# Statistical Arbitrage Session 2
# **DRAFT**

Austin Gerig

February 25, 2005

## 1 Introduction

We will first look at simple time correlations in the market data. It is important to know these correlations no matter what strategy you are developing because they give insight into how the markets react. Any correlations between successive time intervals are immediately exploitable - they in fact answer the question, given the current or previous change in price, is the next or future price change more likely to be up or down? There are oftentimes large autocorrelations in synthetic markets (linear combinations of different market prices) and you may want to look into a trading strategy based on this.

## 2 1-Dimensional Discrete Random Walk

The time evolution for the price of the futures contracts we are studying can be modeled by a simple 1-dimensional discrete step random walk (1-D brownian motion). At each successive time step, in this case 1 second, the market will either stay at the same price, increase by 1 tick, or decrease by 1 tick (for liquid markets, it infrequently moves more than this). Therefore, we can model the 1-second returns for the futures contract as $x_i$, a random variable that is updated every second with the following possible outcomes $\{-1, 0, 1\}$. More generally, if we assume:

$$E[x_i] = 0, \tag{1}$$
$$E[x_i^2] = \sigma^2, \tag{2}$$
$$E[x_i x_j] = \delta_{ij}\sigma^2, \tag{3}$$

then we can derive the following for the sum of the first $n$ random variables, $X(n) = \sum_{i=1}^{n} x_i$,

$$E[X(n)] = \sum_{i=1}^{n} E[x_i] = 0, \tag{4}$$

$$E[X^2(n)] = \sum_{i=1}^{n} E[x_i x_j] = \sum_{i=1}^{n} E[x_i^2] = n\sigma^2. \tag{5}$$

We see that the variance should increase linearly in time. Fig. 1 is a plot of how the variance, $E[X^2(n)]$, increases as a function of time, $n$, for the E-mini S&P 500 futures contract from October 18th-22nd, 2004. Here and throughout this section, prices are taken as the midpoint of the best bid and best ask.

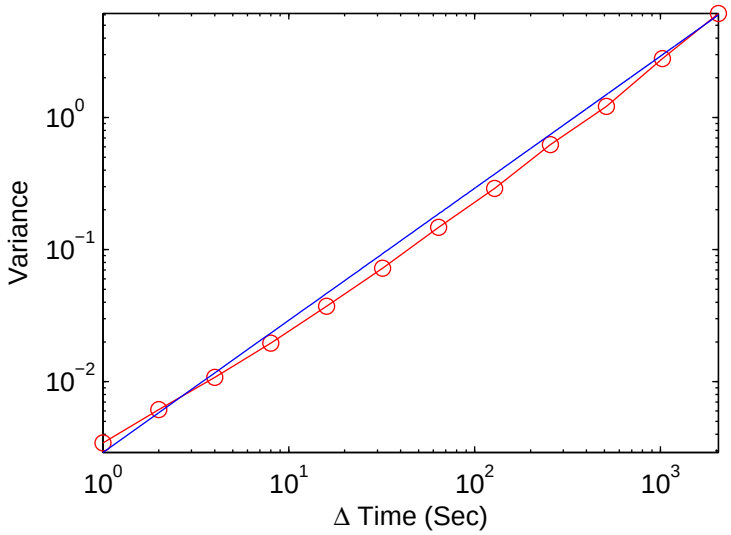


Figure 1: Variance $E[X^2(n)]$ as a function of the step size $n$. Here, the slope is 0.98. For brownian motion, the slope is 1. Data is from the E-mini S&P 500 futures contract from October 18th-22nd, 2004.

On a log-log plot, the slope of the line through the variance points is equal to the exponent of $n$. For brownian motion, this slope is 1. If the slope is greater than 1, the process is superdiffusive, if it is less than 1, the process is subdiffusive. In the case of the E-mini S&P 500, the process begins subdiffusive and later turns superdiffusive. What would a time series with these properties look like? (Possible explanation: subdiffusive = negative autocorrelations, superdiffusive = positive autocorrelations.)

# 3 Autocorrelations

Autocorrelations for the data are plotted in Fig. 2. Negative autocorrelations exist in the data for time steps of 1-8 seconds and decay quickly; no significant autocorrelations exist for time steps of 8-2048 seconds (only 1 and 512 second time steps are shown).

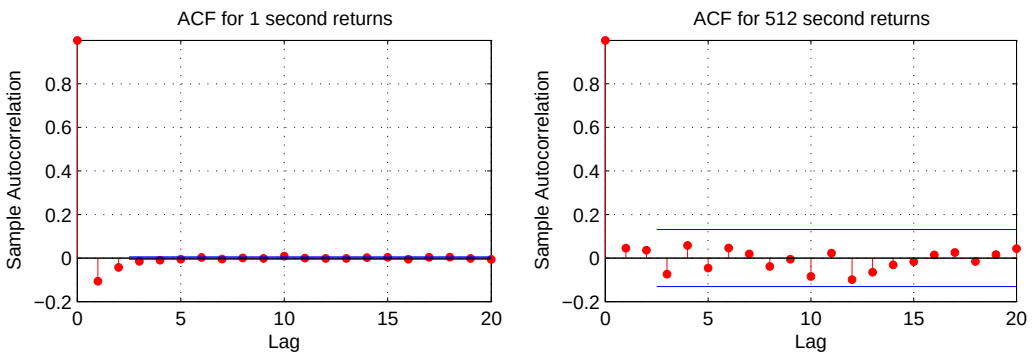


Figure 2: Autocorrelation functions for the E-mini S&P 500 futures contract for the week of October 18th-22nd, 2004 for time steps of 1 second and 512 seconds.

