Online Algorithms in High-frequency Trading

view issue

by Jacob Loveless, Sasha Stoikov, Rolf Waeber |
August 15, 2013

Topic: Development

Online Algorithms in High-frequency Trading

The challenges faced by competing HFT algorithms

JACOB LOVELESS, SASHA STOIKOV, AND ROLF WAEBER

HFT (high-frequency trading) has emerged as a powerful force in modern financial markets. Only 20 years ago, most of the trading volume occurred in exchanges such as the New York Stock Exchange, where humans dressed in brightly colored outfits would gesticulate and scream their trading intentions. Nowadays, trading occurs mostly in electronic servers in data centers, where computers communicate their trading intentions through network messages. This transition from physical exchanges to electronic platforms has been particularly profitable for HFT firms, which invested heavily in the infrastructure of this new environment.

Although the look of the venue and its participants has dramatically changed, the goal of all traders, whether electronic or human, remains the same: to buy an asset from one location/trader and sell it to another location/trader for a higher price. The defining difference between a human trader and an HFT is that the latter can react faster, more frequently, and has very short portfolio holding periods. A typical HFT algorithm operates at the sub-millisecond time scale, where human traders cannot compete, as the blink of a human eye takes approximately 300 milliseconds. As HFT algorithms compete with each other, they face two challenges:

• They receive large amounts of data every microsecond.

• They must be able to act extremely fast on the received data, as the profitability of the signals they are observing decays very quickly.

Online algorithms provide a natural class of algorithms suitable for HFT applications. In an online problem, new input variables are revealed sequentially. After each new input the algorithm needs to make a decision—for example, whether or not to submit a trade. This is in stark contrast to an offline problem, which assumes that the entire input data is available at the time of the decision making. Many practical optimization problems addressed in computer science and operations research applications are online problems.[1]

Besides solving an online problem, HFT algorithms also need to react extremely fast to market updates. To guarantee a fast reaction time, efficient memory handling is a necessity for a live trading algorithm. Keeping a large amount of data in memory will slow down any CPU, so it is important that an algorithm uses only a minimal amount of data and parameters, which can be stored in fast accessible memory such as the L1 cache. In addition, these factors should reflect the current state of the market and must be updated in real time when new data points are observed. In summary, the smaller the number of factors that need to be kept in memory and the simpler the computation required to update each factor, the faster an algorithm is able to react to market updates.

Based on the speed requirement and the online nature of HFT problems, the class of *one-pass algorithms* is especially suitable for HFT applications. These algorithms receive one data point at a time and use it to update a set of factors. After the update, the data point is discarded and only the updated factors are kept in memory.

Three problems can arise in HFT algorithms. The first is the estimation of a running mean of liquidity; this can be useful to an HFT in determining the size of an order that is likely to execute successfully on a particular electronic exchange. The second problem is a running volatility estimation, which can help quantify the short-term risk of a position. The third problem is a running linear regression, which can be used in trading pairs of related assets.

Each of these problems can be solved efficiently using an online one-pass algorithm. In this article we backtest the performance of one-pass algorithms on limit-order-book data for highly liquid ETFs (exchange-traded funds) and describe how to calibrate these algorithms in practice.

RELATED CONTENT

BARBARIANS AT THE GATEWAYS
High-frequency Trading and Exchange Technology
Jacob Loveless

THE ESSENCE OF SOFTWARE ENGINEERING: THE SEMAT KERNEL
A thinking framework in the form of an actionable kernel
Ivar Jacobson, Pan-Wei Ng, Paul McMahon, Ian Spence, Svante Lidman

MANAGING TECHNICAL DEBT
Shortcuts that save money and time today can cost you down the road.
Eric Allman

CODING GUIDELINES:FINDING THE ART IN THE SCIENCE
What separates good code from great code?
Robert Green, Henry Ledgard

BROWSE THIS TOPIC:

DEVELOPMENT

QUEUE ON SLASHDOT
- More Encryption Is Not the Solution
- The Antifragile Organization
- Realtime GPU Audio

QUEUE ON REDDIT
- Nonblocking Algorithms and Scalable Multicore Programming
- FPGA Programming for the Masses
- There's just no getting around it: You are Building a Distributed System

ONLINE ALGORITHMS IN HFT

The one advantage that HFT has over other market participants is reaction speed. HFT firms are able to see every action in the market—that is, the information contained in the limit order book—and react within microseconds. Though some HFT algorithms may base their actions on a source of information outside the market (say, by parsing news reports, measuring temperature, or gauging market sentiment), most base their decisions solely on the messages arriving at the market. By some estimates, there are approximately 215,000 quote updates per second on the New York Stock Exchange.[4] The challenge for HFTs is to process this data in a way that allows them to make decisions, such as when to enter positions or reduce risk. The examples used in this article assume that HFTs can observe every update in the best bid and ask prices, including the best bid and ask sizes. This subset of information contained in the limit order book is often referred to as the Level-I order book information.

