STATS 244 Final Project: Feature Engineering of LOB via Machine Learning Methodologies

Ben Etringer
Adhitya Venkatesh

December 12, 2016

Contents

1 Introduction

The emergence of big data has had tremendous results across many industries over the past decade, especially in finance. It has enabled a data-driven approach to the analysis of markets and securities that has in turn allowed quantitative firms following the trend of applying big data to flourish . High frequency trading (HFT) firms in particular greatly benefit from the rise of big data methodologies. This refers specifically to the advances made in predictive analytics, which obviously increases predictive power of the markets and consequently profit potential.

A central data set in HFT is limit order books (LOB), which are essentially data feeds directly from the exchanges. These indicate the volume of shares and the corresponding price levels at which other market participants are placing orders to buy/sell a particular stock. Other information includes a message book of order placements, executions, and cancellations along with their timestamps.

There is plenty of literature on theoretical models attempting to capture the dynamics of various factors derived from information in the LOB. However, this literature relies on simplifying assumptions about the stochastic properties of how the prices and volumes evolve over time. This is where new age predictive analytics allows us to progress: by providing ways of fitting the data without requiring assumptions of the statistical nature of the data to necessarily hold. Machine learning is one such tool.

We used this tool via a neural network model to capture the dynamics of the bid and ask prices of the stock in question over time. This is a crucial prediction problem for market makers in the HFT sphere as they function to provide liquidity for the market and profit by the spread of these prices per trade. Even non-market making HFT firms benefit from solving this prediction problem to more optimally place and time trades. Starting with the fact that the bid-ask spread is determined by the volatility and liquidity of the stock at a particular point in time, we created features processed from the raw data in the LOB that measure aspects of volatility and liquidity. The remainder of this report is an account of our efforts and results from applying neural networks to model the bid and ask prices as a function of time by incorporating our feature set.

2 Data Preprocessing and Visualization

Before we can implement the proposed neural network model, we must first clean and preprocess the data. In this section, we will propose which features are of the most importance, and provide figures to visualize the data.

2.1 Bid-Ask Price

We being by looking at a time-series plot of the best bid and best ask prices. This is the data we will use as the input/output for the model (see figure 1).

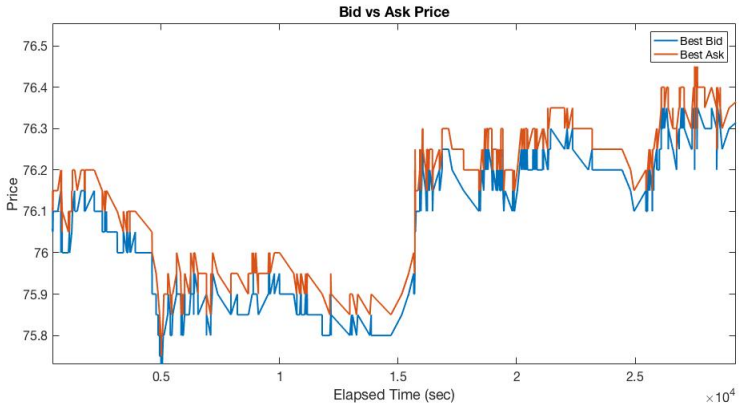

Figure 1: Time Series Plot of Best Bid and Best Ask Price

2.2 Proposed Features

We then wish to create the feature matrix for our neural network in order to predict future price and bid-ask spread. Therefore, we require features that account for the direction of movement, and the amount of movement of the bid and ask prices. We propose that the following six features will provide the best fit: (1) Probability Weighted Volume Price, (2) Inverse Size Weighted Fair Value Price, (3) Duration Intensity, (4) First-Order Derivative of Duration Intensity, (5) Volume Kurtosis, and (6) Volume Skewness.