# 4 Clustered Volatility

Markets will often move in fits and spurts (at least when the movement is viewed on time scales $\Delta T > 30$ seconds). Although price changes (probably) remain uncorrelated, the underlying volatility can vary from its traditional mean - staying there for extended periods of time. This phenomenom is known as volatility clustering (markets are often fitted to a GARCH process because these processes can exhibit volatility clustering). Fig. 3 is a plot of the 2.5 minute returns for the same dataset used in Fig. 1. Clustering is difficult to perceive on this limited dataset and at this time scale.

It's possible that the clustering is what causes market distributions to be leptokurtic (high peaks and fat tails). Markets may exist too frequently in two regimes - one where volatility is low, and another where volatility is high. (See below for further discussion).

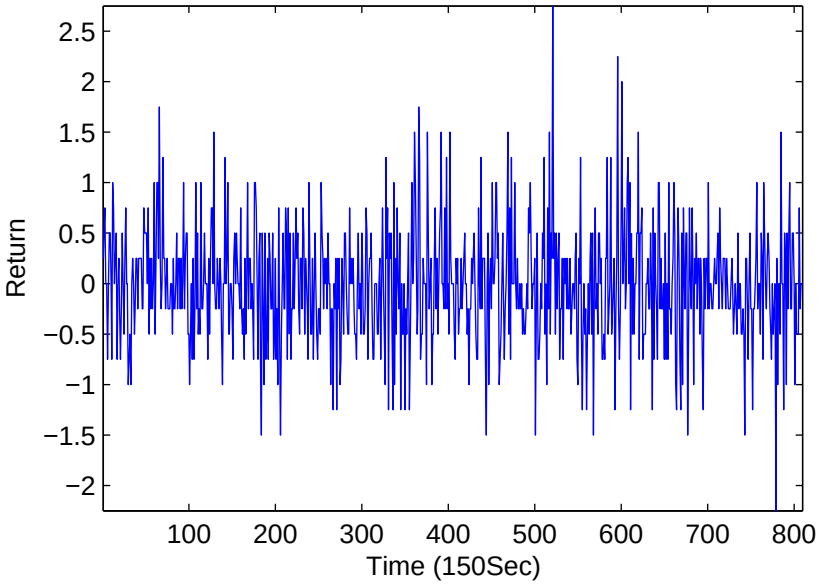


Figure 3: 2.5 minute returns for the E-mini S&P 500 futures contract for the week of October 18th-22nd, 2004.

# 5 Time Inhomogeneous Data

Markets are updated inhomogeneously in time. When we put regular time stamps on data to form a time series, we are forcing the market data into bins that are not natural to it. Another way to form a time series from the data is to consider the fundamental step size to be one update in the price of the market. When we use this series, volatility clustering should only occur if updates to the market arrive clumpy - but this usually is not the case. For example, for the popular E-mini contracts traded on CME, price updates rarely are larger than 1 tick (disregarding periods when economic numbers are released). Considering one tick to be the fundamental step size, the variance curve plotted in Fig. 1 changes to that plotted in Fig. 4. From the figure, the market is subdiffusive from 1 tick step to 2 tick steps (it is negatively correlated on this time scale - this would be difficult to exploit because we can only transact at the bid or ask). Otherwise the series appears to behave like brownian motion. See Fig. 5 for autocorrelations of the data when we use this newly formed series. Negative autocorrelations exist for steps of 1 tick, all other timesteps give insignificant autocorrelations (only ACF's for 1 tick and 32 tick time-steps are shown).

Comparing Figs. 1 and 4, why does the market appear to behave differently when we use different fundamental steps to generate the series? One possible explanation is that the return distributions may become leptokurtic at larger $n$ (this happens because the market data is inhomogeneous in time and volatility clusters - see Clark, 1973). The more leptokurtic the distribution, the larger the variance - which can explain why the slope is greater

than 1 at large $n$ values in Fig. 1.

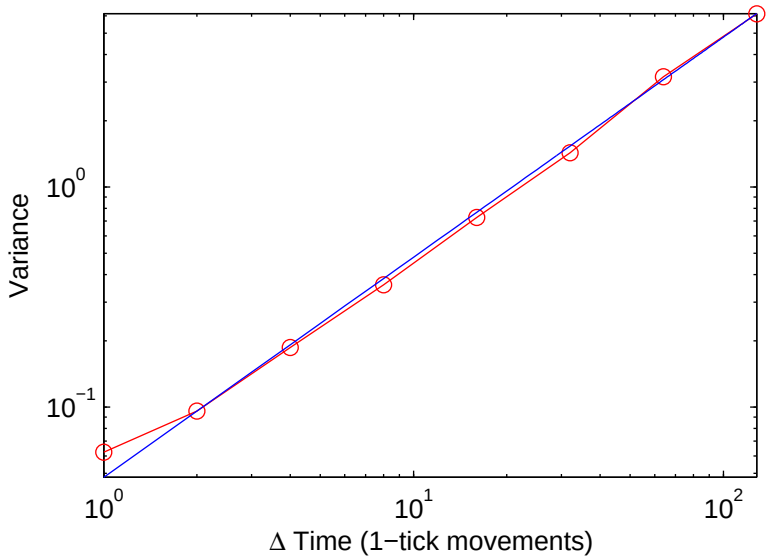


Figure 4: Variance $E[X^2(n)]$ as a function of the step size $n$, in this case, $n$ = 1-tick. The fitted slope is 0.97. For brownian motion, the slope is 1. Data is from the E-mini S&P 500 futures contract from October 18th-22nd, 2004.