The following three examples of online algorithms, each motivated with an application in HFT, are described in detail in this article:

• **Online mean algorithm.** Illustrated by constructing a factor that predicts the available liquidity, defined as the sum of the sizes at the best bid and the best ask, at a fixed horizon in the future. This quantity may be useful in estimating what order size is likely to execute at the best quotes at a given latency.

• **Online variance algorithm.** Illustrated by constructing a factor that predicts the realized volatility over a fixed horizon in the future. This quantity may be useful in estimating the short-term risk of holding inventory.

● **Online regression algorithm.** Illustrated by constructing a factor that predicts the expected PNL (profit and loss) of a long-short position in two related assets. This may be useful in constructing a signal indicating when a long-short position is likely to be profitable.

In all three cases, the algorithm has a single parameter, alpha, which controls the rate at which old information is forgotten. Figure 1 plots the raw liquidity measure (bid size plus ask size) in blue. Red and green represent the online liquidity factor, with alpha=0.9 and alpha=0.99, respectively. Note that as alpha approaches a value of 1, the signal gets smoother and efficiently tracks the trend in the underlying data.

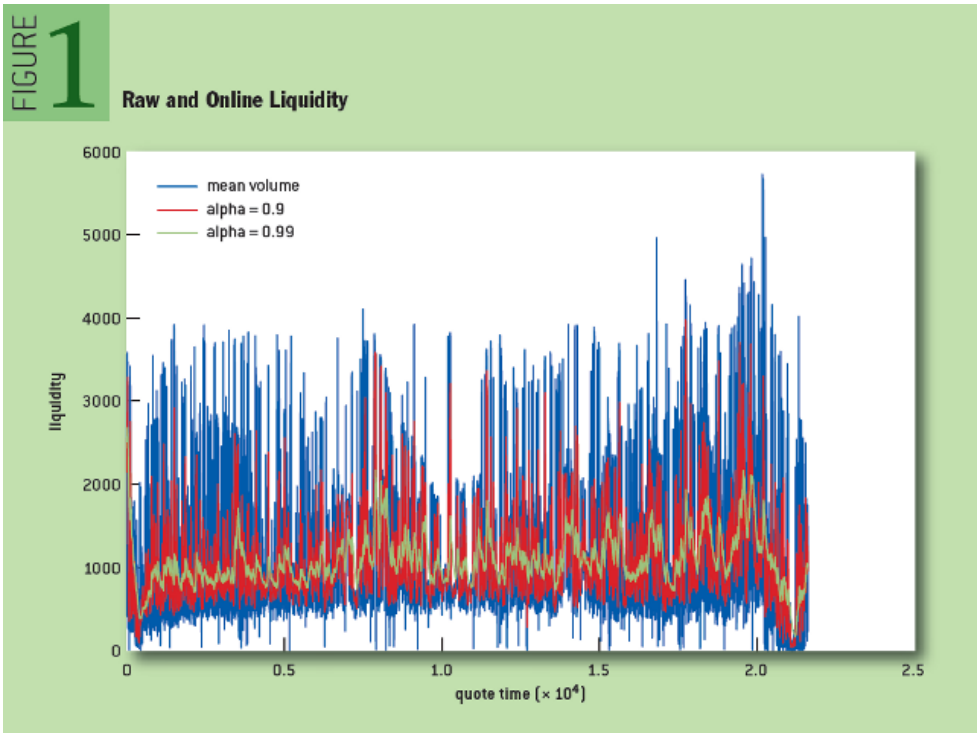
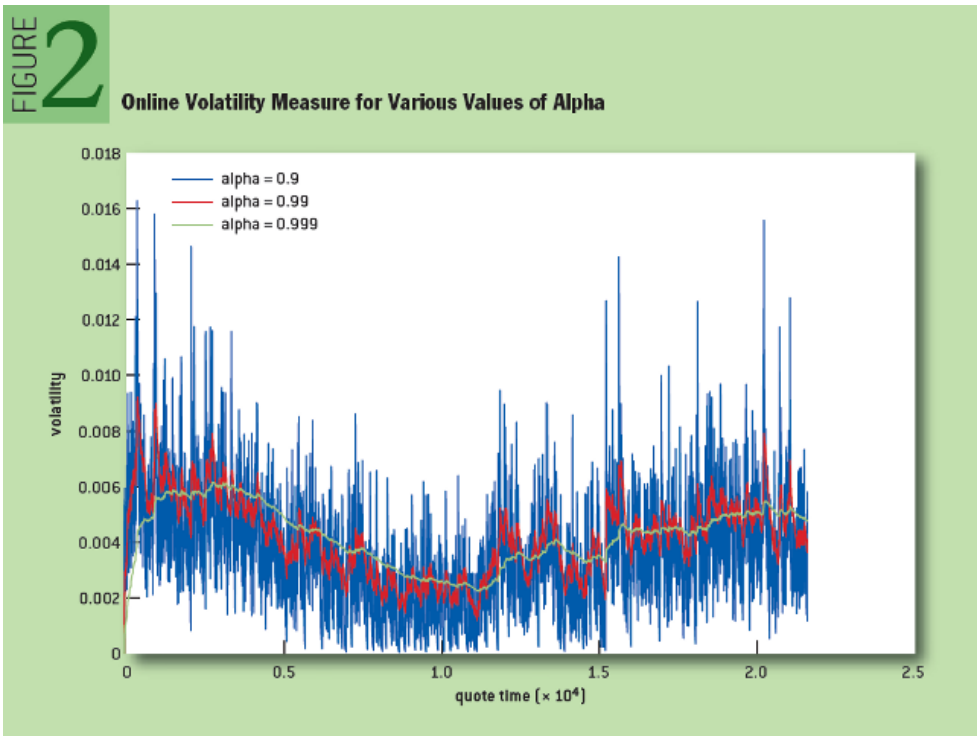


