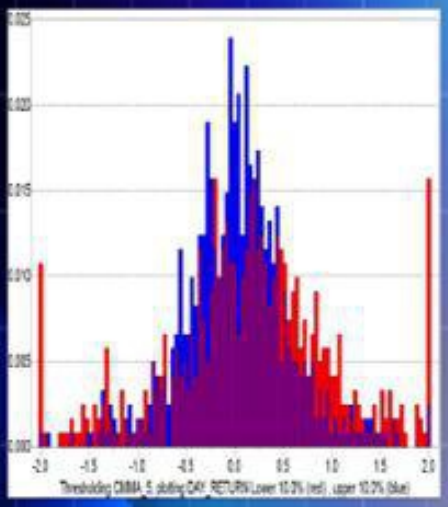
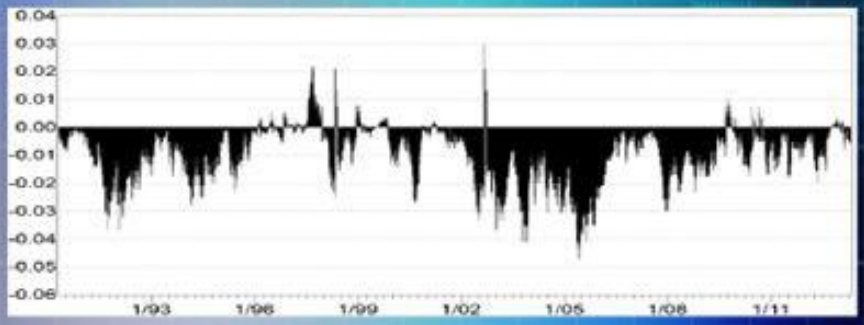
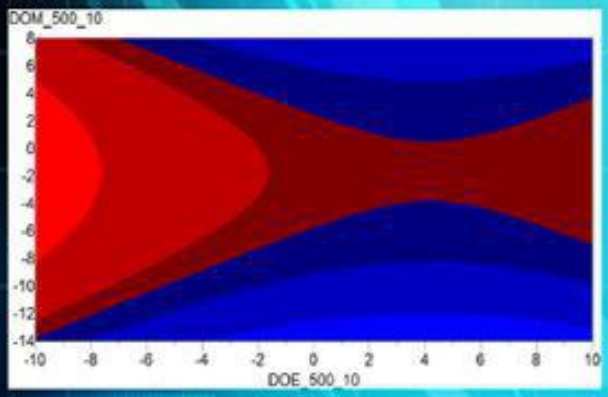
Statistically Sound Machine Learning for Algorithmic Trading of Financial Instruments

Developing Predictive-Model-Based Trading Systems Using TSSB







David Aronson
Timothy Masters

Statistically Sound Machine Learning for Algorithmic Trading of Financial Instruments

Developing Predictive-Model-Based Trading Systems Using *TSSB*

David Aronson

with

Timothy Masters, Ph.D.
Technical Advisor

Edition 1.20

Great effort has been undertaken to ensure that the content of this book, as well as the *TSSB* program, are correct. However, errors and omissions are impossible to avoid in a work of this extent. Neither this book nor the *TSSB* program are meant as providing professional advice. No guarantee is made that these items are free of errors and omissions, and the reader assumes full liability for any losses associated with use of these items.

About this book:

This is a tutorial, with instruction organized from easy to sophisticated. If the reader just skims through the entire text, hoping to gain an idea of how to use the *TSSB* program, the reader will be hopelessly dismayed by the vast complexity of options. The correct approach is to begin with the first, very simple example and implement it. Then progress to the next, and so forth. Each example builds on experience gained from prior examples. In this way, the reader will painlessly become familiar with the program.

The *TSSB* program may be downloaded (free) from the *TSSB* website: http://tssbsoftware.com/

ISBN 978-1489507716

About David Aronson

David Aronson is a pioneer in machine learning and nonlinear trading system development and signal boosting/filtering. He has worked in this field since 1979. His accomplishments include:

- Started Raden Research Group in 1982 where he oversaw the development of PRISM (Pattern Recognition Information Synthesis Modeling).
- Chartered Market Technician certified byTheMarket Technicians Association since 1992.
- Proprietary equities trader for Spear, Leeds and Kellogg 1997 – 2002.
- Was an Adjunct Professor of Finance, teaching a graduate level course in technical analysis, data mining and predictive analytics to MBA and financial engineering students from 2002 to 2011.
- Author of "Evidence Based Technical Analysis" published by John Wiley & Sons 2006. This was the first popular book to deal with data mining bias and the Monte Carlo Permutation Method for generating bias-free p-values.
- Co-designer of TSSB (Trading System Synthesis and Boosting), a software platform for the automated development of statistically sound predictive-model-based trading systems.
- Developed a method for indicator purification and Pure VIX.
- Innovated the concept of signal boosting: using machine learning to enhance the performance of existing strategies.

About Timothy Masters

Timothy Masters received a PhD in mathematical statistics with a specialization in numerical computing. Since then he has continuously worked as a consultant for government and industry.

- Early research involved automated feature detection in high-altitude photographs while developing applications for flood and drought prediction, detection of hidden missile silos, and identification of threatening military vehicles.
- Worked with medical researchers in the development of computer algorithms for distinguishing between benign and malignant cells in needle biopsies.

- Current focus is methods for evaluating automated financial market trading systems.

- Authored five books on prediction, classification, and applications of neural networks:

  Practical Neural Network Recipes in C++ (Academic Press, 1993)
  Signal and Image Processing with Neural Networks (Wiley, 1994)
  Advanced Algorithms for Neural Networks (Wiley, 1995)
  Neural, Novel, and Hybrid Algorithms for Time Series Prediction (Wiley, 1995)
  Assessing and Improving Prediction and Classification (CreateSpace, 2013)

More information can be found on the author's website: TimothyMasters.info

Table of Contents

Volatility Indicators
- ABS PRICE CHANGE OSCILLATOR ShortLen Multiplier
- PRICE VARIANCE RATIO HistLength Multiplier
- MIN PRICE VARIANCE RATIO HistLen Mult Mlength
- CHANGE VARIANCE RATIO HistLength Multiplier
- MIN CHANGE VARIANCE RATIO HistLen Mult Mlen
- ATR RATIO HistLength Multiplier
- DELTA PRICE VARIANCE RATIO HistLength Multiplier
- DELTA CHANGE VARIANCE RATIO HistLength Multiplier
- DELTA ATR RATIO HistLength Multiplier
- BOLLINGER WIDTH HistLength
- DELTA BOLLINGER WIDTH HistLength DeltaLength
- N DAY NARROWER HistLength
- N DAY WIDER HistLength

Indicators Involving Indices
- INDEX CORRELATION HistLength
- DELTA INDEX CORRELATION HistLength DeltaLength
- DEVIATION FROM INDEX FIT HistLength MovAvgLength
- PURIFIED INDEX Norm HistLen Npred Nfam Nlooks Look1

Basic Price Distribution Statistics
- PRICE SKEWNESS HistLength Multiplier
- CHANGE SKEWNESS HistLength Multiplier
- PRICE KURTOSIS HistLength Multiplier
- CHANGE KURTOSIS HistLength Multiplier
- DELTA PRICE SKEWNESS HistLen Multiplier DeltaLen
- DELTA CHANGE SKEWNESS HistLen Multiplier DeltaLen
- DELTA PRICE KURTOSIS HistLen Multiplier DeltaLen
- DELTA CHANGE KURTOSIS HistLen Multiplier DeltaLen

Indicators That Significantly Involve Volume
- VOLUME MOMENTUM HistLength Multiplier
- DELTA VOLUME MOMENTUM HistLen Multiplier DeltaLen
- VOLUME WEIGHTED MA OVER MA HistLength
- DIFF VOLUME WEIGHTED MA OVER MA ShortDist LongDist
- PRICE VOLUME FIT HistLength
- DIFF PRICE VOLUME FIT ShortDist LongDist
- DELTA PRICE VOLUME FIT HistLength DeltaDist
- ON BALANCE VOLUME HistLength
- DELTA ON BALANCE VOLUME HistLength DeltaDist
- POSITIVE VOLUME INDICATOR HistLength

Introduction

Many people who trade financial instruments would like to automate some or all of their trading systems. Automation has several advantages over seat-of-the-pants trading:

- Intelligently designed automated trading systems can and often do outperform human-driven systems. An effective data-mining program can discover subtle patterns in market behavior that most humans would not have a chance of seeing.

- An automated system is absolutely repeatable, while a human-driven system is subject to human whims. Consistency of decision-making is a vital property of a system that can consistently show a profit. Repeatability is also valuable because it allows examination of trades in order to study operation and perhaps improve performance.

- Most properly designed automated trading systems are amenable to rigorous statistical analysis that can assess performance measures such as expected future performance and the probability that the system could have come into existence due to good luck rather than true power.

- Unattended operation is possible.

Automated trading systems are usually used for one or both of two applications. *TSSB* is a state-of-the-art program that is able to generate trading systems that perform both applications:

- *TSSB* produces a complete, stand-alone trading system which makes all trading decisions.

- *TSSB* produces a model which may be used to filter the trades of an existing trading system in order to improve performance. It is often the case that by intelligently selecting a subset of the trades ordered by an existing system, and rejecting the other trades, we can improve the risk/reward ratio. TSSB can also suggest position sizes according to the likelihood of the trade's success.

Two Approaches to Automated Trading

Whether the user's goal is development of a stand-alone trading system or a system to filter signals from an existing trading system, there are two common approaches

to its development and implementation: *rules-based* (IF/THEN rules proposed by a human) and *predictive modeling*.

A rules-based trading system requires that the user specify the exact rules that make trade decisions, although one or more parameters associated with these rules may be optimized by the development software. Here is a simple example of an algorithm-based trading system:

IF the short-term moving average of prices exceeds the long-term moving average of prices, **THEN** hold a long position during the next bar.

The above algorithm explicitly states the rule that decides positions bar-by-bar, although the exact definition of 'short-term' and 'long-term' is left open. The developer might use software to find moving-average lookback distances that maximize some measure of performance. Programs such as TradeStation® include a proprietary language (EasyLanguage® in this case) by which the developer can specify trading rules.

With the widespread availability of high-speed desktop computers, an alternative approach to trading system development has become popular. *Predictive modeling* employs mathematically sophisticated software to examine indicators derived from historical data such as price, volume, and open interest, with the goal of discovering repeatable patterns that have predictive power. A predictive model relates these patterns to a forward-looking variable called a *target* or dependent variable. This is the approach used by *TSSB*, and it has several advantages over algorithm-based system development:

- Intelligent modeling software can discover patterns that are so complex or buried under random noise that no human could ever see them.

- Once a predictive model system is developed, it is usually easy to tweak its operation to adjust the risk/reward ratio to suit applications ranging across a wide spectrum. It can obtain a desired trade off between numerous signals with a lower probability of success and fewer signals with a higher probability of success. This is accomplished by adjusting a threshold that converts model predictions into discrete buy and sell signals.

- Well designed software allows the developer to adjust the degree of automation employed in the discovery of trading systems. Experienced developers can maintain great control over the process and put their knowledge to work creating systems having certain desired properties, while inexperienced developers can take advantage of massive automation, letting the software have majority control.

- In general, predictive modeling is more amenable to advanced statistical analysis than algorithm-based system development. Sophisticated statistical analysis algorithms can be incorporated into the model-generating process more easily than they can be incorporated into systems based on human-specified rules.

Predictive Modeling

The predictive modeling approach to trading system development relies on a basic property of market price movement: all markets contain patterns that tend to repeat throughout history, and hence can often be used to predict future activity. For example, under some conditions a trend can be expected to continue until the move is exhausted. Under other conditions, a sudden violent move will, more often than not, be followed by a retracement toward the recent mean price. A predictive model studies historical market data and attempts to discover the patterns that repeat often enough to be profitable. Once such patterns are discovered, the model will be on the lookout for their reoccurrence. Based on historical observations, the model will then be able to predict whether the market will soon rise, fall, or remain about the same. These predictions can be translated into buy/sell decisions by applying thresholds to the model's predictions. We expand on notion this below.

Indicators and Targets

Predictive models do not normally work with raw market data. Rather, the market prices and other series, such as volume, are usually transformed into two classes of variables called *indicators* and *targets*. This is the data used by the model during its training, testing, and ultimate realtime use. It is in the definition of these variables that the developer exerts his or her own influence on the trading system.

Indicators are variables that look strictly backwards in time. When trading in real time, as of any given bar an indicator will be computable, assuming that we are in possession of sufficient historical price data to satisfy the definition of the indicator. For example, someone may define an indicator called *trend* as the percent change of market price from the close of a bar five bars ago to the close of this bar. As long as we know these two prices, we can compute this *trend* indicator. The numerous indicators that *TSSB* can compute will be discussed in detail here.

Targets are variables that look strictly forward in time. (In classical regression modeling, the target is often referred to as the *dependent variable*.) Targets reveal the future behavior of the market. We can compute targets for historical data as long as we have a sufficient number of future bars to satisfy the definition of the target. Obviously, though, when we are actually trading the system we cannot know the targets unless we have a phenomenal crystal ball. For example, we may define an indicator called *day_return* as the percent market change from the open of the next day to the open of the day after the next. If we have a historical record of prices, we can compute this target for every bar except the last two in the dataset. Targets that

TSSB can compute are discussed here.