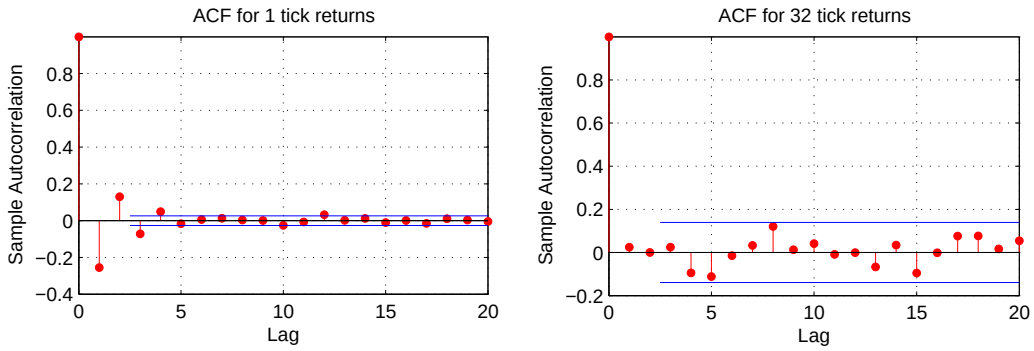


Figure 5: Autocorrelation functions for the E-mini S&P 500 futures contract for the week of October 18th-22nd, 2004 for time steps of 1 price update and 32 price updates.

# 6 Summary

Serial correlations are important because they are immediately exploitable and they provide information about the nature of the time series. The price series of the E-mini S&P futures contract fits fairly nicely to a model of brownian motion when time steps are considered updates in market price

(however, negative autocorrelations exist at 1 time step). When doing analysis, keep in mind that time can be considered in seconds (resulting in an inhomogeneous series) or in market price updates (resulting in a homogeneous series) - and these, in general, will have different properties.

# 7 References

P. K. Clark, 'A Subordinated Stochastic Process Model with Finite Variance for Speculative Prices', *Econometrica* **41**, 135-256 (1973).

# 8 Matlab Source Code

## 8.1 1-Dimensional Discrete Random Walk

```
security1='MiniSP';
tick=.25;
month='10';
days=18:22;
year='04';
Asec=[];
deltatime=[];
CovPlot=[];

for i=days
    day=int2str(i);
    tmp = load([security1 '_' month ...
     '.' day '.' year '_BookSec.csv']);
    Asec=[Asec; ((tmp(:,3)+tmp(:,18))/2)'];
end
Asec=Asec/100;
clear tmp

AReturn=diff(Asec')';
for k=0:11
    i=2^k;
    AReturntmp=[];
    for j=1:length(days)
        for n=1:i:(length(AReturn(j,:)))-(i-1)
            tmp(1+(n-1)/i)=sum(AReturn(j,n:n+(i-1)));
        end
        AReturntmp=[AReturntmp, tmp];
        clear tmp
    end
    deltatime=[deltatime,i];
```

```
    CovPlot=[CovPlot, cov(AReturntmp)];
end

b = regress(CovPlot',deltatime');
fitx = 0:max(deltatime);
fity = b*fitx;

figure(1)
plot(deltatime, CovPlot,'r-o')
hold on
plot(fitx,fity)
xlabel('\Delta Time (Sec)')
ylabel('Variance')

figure(2)
loglog(deltatime, CovPlot,'r-o')
hold on
loglog(fitx,fity)
xlabel('\Delta Time (Sec)')
ylabel('Variance')

ab=log(deltatime');
cd=log(CovPlot');
polyfit(ab,cd,1)
polyfit(ab(1:6),cd(1:6),1)
polyfit(ab(6:12),cd(6:12),1)
```

## 8.2 Autocorrelations

```
security1='MiniSP';
tick=.25;
month='10';
days=18:22;
year='04';
Asec=[];
deltatime=[];
CovPlot=[];

for i=days
    day=int2str(i);
    tmp = load([security1 '_' month ...
     '.' day '.' year '_BookSec.csv']);
    Asec=[Asec; ((tmp(:,3)+tmp(:,18))/2)'];
end
Asec=Asec/100;
```

```
clear tmp

AReturn=diff(Asec')';
for k=1:12
    i=2^(k-1);
    AReturntmp=[];
    for j=1:length(days)
        for n=1:i:(length(AReturn(j,:)))-(i-1)
            tmp(1+(n-1)/i)=sum(AReturn(j,n:n+(i-1)));
        end
        AReturntmp=[AReturntmp, tmp];
        clear tmp
    end
    figure(k)
    autocorr(AReturntmp, [], 2);
    title(['ACF for ' num2str(i) ' second returns'])
end
```