2.2.1 Probability Weighted Volume Price (PWV)

The PWV price is a quantity that accounts for both supply and demand by computing a probability weighted price from the limit order book. That is, let $\pi_{l,i,\tau} = \Pr(t_f < t+\tau | l, i)$ be the probability of an order of type $i \in \{\text{bid}, \text{ask}\}$, submitted at book level l, to be filled at time τ. Then the probability weighted volume is defined by,

$$v(t, \tau, L; i) = \frac{1}{\sum_{i,l} v_{t,l;i}} \sum_{l=0}^{L} v_{t,l;i} \pi_{l,i,\tau}$$

The time series plot of PWV price is shown in figure 2. We then subtract the mid-price at each time-step to obtain our first feature.

2.2.2 Inverse Size Weighted Fair Value Price (IWFV)

The IWFV price is similar to the PWV price, however the factor-loadings are reversed on the limit order book. That is, if p_l^i and v_l^i are the level-l bid/ask price and volume, where

$i \in \{\text{bid}, \text{ask}\}$, then we have

$$P^{(\text{IWFV})} = \frac{p_l^{Bid} v_l^{Ask} + p_l^{Ask} v_l^{Bid}}{v_l^{Ask} + v_l^{Bid}}$$

The time series plot of IWFV price is shown in figure 2. We then subtract the mid-price at each time step to obtain our second feature.

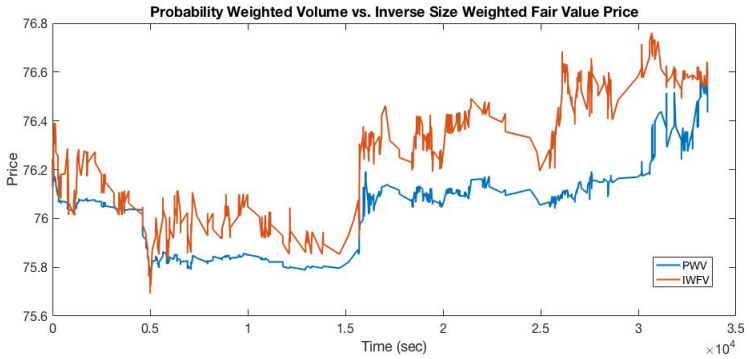

Figure 2: Time Series Plot of PWV vs. IWFV Price

2.2.3 Duration Intensity and its First-Order Derivative

We use implement the kernel smoothing method with Gaussian kernels and an adjusted bandwidth (to account for large tails without saturating the intensity model) (see figure 3 for smoothing plot). We then use the central difference method to numerically approximate the first order derivative. That is,

$$f'(d) \approx \frac{f(d+h) - f(d-h)}{2h}$$

where d is the duration and h is a small constant (we use .05 seconds in our model). See figure 4 for plot of the duration intensity and its first-order derivative. These two vectors are the third and fourth features to our model.

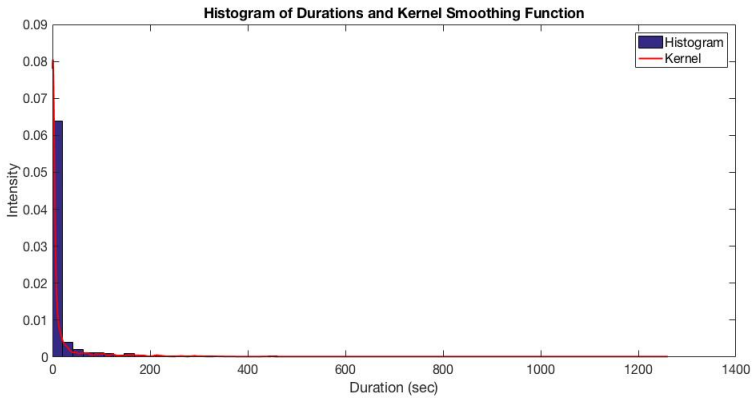

Figure 3: Comparison plot Histogram of Durations and Approximated Duration Intensity

