))

for index_ratio,l1 in enumerate(l1_ratio_grid):
    print(l1)

    for index_a,a in enumerate(a_grid):
        print (a)
        net = linear_model.ElasticNet(alpha=a, l1_ratio=l1, max_iter = 1000000)
        scores = cross_val_score(net, X_train, y_train, cv=5)
        avg_score = sum(scores)/len(scores)
        cross_score[index_ratio,index_a] = avg_score


ind = np.unravel_index(np.argmax(cross_score, axis=None), cross_score.shape)
l1_best = l1_ratio_grid[ind[0]]
a_best = a_grid[ind[1]]
print ('best combination of grid is (l1, a)', l1_best, a_best)

best_model = linear_model.ElasticNet(alpha= a_best, l1_ratio = l1_best)
best_model.fit(X_train,y_train)


R_out = best_model.score(X_test,y_test)
print (R_out)
```

D.6 Procedure to perform Random Forest

```
print ('\nFiting  Random Forest on the data')


X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=0)
performance = []

max_depth_grid = np.arange(3,6,1)
feat_grid = np.arange(1,X.shape[1])
n_est_grid = [200,300,400,500]

for n_est in n_est_grid:
    for index_1,m_depth in enumerate(max_depth_grid):
        print (m_depth)

        for index_2,feat in enumerate(feat_grid):

            forest = sk.ensemble.RandomForestRegressor(n_estimators=n_est , max_depth=m_depth, min_samples_split=10, min_sa
            scores = cross_val_score(forest, X_train, y_train, cv=2)
            avg_score = sum(scores)/len(scores)
            performance.append([n_est,m_depth,feat,avg_score])

performance = pd.DataFrame(performance)
index = performance.iloc[performance[3].argmax()]

best_model_rf = sk.ensemble.RandomForestRegressor(n_estimators=int(index[0]) , max_depth=int(index[1]), min_samples_split=1

best_model_rf.fit(X_train,y_train)

score = best_model_rf.score(X_test,y_test)
print ('best combination of hyperparameters for random forest is n_estimators, max_depth, max_features', index[0],index[1],


## (best_model.feature_importances_,)
q = rankdata(best_model_rf.feature_importances_).astype('int')

imp = np.concatenate((np.expand_dims(X_train.columns,1),np.expand_dims(best_model_rf.feature_importances_,1)),axis=1)
imp = np.concatenate((imp,np.expand_dims(q,1)),axis=1)
imp =  pd.DataFrame(imp)
imp.columns = ['Feature','Feature weight', 'Feature rank']
imp = imp.sort_values(by='Feature weight', axis=0, ascending=False, inplace=False, kind='quicksort', na_position='last')
y_fig = imp[imp['Feature rank']>-1]['Feature weight']
x_fig = np.arange(y_fig.shape[0])

plt.figure(num=None, figsize=(15, 6), dpi=80, facecolor='w', edgecolor='k')
my_xticks = imp['Feature']
plt.xticks(x_fig, my_xticks)
plt.bar(x_fig, y_fig)
plt.title('Feature importances of the random forest')
plt.show()
```

D.7 Procedure to perform Ada-Boost

```
#sklearn.ensemble.AdaBoostRegressor(base_estimator=None, n_estimators=, learning_rate=0.01, loss=?linear?)

max_depth_grid = np.arange(3,6,1)
n_estim_grid = np.arange(200,600,100)
cross_score = np.zeros((max_depth_grid.shape[0], n_estim_grid.shape[0]))
lr_grid = np.arange(0.01,0.111,0.02)
performance = []
for lr in lr_grid:
    for index_1,m_depth in enumerate(max_depth_grid):
        print ("m_depth: ", m_depth)

        for index_2,n_estim in enumerate(n_estim_grid):

            tree = sk.tree.DecisionTreeRegressor(max_depth=m_depth)
            boost = sk.ensemble.AdaBoostRegressor(base_estimator=tree, n_estimators=n_estim, learning_rate=lr, loss='linear
            scores = cross_val_score(forest, X_train, y_train, cv=3)
            avg_score = sum(scores)/len(scores)
            performance.append([lr,m_depth,n_est,avg_score])



performance = pd.DataFrame(performance)
index = performance.iloc[performance[3].argmax()]
best_tree = sk.tree.DecisionTreeRegressor(max_depth=int(index[1]))
best_model_ada = sk.ensemble.AdaBoostRegressor(base_estimator=best_tree, n_estimators=int(index[2]), learning_rate=index[0]




best_model_ada.fit(X_train,y_train)

score = best_model_ada.score(X_test,y_test)
print ('best combination of hyperparameters for random forest is n_estimators, max_depth, max_features', index[0],index[1],


# (best_model.feature_importances_,)
q = rankdata(best_model_ada.feature_importances_).astype('int')

imp = np.concatenate((np.expand_dims(X_train.columns,1),np.expand_dims(best_model_ada.feature_importances_,1)),axis=1)
imp = np.concatenate((imp,np.expand_dims(q,1)),axis=1)
imp =  pd.DataFrame(imp)
imp.columns = ['Feature','Feature weight', 'Feature rank']
imp = imp.sort_values(by='Feature weight', axis=0, ascending=False, inplace=False, kind='quicksort', na_position='last')
y_fig = imp[imp['Feature rank']>-1]['Feature weight']
x_fig = np.arange(y_fig.shape[0])

plt.figure(num=None, figsize=(15, 6), dpi=80, facecolor='w', edgecolor='k')
my_xticks = imp['Feature']
plt.xticks(x_fig, my_xticks)
plt.bar(x_fig, y_fig)
plt.title('Feature importances of the adaboost regressor')
plt.show()
```

D.8 Procedure to perform Neural Nets

```
def generate_model(No_hidden, No_units=200, activation_function = 'relu',dropout = 0):

    # Initialize the constructor
    model = Sequential()

    #model.add(Dropout(dropout))
    model.add(Dropout(dropout, input_shape=(X_train.shape[1],)))

    # Add an input layer
   # model.add(Dense(No_units, activation='relu', input_shape=(X_train.shape[1],)))
    model.add(Dense(No_units, activation='relu', ))

    for i in range(No_hidden):

        # Add one hidden layer
        model.add(Dense(No_units, activation='relu'))

        model.add(Dropout(dropout))


    # Add an output layer
    model.add(Dense(1, activation='linear'))

    return (model)


No_hidden_in = 2
performance = []

for No_units_in in range(100,200,50):

    print ('new number of units: ', No_units_in)

    for dropout in [0.2,0.3]:

        print ('new dropout')

        model = generate_model(No_hidden_in, No_units = No_units_in, dropout = dropout)


        model.compile(loss='mean_squared_error',
                      optimizer='adam',
                      metrics=['mse'])

        model.fit(X_train, y_train,epochs=100, batch_size=10, verbose=0)
        y_pred = model.predict(X_test)
        score = model.evaluate(X_test, y_test,verbose=1)
        print ("MSE score for No_un/layer: ", No_units_in, "and No hidden layers: ", No_hidden_in, "is :", score)
        performance.append([No_units_in,dropout,score])
```