## 8.3 Clustered Volatility

```
security1='MiniSP';
tick=.25;
month='10';
days=18:22;
year='04';
Asec=[];
AReturn150sec=[];

for i=days
    day=int2str(i);
    tmp = load([security1 '_' month ...
     '.' day '.' year '_BookSec.csv']);
    Asec=[Asec; ((tmp(:,3)+tmp(:,18))/2)'];
end
Asec=Asec/100;
clear tmp

AReturn=diff(Asec')';
for i=1:length(days)
    for n=1:150:(length(AReturn(i,:)))-149
        tmp(1+(n-1)/150)=sum(AReturn(i,n:n+149));
    end
    AReturn150sec=[AReturn150sec, tmp];
    clear tmp
end
```

```
figure(1)
plot(AReturn150sec)
xlabel('Time (150Sec)')
ylabel('Return')
```

## 8.4 Time Inhomogeneous Data

### Variance

```
security1='MiniSP';
tick=.25;
month='10';
days=18:22;
year='04';
Asec=[];
deltatime=[];
CovPlot=[];

for i=days
    day=int2str(i);
tmp = load([security1 '_' month ...
     '.' day '.' year '_BookSec.csv']);
    Asec=[Asec; ((tmp(:,3)+tmp(:,18))/2)'];
end
Asec=Asec/100;
clear tmp

AReturn=diff(Asec')';
for k=1:8
    i=2^(k-1);
    AReturntmp=[];
    for j=1:length(days)
        tmp1=AReturn(j,:);
        tmp1=tmp1(tmp1~=0);
        for n=1:i:(length(tmp1)-(i-1))
            tmp(1+(n-1)/i)=sum(tmp1(n:n+(i-1)));
        end
        AReturntmp=[AReturntmp, tmp];
        clear tmp
    end
    deltatime=[deltatime,i];
    CovPlot=[CovPlot, cov(AReturntmp)];
end
```

```
b = regress(CovPlot',deltatime');
fitx = 0:max(deltatime);
fity = b*fitx;

figure(1)
plot(deltatime, CovPlot,'r-o')
hold on
plot(fitx,fity)
xlabel('\Delta Time (1-tick movements)')
ylabel('Variance')

figure(2)
loglog(deltatime, CovPlot,'r-o')
hold on
loglog(fitx,fity)
xlabel('\Delta Time (1-tick movements)')
ylabel('Variance')

ab=log(deltatime');
cd=log(CovPlot');
polyfit(ab,cd,1)
```

### Autocorrelations

```
security1='MiniSP';
tick=.25;
month='10';
days=18:22;
year='04';
Asec=[];
deltatime=[];
CovPlot=[];

for i=days
    day=int2str(i);
    tmp = load([security1 '_' month ...
     '.' day '.' year '_BookSec.csv']);
    Asec=[Asec; ((tmp(:,3)+tmp(:,18))/2)'];
end
Asec=Asec/100;
clear tmp

AReturn=diff(Asec')';
for k=1:8
    i=2^(k-1);
    AReturntmp=[];
```

```
    for j=1:length(days)
        tmp1=AReturn(j,:);
        tmp1=tmp1(tmp1~=0);
        for n=1:i:(length(tmp1)-(i-1))
            tmp(1+(n-1)/i)=sum(tmp1(n:n+(i-1)));
        end
        AReturntmp=[AReturntmp, tmp];
        clear tmp
    end
    figure(k)
    autocorr(AReturntmp, [], 2);
    title(['ACF for ' num2str(i) ' tick returns'])
end
```

# Statistical Arbitrage Session 3
# **DRAFT**

Austin Gerig

March 4th, 2005

## 1 Introduction

Traders use a variety of different technical indicators to determine entrance and exit signals. This chapter reviews several of the more popular ones, and is meant to be used as a reference if you decide to form a strategy based on them.

## 2 Moving Averages

A simple moving average is a time series consisting of the average value of market prices over some time-window. If the original market price series is $x_1, x_2, ..., x_N$. The moving average at time-step $n$ given a window size $s$, $\mathrm{MA}_s[n]$, is defined:

$$\mathrm{MA}_s[n] = (1/s)\sum_{i=0}^{s-1} x_{n-i}. \tag{1}$$

An exponential moving average is similar to a simple moving average, but the original price series is weighted exponentially, giving more importance to recent prices. It is most often defined iteratively:

$$\mathrm{EMA}_s[n] = \mu \cdot x_n + (1-\mu)\mathrm{EMA}_s[n-1], \tag{2}$$

where,

$$\mu = \frac{2}{1+s}. \tag{3}$$

Moving averages are used as signal entrances in several ways. Most often, signals are generated by the underlying market price crossing one of the moving averages, or by two or more moving averages crossing each other. For example, when a stock crosses below its 50-day moving average, this is typically considered bearish and a sell signal is generated (this is a continuation strategy).

# 3 Bollinger Bands

Bollinger bands consist of three series, a simple moving average, $\mathrm{MA}_s[n]$, and two series defined by $(\mathrm{MA}_s[n] \pm 2\sigma_s[n])$ where $\sigma_s[n]$ is the standard deviation at time $n$:

$$\sigma_s[n] = \sqrt{(1/s)\sum_{i=0}^{s-1}(x_{n-i} - \bar{x}_{n-i})^2}. \tag{4}$$

Bollinger bands not only monitor the price activity of the underlying, but they also offer a picture of the underlying's current volatility. They are used to determine extreme price movements.

# 4 Momentum

Momentum is defined as the time series of price returns:

$$\mathrm{M}_s[n] = x_n - x_{n-s}. \tag{5}$$

Momentum series can also be generated for moving averages and other technical indicators - in this case momentum is defined as the difference of values for the underlying indicator. Signals are triggered when momentum crosses above or below a certain threshold. Typically, traders will buy into large momentum and sell into large negative momentum - this is a continuation strategy.