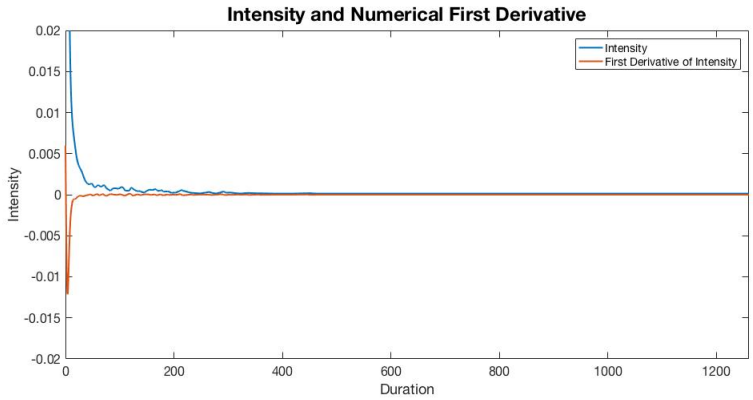

Figure 4: Comparison plot of Duration Intensity and its First-Order Derivative

2.2.4 Skewness and Kurtosis

We then look at the skewness and kurtosis of the limit order book for the best 10 bid/ask prices. Any asymmetry in the limit order book could provide insight into a future price shift. In order to compute each of these, we invoke MatLab's `skewness` and `kurtosis` functions on the LOB at each time step. Figures 5 and 6 show the time series plot of the resulting vector.

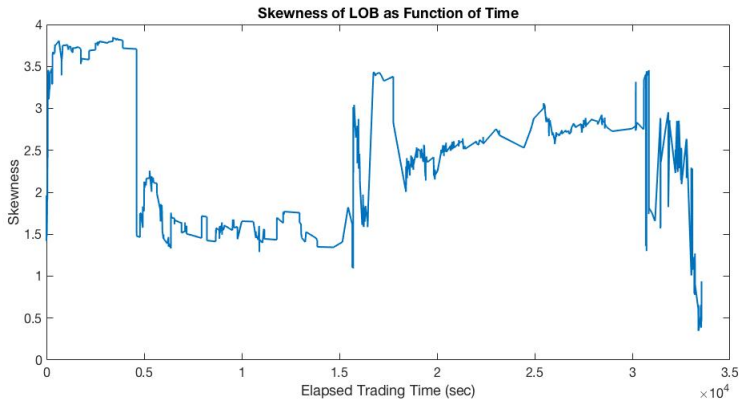

Figure 5: Time Series Plot of Skewness from Volumes in LOB

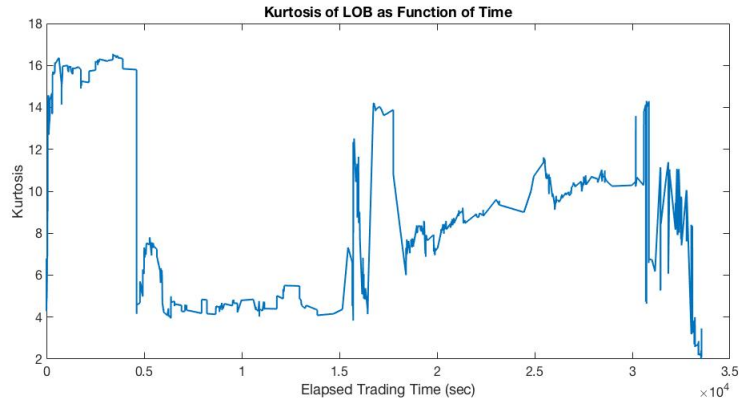

Figure 6: Time Series Plot of Kurtosis from Volumes in LOB

3 Model

After the aforementioned features were extracted from the raw level 1 data, the processed data set was decomposed into 70% training data, 15% validation data, and the remaining 15% as testing data by random sampling. The training data was then used to fit a function of the form

$$[\text{Bid\_Price}(t), \text{Ask\_Price}(t)] = f(\text{Bid\_Price}(t-h), \text{Ask\_Price}(t-h), \text{Features}(t-h))$$

Where h is all iterates from 1 to d.