The fundamental idea behind predictive modeling is that *indicators* maycontain information that can be used to predict *targets*. The task of a predictive model is to find and exploit any such information. Consider the following hypothetical values of two indicators that we choose to call *trend* and *volatility*, along with a target variable that we will call *day_return*:

Date	trend	volatility	day_return
19950214	0.251	1.572	0.144
19950215	0.101	1.778	0.055
19950216	-0.167	2.004	-0.013
...			

Suppose we provide several years of this data to a model and ask it to learn how to predict *day_return* from *trend* and *volatility*. (This process is called model training.) Then, we may at a later date calculate from recent prices that *trend*=0.225 and *volatility*=1.244 as of that day. The trained model may then make a prediction that *day_return* will be 0.152. (These are all made-up numbers.) Based on this prediction that the market is about to rise substantially, we may choose to take a long position.

Converting Predictions to Trade Decisions

Intuition tells us that we should put more faith in extreme predictions than in more common predictions near the center of the model's prediction range. If a model predicts that the market will rise by 0.001 percent tomorrow, we would not be nearly as inclined to take a long position as if the model predicts a 5.8 percent rise. This intuition is correct, because in general there is a large correspondence between the magnitude of a prediction and the likelihood of success of the associated trade. Predictions of large magnitude are more likely to signal profitable market moves than predictions of small magnitude.

The standard method for making trade decisions based on predicted market moves is to compare the prediction to a fixed threshold. If the prediction is greater than or equal to an upper threshold (usually positive), take a long position. If the prediction is less than or equal to a lower threshold (usually negative), take a short position. The holding period for a position is implicit in the definition of the target. This will be discussed in detail here.

It should be obvious that the threshold determines a tradeoff in the number of trades versus the accuracy rate of the trades. If we set a threshold near zero, the magnitude

of the predictions will frequently exceed the threshold, and a position will be taken often. Conversely, if we set a threshold that is far from zero, predicted market moves will only rarely lie beyond the threshold, so trades will be rare. We already noted that there is a large correspondence between the magnitude of a prediction and the likelihood of a trade's success. Thus, by choosing an appropriate threshold, we can control whether we have a system that trades often but with only mediocre accuracy, or a system that trades rarely but with excellent accuracy.

TSSB automatically chooses optimal long and short thresholds by choosing them so as to maximize the profit factor for long systems and short systems separately. (See here for the definition of profit factor.) In order to prevent degenerate situations in which there is only one trade or very few trades, the user specifies a minimum number of trades that must be taken, either as an absolute number or as a minimum fraction of bars. In addition, *TSSB* has an option for using two thresholds on each side (long and short) so as to produce two sets of signals, one set for 'normal reliability' trades, and a more conservative set for 'high reliability' trades. Finally, in many applications, *TSSB* prints tables that show performance figures that would be obtained with varying thresholds.

Computation of thresholds and interpretation of trade results based on predictions relative to these thresholds are advanced topics that will be discussed in detail here. For now, the user needs to understand only the following concepts:

- The user specifies *indicator* variables based on recent observed history and *target* variables that portray future price movement.

- *TSSB* is given raw historical market data (prices and perhaps other data, such as volume) and it generates an extensive database of indicator and target variables. One or more models are trained to predict the target given a set of indicators. In other words, the model learns to use the predictive information contained in the indicators in order to predict the future as exemplified by the target.

- Every time a prediction is made, the numerical value of this prediction is compared to a *long* or *upper* threshold. If the prediction is greater than or equal to the long threshold, a long position is taken. Similarly, the prediction is compared to a *short* or *lower* threshold, which will nearly always be less than the *long* threshold. If the prediction is less than or equal to the short threshold, a short position is taken.

- The holding period for a position is inherent in the target variable. This will be discussed in detail here.

- *TSSB* will report results for long and short systems separately, as well as net

results for the combined systems.

Testing the Trading System

TSSB provides the ability to perform many tests of a predictive model trading or filtering system. The available testing methodologies will be discussed in detail here. However, so that the user may understand the elementary trading/filtering system development and evaluation presented in the next chapter, we now discuss two general testing methodologies: cross validation and walkforward testing. These are the primary standards in many prediction applications, and both are available in *TSSB* in a variety of forms.

The principle underlying the vast majority of testing methodologies, including those included in *TSSB*, is that the complete historical dataset available to the developer is split into separate subsets. One subset, called the *training set* or the *development set*, is used to train the predictive model. The other subset, called the *test set* or the *validation set*, is used to evaluate performance of the trained model. (Note that the distinction between the terms *test set* and *validation set* is not consistent among experts, so the increasingly common convention is to use them interchangeably. The same is true of *training set* and *development set*.)

The key here is that no data that takes part in the training of the model is permitted to take part in its performance evaluation. Under fairly general conditions, this mutually exclusive separation guarantees that the performance measured in the test set is an *unbiased* estimate of future performance. In other words, although the observed performance will almost certainly not exactly equal the performance that will be seen in the future, it does not have a systematic bias toward optimistic or pessimistic values. Having an unbiased estimate of future performance is one of the two main goals of a trading system development and testing operation. The other goal is being able to perform a statistical significance test to estimate the probability that the performance level achieved could have been due to good luck. This advanced concept will be discussed on Pages here and here.

In the earliest days of model building and testing, when high speed computers were not readily available, splitting of the data into a training set and a test set was done exactly once. The developer would typically train the model using data through a date several years prior to the current date, and then test the model on subsequent data, ending with the most recent data available. This is an extremely inefficient use of the data. Modern development platforms should make available *cross validation*, *walkforward*, or both. These techniques split the available data into training sets and test sets many times, and pool the performance statistics into a single unbiased estimate of the model-based trading system's true capability. This

extensive reuse of the data for both training and testing makes efficient use of precious and limited market history.

Walkforward Testing

Walkforward testing is straightforward, intuitive, and widely used. The principle is that we train the model on a relatively long block of data that ends a considerable time in the past. We test the trained model on a relatively short section of data that immediately follows the training block. Then we shift the training and testing blocks forward in time by an amount equal to the length of the test block and repeat the prior steps. Walkforward testing ends when we reach the end of the dataset. We compute the net performance figure by pooling all of the test block trades. Here is a simple example of walkforward testing:

1) Train the model using data from 1990 through 2007. Test the model on 2008 data.
2) Train the model using data from 1991 through 2008. Test the model on 2009 data.
3) Train the model using data from 1992 through 2009. Test the model on 2010 data.

Pool all trades from the tests of 2008, 2009, and 2010. These trades are used to compute an unbiased estimate of the performance of the model.

The primary advantage of walkforward testing is that it mimics real life. Most developers of automated trading systems periodically retrain or otherwise refine their model. Thus, the results of a walkforward test simulate the results that would have been obtained if the system had been actually traded. This is a compelling argument in favor of this testing methodology.

Another advantage of walkforward testing is that it correctly reflects the response of the model to *nonstationarity* in the market. All markets evolve and change their behavior over time, sometimes rotating through a number of different regimes. Loosely speaking, this change in market dynamics, and hence in relationships between indicator and target variables, is called *nonstationarity*. The best predictive models have a significant degree of robustness against such changes, and walkforward testing allows us to judge the robustness of a model.

TSSB's ability to use a variety of testing block lengths makes it easy to evaluate the robustness of a model against nonstationarity. Suppose a model achieves excellent walkforward results when the test block is very short. In other words, the model is never asked to make predictions for data that is far past the date on which its training block ended. Now suppose the walkforward performance deteriorates if the test block is made longer. This indicates that the market is rapidly changing in ways that the model is not capable of handling. Such a model is risky and will require frequent retraining if it is to keep abreast of current market conditions. On the other hand, if walkforward performance holds up well as the length of the test

block is increased, the model is robust against nonstationarity. This is a valuable attribute of a predictive model. Look at Figure 1 on the next page, which depicts the placement of the training and testing blocks (periods) along the time axis.

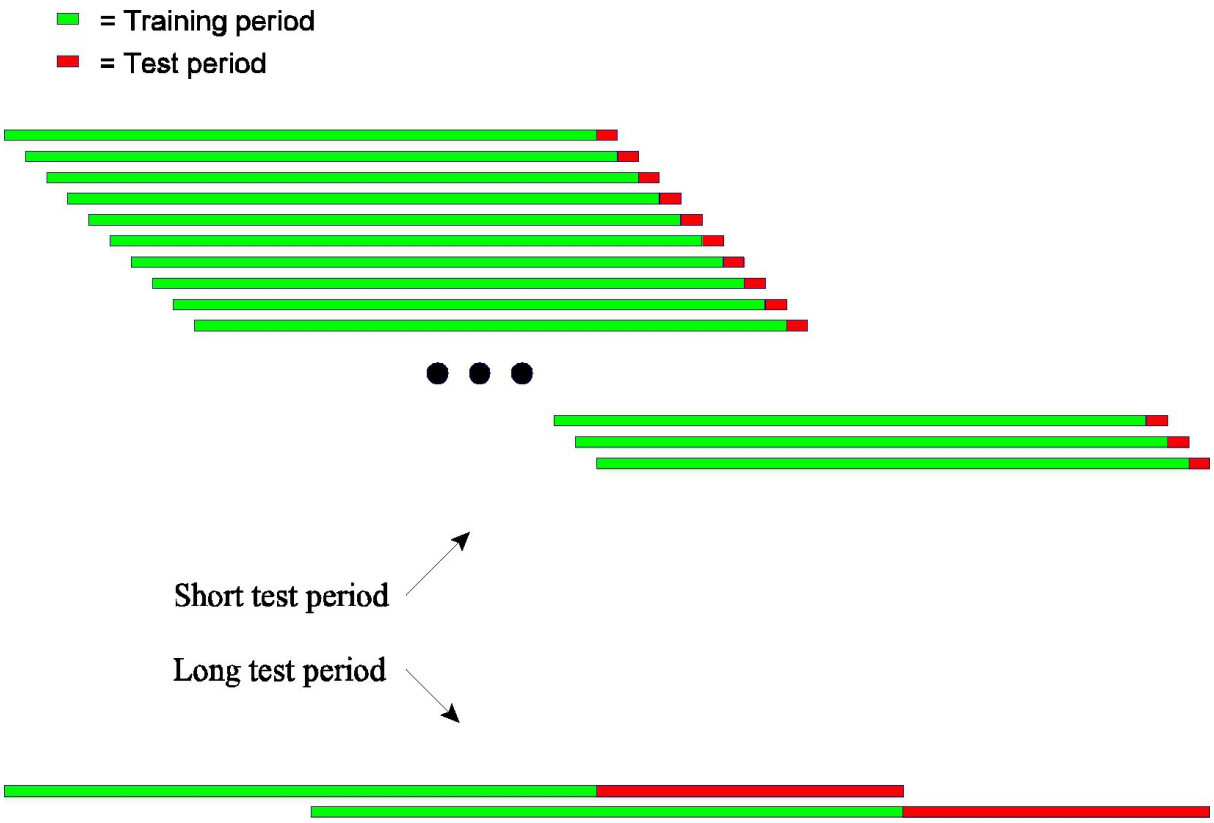


Figure 1: Walkforward testing with short and long test periods

Figure 1 above shows two situations. The top section of the figure depicts walkforward with very short test blocks. The bottom section depicts very long test blocks. It can be useful to perform several walkforward tests of varying test block lengths in order to evaluate the degree to which the prediction model is robust against nonstationarity.

Walkforward testing has only one disadvantage relative to alternative testing methods such as cross validation: it is relatively inefficient when it comes to use of the available data. Only cases past the end of the first training block are ever used for testing. If you are willing to believe that the indicators and targets are reasonably stationary, this is a tragic waste of data. Cross validation, discussed in the next section, addresses this weakness.

Cross Validation

Rather than segregating all test cases at the end of the historical data block, as is done with walkforward testing, we can evenly distribute them throughout the

available history. This is called *cross validation*. For example, we may test as follows:

1) Train using data from 2006 through 2008. Test the model on 2005 data.
2) Train using data from 2005 through 2008, excluding 2006. Test the model on 2006 data.
3) Train using data from 2005 through 2008, excluding 2007. Test the model on 2007 data.
4) Train using data from 2005 through 2008, excluding 2008. Test the model on 2008 data.

This idea of withholding interior 'test' blocks of data while training with the surrounding data is illustrated in Figure 2 below. In cross validation, each step is commonly called a *fold*.