# 5 Relative Strength Index

The relative strength index (RSI) was developed by Welles Wilder in 1978 and is a widely used indicator of overbought/oversold market conditions. It is defined:

$$\mathrm{RSI}_s[n] = 100 - \frac{100}{1 + U/D}, \tag{6}$$

$$U = E[\,\mathrm{M}_1[n-i] \,:\, \mathrm{M}_1[n-i] > 0 \text{ where } i = 0, \ldots, (s-1)\,], \tag{7}$$

$$D = E[\,\mathrm{M}_1[n-i] \,:\, \mathrm{M}_1[n-i] < 0 \text{ where } i = 0, \ldots, (s-1)\,]. \tag{8}$$

$U$ and $D$ are the average of all of the up and down price moves respectively. Markets are supposed to top when the indicator reaches 70 and bottom when the indicator reaches 30. It is used in reversal strategies.

# 6 Moving Average Convergence Divergence

The moving average convergence divergence consists of two series:

$$\mathrm{MACD}_{s_1,s_2}[n] = \mathrm{EMA}_{s_1}[n] - \mathrm{EMA}_{s_2}[n], \tag{9}$$

$$\mathrm{EMA}_{s_3}^{\mathrm{MACD}_{s_1,s_2}}[n], \tag{10}$$

where $\mathrm{EMA}_{s_3}^{\mathrm{MACD}_{s_1,s_2}}[n]$ means the exponential moving average is of $\mathrm{MACD}_{s_1,s_2}[n]$ and not the usual $x_1, x_2, ..., x_N$. The collection of these series is denoted $\mathrm{MACD}(s_1, s_2, s_3)$... typically $\mathrm{MACD}(12, 26, 9)$ is used. Signals are generated when the two series cross, or by the overall trend of $\mathrm{MACD}_{s_1,s_2}[n]$.

# 7 Pivot Points

Pivot points are used to determine support and resistance levels for the market price. They are calculated as follows:

$$\begin{aligned}
R2_s[n] &= P_s[n] + (H - L), && (11)\\
R1_s[n] &= 2P_s[n] - L, && (12)\\
P_s[n] &= (H + L + C)/3, && (13)\\
S1_s[n] &= 2P_s[n] - H, && (14)\\
S2_s[n] &= P_s[n] - (H - L), && (15)
\end{aligned}$$

where,

$$\begin{aligned}
H &= \max\{x_n, x_{n-1}, ..., x_{n-s+1}\}, && (16)\\
L &= \min\{x_n, x_{n-1}, ..., x_{n-s+1}\}, && (17)\\
C &= x_n. && (18)
\end{aligned}$$

$R1$ and $R2$ are levels of resistance and $S1$ and $S2$ are levels of support.

# 8 Market Patterns

See the brochure from Focus Trading, Inc.

# 9 Summary

Several common technical indicators were introduced. Widespread knowledge of indicators by market participants can affect markets - a relatively large subset of traders may 'synchronize' their trades by following and trading a popular indicator, self-fulfilling its prophecy (see the Lamper paper below). It would be useful to statistically analyze the predictive power of these indicators.

# 10 References

L. N. Kestner, *Quantitative Trading Strategies*, (McGraw-Hill, New York, 2003).
D. Lamper, S.D. Howison, N.F. Johnson, Phys. Rev. Lett. **88**, 017902 (2002).

# Statistical Arbitrage Session 4
# **DRAFT**

Austin Gerig

March 11th, 2005

## 1 Introduction

In the past decade, several methods originally developed for computer science/complexity theory have been used to search for patterns or trends in market data. In this session, we will examine two of these methods - neural networks and genetic algorithms.

## 2 Neural Networks

A neural network is a collection of interconnected processing units (the human brain is a neural network). The processing units, called neurons or nodes, receive information from other nodes or an outside source, process it, and then communicate their results to other nodes in the network. We will be discussing artificial neural networks (although I will just call them neural networks here). These are artificially created neural networks used to model complex processes. Neural networks can be used to find patterns in market data.

### 2.1 A Simple Example

There are several different ways to setup the structure of a neural net. Here, I will use a simple feedforward (one-directional) network with four input nodes, two hidden layer nodes, and one output node. A diagram of the model is shown is Fig. 1.

The details of the model are as follows. The 1-second returns of the E-mini S&P 500 futures contract (price is assumed to be the midpoint of the best bid and best ask and data is from October 18th-22nd, 2004) are mapped to 0 if negative, 1 if positive, and are ignored if zero. This new series is denoted $x_i$. The previous four instances of this series are treated as the input vector, $I = [x_{t-1}, x_{t-2}, x_{t-3}, x_{t-4}]$. The lines in the diagram from the input nodes to the hidden nodes represent different weightings - these