In order to best model the function f by incorporating the feature set, the machine learning model used was a neural network, shown in figure 7. More specifically, a non-linear autoregressive model with external input (NARX) framework. We fit multiple neural network models, with only a single hidden-layer, each with varying number of time delays d up to 5 and number of hidden-layer nodes up to 20. The optimal choice resulting in a time delay d of 3 and 12 hidden-layer nodes (not including bias nodes at each level) based on the lowest MSE when the model was ran on the testing data.

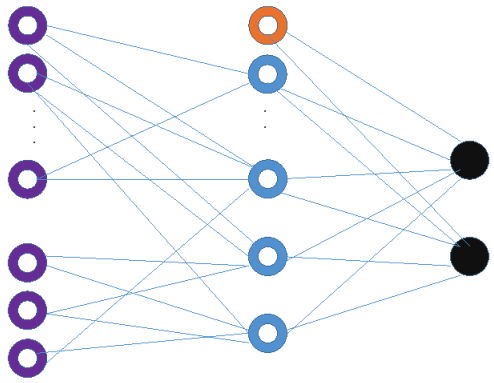

Figure 7: Neural Network Diagram

: Represents input nodes (21 nodes) with 3 nodes per feature and each node representing a time step from the current time step t.
: Represents hidden layer nodes (12 nodes). Optimal number determined by model selection (choosing the smallest MSE when predicting on test data).
: Represents the bias node in the hidden layer.
: Represents output vector. One node representing the bid price the other the ask price.

Further, to avoid overfitting the data, the training algorithm used was Bayesian regularization. The `trainbr` package automatically handles the calibration of the penalty term as the neural network is trained. This term is given as the variable μ in figure 8 below which shows the convergence results of the parameters as the network is trained.

The training and test errors are shown in figure 9. Applicable code is found in the Supporting Code section.

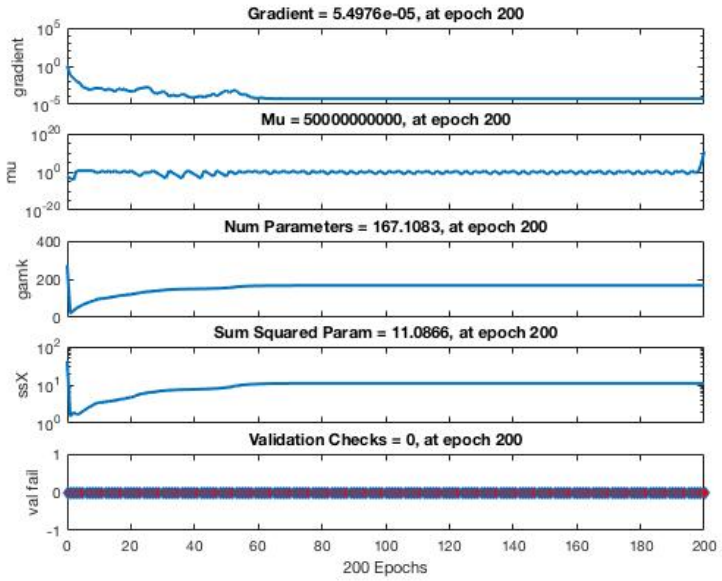

Figure 8: Neural Network Training State for Bid-Ask Price

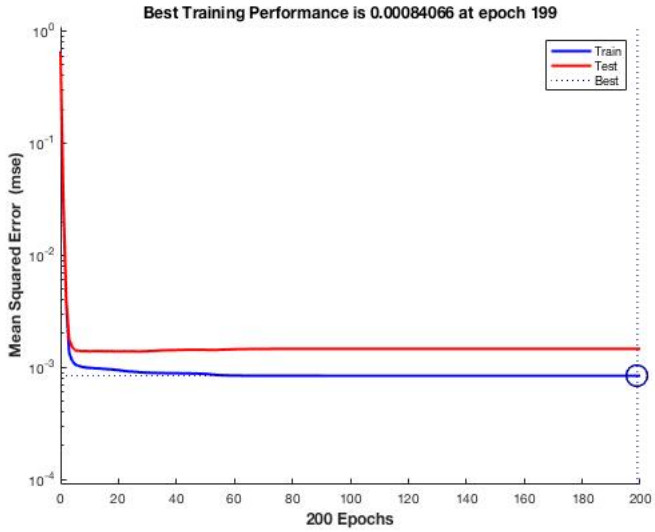

Figure 9: Training and Test Errors

4 Results and Analysis

We start with a time series graph of the fitted model vs the data and the plot of the error autocorrelation, see figure 10.