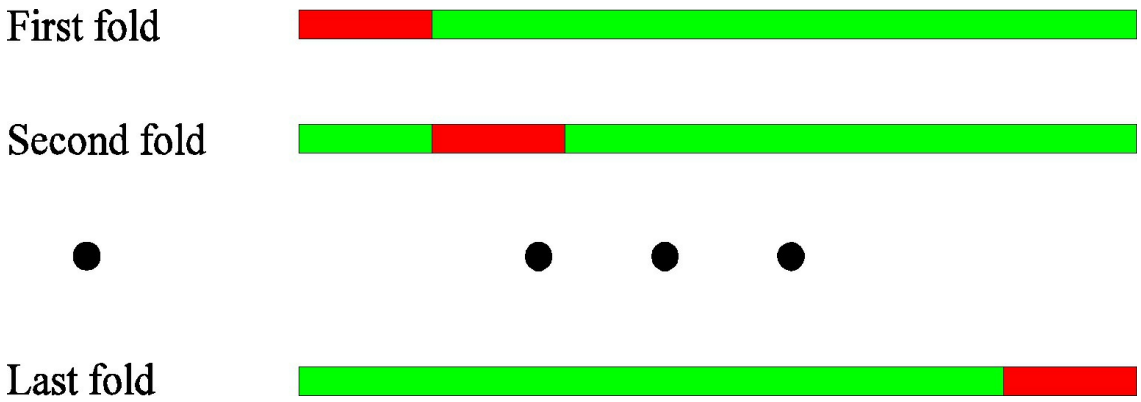


Figure 2: Cross validation

The obvious advantage of cross validation over walkforward testing is that every available case becomes a test case at some point. However, there are several disadvantages to note. The most serious potential problem is that cross validation is sensitive to nonstationarity. In a walkforward test, only relatively recent cases serve as test subjects. But in cross validation, cases all the way back to the beginning of the dataset contribute to test performance results. If the behavior of the market in early days was so different than in later days that the relationship between indicators and the target has seriously changed, incorporating test results from those early days may not be advisable.

Another disadvantage is more philosophical than practical, but it is worthy of note. Unlike a walkforward test, cross validation does not mimic the real-life behavior of a trading system. In cross validation, except for the last fold, we are using data from the future to train the model being tested. In real life this data would not be known at the time that test cases are processed. Some skeptics will raise their eyebrows at this, even though when done correctly it is legitimate, providing nearly unbiased performance estimates. Finally, overlap problems, discussed in the next

section, are more troublesome in cross validation than in walkforward tests.

Overlap Considerations

The discussions of cross validation and walkforward testing just presented assume that each case is independent of other cases. In other words, the assumption is that the values of variables for a case are not related to the values of other cases in the dataset. Unfortunately, this is almost never the situation. Cases that are near one another in time will tend to have similar values of indicators and/or targets. This generally comes about in one or both of the following ways:

- Many of the targets available in *TSSB* look further ahead than just the next bar. For example, suppose our target is the market trend over the next ten bars. This is the quantity we wish to predict in order to make trade decisions. If this value is high on a particular day, indicating that the market trends strongly upward over the subsequent ten days, then in all likelihood this value will also be high the following day, and it was probably high the prior day. Shifting ahead or back one day still leaves an overlap of nine days in that ten-day target window. Such case-to-case correlation in time series data is called *serial correlation*.

- In most trading systems, the indicators look back over a considerable time block. For example, an indicator may be the market trend over the prior 50 days, or a measure of volatility over the prior 100 days. As a result, indicators change very slowly over time. The values of indicators for a particular day are almost identical to the values in nearby days, before and after.

These facts have several important implications. Because indicators change only slowly, the model's predictions also change slowly. Hence market positions change slowly; if a prediction is above a threshold, it will tend to remain above the threshold for multiple bars. Conversely, if a prediction is below a threshold, it will tend to remain below that threshold for some time. If the target is looking ahead more than one bar, which results in serial correlation as discussed above, then the result of serial correlation in both positions and targets is serial correlation in returns for the trading system. This immediately invalidates most common statistical significance tests such as the t-test, ordinary bootstrap, and Monte-Carlo permutation test. *TSSB* does include several statistical significance tests that can lessen the impact of serial correlation. In particular, the stationary bootstrap and tapered block bootstrap will be discussed here. Unfortunately, both of these tests rely on assumptions that are often shaky. We'll return to this issue in more detail later when statistical tests are discussed. For the moment, understand that targets

that look ahead more than one bar usually preclude tests of significance or force one to rely on tests having questionable validity.

Lack of independence in indicators and targets has another implication, this one potentially more serious than just invalidating significance tests. The legitimacy of the test results themselves can be undermined by bias. Luckily, this problem is easily solved with a *TSSB* option called OVERLAP. This will be discussedhere. For now we will simply explore the nature of the problem.

The problem occurs near the boundaries between training data and test data. The simplest situation is for walkforward testing, because there is only one (moving) boundary. Suppose the target involves market movement ten days into the future. Consider the last case in the training block. Its target involves the first ten days after the test block begins. This case, like all training set cases, plays a role in the development of the predictive model. Now consider the case that immediately follows it, the first case in the test block. As has already been noted, its indicator values will be very similar to the indicator values of the prior case. Thus, the model's prediction will also be similar to that of the prior case. Because the target looks ahead ten days and we have moved ahead only one day, leaving a nine-day overlap, the target for this test case will be similar to the target for the prior case. But the prior case, which is practically identical to this test case, took part in the training of the model! So we have a strong prejudice for the model to do a good job of predicting this case, whose indicators and target are similar to the training case. The result is optimistic bias, the worst sort. Our test results will exceed the results that would have been obtained from an honest test.

This boundary effect manifests itself in an additional fashion in cross validation. Of course, we still have the effect just described when we are near the end of the early section of the training set and the start of the test set. This is the left edge of the red regions in Figure 2. But we also have a boundary effect when we are near the end of the test set and the start of the later part of the training set. This is the right edge of each red region. As before, cases near each other but on opposite sides of the training set / test set boundary have similar values for indicators and the target, which results in optimistic bias in the performance estimate.

The bottom line is that bias due to overlap at the boundary between training data and test data is a serious problem for both cross validation and walkforward testing. Fortunately, the user can invoke the OVERLAP option to alleviate this problem, as will be discussed here.

Performance Criteria

Prior sections discussed the evaluation of performance using cross validation or walkforward testing. Those sections dealt with the mechanics of partitioning the data into training and test sets. Other considerations will be presented here. Several terms should be defined first:

- A *fold* is a single partitioning of the available data into a training set, a test set, and perhaps some cases from the dataset that are temporarily omitted in order to handle overlap. In Figures 1 and 2, a fold would be one of the horizontal strips consisting of the green training block and the red test block.

- The *in-sample* (or *IS*) performance for a fold is the performance of the model or trading system in the current training set. Because the training process for the model optimized some aspect of its performance, the in-sample performance will usually have an optimistic bias, often called the *training bias*.

- The *out-of-sample* (or *OOS*) performance for a fold is the performance of the model or trading system in the current test set. Because this data did not take part in training the model, it is, for all practical purposes, unbiased. In other words, on average it will reflect the true capability of the model or trading system.

Model Performance Versus Financial Performance

Performance statistics for model-based trading systems fall into two categories. One is the predictive performance of the model that determines trade decisions. This may include statistics such as the mean squared error of the predictions, or the model's R-squared. The other is the financial performance of the trading system, such as its profit factor or Sharpe ratio. Naturally, there is a degree of correspondence between statistics in these two categories. If a model has excellent performance, the trading system probably will as well. By the same token, a poorly performing model will most likely produce a poorly performing trading system.

On the other hand, the degree of correspondence between model and trade performance may not always be as high as one might expect. For example, the venerable *R-squared* is a staple in many fields. Yet it has a shockingly low relationship with the profit factor of the trading system, a commonly used measure of financial performance. In fact, it is not uncommon for a model-based trading system with respectable financial performance to be driven by a predictive model that has a negative R-squared! (Roughlyspeaking, a negative R-squaredmeans that the model's error variance exceeds that of just guessing the target mean for every case.) This peculiar behavior happens because trades are signaled only for

extremely high or low predictions, which are almost always the most reliable decisions. For less extreme predictions, those lying between the short and long thresholds where no trades are taken, errors can be so large that they overwhelm the predictive quality at the extremes. The result is a negative R-square.

Because of this frequent discrepancy, financial performance statistics for the trading system, such as profit factor, are much more useful than predictive performance of the model. Nonetheless, model accuracy is of some interest and should always be examined, if for no other reason than to check for anomalous behavior.

Financial Relevance and Generalizability

Performance statistics, especially for the trading system, can be graded on two scales: *financial relevance* and *generalizability*. This is not to say that these two criteria are mutually exclusive, or even at opposing ends of a continuum. Still, they are often independent of one another, and some discussion is warranted.

A *financially relevant* statistic is one that is important from a profit-making or money management perspective. For example, the ratio of annual percent return to average annual drawdown is of great interest to a person responsible for money management.

The *generalizability* of a statistic is a somewhat vague term, though important. It refers to the degree to which its IS (in-sample) value tends to hold up OOS (out-of-sample). The reason this is important is that when we train a model to predict a target, we do so by optimizing some measure of performance. Obviously, our ultimate goal is OOS performance. A model's or trading system's IS performance is of academic interest only; it is the OOS performance that determines whether the system will make money for us. Thus, if the training process optimizes a performance statistic that does not tend to hold up OOS, we have gained little or nothing. When we choose a performance statistic to optimize during training, we are best off if we choose one whose performance OOS is likely to reflect its IS value. In other words, wewant to optimize a performance statistic that has good generalizability.

The generalizability of a statistic is somewhat dependent on the particular model, the indicators, and the target. For this reason, the developer should experiment and assess this property by examining individual fold results for several candidate statistics. However, experience provides a few guidelines. For example, profit factor tends to have good generalizability. In contrast, performance statistics that depend on the temporal order of gains and losses, such as anything involving

drawdown, have poor generalizability. Finally, the more trades that go into a statistic, the more likely it is to generalize well.

Performance Statistics in *TSSB*

The *TSSB* program computes and prints a wide variety of predictive accuracy and financial performance statistics. These are provided for the underlying model(s) as well as the complete trading system that may be based on numerous models, committees, and so forth. We will regularly refer to them throughout the remainder of this document, so rather than repeat definitions every time they appear, they will all be defined here. The program also prints some quantities that are not exactly performance measures, but are related to performance. These quantities are included here as well. The following items are listed in the approximate order that they usually appear in the program's log file.

Final best crit - This depends on the optimization criterion employed by the user. It refers to the stepwise selection of the indicators chosen by the program. This figure is of little or no interest to most users, and is included only for the use of advanced users who are interested in technical details of operation.

Target grand mean - The mean value of the target variable. This is especially useful when examining individual fold results. If the in-sample target mean is very different from the out-of-sample target mean, poor performance on this fold might be excused.

Outer hi thresh and ***means*** - We saw here that trade decisions are made by comparing the model's predictions with one threshold for long trades and another (usually lower) threshold for short trades. The *Outer hi thresh* line in the log file shows the optimal long threshold computed by the program, the number of cases that equaled or exceeded this threshold, the mean target for cases whose predictions lay at or beyond the threshold, and the mean target for the cases below this threshold. We hope that the mean target value of cases beyond the outer hi threshold exceeds that of cases below the threshold. In some applications, *Inner* thresholds and means will also be printed. This is a very advanced concept. See the manual for details.

Outer lo thresh and ***means*** - This is the corresponding information for the lower (sell short trades) threshold.

Target statistics at various percentages kept - If the model is effective at

predicting the target, one would expect that cases having large predicted values will have large actual target values, and cases having small predictions will have small actual target values. This table lets us assess in a variety of ways the degree to which this is happening. The table contains five columns, corresponding to 10, 25, 50, 75, and 90 percent of the cases having predictions beyond upper (long) and lower (short) thresholds. The various statistics printed in this table are best explained in the context of a specific application with actual numbers, so details are deferred until here.

MSE -Mean squared error of the model. In financial applications this figure is nearly meaningless, but it is printed for completeness.

R-squared - Fraction of the target variance which is predictable by the model. In pathological cases, which are not uncommon in financial applications, this widely used predictive accuracy measure may be negative. A negative R-squared means that the model's predictions are actually worse than just guessing the target mean for every case. In financial applications this figure is nearly meaningless, but it is printed for completeness.

ROC Area - Early communication theory used a *Receiver Operating Characteristic* (*ROC*)curve to depict the performance of a communication system. The area under this curve is strongly correlated with the ability of the system to separate information from noise. The *TSSB* program generalizes the original communication version to financial modeling. Roughly speaking, the *ROC Area* measures the degree to which large predictions correspond to large target values, and small predictions correspond to small target values. The ROC area ranges from zero (a model that gets its predictions entirely backwards) to one (a perfect model). A value of 0.5 corresponds to random guessing.

Buy-and-hold profit factor - The profit factor that would be obtained by treating each case as a single trade, taking a long position for every one of them and holding each for the look-ahead distance of the target variable. The buy-and-hold profit factor represents the financial performance of a naive trading system, and thus serves as a basis of comparison for the *long profit factor* of the trained model. *TSSB* allows targets that may not represent profits. Therefore this value is printed only if the target is interpretable as a profit, or if the user specified a profit variable. See here for a discussion of this issue.

Sell-short-and-hold - The profit factor that would be obtained by treating each case as a single trade, taking a short position for every one of them, and holding that position for the look-ahead distance of the target. This is the reciprocal of the *Buy-and-hold profit factor*. This serves as a naive-trading-system baseline for comparison with the *short profit factor* of the trained model. This value is printed only if the target is interpretable as a profit, or if the user specified a profit variable. See here for a discussion of this issue.