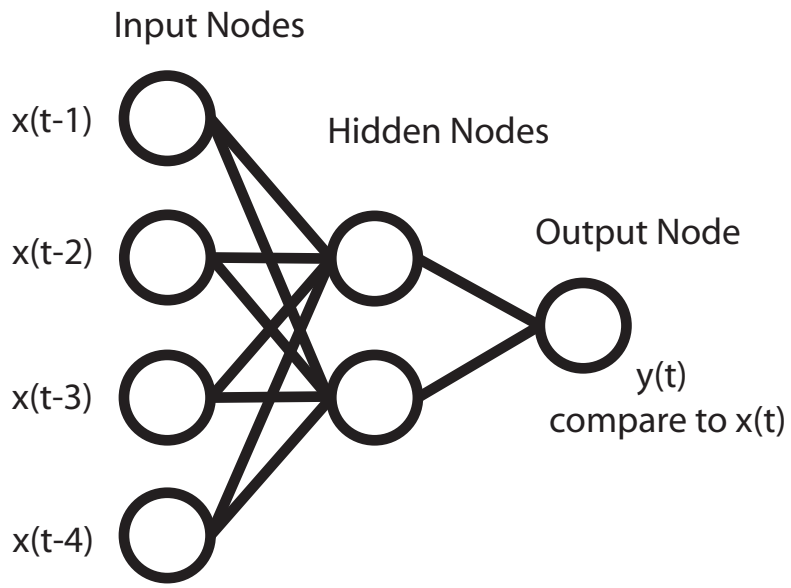


Figure 1: Diagram of the neural network discussed above. The previous four changes in price for the E-mini S&P 500 futures contract are used as inputs. The neural net is trained with the current change in price as the output.

weightings are $W_j^h = [w_{1j}^h, w_{2j}^h, w_{3j}^h, w_{4j}^h]$ where $j$ indexes the hidden nodes. The output values of the hidden layer, $h_j$ are calculated as follows:

$$h_j(t) = f(\sum_{i=1}^{4} x_{t-i} \cdot w_{ij}^h), \tag{1}$$

where,

$$f(X) = \frac{1}{1+e^{-X}}. \tag{2}$$

$f(\cdot)$ is used to keep input and output values between zero and one. The outputs of the hidden layer are then treated as inputs to the final output node:

$$y_t = f(\sum_{j=1}^{2} h_j(t) \cdot w_j^o), \tag{3}$$

where the superscript $o$ refers to the output weights, $W^o = [w_1^o, w_2^o]$. This final value, $y_t$, can then be compared to $x_t$.

The neural net is trained to output $x_t$, meaning that the error $e_t = x_t - y_t$ is used to adjust the original weightings $W_j^h$ and $W^o$ via a backpropagation algorithm:

$$w_j^o = w_j^o + r \cdot e_t \cdot h_j(t) \cdot y_t(1-y_t), \tag{4}$$

$$w_{ij}^h = w_{ij}^h + r \cdot e_t \cdot x_{t-i} \cdot h_j(t)(1-h_j(t)), \tag{5}$$

where $r$ is the learning rate, a value between 0 and 1. The neural net is trained using the first 3 days of data. A trading strategy is developed based on the

resulting weights and is tested on the final 2 days of data (see source code for details). The cumulative profit of the trading strategy is plotted in Fig. 2. The average profit per trade was .32 ticks - meaning the neural net was correctly predicting 2 out of 3 movements in the market. This result is not surprising - the neural net is just picking up on the negative autocorrelations of the 1-second returns for the E-mini S&P contract.

Instead of using a backpropagation algorithm to determine weights, it is oftentimes beneficial to use a more brute force approach (notice that the weightings determined above will be skewed towards minimizing error for the latter values of $x_i$ - this is because the neural net is always adjusting to the most recent data). In order to treat all errors equally, the following algorithm is used. A set of weightings is determined randomly or according to a 'best guess' and then the mean squared error is calculated, MSE $= E[e_t^2]$. This is calculated for many different sets of weightings and the set with the least MSE is chosen. (Instead of blindly sampling sets of weightings, genetic algorithms can be used to search the state space of weightings - see below for an explanation of GA's).

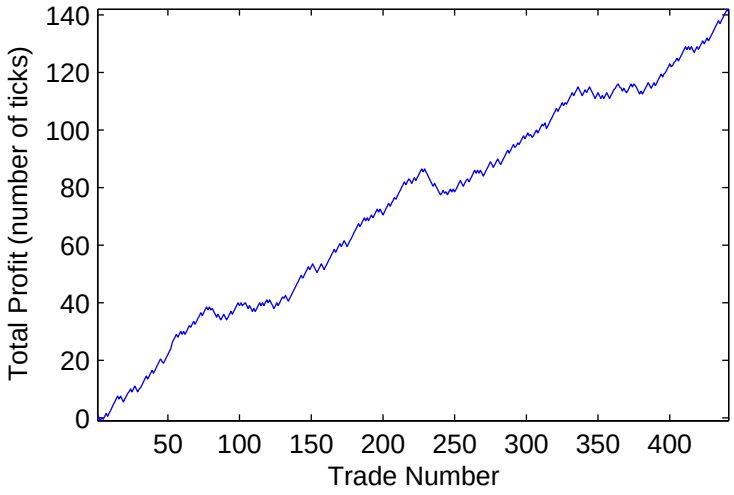


Figure 2: Cumulative profit (in ticks) for a trading strategy based on the predictions of the neural net described above.

# 3 Genetic Algorithms

A genetic algorithm is an algorithm that uses techniques from evolutionary biology to find approximate solutions for an optimization problem. In the same way that evolutionary processes optimize an organism's structure for survival, a genetic algorithm will optimize a collection of parameters for a predefined fitness. Genetic algorithms are useful when a brute force approach is too time consuming.

A genetic algorithm can be described as follows:

1. Choose an initial population of $N$ possible solutions (chromosomes).
2. Assign a weight to each chromosome based on its fitness.
3. Select two chromosomes at random based on their weight.
4. Crossover and mutate these chromosomes with predefined probabilities.
5. Repeat steps 2 and 3 until the new population has $N$ members.
6. Repeat steps 2-5 until a satisfactory solution is found.