Figure 2 plots the online volatility measure for various values of alpha. Once again, notice that the measure is smoother for larger alpha. Although a larger alpha provides a smoother signal, it also lags further behind the underlying trend as it gives a lot of weight to older data. As discussed later, choosing a value for alpha translates into a tradeoff between a smooth signal and a reduced lagging of the trend.



To illustrate the online regression algorithm, we look at the time series of mid prices for SPY and SSO, two highly related ETFs (SSO is the double-leveraged version of SPY). As shown in figure 3, the relationship between the two assets seems very close to linear over the course of a day. Figure 4 plots the online mean and intercept for two values of alpha.

FIGURE 3

Online Regression Algorithm

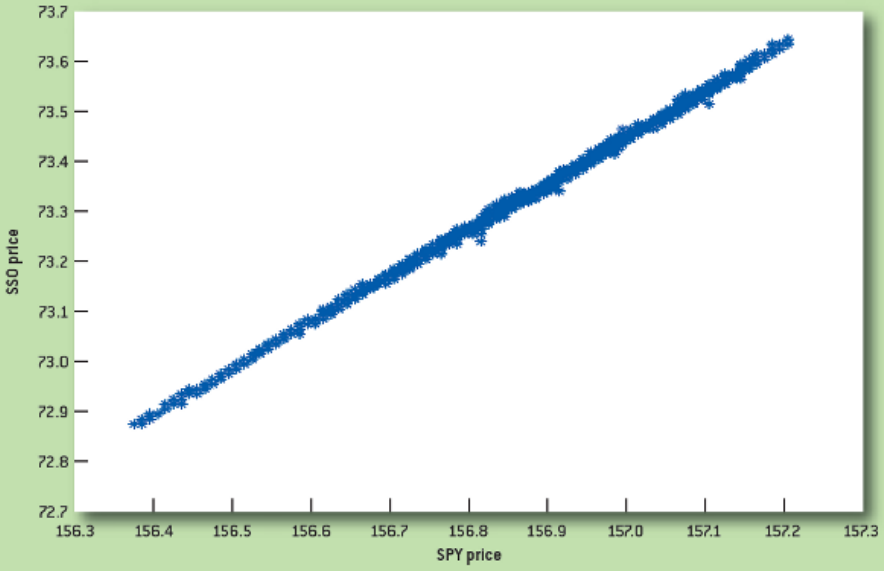
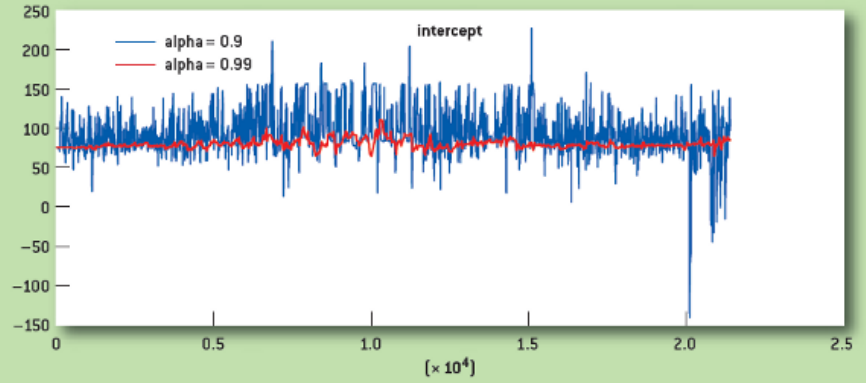
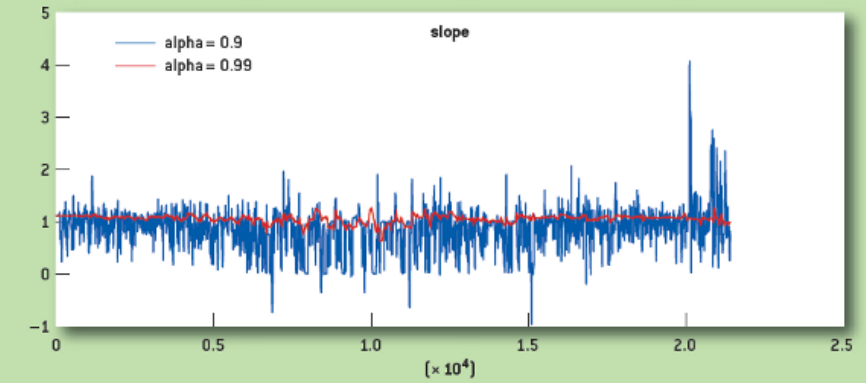