Dual-thresholded outer PF - The profit factor that would be obtained by treating each case as a single trade and taking a long position for each case whose prediction equals or exceeds the upper threshold, and taking a short position for each case whose prediction is less than or equal to the lower threshold. The duration of the trade is the look-ahead distance of the target variable. This statistic measures both long and short trading performance. This value is printed only if the target is interpretable as a profit, or if the user specified a profit variable. See here for a discussion of this issue.

Outer long-only PF and ***Improvement Ratio*** - Outer long-only PF is the profit factor that would be obtained by treating each case as a single trade and taking a long position for each case whose prediction equals or exceeds the upper threshold. The duration of the trade is the look-ahead distance of the target variable. Thus, this statistic measures only the performance of *buy* signals. The *Improvement Ratio* is the *Outer long-only PF* divided by the *Buy-and-hold profit factor*. In other words, this is the factor by which the long-only version of the model improves profit factor over a simple buy-and-hold strategy. These values are printed only if the target is interpretable as a profit, or if the user specified a profit variable. See here for a discussion of this issue.

Outer short-only PF and ***Improvement Ratio*** - As above, except for short positions only.

Balanced (0.01 each side) PF - This statistic is printed only if the FRACTIL THRESHOLD option (here) is invoked in a multiple-market situation, such as the stocks in the S&P 500 or some other basket of equities. This is the profit factor that would be obtained by treating each bar as a single trade opportunity in each market (or financial instrument), and taking a long position for each market whose prediction is in the upper (highest ranked) one percent of predictions across all of the markets being traded, and taking a short position for each market whose prediction is in the lower (lowest ranked) one percent of predictions

across all of the markets. The idea is that we hold a balanced position, long and short an equal number of markets. This is known as a *market-neutral* strategy. This value is printed only if the target is interpretable as a profit, or if the user specified a profit variable. See here for a discussion of this issue. See here for a detailed description of the various *Balanced* optimization criteria and the concept of balanced multiple-market trading.

Balanced (0.05 / 0.10 / 0.25 / 0.50 each side) PF - As above, except that the 10, 25, and 50 percentile subsets of predictions are taken across all markets.

Profit factor above and below various thresholds - This table is printed only in a cross-validation or walkforward summary, and only if the FRACTILE THRESHOLD option (here) is invoked in a multiple-market situation. The table contains five columns. The first column lists a variety of thresholds. The second column is the fraction (0-1) of cases whose predicted target is greater than or equal to the threshold for that row. The third is the profit factor one would obtain from taking a long position for those cases. The fourth is the fraction of cases whose predicted target is strictly less than the threshold. The last column is the profit factor one would obtain from taking a short position for those cases. This table is useful for evaluating the tradeoff between the number of trades taken and the profit factor of the corresponding trading system.

Desirable Program Features

It is possible to use a general-purpose statistical modeling package to develop and test a financial market trading system based on predictive modeling. The best statistical analysis programs contain a data transform language that can be used to create indicator and target variables. Or, these variables can be computed with specialized software. Then, model-building methods in the package can create a predictive model from a specified training set. This model can be applied to a test set, and the predictions exported to a spreadsheet program. With some work, the spreadsheet program can then be used to compute basic financial performance statistics.

It should be obvious that the procedure just described is awkward, tedious, and of limited versatility. The development and testing of predictive-model trading systems is best done with software written specifically for this task. A professional program will do the following, at a minimum:

- Be able to compute a wide variety of indicators and targets, saving the user from the need to write or purchase specialized software to do this.
- Contain a scripting language that will let the user define variables that are not predefined in the program. The language will also enable the user to modify existing variables.
- Be capable of developing and testing both stand-alone trading systems and signal filters for existing systems.
- Handle both daily and intraday data.
- Process multiple markets, including the ability to compute cross-sectional indicators based on the behavior of individual markets in the context of a universe of markets, such as the S&P 500, a variety of currencies, or interest-rate futures.
- Be able to export standard-format databases to other programs, and read externally produced databases.
- Supply a variety of modeling methods so that the user can select the one having the best combination of power, resistance to overfitting, and speed.
- Have the ability to automate selection of indicators from a list of candidates.
- Offer a wide and useful variety of optimization criteria so that the user can choose a criterion that best suits his or her purpose.

- Support regime training and testing to facilitate models that specialize in specific regimes such as high (or low) volatility, up (or down) trends, and so forth.

- Supply both cross validation and walkforward testing at a variety of granularities (day, month, year, and so forth).

- Provide details on the model(s)' predictive accuracy and financial performance statistics for both the training set and the test set individually for every fold, as well as pooled test set results.

- When possible, compute statistical significance levels for financial performance statistics.

- Preserve predictions for examination within the program as well as export to other programs.

- Implement a variety of model-combining committees to enable state-of-the-art prediction capabilities.

- Include built-in graphics capabilities to study variables and their relationships.

TSSB includes all of these capabilities, along with many more features that are useful in the development and testing of predictive-model trading systems and signal filters. In the next chapter we will delve into the program with a specific example of the development and testing of a simple automated trading system based on predictive modeling.

A Simple Standalone Trading System

To illustrate the first of TSSB's two uses, we begin with the development of a simple standalone trading system. (Development of a system for filtering trades of an existing system will be presented in the next chapter.) This chapter will show and discuss all of the files that are necessary to implement this system, although the discussion of this first system will be limited to an overview. Extensive details will appear later in this document, and we will provide references to these details here to satisfy readers who want to occasionally flip ahead.

The Script File

All *TSSB* operations are controlled by a script file. The default extension for the file's name is. SCR, although the user is free to use any extension, such as .TXT. The .SCR extension was a standard for script files for many years, but since Microsoft Windows chose it for screen savers, this extension has become problematic on some computers.

The script file for this example is named SIMPLE_STANDALONE.SCR. contents are now listed, and a discussion of each line follows.

```
READ MARKET LIST "SYMBOLS.TXT" ;
READ MARKET HISTORIES "E:\SP100\IBM.TXT" ;
READ VARIABLE LIST "TREND_VOLATILITY.TXT" ;

MODEL SIMPLE_MODEL IS LINREG [
   INPUT = [ P_TREND P_VOLATILITY ]
   OUTPUT = DAY_RETURN
   MAX STEPWISE = 0
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   ] ;

WALK FORWARD BY YEAR 10 1999 ;
```

Here is a brief discussion of each line of the script file:

```
READ MARKET LIST "SYMBOLS.TXT";
```

TSSB needs to know which markets (or tradeable instruments) will take part in the development. These markets are listed in the file SYMBOLS.TXT. Keeping the market list in a separate file facilitate fast and easy changes to the markets that are used for development. Note that like all files named in a script file, the name is enclosed in quotes

and must not include blanks or any other character that Windows deems illegal for file names. The READ MARKET LIST command discussed in detail here.

```
READ MARKET HISTORIES "E:\SP100\IBM.TXT";
```

The market histories are ASCII files that contain the date, optional time market symbol, open, high, low, close, and volume. The user must name one of the files (IBM.TXT here) so that*TSSB* knows where to find all of them. This named file need not appear in the market list specified in the READ MARKET LIST command shown above, but the named fil must be present in the specified directory. The READ MARKE HISTORIES command must appear on any line after the REA MARKET LIST command so that the program knows which markets t read. The READ MARKET HISTORIES command is discussed detail here.

```
READ VARIABLE LIST "TREND_VOLATILITY.TXT";
```

TSSB has the ability to internally compute a wide variety of indicators and targets. The user creates an ASCII text file, often called a*variable definition list* file, that names and defines the indicators and targets that the user wishes to compute. The variable definition list file that we wish to use is specified with this command (READ VARIABLE LIST This command must appear on any line after the READ MARKE HISTORIES command so that the market data has been read and i ready to be processed. The READ VARIABLE LIST command discussed in detail here.

```
MODEL SIMPLE_MODEL IS LINREG [
```

The MODEL command specifies the modeling method to be used fo model development and assigns a name for the resulting prediction model. Here, the user chose to name the model SIMPLE_MODEL. name such as JOSEPH would be allowed as well, although it woul probably be confusing to users. Blanks and special characters other than an underscore are not allowed in the name. The type of model in this example is simple linear regression, which is specified by means of the keyword LINREG. Specifications for the model are enclosed in squar brackets. Because it is legal (though not required) to spread the specifications across multiple lines, clarity is often enhanced by putting the opening bracket on the model definition line and putting individual specifications on separate lines. The MODEL command is discussed i detail here.

The following five items are specifications for the model developed in this example:

INPUT = [P_TREND P_VOLATILITY]

The INPUT line specifies the indicators that will be used by the model This indicator list is enclosed in square brackets. The two indicators here, P_TREND and P_VOLATILITY, are defined in the variabl definition file, which will be discussed soon. The INPUT command is discussed in detail here.

OUTPUT = DAY_RETURN

The OUTPUT line specifies the target variable, the true values of the quantity that the model will be asked to predict. The target here, DAY_RETURN, is defined in the variable definition file, which wil be discussed soon. The OUTPUT command is discussed in detail here.

MAX STEPWISE = 0

The MAX STEPWISE specification tells the automated stepwis indicator selection algorithm in *TSSB* the maximum number of indicators that may be used as inputs to the developed model. By setting it to zero, we tell the program that it is to skip automatic selection, and use every indicator in the INPUT list. The MAX STEPWISE comma is discussed in detail here.

CRITERION = PROFIT FACTOR

TSSB allows the user to choose from among a variety of optimization criteria (sometimes called the *objective function*) for several aspects of model training. The PROFIT FACTOR criterion tells the program tha we are most interested in maximizing the two-sided (long and short) profit factor of the trading system. The CRITERION command i discussed in detail here.

MIN CRITERION FRACTION = 0.1

For performance reports as well as some optimization criteria, *TSSB* generally chooses long and short prediction thresholds that trigger trades in such a way that the profit factor of the trades is maximized. (The actual mechanism can be much more involved, too complex to address here. Further details will be provided in subsequent sections.) The problem with choosing a threshold that maximizes the trade profit factor is that unless constraints are imposed, the program may set such an extreme threshold that only one, or very few, winning trades are executed, leading to a huge or infinite profit factor. The MIN CRITERION FRACTION specifies a minimum fraction of tra opportunities that result in actual trades on each side (long and short, counted separately). (In TSSB, by default each bar in each market is a trade opportunity.) Here, the user has specified that at least 0.1 (ten percent) of the trade opportunities must result in a long position being

taken, and another ten percent must result in a short position being taken. The MIN CRITERION FRACTION command is discussed detail here.

```
];
```

This closing bracket (along with the obligatory semicolon that ends commands) completes the model specifications begun with the MODEI command.

```
WALK FORWARD BY YEAR 10 1999;
```

This command specifies that the trading system will be evaluated by means of a walk forward test. BY YEAR means that the testing window will encompass one calendar year, each training window will encompass the ten years prior to the test window, and the first test window will be in 1999. The WALK FORWARD command i discussed in detail here.

We'll now examine the files that are referenced in the SIMPLE_STANDALONE.SCRscript file just discussed. The first file referenced the market list, SYMBOLS.TXT. It is just a list of the trading instruments that wil be processed:

```
AA
DELL
DOW
GE
...
XOM
XRX
```

Next, the script file names a market history file. Here are a few lines from such a file:

```
20110301 16.90 16.94 16.21 16.23 297967
20110302 16.20 16.43 16.13 16.18 201850
20110303 16.37 16.77 16.36 16.63 191486
20110304 16.77 16.80 16.37 16.58 186114
20110307 16.58 16.75 16.16 16.17 119222
```

In the market history file excerpted above, the date appears first as YYYYMMDD This is followed by the open, high, low, and close prices. Volume appears last on the line.

Finally, the script file references a file called the variable definition list. Here is this file, which defines three variables (two indicators and a target):

```
P_TREND:      LINEAR PER ATR 5 100
P_VOLATILITY: PRICE VARIANCE RATIO 5 4
DAY_RETURN:   NEXT DAY ATR RETURN 250
```

The first line defines a variable called P_TREND, which will be used as a indicator. This could just as well have been named MARY or PHIL, but it is always good to make the name descriptive. The maximum length of a name is 15 characters. Blanks and special characters other than an underscore (_) are not allowed. The P_TREND variable is defined as an indicator that is built into the *TSSB* library: LINEAR PER ATR. This is the least-squares linear slope divided b ATR (average true range) as a normalizer. The linear slope is fit over a window of 5 bars, and ATR is computed over a window of 100 bars.

The volatility variable (called P_VOLATILITY here) is the ratio of the variance o the log of price over a 5-bar window, divided by the variance over a 5*4=20 bar window. This built-in variable will serve as another indicator for the model.

The variable that the model uses as a target is given the name DAY_RETURN which is nicely descriptive. It is defined as the *TSSB* built-in variable NEXT DAY ATR RETURN, which is the price change from tomorrow's open to the next day's open, normalized by the 250-day ATR.

TSSB's built-in variables will be discussed in somewhat more depth here of this manual. However, the User's Manual is the ultimate reference for all built-in variables.