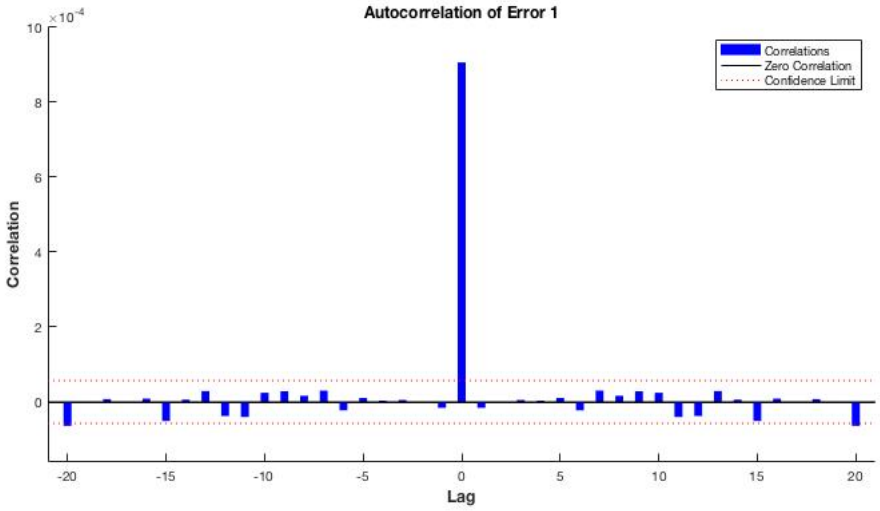

Figure 10: Plot of Error Autocorrelation

It is apparent that the errors are uncorrelated and the fitted function captures the data quite accurately. In order to test the validity of the model, regressions were done comparing the output to the fitted values as shown in figure 11.

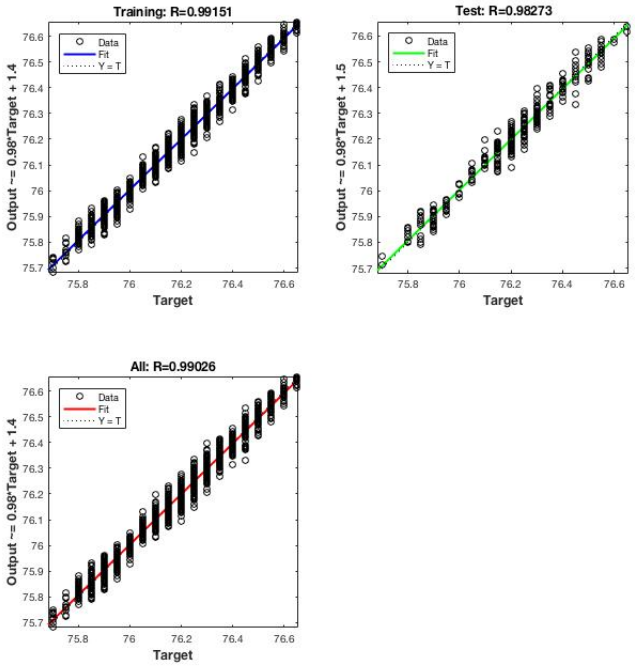

Figure 11: Regression of Output to Fitted Values

4.1 Concluding Remarks

Given the near-1 R-squared value, the model is extremely accurate and can be improved even further with additional features, parameter tuning, and of course data. Luckily, the feature set chosen provided strong predictive power of the time series evolution of bid-ask prices. This is quite a useful problem to solve using limit order book data, especially high-frequency trading firms functioning as market-makers. Accurate prediction of future bid-ask prices is central to onloading and offloading shares to the market without running the risk of too much or too little inventory at any given time. Since a neural network was chosen to model this market-making problem, the resulting fit f is not easily interpretable and thus becomes a difficult task to extricate how the model has learned to relate the features measuring volume imbalance/liquidity to those measuring the volatility of the limit order book in that given snapshot of time. However, the results from the previous figures suggests high predictive capability despite lack of interpretability. This tradeoff is justified, as the primary objective was prediction and not interpretation of the model.