FIGURE 4

Online Mean and Intercept for Two Values of Alpha





ONE-PASS ALGORITHMS

As indicated by its name, a *one-pass algorithm* reads every input variable exactly once and then discards it. This type of algorithm is very efficient in terms of memory handling, as it requires only a minimal amount of data to be stored in memory. This section presents three important examples of online one-pass algorithms: the exponential moving average, the exponentially weighted variance, and the exponentially weighted regression. The next section then describes the application of these algorithms for HFT.

First, let's look briefly at the *simple moving average* of a time series. This is an estimate of the mean of a time series over a moving window of a fixed size. In finance, it is often used to detect trends in price, in particular by comparing two simple moving averages: one over a long window and one over

a short window. In another application, the average traded volume over the past five minutes can serve as a prediction of the volume traded in the next minute. In contrast to the exponential moving average, the simple moving average cannot be solved with a one-pass algorithm.

Let $(X_t)_t = X_0, X_1, X_2, \ldots$ be the observed sequence of input variables. At any given time t we want to predict the next outcome X_{t+1}. For $M > 0$ and $t \geq M$, the simple moving average with window size M is defined as the average of the last M observations in the time series $(X_t)_t$ —that is,

$$\hat{X}_{t,M} = \frac{1}{M}\sum_{j=0}^{M-1} X_{t-j}$$

. The moving average can also be computed via the following recursion:

$$\hat{X}_{t,M} = \hat{X}_{t-1,M} - \frac{X_{t-M}}{M} + \frac{X_t}{M}$$

. (1)

While this is an online algorithm, it is not a one-pass algorithm, as it needs to access every input data point exactly twice: once to add it to the moving average and then again to drop it out of the moving average estimate. Such an algorithm is referred to as a two-pass algorithm and requires keeping an entire array of size M in memory.

EXAMPLE 1: ONE-PASS EXPONENTIAL WEIGHTED AVERAGE

In contrast to the regular average

$$\bar{X} = \frac{1}{t+1}\sum_{j=0}^{t} X_j$$

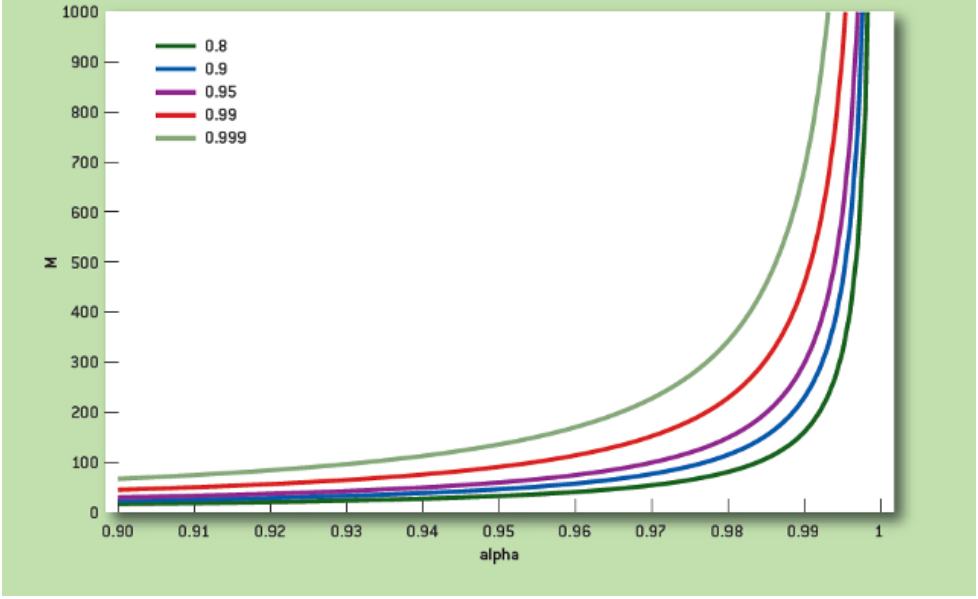
, the exponential weighted average assigns an exponentially decreasing weight to older observations:

$$\hat{X}_{t,\alpha} = (1-\alpha)\sum_{j=0}^{t-1} \alpha^j X_{t-j} + \alpha^t X_0.$$

Here α is a weighting parameter chosen by the user and needs to satisfy $0 < \alpha \leq 1$. As this exponential weighted average gives more importance to more recent input compared with older data points, it is often considered to be a good approximation of the simple moving average.