The Audit Log

TSSB always writes an ASCII text file called AUDIT.LOG that contains the resul of its operations. If the program encountered an error, in many cases the audit log will contain (usually as its last line) a more detailed explanation of the error than appeared on the screen. We now examine the audit log produced by the example script file shown above. For clarity, it will be broken into sections, with each explained separately.

```
COMMAND ---> READ MARKET LIST "SYMBOLS.TXT" ;
User specified 12 markets.
AA DELL DOW GE HAL HNZ IBM JNJ JPM WMT XOM XRX
```

Each command in the script file is echoed to the log file. Here, we see the READ MARKET LIST command and its results. The log file tells us that we are processing 12 markets, and they are listed.

```
COMMAND ---> READ MARKET HISTORIES "E:\SP100\IBM.TXT" ;

Reading market histories (*.TXT) from path E:\SP100\

AA had 10393 cases (19700102 000000 through 20110307
000000)
Max ratio = 1.29 on 20081010 000000

DELL had 5724 cases (19880623 000000 through 20110307
000000)
Max ratio = 1.46 on 19930525 000000

DOW had 6855 cases (19840104 000000 through 20110307
000000)
Max ratio = 1.22 on 19871020 000000

.....

XRX had 6855 cases (19840104 000000 through 20110307
000000)
Max ratio = 1.42 on 19991008 000000
```

The READ MARKET HISTORIES command is processed. For each market, t following information is printed:

- Name of the market and the number of cases in the history file
- Starting date as YYYYMMDD, and time as HHMMSS. For daily data, tl time is zero.
- Ending date in the same format
- Maximum ratio of open prices for any two consecutive bars, and the date/time it occurred. Unusually high ratios indicate a very large change in price from one bar to the next, and while this may be legitimate, it may indicate an error in the market data.

```
COMMAND ---> READ VARIABLE LIST "TREND_VOLATILITY.TXT" ;

Temporary work file is
D:\BOOSTER\DOC\Tutorial\Examples\BoosterTempFile0.tmp

Database file is
D:\BOOSTER\DOC\Tutorial\Examples\BoosterTempDatabase.tmp
```

The variable definition file “TREND_VOLATILITY.TXT” is read. Although it of little practical importance in most cases, the log file (AUDIT.LOG) names an special temporary work files that are created. These files can be gigantic (gigabytes

for large applications), so the user should confirm that the drive on which the files are created contains sufficient free space. Normally, all work files will be automatically deleted when the program ends, and if any happen to be left from a crashed prior session, they will be deleted when TSSB starts again. However, in rare cases of user error leading to abnormal program termination, enormous work files will remain on the hard drive until *TSSB* is run again. If the hard drive suddenly seems full, it would not hurt to search it for any .tmp files whose name begins with *Booster*. As long as the program is not running, it is safe to delete them.

```
Summary information for variable P_TREND:
Market Fbad Bbad Nvalid First    Last     Min    Max   Mean
   AA  100   0  10293 19700526 20110307 -50.000 49.467 0.719
 DELL  100   0   5624 19881114 20110307 -49.972 49.783 1.007
  DOW  100   0   6755 19840525 20110307 -50.000 49.582 0.821
...

XRX  100   0   6755 19840525 20110307 -50.000 49.721 0.719
```

For each variable in the definition file, summary information is printed for each market. This information is as follows:

Market - The symbol for the market

Fbad - The number of undefined values at the front (that's what F stands for) of the file. In this case, there are 100 undefined values, which makes sense. The P_TREND variable is normalized by ATR (average true range) in a 100-bar window. So we need to pass 100 bars before this indicator's initial value can be computed.

Bbad - The number of undefined values at the back (end) of the file. For indicators this will always be zero, because indicators look only backwards in time. But targets look forward in time, so as the end of the dataset is approached, targets will become undefined. For example, the DAY_RETURN target here would have Bbad equal to 2, because it is based on the price change from tomorrow's open to the next day's open. It needs to look two days into the future.

Nvalid - The number of cases in this market that have valid (computable) values of this variable. A variable cannot be computed for a case if the case is too early in the database for required history to be available (Fbad), too late in the database for required future prices to be available (Bbad), or if the lookback or lookahead window contains missing or invalid data. The most common cause for invalid data in the interior of the dataset is suspicious price jumps flagged by the optional CLEAN RAW DATA

command described here.

First - The date (YYYYMMDD) of the first valid occurrence of this variable.

Last - The date (YYYYMMDD) of the last valid occurrence of this variable. It i not guaranteed that all cases within this date range contain valid values of this variable. The most common cause for invalid data in the interior of the dataset is suspicious price jumps flagged by the optional CLEAN RAW DATA command described here.

Min - The minimum value of this variable in the market.

Max - The maximum value of this variable in the market.

Mean - The mean value of this variable in the market.

```
User defined 3 variables
Wrote 87168 records to the database

      AA  10141
    DELL   5472
     DOW   6603
      GE   6603
     HAL   6603
     HNZ   6603
     IBM  12128
     JNJ   6603
     JPM   6603
     WMT   6603
     XOM   6603
     XRX   6603
```

The last thing done by *TSSB* when processing the READ VARIABLE LIS command is to report the number of variables found in the variable definition file, the number of cases (records) placed in the database, and the number of those cases attributed to each market. Note that some markets have more cases because these markets begin at an earlier date.

```
COMMAND ---> MODEL SIMPLE_MODEL IS LINREG [
                INPUT = [ P_TREND P_VOLATILITY ]
                OUTPUT = DAY_RETURN
                MAX STEPWISE = 0
                CRITERION = PROFIT FACTOR
                MIN CRITERION FRACTION = 0.1
                ] ;
```

TSSB echoes all script file commands to the log file. These lines are the definition

of the model and its specifications, which we already discussed.

A Walkforward Fold

We continue this tour of the log file AUDIT.LOG with the results of a walkforward fold.

```
COMMAND ---> WALK FORWARD BY YEAR 10 1999 ;
Beginning walk forward by year
```

Walkforward is about to begin. Both IS (in-sample, the training set) and OOS (out-of-sample, the test set) results will be written for each fold. For economy, we will now examine only the first fold. The output generated for the fold will be broken up into sections for clarity. All in-sample results appear first, followed by the out-of-sample results.

```
------------------------------------------------------------
Walkforward test date 1999 train 30219 cases, testing 3024
------------------------------------------------------------

LINREG Model SIMPLE_MODEL predicting DAY_RETURN
Stepwise not used; all predictors available to model
Regression coefficients:
         -0.001424  P_TREND
          0.000257  P_VOLATILITY
          0.038171  CONSTANT
```

We see that this walkforward fold is testing the year 1999. The training set for this fold contains 30219 cases, and the test set contains 3024 cases.

The model is linear regression (LINREG), the user named the model SIMPLE_MODEL, and its target variable is DAY_RETURN.

Stepwise indicator selection was not used. (Recall that the user specified MAX STEPWISE = 0.) This forces all indicators (often called *predictors* in the context of a predictive model) to be made available to the model being defined. In the case of linear regression, this implies that all variables will be used by the model. Some models, which will be discussed later, may choose to ignore some indicators, even if those indicators are made available to them.

The coefficients of the linear regression model, as well as the constant offset, are printed.

```
Target grand mean = 0.03721

Outer hi thresh = 0.07134  with 4040 of 30219 cases at or
above (13.37 %)  Mean = 0.09306 versus 0.02859

Outer lo thresh = 0.00428  with 3025 of 30219 cases at or
below (10.01 %)  Mean = -0.04828 versus 0.04672
```

We are still dealing with the training set in this fold. The mean of the target variable, DAY_RETURN, in the training set is 0.03721.

TSSB found an upper (long trades) threshold of 0.07134 as the prediction threshold that maximizes the profit factor of long trades, subject to the condition that at least ten percent of the potential trades (training cases, which are bars in markets by default) be taken as long trades (MIN CRITERION FRACTION = 0.1). Of t 30219 training cases, 4040 of them, or 13.37 percent, had predictions at or above this threshold and hence signaled long trades. The mean of the target variable, DAY_RETURN, in these long trades was 0.09306, while the mean of the target fo the trades not taken (those trade opportunities in which a trade was not signaled) was 0.02859. Thus, in the in-sample data (training data), the model discriminated well between trade opportunities that offered good buy opportunities and those that did not. This is no surprise.

Corresponding results are reported for the low (short trades) threshold. Note that its optimal threshold came almost right up against the restriction of taking at least ten percent of the trades as short, with 10.01 percent of possible trades signaling as short. As would be expected for in-sample (training set) data, the mean target for cases whose predictions are at or below the low threshold is less than the mean target of cases above the short threshold (-0.04828 vs. +0.04672).

```
Target statistics at various percentages kept...
Statistic              10      25      50      75      90

Mean target above   0.0807  0.0745  0.0513  0.0508  0.0466
N wins above          1475    3688    7146   10723   12820
Mean win above      0.8105  0.7500  0.7140  0.7015  0.6979
Total win above     1195.5  2765.9  5102.1  7522.6  8947.6
N losses above        1270    3208    6605    9944   11981
Mean loss above     0.7493  0.6868  0.6550  0.6407  0.6410
Total loss above     951.6  2203.1  4326.5  6370.7  7679.7
Profit factor above 1.2563  1.2555  1.1793  1.1808  1.1651
```

The lines shown above are only half of the complete chart, statistics for cases whose predictions are above a threshold. The second half of the chart, for cases below a threshold, will appear soon. However, this chart contains an enormous amount of information, so it's best to take it one half at a time.

Each of the five columns represents a different threshold. Rather than specifying numeric values, they are specified as percentiles. So, for example, the first column represents whatever threshold results in ten percent of the cases lying at or above the threshold, and the statistics shown in that column are for that ten percent of cases. Similarly, the rightmost column represents a much lower threshold, so low that ninety percent of the cases lie at or above it.

We'll now examine each row statistic:

Mean target above - The mean of the target variable (DAY_RETURN) in thi column's percent of the cases. (A case is a trading opportunity, which in most applications is a bar in a market.) Notice the 'ideal' behavior: the mean monotonically decreases from 0.0807 to 0.0466 as the number of cases kept increases. In other words, as the threshold for signaling a long trade is loosened (decreased) to produce more trades, the mean win for those trades decreases. The best long trades, with a mean DAY_RETURN of 0.0807, are obtained by keeping only the top te percent of predictions. But remember, we are still in the training set! OOS results, which are what we are really interested in, will appea later.

N wins above - The number of trades in this 'at or beyond the threshold' set which have a positive target value. (A future version of the *TSSB* program will also express this as a percentage of trades.)

Mean win above - The mean target value for winning trades (cases whose target value is positive).

Total win above - The total target value of all winning trades. In other words, if the target measures profit then this is the sum of gains on all winning trades.

N losses above, Mean loss above, Total loss above - As above, except for losing trades (those whose target value is negative).

Profit factor above - The profit factor of the cases whose prediction equals or exceeds the threshold. This is the total wins divided by the total losses, which is the industry standard definition.

The second half of this table does not add much to the prior discussion except for one thing to note: These results apply to short trades, since they involves cases whose predictions are at or below a threshold. Observe 'ideal' behavior similar to what was seen in the prior table: the mean target value is at its smallest (which is what we want for short trades) when we keep only the smallest (usually but not necessarily the most negative) ten percent of predicted values. The target mean

monotonically increases as we loosen the threshold (make it larger) so as to produce more short trades.

```
Statistic               10       25       50       75       90

Mean target below  -0.0478  -0.0035   0.0230   0.0247   0.0324
N wins below          1488     3524     6864    10261    12198
Mean win below      0.7120   0.6718   0.6429   0.6370   0.6384
Total win below     1059.5   2367.6   4412.8   6536.3   7786.6
N losses below        1285     3383     6959    10417    12631
Mean loss below     0.7120   0.6920   0.6841   0.6813   0.6862
Total loss below     915.0   2341.0   4760.5   7096.7   8667.5
Profit factor below 1.1580   1.0113   0.9270   0.9210   0.8984
```

Here is the final information for the training section of the 1999 fold:

```
MSE = 0.72075   R-squared = 0.00108   ROC area = 0.52323
Buy-and-hold profit factor=1.129 Sell-short-and-hold=0.886
Dual-thresholded outer PF = 1.245
   Outer long-only PF = 1.310   Improvement Ratio = 1.161
   Outer short-only PF = 1.160  Improvement Ratio = 1.309
```

MSE is the mean squared error of the linear regression model, and R-squared is the corresponding fraction of the target variance accounted for by the model's predictions. It's extremely low, as is the rule in financial applications. In most situations, the best predictions (those most likely to result in winning trades) are in the tails of the model's distribution of predictions, the most extreme large and small values. The bulk of the cases, whose predictions lie in the vast midland, tend to be more or less random. This is why R-squared is usually tiny.

The ROC area, which was briefly discussedhere, is of only modest interest. Recall that the value of ROC area ranges from zero (a model that gets every decision exactly wrong) to one (a model that gets every decision exactly correct), with 0.5 corresponding to random guessing. So it is nice, though not unexpected, to see a ROC area in excess of 0.5, which indicates some degree of predictive power. This is the result for the training set, after all, where we expect results to be infected with data mining bias.