5 Works Cited

References

[1] Avellaneda, M., & Stoikov, S. (2008). High-frequency trading in a limit order book. *Quantitative Finance*. 8(3), 217-224. doi:10.1080/14697680701381228

[2] Guo, Xin, Tze Lung Lai, Howard Shek, and Samuel Po-Shing Wong. *Quantitative Trading: Algorithms, Analytics, Data, Models, Optimization.*. Place of Publication Not Identified: Chapman & Hall Crc, 2016. Print.

[3] Kercheval, A. N., & Zhang, Y. (2015). Modelling high-frequency limit order book dynamics with support vector machines. . *Quantitative Finance,*. 15(8), 1315-1329. doi:10.1080/14697688.2015.1032546

6 Supporting Code

6.1 Preprocessing Code

Note, the following script assumes the user has imported the .csv files into MatLab and has used Excel (or another program) to clean the * from the file.

```
% Clean data into usable dataframe
A = HSBCL2Proj(HSBCL2Proj(:,2)>=0,:);
B = HSBCL2Proj1(HSBCL2Proj1(:,2)>=0,:);
A = A(:,~isnan(A(1,:)));
B = B(:,~isnan(B(1,:)));
for i = 4:2:42
    A(isnan(A(:,i)),i)=0;
    B(isnan(B(:,i)),i)=0;
end
for i = 3:2:41
    for j=1:length(A)
        if isnan(A(j,i))
            A(j,i)=A(j,i-2)+.05;
        end
    end
end
for i = 41:-2:3
    for j=1:length(A)
        if isnan(A(j,i))
            A(j,i)=A(j,i+2)-.05;
        end
    end
end
for i = 3:2:41
    for j=1:length(B)
        if isnan(B(j,i))
            B(j,i)=B(j,i-2)+.05;
        end
    end
end
for i = 41:-2:3
    for j=1:length(B)
        if isnan(B(j,i))
            B(j,i)=B(j,i+2)-.05;
        end
```

```
    end
end
dataframe = [A;B];

% Determine VWP and Inverse VWP
VWP = zeros(length(dataframe),1);
IVWP = zeros(size(VWP));
% Determine Skewness and Kurtosis of Volumes
vol_skew = zeros(size(VWP));
vol_kurt = zeros(size(VWP));
for i = 1:length(dataframe)
    vol_vec = zeros(20,1);
    price_vec = zeros(20,1);
    count =0;
    for j = 4:2:42
        count = count +1;
        vol_vec(count) = dataframe(i,j);
        price_vec(count) = dataframe(i,j-1);
    end
    vol_skew(i) = skewness(vol_vec);
    vol_kurt(i) = kurtosis(vol_vec);
    vol_vec = vol_vec/sum(vol_vec);
    inv_vol_vec = flip(vol_vec);
    VWP(i) = sum(vol_vec.*price_vec);
    IVWP(i) = sum(inv_vol_vec.*price_vec);
end

dur = dataframe(:,2);
t = cumsum(dur);
% Plot VWP vs IWFV
figure
plot(t, VWP)
hold on
plot(t,IVWP)
hold off
title('Probability Weighted Volume vs. Inverse Size Weighted Fair Value Price')
legend('PWV', 'IWFV')
xlabel('Time (sec)')
ylabel('Price')

% Convert durations to intensity of durations
```

```
% and plot results

pd = fitdist(dur,'kernel', 'width' , 3);
plot_y = pdf(pd, sort(dur));
dur_intensity = pdf(pd,dur);
[f,x] = hist(dur,60);
figure
bar(x,f/trapz(x,f))
hold on
plot(sort(dur),plot_y,'r')
hold off
title('Histogram of Durations and Kernel Smoothing Function')
legend('Histogram','Kernel')
xlabel('Duration (sec)')
ylabel('Intensity')

% Determine bid-ask spread
spreads = dataframe(:,23)-dataframe(:,21);

% Plot duration intensity with its first derivative
dur_int_1D = (pdf(pd,sort(dur)+.05)-pdf(pd,sort(dur)-.05))/0.1;
plot_me = pdf(pd,sort(dur));
figure
plot(sort(dur),plot_me)
hold on
plot(sort(dur),dur_int_1D)
hold off
axis([0,max(dur), -.02, .02])
title('Intensity and Numerical First Derivative')
xlabel('Duration')
ylabel('Intensity')
legend('Intensity','First Derivative of Intensity')

dur_int_1D = (pdf(pd,dur+.05)-pdf(pd,dur-.05))/.1;

% Plot skewness and kurtosis
figure
plot(t, vol_skew)
title('Skewness of LOB as Function of Time')
xlabel('Elapsed Trading Time (sec)')
ylabel('Skewness')
```

```
figure
plot(t, vol_kurt)
title('Kurtosis of LOB as Function of Time')
xlabel('Elapsed Trading Time (sec)')
ylabel('Kurtosis')

price = (dataframe(:,23)+dataframe(:,21))/2;
bid_price = dataframe(:,21);
ask_price = dataframe(:,23);

% Plot Best Bid and Best Ask as time series
figure
plot(t, bid_price)
hold on
plot(t, ask_price)
hold off
title('Bid vs Ask Price')
legend('Best Bid', 'Best Ask')
xlabel('Elapsed Time (sec)')
ylabel('Price')

% Clear workspace of unneeded variables
clear x t vol_vec price_vec plot_y plot_me pd j i f dur ...
    dataframe count A B inv_vol_vec
```