Compared with the simple moving average, the exponential weighted average takes all previous data into account, not just the last M observations. To compare the simple moving average and the exponential weighted average further, figure 5 shows how many data points receive 80, 90, 95, 99, and 99.9 percent of the weight in the estimation as a function of α. For example, if $\alpha = 0.95$, then the last $M = 90$ observed data points contribute to 99 percent of the estimated value. As a warning, if the time series $(X_t)_t$ has very heavy tails, then the exponential smoothed average might be dominated by an extreme observation, whereas the moving average is less prone to extreme observations as these eventually drop out of the observation window. Frequent restarting of the estimation procedure can solve this long-term memory effect of exponential smoothing.

FIGURE 5 **The Moving Average and the Weighting Parameter**



The reason for favoring the exponential moving average over the simple moving average in HFT is that it can be efficiently solved using a one-pass algorithm, initially introduced in Brown (1956).[3]

$$\begin{aligned} \hat{X}_{0,\alpha} &= X_0, \\ \hat{X}_{t,\alpha} &= (1-\alpha)X_t + \alpha\hat{X}_{t-1,\alpha}. \end{aligned}$$

(2)

This formula also provides a simple interpretation of the parameter α as a control of how much weight is given to the most recent observation, compared with all previous observations.

EXAMPLE 2: ONE-PASS EXPONENTIALLY WEIGHTED VARIANCE

The exponential smoothing described in the previous section estimates a moving average of a time series. In finance, the volatility of a time series is

often an important factor as well. Broadly speaking, volatility should capture how much a time series fluctuates around its mean. There is no widely accepted definition of volatility for high-frequency financial data. This section considers the volatility to be the standard deviation (square root of variance) of a data point in the time series $(X_t)_t$. Similar to the exponentially weighted moving average from the previous section, an online one-pass algorithm can be constructed that estimates the volatility of the time series $(X_t)_t$ with an exponential weighting scheme.

The variance of a random variable is defined as $Var(X) = E[X - E[X])^2]$. Estimating the exponential weighted variance of the time series requires two estimators: one that estimates the mean $E[X]$ and one that estimates the variance:

$$\begin{aligned} \hat{X}_{0,\alpha} &= X_0 \\ \hat{V}_0 &= 1 \\ \hat{X}_{t,\alpha} &= (1-\alpha)X_{t,\alpha} + \alpha\hat{X}_{t-1,\alpha} \\ \hat{V}_{t,\alpha} &= (1-\alpha)(X_{t,\alpha} - \hat{X}_{t,\alpha})^2 + \alpha\hat{V}_{t-1,\alpha} \end{aligned}$$

The standard deviation of the next measurement point X_{t+1} is then estimated as

$$\sqrt{\hat{V}_{t,\alpha}}$$

. Again, the input parameter $\alpha \in (0,1)$ is chosen by the user and reflects how much weight is assigned to older data points compared with the latest observed data input. Here, we initialized the estimator of the variance with 1, which is a rather arbitrary choice. Another way is to have an initial "burn-in" period for which the time series $(X_t)_t$ is observed and a standard variance estimator of the series over this burn-in time window can be used to initialize the estimator. Of course, a similar method can be used to initialize the estimator of the exponentially weighted average estimator.

EXAMPLE 3: ONE-PASS ALGORITHM FOR EXPONENTIALLY WEIGHTED LINEAR REGRESSION

The last example is an online one-pass algorithm for the exponentially weighted linear regression model. This model is similar to ordinary linear regression, but again gives more importance (according to an exponential weighting) to recent observations than to older observations. As already shown, such regression methods are very useful in HFT strategies to estimate the relation of different assets, which can be, for example, exploited in creating pair trading strategies.

In this model we consider a two-dimensional time series $(X_t,Y_t)_t$ and conjecture that the variables X and Y are related via a linear relation that is corrupted by a noise term ε_t with zero mean. That is,

$$Y_t = \beta_0 + \beta_1 X_t + \varepsilon_t. \quad (3)$$

The variable Y is referred to as the response variable, whereas X is called the explanatory variable. For simplicity let's assume just one explanatory variable here, but the extension to several explanatory variables is straightforward. In the standard offline approach to linear regression, the parameters β_0 and β_1 are calibrated after all the data points are observed. These data points are collected in a vector $\boldsymbol{Y} = (Y_0, Y_1, \ldots, Y_t)^T$ and a matrix

$$\boldsymbol{X} = \begin{bmatrix} 1 & X_0 \\ \vdots & \vdots \\ 1 & X_t \end{bmatrix}$$

The column of ones in the matrix $\boldsymbol{X}$ corresponds to the intercept in equation 3. If we further write the parameters β_0 and β_1 as a vector—that is, $\boldsymbol{\beta} = (\beta_0,\beta_1)T$—then the relationship between $\boldsymbol{Y}$ and $\boldsymbol{X}$ can conveniently be written in matrix notation as

$$\boldsymbol{Y = X\beta + \varepsilon}$$

where $\boldsymbol{\varepsilon}$ is a vector of stochastic noise terms, and each of these error terms has zero mean.