The remaining parameters were also discussed in that earlier chapter following the ROC area definition, so we will not dwell on them except to note that the model improved performance over naive holding on both the long and short side. Again, this is the training set (in-sample data) being evaluated here, so we should not be surprised.

Out-of-Sample Results for This Fold

The prior results concerned the training set (1989-1998) in the first walkfoward fold. Results for the test set, 1999, now appear in the log file:

```
Out-of-sample results...

Target grand mean = 0.02750

Outer hi thresh = 0.07134  with 430 of 3024 cases at or
above (14.22 %)  Mean = 0.13452 versus 0.00976

Outer lo thresh = 0.00428  with 264 of 3024 cases at or
below (8.73 %)  Mean = -0.05468 versus 0.03536
```

The mean of the target variable DAY_RETURNis a little smaller in the test se (0.02750) than it was in the training set (0.03721). Notice that the thresholds here are the same as those in the training set. (Of course! These are the optimal thresholds that the program found for the training set, so naturally we must keep them. It would be cheating to find new optimal thresholds for the test set!) We see that 14.22 percent of the 1999 trade opportunities, 430 of the 3024, were long trades because their predictions lay at or above the upper threshold. The mean of DAY_RETURN in these long trades was 0.13452, which nicely exceeds the mea of 0.00976 for the other cases. In other words, the model performed well for this measure of success in the out-of-sample data.

We also see that 8.73 percent of the 1999 trade opportunities, 264 of the 3024, were short trades because their predictions lay at or below the lower threshold. The mean of DAY_RETURN for these short trades was-0.05468, wonderfully les than the mean of 0.03536 for the other trade opportunities that were not selected as short trades. As for long trades, the model also performed well for this measure of success for short trades.

In the script file, the user set MIN CRITERION FRACTION = 0.1 to impose restriction that at least ten percent of the trade opportunities be long trades, and another ten percent be short trades. So how can it be that only 8.73 percent of these trades were short? The answer is that any such restriction can apply only to the training data. The program must never be told anything about the test data, so it has no way of knowing how the test set will fare. In fact, it is not unusual, in rapidly changing market conditions, to have no test-set cases produce a trade, or for all cases in the test set to produce a trade. This is yet another manifestation of the non-stationarity problem.

More details are revealed in the table of results that follows. A discussion of these results follows the table.

```
Target statistics at various percentages kept...
Statistic                10       25       50       75       90

Mean target above    0.1697   0.0883   0.0487   0.0404   0.0374
N wins above            162      395      768     1143     1363
Mean win above       0.8046   0.7189   0.6709   0.6514   0.6595
Total win above       130.3    284.0    515.3    744.5    898.9
N losses above          137      347      720     1089     1312
Mean loss above      0.5774   0.6260   0.6135   0.5995   0.6076
Total loss above       79.1    217.2    441.7    652.8    797.2
Profit factor above 1.6479   1.3073   1.1666   1.1404   1.1277

Mean target below   -0.0616  -0.0113   0.0063   0.0072   0.0117
N wins below            154      377      746     1119     1329
Mean win below       0.7391   0.6848   0.6291   0.6200   0.6259
Total win below       113.8    258.2    469.3    693.7    831.9
N losses below          140      360      735     1108     1341
Mean loss below      0.6801   0.6935   0.6515   0.6409   0.6441
Total loss below       95.2    249.6    478.9    710.2    863.8
Profit factor below 1.1954   1.0341   0.9800   0.9769   0.9630
```

It is interesting and heartening to observe that even though this is out-of-sample data, we have the same monotonic relations among target means for both long and short trades as we had in the training set. The mean target for long trades runs from a high of 0.1697 for the highest ten percent of predictions, to a low of 0.0374 for the highest 90 percent predictions. Similarly, on the short side, the mean DAY_RETURN for the ten percent smallest predictions is -0.0616 (which is good because when we are short we want the market to go down). This target mean steadily rises to 0.0117 when we examine the 90 percent smallest predictions.

Just to be clear, there is no way we could in real life choose to trade only some fixed percent of the trade opportunities in the test set. We would need to be able to see into the future to know the predictions for the entire year in order to compute the threshold required to generate trades on a at least a specified percentage of trade opportunities. The trading thresholds can only be be computed from the training set. Still, after-the-fact examination like this can reveal a lot about the behavior and quality of the trading system.

```
MSE = 0.73187   R-squared = 0.00205   ROC area = 0.53318
Buy-and-hold profit factor=1.091  Sell-short-and-hold=0.916
Dual-thresholded outer PF = 1.352
   Outer long-only PF = 1.488   Improvement Ratio = 1.364
   Outer short-only PF = 1.166  Improvement Ratio = 1.272
```

These statistics were discussed earlier, so we will not dwell on them here other than to note that improvement ratios of 1.364 for long trades and 1.272 for short trades is quite respectable for a simple trading system like this.

The Walkforward Summary

After the results for every individual fold have been reported, *TSSB* pools all OOS folds into a single set and computes various performance statistics.

```
-------------------------------------------------
Walkforward is complete.  Summary...
-------------------------------------------------

Pooled out-of-sample...

Target grand mean = 0.00679

4674 of 36732 cases (12.72%) at or
above outer high threshold  (Mean = 0.02957 versus 0.00346)

3379 of 36732 cases (9.20%) at or
below outer low threshold  (Mean = -0.00963 versus 0.00845)

MSE = 0.51079   R-squared = -0.00101   ROC area = 0.50802
Buy-and-hold profit factor=1.026  Sell-short-and-hold=0.974
Dual-thresholded outer PF = 1.075
   Outer long-only PF = 1.099   Improvement Ratio = 1.071
   Outer short-only PF = 1.036  Improvement Ratio = 1.063
```

None of these quantities is new. They are the same items that were reported for individual folds and discussed earlier. The only difference is that these results are computed from the pooled decisions of all test (out-of-sample) folds.

There is one aspect of this pooled summary that will not concern casual users, but that may be of interest to those who want to understand the inner workings of the program. For each individual fold, both IS (training set) and OOS (test set) tables are printed that contain detailed performance statistics at each of 5 different percentiles. Such a table appeared here. But no such table appears for the pooled summary. Why?

The answer is that it would not make sense. In the discussion of the table here, it was noted that the table is of marginal meaning for OOS data, because the thresholds to obtain the five percentiles are computed with OOS predictions that would not be known in advance. When the data is pooled across folds, the situation becomes even more inappropriate. Suppose that due to normal market variation, the predictions for some OOS fold are unusually large. Such variation is common Then, when we compute a threshold based on a percentile of the pooled data, the highest set will be dominated by that unusually high fold. To have the results in a table like this be so dependent on the behavior of the model in one particular fold or small subset of folds would be misleading in the extreme, so the program refrains from printing the table.

Astute readers will then wonder about the veracity of the means of the target variable above and below the long and short thresholds, such as '**`Mean = 0.02957 versus 0.00346`**' in the example just shown. Would these not suffer from the same problem? No. The reason is that rather than using a single threshold to cover all of the pooled OOS data, these means are based on the separate trading thresholds computed from the training data in each individual fold. Thus, the fact that these thresholds are based on training data, not OOS data, and the fact that each fold's threshold is considered separately, tells us that that these comparative means are legitimate and meaningful.

If you do not understand this explanation, don't worry. It's not crucial to correct use of *TSSB*. Just remember that the difference in means above and below the threshold printed in the pooled summary is an honest measure of the results that would have been obtained in real life had the model been used. Also remember that the table of detailed statistics is not printed for the pooled summary, and there is a good reason for this omission: it would be misleading.

A Simple Filter System

We now advance to a prediction-model-based method for filtering the trades of an existing trading system. It is assumed that the reader has read and digested the prior chapter that presented a standalone trading system. Repeated explanations will be minimized.

The idea behind filtering is that we already have a trading system that performs fairly well, and we wish to improve its performance by executing only a superior subset of its suggested trades. One could filter a worthless trading system, such as one that is based on coin tosses, and often achieve decent results. However, there is little point in doing so. One would be better off just designing a standalone system.

This chapter will show and discuss all of the files that are necessary to implement a simple filter, although the discussion of this filter system will be limited to an overview of its operation. Extensive details of individual components will appear later in this document, and we will provide references to these details here to satisfy readers who want to occasionally flip ahead.

The Trade File

In order to build and test filter systems, we must provide *TSSB* with a file that contains the trades produced by the existing system that we wish to filter. This is an ASCII text file in what we call the*TSSB Database Format* This format has some requirements that will be discussed in detail here. However, we will touch on the requirements here in order to orient the reader to the basic principles involved. Note that this format is a subset of universally recognized database standards, and *TSSB* database files can be easily written and read by common statistical analysis and spreadsheet programs such as Microsoft Excel® and similar packages.

Here are a few early lines from the trade file used in this example:

```
Date Market ATR65 PROFIT SCALEDPROFIT
19881215     SPY    0.25461501    0.20000000    0.78549898
19890508     SPY    0.25753799    0.10000000    0.38829201
19890615     SPY    0.26400000    0.23000000    0.87121201
19910429     SPY    0.47338501    0.66000003    1.39421499
19910430    QQQQ    0.11830800    0.08000000    0.67620301
19910607    QQQQ    0.11230800   -0.07000000   -0.62328798
19910607     SPY    0.42815399    0.28000000    0.65397000
19910624    QQQQ    0.10753800    0.10000000    0.92989999
```

The first line of the file lists the fields that will appear on subsequent lines. Here, there are five fields: Date, Market, ATR65, PROFIT, and SCALEDPROFIT. The field names are separated by blanks, although commas or tabs are also legal. Case (upper or lower) is ignored because *TSSB* will automatically convert all letters to upper case. This means that the user can employ whatever capitalization scheme is most clear, and yet be confident that inconsistent capitalization will not cause problems in the program. Because this first line is just a list of names, there is no requirement that they line up with their associated columns, although of course the user can include extra spaces if, for the sake of visual appearance, the user wants them to line up.

Note that ATR65 and PROFIT will not be used in this example. The target variable will be SCALEDPROFIT, as will be seen soon. It is perfectly legal to include an number of extraneous variables in the file.

Also note that this file should be in chronological order. There are ways for *TSSB* to handle database files that are not chronological, but operation is always faster and easier if the file is ordered by date (and time, for intraday data).

We'll now examine one line, the first data line, in this file. The date is specified as YYYYMMDD, so this trade was signaled as of the close on December 15, 1988. This is the date on which all indicators are known, but before the trade is initiated. So in this example, all indicators would have been computed after the market closed on December 15, and the position would be entered the next trading day, or perhaps some time after the close in after-market trading. This timing of 'after indicators are known' as well as 'before the trade is opened' is necessary for any filtering system. We need 'after indicators are known' in order for the filtering system to have access to all of the information it needs to make a decision, and we need 'before the trade is opened' so that the filtering system can make a decision to execute or abort the trade before a position is actually opened.

Continuing on this data line in the file, we see that the trade occurred in the market SPY. The ATR65 variable (average true range over the past 65 days) as of the close of trading that day was 0.25461501 points, and the trading system made a profit of 0.2 points. Dividing this profit by ATR65 gives a volatility-normalized profit of 0.78549898. This last quantity is the target that this example will use. Note that *TSSB* neither knows nor cares about exactly when or how the system being filtered opened its position. All it has and needs is the profit/loss of the trade and the assurance that the trade was initiated after the close of the market as of the specified date.

When designing a filter system, it is best for the filter to specialize in either long trades or short trades. Thus, if we want to develop a filter for an existing system that executes both long and short trades, we need to create two separate filters and

employ the long filter when a long trade is signaled by the existing system, and employ the short filter when a short trade is signaled. In the example presented here, we are considering only long trades. This distinction will be crucial in specifying model options in the script file, and in interpreting results in the AUDIT.LOG results file.

The Script File

The script file for this example is named SIMPLE_FILTER.SCR. Its contents a
now listed, and a discussion of each line follows.

```
READ MARKET LIST "ETF20.TXT" ;
READ MARKET HISTORIES "E:\ETF\AADR.TXT" ;
CLEAN RAW DATA 0.65 ;
READ VARIABLE LIST "TREND_VOLATILITY.TXT" ;
APPEND DATABASE "MEANREV.DAT" ;
ScaledProfit IS PROFIT ;

MODEL FILTMOD IS GRNN [
   INPUT = [ P_TREND P_VOLATILITY ]
   OUTPUT = ScaledProfit
   MAX STEPWISE = 2
   CRITERION = LONG PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   ] ;

WALK FORWARD BY YEAR 5 2007 ;
```

Here is a brief discussion of each line in the script file. Note that most of these lines were already discussed in the prior chapter covering a simple standalone trading system. Repetition will be minimized here.

READ MARKET LIST "ETF20.TXT";

This tells *TSSB* which markets will take part in the development. These markets are listed in the file ETF20.TXT, analogous to the file SYMBOLS.TXT in the SIMPLE_STANDALONE.SCR script file s in the prior chapter. The READ MARKET LIST command is discusse in detail here.