6.2 MyNeuralNetworkFunction

```
function [y1,xf1,xf2] = myNeuralNetworkFunction(x1,x2,xi1,xi2)

%MYNEURALNETWORKFUNCTION neural network simulation function.
%
% Generated by Neural Network Toolbox function genFunction,
% 12-Dec-2016 00:09:20.
%
% [y1,xf1,xf2] = myNeuralNetworkFunction(x1,x2,xi1,xi2) takes these
% arguments:
%   x1 = 6xTS matrix, input #1
%   x2 = 2xTS matrix, input #2
%   xi1 = 6x3 matrix, initial 3 delay states for input #1.
```

```
%   xi2 = 2x3 matrix, initial 3 delay states for input #2.
% and returns:
%   y1 = 2xTS matrix, output #1
%   xf1 = 6x3 matrix, final 3 delay states for input #1.
%   xf2 = 2x3 matrix, final 3 delay states for input #2.
% where TS is the number of timesteps.
```

6.3 Neural Network Constants

In the following subsections, we layout the neural network constants

6.3.1 Input 1

```
% Input 1
x1_step1.xoffset = [-0.226368691189052;-0.270265041594143;0.00013231916431225;
          -0.0121363805269234;2.00488990562017;0.346400848282155];

x1_step1.gain = [3.73401046500632;6.19937202524666;24.8799130679515;
          110.417004205609;0.137468717118483;0.570690315839459];
x1_step1.ymin = -1;
```

6.3.2 Input 2

```
% Input 2
x2_step1.xoffset = [75.7;75.75];
x2_step1.gain = [2.22222222222224;2.22222222222221];
x2_step1.ymin = -1;
```