## 3.1 A Simple Example

Suppose we wish to optimize a collection of $n$ parameters

$$C = \{p_i : i = 1, 2, \ldots, n \text{ and } 0 \leq p_i \leq 9\} \tag{6}$$

for a predefined fitness, let's say

$$F = \sum_{i=1}^{n} p_i^2. \tag{7}$$

This means we wish to find a $C$ such that $F$ is maximized. If the solution is given in the following form $C^* = \{p_i^*, p_{i+1}^*, \ldots, p_n^*\}$ where $^*$ denotes optimal, then it can easily be shown that $C^* = \{9, 9, \ldots, 9\}$.

Suppose, however, that we didn't know $C^*$. We could solve the problem by brute force, calculating $F$ for all possible $C$ and choosing the $C$ that gives a maximum $F$; this would involve $10^n$ calculations. Alternatively, we could use a genetic algorithm. The results of using a genetic algorithm to solve this problem for $n = 7$ are shown below. Fig. 3 plots fitness of the population as a function of time (generations). The population rather quickly approaches a limiting value - and the fittest chromosome for this run was $\{9, 9, 9, 9, 9, 9, 9\}$, which is the optimal solution. Good solutions are common ($F > 500$) after only 50 generations - or $(50 \cdot 100 = 5000)$ calculations of $F$. In comparison, brute force would require $(10^7 = 10,000,000)$ calculations of $F$.

## 3.2 Optimizing Strategies

Genetic algorithms are useful for optimizing the parameters of a trading strategy, especially when there are many parameters to be optimized and a brute force approach is not possible. Because we wish to optimize profitability, the fitness function will be a function of profitability. Several references are provided below where a genetic algorithm is used to optimize a trading strategy.

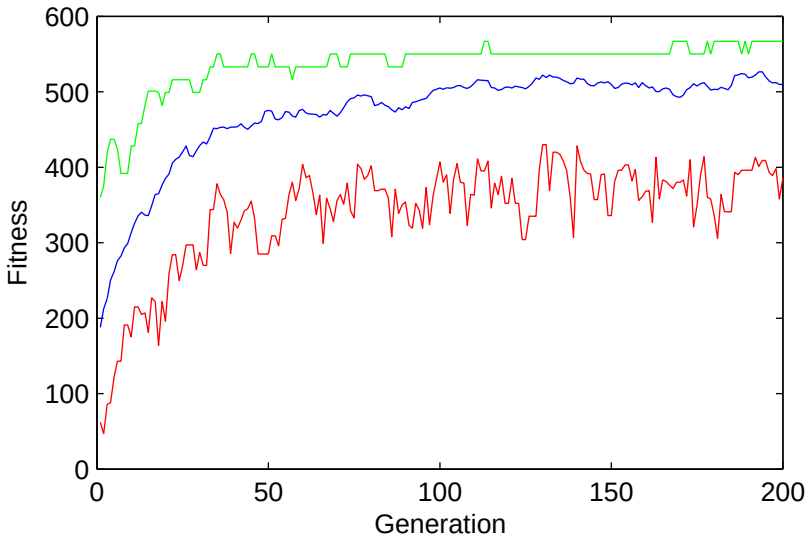


Figure 3: Fitness as a function of the number of generations. The top plot is the fitness, $F$, of the fittest chromosome in the population, the middle plot is the average fitness of the population, and the bottom plot is the fitness of the least fit chromosome in the population. See source code below for parameter values

### 3.3 Pattern Recognition

Genetic algorithms can also be used for pattern recognition in market data. Consider the following. Create a population of chromosomes that are each a binary string, e.g., $\{1,0,1,1,1,0\}, \{1,0,1,1,1,0\}, \ldots$ Using the same time series, $x_t$, from the neural net example, define a fitness function:

$$F = \sum_t \sum_{i=1}^{n} \left| (p_i - 0.5) \cdot (x_{t-n+i} - 0.5) \right| . \tag{8}$$

This would select for chromosomes that match the structure of the market.

## 4 Other Models

Markov Models (Epsilon Machines), Independent Component Analysis (Principal Component Analysis), Wavelet Transform (Windowed Fourier Transform), Power Spectrum, Optimal Portfolio Rebalancing (Switching Portfolios).

## 5 Summary

Neural networks and genetic algorithms can be used to search for patterns in market data. Genetic algorithms can also be used to optimize the parameters of a trading strategy.

# 6 References

## 6.1 Neural Networks

C. L. Dunis, X. Huang, "Forecasting and Trading Currency Volatility: An Application of Recurrent Neural Regression and Model Combination," *Journal of Forecasting*, **21**, 317-354 (2002).

P. J. Bolland, J. T. Connor, A. N. Refenes, "Application of Neural Networks to Forecast High Frequency Data: Foreign Exchange," *Nonlinear Modelling of High Frequency Financial Time Series*, Edited by C. Dunis and B. Zhou, 225-246 (1998).

## 6.2 Genetic Algorithms

C. Dunis et al., "An Application of Genetic Algorithms to High Frequency Trading Models: A Case Study," *Nonlinear Modelling of High Frequency Financial Time Series*, Edited by C. Dunis and B. Zhou, 247-278 (1998).