READ MARKET HISTORIES "E:\ETF\AADR.TXT";

The market histories are ASCII files that contain the date, optional time market symbol, open, high, low, close, and volume. We saw this same command, though naming a different file, in the SIMPLE_STANDALONE.SCR example in the prior chapter. T READ MARKET HISTORIES command is discussed in detail here.

CLEAN RAW DATA 0.65;

This is an optional command that could have been used in the simple standalone system of the prior chapter, but was omitted to avoid hitting the reader with too many ideas at once. Because this command is almost always employed, we'll introduce it here. The function of the CLEAN RAW DATA command is to protect the study from wild, likely

erroneous prices in the market history files. For the entire price history of each market, the closing price of each bar is divided by the closing price of the prior bar. If this ratio, or its reciprocal, is less than the specified quantity (0.65 in this example) then this bar is flagged as erroneous data and is not used for computing any variable.

READ VARIABLE LIST "TREND_VOLATILITY.TXT";

This is the same *variable definition list* file that appeared in the simple standalone example in the prior chapter. The READ VARIABLE LIS command is discussed in detail here.

APPEND DATABASE "MEANREV.DAT";

here we saw that in a filter application, the user supplies *TSSB* with an ASCII text file that contains the trades produced by the system being filtered. This file is read by means of the APPEND DATABAS command. Note that the prior command, READ VARIABLE LIS created a database that contains the variables specified in the variable definition list file, TREND_VOLATILITY.TXT. So at the time th program reads the trade file MEANREV.DAT, a database already exists. Thus, we need to *append* the trade file information to the existing database of internally computed indicators. *TSSB* also supports a READ DATABASE (here) command which lets the user avoid having to deal with market files and variable definition list files. However, the READ DATABASE command cannot be used when a database alread exists, which is the situation in this filtering example. When a database already exists, we need to use the APPEND DATABASE comman which is discussed in detail here.

ScaledProfit IS PROFIT;

The IS PROFIT command will usually be needed when the targe variable (named in the OUTPUT statement) is read from a database file, as opposed to being computed internally. In this simple filter example, the target variable is ScaledProfit, the profit of each trade produced by the existing system that is to be filtered. This variable is read from the database MEANREV.DAT by the APPEND DATABASE commar shown above. The reason we need the IS PROFIT command is tha *TSSB* contains a safety feature to prevent the user from accidentally misinterpreting results. This safety feature suppresses printing of any performance measure that involves profit (such as profit factors) when the target variable is not actually a profit. Such targets do exist. For example, the trend of prices in the near future can be used as a target being predicted, but it is certainly not a profit. Profit factors based on slope would be meaningless. When targets are computed internally, *TSSB* knows whether the target is indeed a profit. But when it's read

from a database file the program has no way of knowing whether the target is a profit. The IS PROFIT command, described in detailhere, informs *TSSB* that the target is a profit so that profit-based performance measures will be printed.

MODEL FILTMOD IS GRNN [

The MODEL command defines the modeling methodology used by *TSSB* and allows the user to name the model produced. The simple standalone example in the prior chapter employed a linear regression (LINREG) model. Just for variety, this filter example employs a different modeling approach called the General Regression Neural Network (GRNN). This model type is describedhere. The MODEI command is discussed in detail here.

INPUT = [P_TREND P_VOLATILITY]

The INPUT line specifies the indicators that will be considered as candidate inputs for the model. The two indicators here, P_TREND and P_VOLATILITY, are the same ones that were used in the simple standalone example presented in the prior chapter. The INPUT command is discussed in detail here.

OUTPUT = ScaledProfit

The OUTPUT line names the target variable, the true values of the quantity that the model will be asked to predict. The target here, ScaledProfit, was read by the APPEND DATABASE command that appeared above. Note that case (upper or lower) is ignored. *TSSB* converts all letters to upper case, which allows the user to employ case for clarity. The OUTPUT command is discussed in detail here.

MAX STEPWISE = 2

The MAX STEPWISE specification tells the stepwise indicator selection algorithm in *TSSB* the maximum number of indicators that it may use. *TSSB* may elect to use fewer than this maximum. This command is discussed in detail here.

CRITERION = LONG PROFIT FACTOR

TSSB allows the user to choose from among a variety of optimization criteria (sometimes called objective functions) for several aspects of model training. In the simple standalone example of the prior chapter we optimized PROFIT FACTOR, which takes into account performance for both long trades and short trades. In this filtering example, we are considering only long trades signaled by the existing system that we wish to filter. The MEANREV.DAT file contains only long trades Thus, it makes sense for the filter we are developing to optimize only

the profit factor of long trades. Performance of this example's filter on short trades would involve taking a short position when the existing system being filtered signals a long trade! That would be silly. The CRITERION command is discussed in detail here.

```
MIN CRITERION FRACTION = 0.1
```

This command, which we also saw in the simple standalone example of the prior chapter, specifies the minimum fraction of trade opportunities that result in actual trades on each side (long and short, counted separately). In signal filtering, a trade opportunity is a trade signaled by the system being filtered. The MIN CRITERION FRACTION comma is discussed in detail here.

```
];
```

This closing bracket completes the model specifications begun with the MODEL command. The semicolon terminates the MODEL command.

```
WALK FORWARD BY YEAR 5 2007;
```

This command specifies that the trading system will be evaluated by means of a walkforward test. BY YEAR means that the testing window will encompass one calendar year, each training period will encompass the five years prior to the test window, and the first test window will be in 2007. This command is discussed in detail here.

We examined the market and variable definition list files in the prior chapter, so we will not repeat the examination here.

The Audit Log

Everything in the audit log for this simple filter example also appeared in the prior chapter when we discussed a simple standalone system. The reader might want to glance back at that chapter for a quick review, as repetition of details will be avoided here. However, because this example is a filtering system, interpretation of some performance measures is somewhat different. We will now focus on these differences.

Here is the first part of the in-sample (training set) results for the first walkforward fold:

```
-------------------------------------------------------
Walkforward test date 2007 training 322 cases, testing 83
-------------------------------------------------------

GRNN Model FILTMOD predicting SCALEDPROFIT
Stepwise based on long pf with min fraction=0.100
(Final best crit = 1.6797)

Sigma weights:
        477.578448  P_TREND

Target grand mean = 0.37588
Outer hi thresh = 0.37587  with 36 of 322 cases
at or above (11.18 %)  Mean = 0.58157 versus 0.34999

Outer lo thresh = 0.37586  with 107 of 322 cases
at or below (33.23 %)  Mean = 0.23327 versus 0.44685
```

A detailed explanation of most of these quantities appeared in the prior chapter. The primary exception is that the prior example employed a linear regression (LINREG) model, while in the interest of variety, this example employs a Genera Regression Neural Network (GRNN) model. Thus, the printed model parameter are different. The GRNN utilizes sigma weights. Large values, such as the 477 seei here, indicate that there is only a slight relationship between the predictor (P_TREND in this fold) and the target (ScaledProfit). The GRNN model, as we as interpretation of its parameters, will be discussed in detail here.

One other small difference is that the prior chapter's example directly specified the indicators to be used by the model, while this example employs stepwise selection. When stepwise selection is used, the final value of the selection criterion is printed as the *Final best crit*. This is of little interest to most users, although advanced users may be interested in the obtained value, especially if different variables are used for the training target and price performance, as described here. In this example, the *Final best crit* of 1.6797 is the log of the long profit factor.

Recall that each record in the trade file MEANREV.DAT is the return of a single trade executed by the existing system being filtered. The implication is that in a filtering application such as this example, the Target Grand Mean (0.37588 here) has special meaning: it is the mean return of all of the trades signaled by the existing system being filtered. In other words, it is an indication of the performance of the existing system alone, with no filtering. It has nothing to do with gains produced by *TSSB*, although knowing this figure for the unfiltered system is always useful.

The means above and below the upper (long) threshold are the first indication we see of the ability of this model to effectively filter an existing system. We see that 36 of the 322 training cases lie at or above the optimal threshold, and these trades had a mean target (ScaledProfit) of 0.58157, as opposed to a mean of 0.34999 for those trades below the threshold. This is not as impressive as it sounds, though, because these are in-sample (training set) results and hence infected with training bias.

The means relative to the lower (short) threshold are of no interest whatsoever in this filtering example. As was discussed earlier in this chapter, any results pertaining to short trades would involve taking a short position when the existing system being filtered signaled a long trade. This would be absurd, so short-side results must be ignored.

This brings up a subtle but important issue. The example of this chapter filters a long system, and for this reason short-side results are meaningless. What about when we filter a short system? Do we examine long-side or short-side results? The answer is that it depends on what the target variable in the trade file represents. If the target is (possibly scaled) market moves (future market returns), a short trade scores a win when the market goes down (the target is negative). For this reason, we would ignore long-side results, and pay attention to only short-side results. On the other hand, if the target in the file is the profit of the short trade, then positive values are wins, so we examine long-side results and ignore the short side. For a system that generates only signals to enter long trades, market moves and profits are the same thing, so this issue does not arise. But for a system that generates only short trades, the profit is the negative of the market move. Thus, for short systems we need to act according to whether the trade file contains market moves or profits.

The next item in the log file is the complex table of performance statistics at various percentiles, and the final item is information on profit factors. We'll skip both of these for now because this is still training set results. We'll cover these items in the next section when test set results are presented.

Out-of-Sample Results for This Fold

The prior results concerned the training set (2002-2006) in the first walkfoward fold. Results for the test set, 2007, now appear in the log file:

```
Out-of-sample results...

Target grand mean = 0.23086

Outer hi thresh = 0.37587  with 5 of 83 cases
at or above (6.02 %)  Mean = 0.72830 versus 0.19897

Outer lo thresh = 0.37586  with 39 of 83 cases
at or below (46.99 %)  Mean = 0.42834 versus 0.05582

Target statistics at various percentages kept...
Statistic               10      25      50      75      90

Mean target above  -0.0414 -0.2261  0.0128  0.1676  0.2518
N wins above             5      15      30      47      55
Mean win above      0.9991  0.7390  0.9494  0.9217  1.0011
Total win above        5.0    11.1    28.5    43.3    55.1
N losses above           3       6      12      16      20
Mean loss above     1.7756  2.6389  2.3285  2.0477  1.8090
Total loss above       5.3    15.8    27.9    32.8    36.2
Profit factor above 0.9378  0.7001  1.0193  1.3222  1.5220
```

The target mean in the test set (2007) is considerably less than the target mean in the training set (0.23086 versus 0.37588). This, of course, has nothing to do with *TSSB*. This disparity simply signifies the fact that the system being filtered had much lower mean profit per trade in 2007 than it did in the 2002-2006 training period.

There were 83 cases (trades signaled by the system being filtered) in 2007, of which 5 had predictions that equaled or exceeded the threshold computed during training. The mean profit of these trades was 0.72830, while the mean of the 83-5=78 trades signaled by the existing system but rejected by this filter was just 0.19897. This is impressive performance, considering that this is out-of-sample data. As discussed earlier, results relative to the low (short) threshold are meaningless, because we are filtering a long system.

The table of detailed performance measures reveals something interesting that should temper our enthusiasm over the results that are relative to the optimal training-set threshold. We just saw that when we kept the 5 of 83 (6.02 percent) of trades having the highest predicted outcomes, we had an impressive mean profit of 0.72830. But this table shows that if we kept just a few more, 10 percent, we would actually incur a loss (-0.0414)! Looking further down the first column, we see that

these 8 cases (10 percent of 83) had 5 wins totaling 5.0, and 3 losses totaling -5.3, for a net loss. A reasonable explanation would be that one of these 8 trades was an unusually large loss. Later in this tutorial, here, we will present a graphic capability built into *TSSB* that will let us take a close look at the distribution of wins and losses above and below any threshold.

Finally, the test-set model and profit factor breakdown is as follows:

```
MSE = 2.30199   R-squared = -0.00922   ROC area = 0.37374

Buy-and-hold profit factor = 1.507
Sell-short-and-hold = 0.664

Dual-thresholded outer PF = 0.510
   Outer long-only PF = 30.244   Improvement Ratio = 20.068
   Outer short-only PF = 0.371  Improvement Ratio = 0.559
```

The results just shown present an excellent example of seemingly anomalous but common behavior in financial modeling. R-squared is negative, indicating that overall, the model did worse at predicting trade outcomes than simply guessing the mean trade would have done. Yet despite this, the model did an excellent job of filtering. In particular, the *Buy-and-hold profit factor* of 1.507 is the profit factor that would be obtained by trading the existing system exactly according to its signals, with no filtering. With filtering, we obtain a profit factor of 30.244, an improvement by a factor of 20.068! (As has been mentioned several times, figures involving short trades are meaningless because we are filtering a long system. For this reason, the dual-thresholded profit factor, which includes both long and short trades, is also meaningless.)

So how can we have such phenomenal results, in the out-of-sample set no less, and yet still have a negative R-squared? The performance table shown on the prior page provides a clear illustration of the answer. We obtained a wonderful mean profit per trade when we used a threshold that kept about six percent of the signaled trades, but the mean profit per trade turned negative when we kept ten percent of the trades. A very common effect in financial applications is that most or all of the useful predictive information is out in the tails, the extreme values of indicators and predictions. A good training algorithm will find and capitalize on this information. Meanwhile, the model will perform terribly everywhere else, so badly that its overall performance as measured by R-squared will be tiny or even negative.