6.3.3 Layer 1

```
% Layer 1
b1 = [0.16023374528036077313;0.30815378349772115651;0.12516773019380034171;
    -0.059368878057112439706;-0.003565546231338616832 4;0.15066453668522555853;
    -0.31359109335207147051;-0.42576980571580108625;0.342652470974950496;
    0.27122109008684697207];


IW1_1 = [-0.009439544536299682 0.1855059278073397 0.39149075923728643
 0.18258340544895670932 -0.19254541362616234279 -0.062135134110332057578
 -0.18619833446611094852 0.22876912632492199529 -0.012609215666220197072
 -0.2628296315477101075 -0.23916239670871497491 -0.1003855421219243399
 0.25220314062051801818 -0.13269787130859425628 -0.036416122762133011637
```

```
 -0.24809042792419558343 -0.043812721304375482068 0.15428545008152005913;

                                        ⋮

    0.2128401545129340322 -0.075690388181967674375 -0.3027748428485065868
    -0.13047297654624218088 -0.13456955818537907899 0.10649271512008275098
    -0.22911817130871936499 0.24209748786737034232 0.001920337717883312082 8
    0.039596741622636696945 -0.10298667563021463023 0.077172494653410639298
     0.12460982773129457291 0.12474024508035765069 -0.054837568132443635649
     0.047397578531584667882 0.26065630850593080048 0.39205164410801773833];


IW1_2 = [0.23707464836841626865 0.28771109971665148608 0.14166505386453726034
-0.1826793716919410393 -0.0415004369925778549 -0.11773488623395837738;

                                        ⋮

-0.32965074155695911484 0.043974383003349035159 0.42628596573863908326
0.19122656805211357689 0.1816159603667706901 0.0830664178574416489 64];
```

6.3.4 Layer 2

```
b2 = [-0.038952649132664898812;-0.080272995377121070248];

LW2_1 = [0.46204380267965589058 0.32510260034156163167 -0.41743828449706010808
0.26539171828674007481 0.42257288823339295147 -0.22886852211427280435
0.33154299010265442238 -0.40261055958560659773 -0.56168279793113251586
0.46955975642944697279;
0.39540895848020701786 0.58319474013746364083 -0.34020448110555984167
0.29658783818575634728 -0.38310680362331106519 -0.34452349303493967625
0.47872107961515347752 -0.51593391770612717639 -0.42069343756141402979
0.53074353139763619236];
```

6.3.5 Simulation

```
% Output 1
y1_step1.ymin = -1;
y1_step1.gain = [2.22222222222224;2.22222222222221];
y1_step1.xoffset = [75.7;75.75];

% ===== SIMULATION ========
```

```
% Dimensions
TS = size(x1,2); % timesteps
% Input 1 Delay States
xd1 = mapminmax_apply(xi1,x1_step1);
xd1 = [xd1 zeros(6,1)];

% Input 2 Delay States
xd2 = mapminmax_apply(xi2,x2_step1);
xd2 = [xd2 zeros(2,1)];

% Allocate Outputs
y1 = zeros(2,TS);

% Time loop
for ts=1:TS

    % Rotating delay state position
    xdts = mod(ts+2,4)+1;

    % Input 1
    xd1(:,xdts) = mapminmax_apply(x1(:,ts),x1_step1);

    % Input 2
    xd2(:,xdts) = mapminmax_apply(x2(:,ts),x2_step1);

    % Layer 1
    tapdelay1 = reshape(xd1(:,mod(xdts-[1 2 3]-1,4)+1),18,1);
    tapdelay2 = reshape(xd2(:,mod(xdts-[1 2 3]-1,4)+1),6,1);
    a1 = tansig_apply(b1 + IW1_1*tapdelay1 + IW1_2*tapdelay2);

    % Layer 2
    a2 = b2 + LW2_1*a1;

    % Output 1
    y1(:,ts) = mapminmax_reverse(a2,y1_step1);
end

% Final delay states
finalxts = TS+(1: 3);
xits = finalxts(finalxts<=3);
xts = finalxts(finalxts>3)-3;
```

```
xf1 = [xi1(:,xits) x1(:,xts)];
xf2 = [xi2(:,xits) x2(:,xts)];
end
```