The most common approach to estimating the parameter $\boldsymbol{\beta}$ is using ordinary least squares estimation—that is, $\boldsymbol{\beta}$ is chosen such that it minimizes the sum of squared residuals

$$\sum_{j=0}^{t}(Y_j - (\beta_0 + \beta_1 X_j))^2$$

. The solution to this minimization problem is

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y}$$

.

As in mean and variance estimations, more recent data points should be more important for the estimation of the parameter $\boldsymbol{\beta}$. Also, a one-pass algorithm for $\boldsymbol{\beta}$ is required for fast computation.

Next let's consider a recursive method that updates $\boldsymbol{\beta}$ sequentially and minimizes

$$\sum_{j=0}^{t} \alpha^{(t-j)} \left(Y_j - \left(\beta_0 + \beta_1 X_j\right)\right)^2.$$

Again, the parameter α needs to be in the range (0,1) and is chosen by the user. The parameters $\beta_0$and β_1 of the weighted least squares estimation can be computed with an efficient online one- pass algorithm. At each step of the algorithm a 2 × 2 matrix $\boldsymbol{M}_t$ and a 2 × 1 vector $\boldsymbol{V}_t$ need to be saved in memory and updated with a new data point according to the following recursion:

$$\begin{aligned} \boldsymbol{M}_t &= \alpha\boldsymbol{M}_{t-1} + \boldsymbol{X}_t^T\boldsymbol{X}_t, \\ \boldsymbol{V}_t &= \alpha\boldsymbol{V}_{t-1} + \boldsymbol{X}_t^T\boldsymbol{Y}_t. \end{aligned}$$

As for the mean and variance estimator, the initialization of the recursion can be done using a burn-in period. Finally, after time t, the best estimate of $\boldsymbol{\beta}$ is

$$\hat{\boldsymbol{\beta}}_t = \boldsymbol{M}_t^{-1}\boldsymbol{V}_t$$

. In the literature this method is also called recursive least squares with exponential forgetting.[2]

ESTIMATING ALPHA

How does one decide on the optimal value of alpha, the one parameter of all these online models? Our approach for all three models is to define a response function that we aim to predict, and minimize the squared error between the response r_i and our factor f_i:

$$\min_{\alpha} \sum_{i} (f_i(\alpha) - r_i)^2$$

This method finds the optimal alpha on a historical time series. Another approach would be to estimate the optimal alpha online as well. This, however, requires more work and goes beyond the scope of this article.

We now provide the details on the online estimators described and estimate the optimal alpha on a given data set.

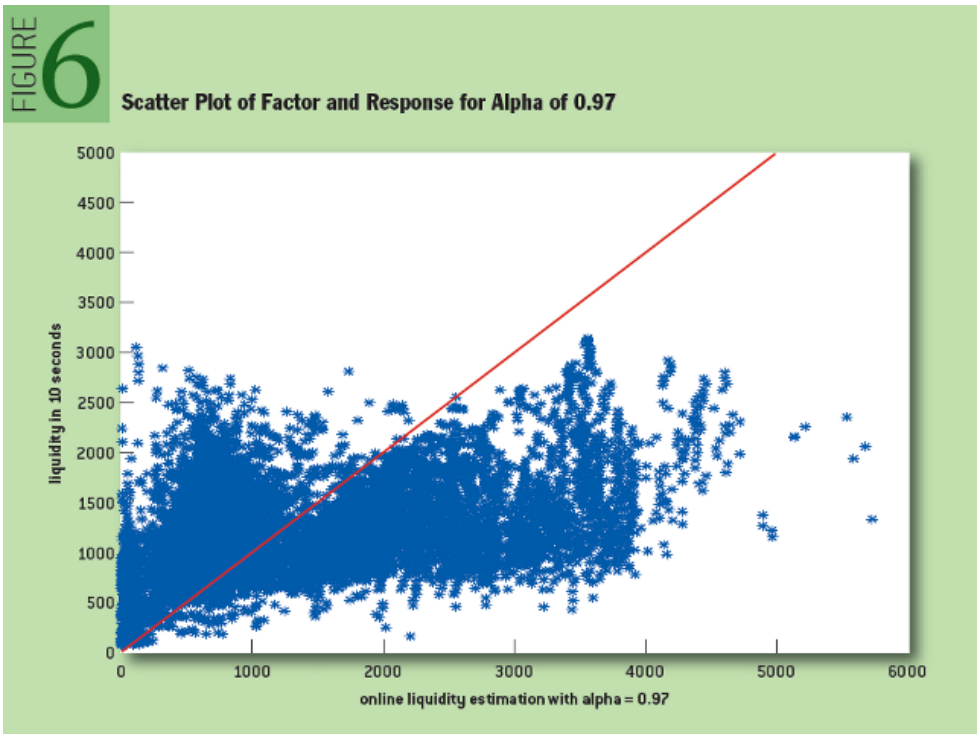
1. The mean liquidity estimator is defined as

$$f_i^{liq}(\alpha) = \alpha f_{i-1}^{liq}(\alpha) + (1-\alpha)\, bs_i + as_i$$

where the index *i* represents quote time. The response is defined to be the liquidity in 10 seconds:

$$r_i = bs_i(10) + as_i(10)$$

where $bs_i(10)$ represents the bid size 10 seconds after the i-th quote. Running an optimization routine over alpha shows that the optimal alpha for the given data is 0.97, displayed in figure 6 as a scatter plot of the factor and the response.