C. Neely, P. Weller, R. Dittmar, "Is Technical Analysis in the Foreign Exchange Market Profitable? A Genetic Programming Approach," *The Journal of Financial and Quantitative Analysis*, **32**, 405-426 (1997).

## 6.3 Other Models

Do a Google search for these names.

# 7 Matlab Source Code

## 7.1 Neural Networks

```
security1='MiniSP';
tick=.25;
month='10';
days=18:22;
year='04';
Asec=[];
input_nodes=4;
hidden_nodes=2;
learning_rate=.25;

for i=days
    day=int2str(i);
    tmp = load([security1 '_' month ...
    '.' day '.' year '_BookSec.csv']);
```

```
    Asec=[Asec; ((tmp(:,3)+tmp(:,18))/2)'];
end
Asec=Asec/100;
clear tmp
AReturn=diff(Asec')';

h_input_weights=rand(input_nodes,hidden_nodes)
h_output_weights=rand(hidden_nodes,1)

for i=1:length(days)-2
    tmp=AReturn(i,:);
    tmp=tmp(tmp~=0);
    tmp1=tmp>0;

    for j=1:length(tmp1)-input_nodes
        h_input=tmp1(j:j+input_nodes-1);
        h_output=1./(1+exp(-h_input*h_input_weights));
        output=1./(1+exp(-h_output*h_output_weights));
        target=tmp1(j+input_nodes);
        error=target-output;

        h_output_weights=h_output_weights+learning_rate*...
        error*h_output'*output*(1-output);
        for k=1:hidden_nodes
            temp(:,k)=h_input'*h_output(k)*(1-h_output(k));
        end
        h_input_weights=h_input_weights+learning_rate...
        *error*temp;
    end
end

h_input_weights
h_output_weights

for i=1:2^input_nodes
    state(i,:)=dec2binvec(i-1,input_nodes);
    h_output=1./(1+exp(-state(i,:)*h_input_weights));
    output=1./(1+exp(-h_output*h_output_weights));
    prediction(i)=output;
end
prediction
state(prediction==max(prediction),:)
state(prediction==min(prediction),:)

profit=0;
```

```
profitplot=[];
for i=length(days)-1:length(days)
    tmp=AReturn(i,:);
    tmp=tmp(tmp~=0);
    tmp1=tmp>0;

    for j=1:length(tmp1)-input_nodes
        h_input=tmp1(j:j+input_nodes-1);
        h_output=1./(1+exp(-h_input*h_input_weights));
        output=1./(1+exp(-h_output*h_output_weights));

        if output==max(prediction)
            profit=profit+tmp(j+input_nodes);
            profitplot=[profitplot, profit];
        end
        if output==min(prediction)-1
            profit=profit-tmp(j+input_nodes);
            profitplot=[profitplot, profit];
        end
    end
end

profitplot=profitplot/tick;
profitplot(length(profitplot))/length(profitplot)

figure(1)
plot(profitplot)
xlabel('Trade Number')
ylabel('Total Profit (number of ticks)')
```

## 7.2 Genetic Algorithms

```
numgens=200;    %number of generations
n=7;            %number of parameters in chromosome
maxvalue=9;     %max value of parameter
popsize=100;    %number of chromosomes in population
pcross=.7;      %probability of crossover
pmutate=.01;    %probability of mutation
pop=[];

%initialize population
for i=1:popsize
    pop=[pop;round(maxvalue*rand(1,n))];
end
pop
```

```
%determine fitness
for i=1:popsize
    fit(i)=sum(pop(i,:).^2);
end
avgfit=[(sum(fit))/popsize];
maxfit=[max(fit)];
minfit=[min(fit)];


%main loop
for i=1:numgens

    %create new population
    poptmp=[];
    for j=1:round(popsize/2)

        %select two chromosomes based on fitness
        for k=1:popsize
            prob(k)=fit(k)/sum(fit);
        end
        [tmp1,tmp2]=sort([rand(1),cumsum(prob)]);
        selected1=pop(find(tmp2==1),:);
        [tmp1,tmp2]=sort([rand(1),cumsum(prob)]);
        selected2=pop(find(tmp2==1),:);
        clear tmp1 tmp2

        %crossover and mutate
        if pcross>=rand(1)
            point=1+round((n-2)*rand(1));
            tmp1=[selected1(1:point),selected2(point+1:n)];
            tmp2=[selected2(1:point),selected1(point+1:n)];
        else
            tmp1=selected1;
            tmp2=selected2;
        end
        for k=1:n
            if pmutate>=rand(1)
                tmp1(k)=round(maxvalue*rand(1));
            end
        end
        for k=1:n
            if pmutate>=rand(1)
                tmp2(k)=round(maxvalue*rand(1));
            end
        end
```

```
        poptmp=[poptmp; tmp1; tmp2];
    end

    %set to new population
    if length(poptmp(:,1)>popsize)
        poptmp=poptmp(1:popsize,:);
    end
    pop=poptmp;

    %determine fitness
    for j=1:popsize
        fit(j)=sum(pop(j,:).^2);
    end
    avgfit=[avgfit,(sum(fit))/popsize];
    maxfit=[maxfit,max(fit)];
    minfit=[minfit,min(fit)];

end

%display results
pop
tmp=find(fit==max(fit));
pop(tmp(1),:)
figure(1)
plot(avgfit,'b')
hold on
plot(maxfit,'g')
plot(minfit,'r')
xlabel('Generation')
ylabel('Fitness')
```