2. The volatility estimator is defined as

$$m_i(\alpha) = \alpha m_{i-1}(\alpha) + (1-\alpha)(p_i - p_{i-1})$$

$$v_i(\alpha) = \alpha v_{i-1}(\alpha) + (1-\alpha)(p_i - p_{i-1} - m_i(\alpha))^2$$

$$f_i^{vol}(\alpha) = \sqrt{v_i(\alpha)}$$

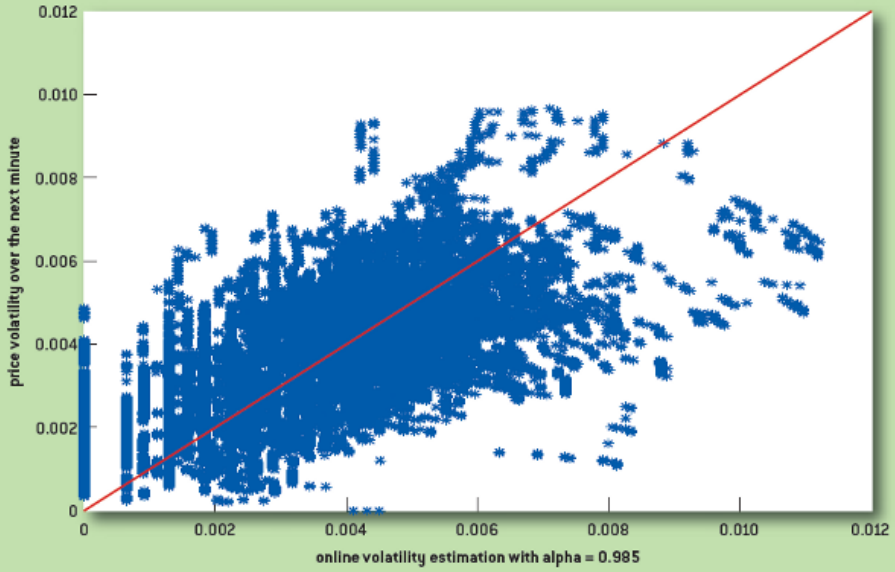
where the index *i* represents realtime in seconds. The response is defined to be the realized volatility over the next minute:

$$r_i = \sqrt{\sum_{j=1}^{60} \left(p_{i+j} - p_{i+j-1} - \frac{1}{60}(p_{i+60} - p_i)\right)^2}$$

Again, searching over different values of alpha yields an optimal alpha of 0.985 for the given data set. Figure 7 displays a scatter plot of the factor and the response.

FIGURE 7

Scatter Plot of Factor and Response for Alpha of 0.985



3. The pairs trading regression estimator is defined as

$$M_i(\alpha) = \alpha M_{i-1}(\alpha) + (1-\alpha)\left(\frac{1}{p_i^{SSO}}\right)(1 \;\; p_i^{SSO})$$

$$V_i(\alpha) = \alpha V_{i-1}(\alpha) + (1-\alpha)\left(\frac{1}{p_i^{SSO}}\right)p_i^{SPY}$$

$$\beta_i^{reg}(\alpha) = M_i(\alpha)^{-1}V_i(\alpha)$$

$$f_i^{reg}(\alpha) = (1 \;\; p_i^{SSO})\beta_i^{reg}(\alpha) - p_i^{SPY}$$

where the index *i* represents quote time. The factor

$f_i^{reg}(\alpha)$

represents the value of SPY relative to SSO—that is, if the quantity is positive, then SPY is relatively cheap and a trade that is long SPY is likely to be profitable.

The response is defined as the PNL over the next minute of a trade that is long one share of SPY and short β shares of SSO:

$$r_i = \left(p_i^{SPY}(60) - \beta_i^{reg}(\alpha)\left(\frac{1}{p_i^{SSO}(60)}\right)\right) - \left(p_i^{SPY} - \beta_i^{reg}(\alpha)\left(\frac{1}{p_i^{SSO}}\right)\right)$$

where

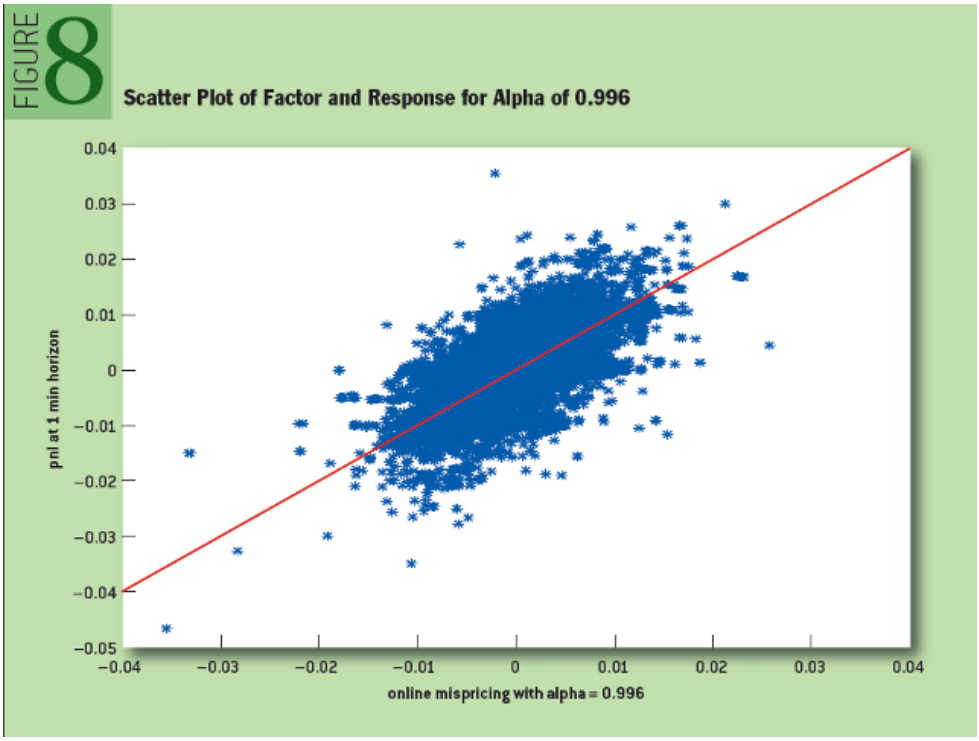
$p_i^{SPY}(60)$

represents the price of SPY 60 seconds after

p_i^{SPY}

. The response r_i represents the PNL of the following long-short strategy: "Buy 1 share of SPY and sell β shares of SSO at time *i* , exit the position after 60 seconds."

In the analyzed data set the optimal alpha turns out to be 0.996. Figure 8 is a scatter plot of the factor and the response.



CONCLUSION

Online one-pass algorithms are instrumental in high-frequency trading, where they receive large amounts of data every microsecond and must be able to act extremely fast on the received data. This article has addressed three problems that HFT algorithms face: the estimation of a running mean of liquidity, which can be useful in determining the size of an order that is likely to execute successfully on a particular electronic exchange; a running volatility estimation, which can help quantify the short-term risk of a position; and a running linear regression, which can be used in trading pairs of related assets. Online one-pass algorithms can help solve each of these problems.

REFERENCES

1. Albers, S. 2003. Online algorithms: a survey. Mathematical Programming 97(1-2): 3-26.

2. Astrom, A., Wittenmark, B. 1994. Adaptive Control, second edition. Addison Wesley.

3. Brown, R. G. 1956. Exponential Smoothing for Predicting Demand. Arthur D. Little Inc., p. 15

4. Clark, C. 2011. Improving speed and transparency of market data. Exchanges; https://exchanges.nyx.com/cclark/improving-speed-and-transparency-market-data.

LOVE IT, HATE IT? LET US KNOW

feedback@queue.acm.org

JACOB LOVELESS is the CEO of Lucera and former head of High Frequency Trading for Cantor Fitzgerald. Mr. Loveless has worked for both high frequency trading groups and exchanges for the past 10 years in nearly every electronic asset. Prior to a life in finance, Mr. Loveless was a special contractor for the US Department of Defense with a focus on heuristic analysis on things which cannot be discussed. Prior to that, he was the CTO and a founder of Data Scientific, a pioneer in distributed systems analysis.

SASHA STOIKOV is a senior research associate at Cornell Financial Engineering Manhattan (CFEM) and a former VP in the High Frequency Trading group at Cantor Fitzgerald. He has worked as a consultant at the Galleon Group and Morgan Stanley and was an instructor at the Courant Institute of NYU and at Columbia's IEOR department. He holds a Ph.D. from the University of Texas and a BS from MIT.

ROLF WAEBER is a Quantitative Research Associate at Lucera and previously served as a Quantitative Researcher at Cantor Fitzgerald's High Frequency Trading Group. He participated in studies on liquidity risk adjustments within the Basel II/III regulation frameworks at the Deutsche Bundesbank. Rolf earned his Ph.D. in Operations Research and Information Engineering from Cornell University in 2013. He holds a BS and an MS in Mathematics from ETH Zurich, Switzerland.

Originally published in Queue vol. 11, no. 8—
see this item in the ACM Digital Library

COMMENTS

xxx | Mon, 21 Oct 2013 07:48:57 UTC

Name of the Author, Loveless... tai diao le