The Walkforward Summary

We conclude this discussion of a simple filter with a look at the walkforward

summary. The contents of such a summary were discussed in depth in the prior chapter. Here we will focus on only those aspects of the summary that specially pertain to signal filtering.

```
-----------------------------------------------------
Walkforward is complete.  Summary...
-----------------------------------------------------

Pooled out-of-sample...
Target grand mean = 0.37774

28 of 236 cases (11.86%) at or above outer high threshold
(Mean = 0.76369 versus 0.32579)

56 of 236 cases (23.73%) at or below outer low threshold
(Mean = 0.42975 versus 0.36156)

MSE = 1.58079   R-squared = -0.00510   ROC area = 0.35389

Buy-and-hold profit factor = 2.176
Sell-short-and-hold = 0.460

Dual-thresholded outer PF = 0.934
   Outer long-only PF = 13.975   Improvement Ratio = 6.423
   Outer short-only PF = 0.378  Improvement Ratio = 0.823
```

After pooling all test periods into one set, the total number of cases (trades signaled by the system being filtered) is 236. Of these, 28 had predictions in the test period that equaled or exceeded the threshold computed in the corresponding training period. The mean target (ScaledProfit) for these select trades was 0.76369, which is more than twice the mean for the trades that the *TSSB* filter rejected.

As we saw in the examination of the first fold, we have a negative R-squared despite having excellent performance as a trade filter. The *Buy-and-hold profit factor* of 2.176 is what the original, unfiltered system obtained across the pooled test period. By accepting only those trades whose predicted value equaled or exceeded the trained optimal value, we obtained a profit factor of 13.975. This greatly reduces the number of trades, as we kept only 28 of 236 signaled trades. Still, this degree of performance is exemplary.

Finally, note that if keeping so few trades is not an option, it is easy to increase the number of trades kept. The following line appeared in the model definition part of the script file:

```
MIN CRITERION FRACTION = 0.1
```

This tells the training algorithm that when it sets the optimal threshold for accepting

or rejecting signaled trades, it must set the threshold such that at least ten percent (0.1) of the trades in the training set are kept. If we raise this to a larger quantity, the threshold will be adjusted accordingly. More trades will be kept in the training set, and almost certainly more trades will also be kept when the model is tested or put to actual use. But be warned... we must never forget the fundamental principle that the best predictive information is almost always in extreme values. If we raise the MINIMUM CRITERION FRACTION too much, we will be accepting a la number of cases (signaled trades) whose predictions are of little value. Performance will suffer, probably a lot.

Common Initial Commands

In most *TSSB* applications, the first step is to read the market data and compute variables to build a database of indicators and targets. This may be done with every execution of the development script, or it may be done just once, saving the resultant database, and then reading the saved database for subsequent executions. Saving and restoring databases will be discussed on Pages here and here, respectively. This chapter will discuss reading market files, selecting markets and time periods, and reading variable definition list files. The emphasis of this chapter will be the specific commands used to perform these operations, and the order in which the operations are performed.

Market Price Histories and Variables

TSSB references two types of data: *market price histories* and *variables*. *Market price histories* are the price and (usually) volume information for financial markets. The price generally includes the open, high, low, and close of each bar, although it is perfectly legal for all four of these quantities to be equal if the market information is ticks or some other quantity that the user wants to use as if it were a market.

The other data type, *variables*, comprise quantities that are computed from market price histories, volume, or other financial series. Variables may be things such as measures of trend or volatility, or more sophisticated quantities. They are stored in the *database*, and they may be used as indicators or targets. The database of variables may be written to or read from a text file to avoid having to recompute them every time a study is performed. This text file of variables may also be used to transfer data to and from third-party programs such as Excel and statistical packages.

TSSB has built-in functions that can operate on market price histories and variables. It contains a library of over 100 predefined indicator and target families from which the user can create variables from market histories. This topic will be discussed here. The program also allows the user to apply built-in functions and transformations, including fully general arithmetic scalar and vector operations, to both market price histories and variables. This is so even for variables created by other programs and read in as a database. These operations, called *expression transforms*, are discussed here.

In summary, the terminology essential to understanding this chapter is as follows:

- *Market price histories* are the price and volume information for markets.
- *Variables* are quantities computed from market prices histories. They reside in the *database* and may be used as *indicator*s and *target*s.

Quick Reference to Initial Commands

We begin with a quick reference guide to the commands that will be presented in this chapter. Every command here is related in some way to creation of the database of indicators and targets from market files and a variable definition list file. This list is presented in an order which would be legal and common in *TSSB*. However, some other orderings are also legitimate. See the notes for individual commands regarding order.

INTRADAY BY MINUTE Specifies that all market files are intraday, with time resolved to minutes. If used, this must precede the READ MARKE HISTORIES command. See here.

INTRADAY BY SECOND Specifies that all market files are intraday, with time resolved to seconds. If used, this must precede the READ MARKE HISTORIES command. See here.

MARKET DATE FORMAT YYMMDD Alters the date format in market files from the default YYYYMMDD to YYMMDD. In this case, the leg date range is 1920 through 2019, inclusive. If used, this must precede the READ MARKET HISTORIES command. See here.

MARKET DATE FORMAT M_D_YYYY Alters the date format in market files from the default YYYYMMDD to Month/Day/Year. If used, this mus precede the READ MARKET HISTORIES command. See here.

MARKET DATE FORMAT AUTOMATIC Automatically, for each market file, determines whether the date format is YYYYMMDD o Month/Day/Year. If used, this must precede the READ MARKE HISTORIES command. See here.

REMOVE ZERO VOLUME Removes all market history records having zero volume. If used, this must precede the READ MARKET HISTORI command. See here.

READ MARKET LIST - Reads a text file that lists the markets that will be used in the study. This must precede the READ MARKET HISTORI command. See here.

READ MARKET HISTORIES Reads all of the market history files that were listed in the file specified with the READ MARKET LIST comman See here.

MARKET SCAN- Scans one or all markets for unusual price jumps that may indicate errors in the market file. This optional command must not appear until after the market(s) have been read with the REAL MARKET HISTORIES command. See here.

The preceding commands are all associated with reading one or more market history files into *TSSB*. The following commands are associated with using these market histories to compute a database of indicator and/or target variables.

RETAIN YEARS- Specifies that only a specified range of years be read from the

market file(s) and/or computed for the database. If used, this must precede the READ VARIABLE LIST command. Its position before c after the READ MARKET HISTORIES command is significant, and most cases it should follow this command. See here.

RETAIN MOD- Specifies that every *n*'th database date (and time, if intraday) be kept. This is useful for temporarily reducing the size of the database to speed operation during early development phases. (The term *MOD* comes from the algebraic operation of *modulo arithmetic*.) If used, this must precede the READ VARIABLE LIST command. See here.

CLEAN RAW DATA - Decrees that suspiciously large market price jumps be flagged as invalid data so that no variable will be computed using this data. If used, this must precede the READ VARIABLE LIST comman See here.

INDEX IS- Names a particular market as an ***index***. Index markets have special uses when computing indicators. If used, this must precede the REAL VARIABLE LIST command. See here.

READ VARIABLE LIST- Reads the variable definition list file and builds a database of all variables. See here.

OUTLIER SCAN- Scans one or all variables in the computed database for outliers that may indicate errors in the market file or unstable market conditions. This command must not appear until after the database of variables has been computed with the READ VARIABLE LIS command. See here.

DESCRIBE - Prints a statistical summary of a variable. See here.

CROSS MARKET AD- Performs an Anderson-Darling test for conformity of a variable across multiple markets. See here.

CROSS MARKET KL- Performs a Kullbach-Liebler test for conformity of a variable across multiple markets. See here.

CROSS MARKET IQ Performs an interquartile range overlap test for conformity of a variable across multiple markets. See here.

STATIONARITY - Performs a suite of tests to identify nonstationary behavior in a variable. See here.

Detailed Descriptions

This section provides more detailed descriptions of the commands that read market history files and create a database of indicators and targets.

INTRADAY BY MINUTE

The format for this command is as follows:

```
INTRADAY BY MINUTE ;
```

By default, there is no time field in a market history file. Each record is for a single day, so only the date appears. However, if the INTRADAY BY MINUTE comman appears before the READ MARKET HISTORIES command, it is assumed that each market record the date is followed by the time of each bar. (If the INTRADAY BYMINUTE command appears *after* the READ MARKET HISTORIES comman it is ignored and has no effect.) This time is encoded as HHMM in 24-hour format Here are a few typical records in an intraday market history file that resolves to the closest minute. The remaining fields on each record are the price open, high, low, and close, and finally the volume.

```
20110301 0830 16.90 16.94 16.21 16.23 297967
20110301 0831 16.20 16.43 16.13 16.18 201850
20110301 0832 16.37 16.77 16.36 16.63 191486
20110301 0833 16.77 16.80 16.37 16.58 186114
20110301 0834 16.58 16.75 16.16 16.17 119222
```

INTRADAY BY SECOND

This is identical to the INTRADAY BY MINUTE option just described, except th the time field includes seconds, as HHMMSS. A typical intraday market history file that resolves to the second might have records that look like the following lines:

```
20110301 083000 16.90 16.94 16.21 16.23 297967
20110301 083001 16.20 16.43 16.13 16.18 201850
20110301 083002 16.37 16.77 16.36 16.63 191486
20110301 083003 16.77 16.80 16.37 16.58 186114
20110301 083004 16.58 16.75 16.16 16.17 119222
```

MARKET DATE FORMAT YYMMDD

The format for this command is as follows:

```
MARKET DATE FORMAT YYMMDD ;
```

By default, the date in a market history file contains the full 4-digit year. In other words, the date is expressed as YYYYMMDD. If the file expresses the date using only two digits as YYMMDD, then the TWO DIGIT YEAR command must appear before the READ MARKET HISTORIES command. In this case, the legal date range is 1920 through 2019, inclusive. A file that uses two digit years might contain records that look like those shown below. These records are for the year 2011.

```
110301 16.90 16.94 16.21 16.23 297967
110301 16.20 16.43 16.13 16.18 201850
110301 16.37 16.77 16.36 16.63 191486
110301 16.77 16.80 16.37 16.58 186114
110301 16.58 16.75 16.16 16.17 119222
```

MARKET DATE FORMAT M_D_YYYY

The format for this command is as follows:

```
MARKET DATE FORMAT M_D_YYYY ;
```

This changes the date format for all market files to the usual American standard. The month and the day can be one or two digits. They year must be four digits.

```
3/1/2011 16.90 16.94 16.21 16.23 297967
```

MARKET DATE FORMAT AUTOMATIC

The format for this command is as follows:

```
MARKET DATE FORMAT AUTOMATIC ;
```

This allows different market files to have different date formats. The first record of the file will be read and the date format determined from that. At this time, only YYYYMMDD and M_D_YYYY are allowed.

REMOVE ZERO VOLUME

Sometimes the market history supplied by a vendor may include records that have a volume of zero. For example, on 9/27/1985 there was a severe weather situation that interfered with trading in New York markets. Many issues did not trade at all that day. Nonetheless, since this was a legal trading day and some markets did trade, instead of simply deleting this date from market histories, some vendors include it but set the volume to zero. This has a minor impact on most operations. However, it severely impacts the TRAIN PERMUTED operation (here). For this reason, the following command is provided:

```
REMOVE ZERO VOLUME ;
```

This command causes market records having zero volume to be skipped as their history files are read. (In Version 1 of *TSSB*, zero-volume records are always kept.) If used, this command must precede the READ MARKET HISTORIE command. This command is required if a TRAIN PERMUTED command appea in the script file, even if the market(s) read do not contain any zero-volume records.

READ MARKET LIST

The user may wish to repeat the same study on several different collections of markets. The easiest way to facilitate this versatility is to allow the user to list all of the desired markets in a single text file, and then reference this text file in the script file that controls the study. Such a file is, not surprisingly, called a *market list file*. So, for example, the user might have one market list file that lists the components of the Dow Jones composite index, another that lists the markets in the S&P 500, another that lists numerous bank stocks, and so forth. The lines in a market list file might look like this:

```
AA
DELL
DOW
GE
...
```

The READ MARKET LIST command specifies the name of the market list fil enclosed in double quotes ("). The file name may not include blanks or special characters other than the underscore (_). If a full path name is not specified, *TSSB* will assume that the file is in the current working directory as defined by the Windows operating system. The format of this command is as follows:

```
READ MARKET LIST  "FileName" ;
```

By listing the markets in a single file, the user can easily repeat the study in

different sets of markets by changing only the name of the market list file in the READ MARKET LIST command. For example, the following command woul perform the study on a set of markets that is made up of utilities on the east coast (assuming that the user named the file intelligently!):

```
READ MARKET LIST  "EastCoastUtilities.txt" ;
```

READ MARKET HISTORIES

All of the commands prior to this one are preparatory; they tell *TSSB* about the format of the market history file(s) and specify which files are to be read. It is the READ MARKET HISTORIES command that actually reads the files. For th reason, if any of the preceding commands appear *after* the READ MARKE HISTORIES command, they are ignored; it's too late.

This command must name one market history file in the directory in which all history files reside. The file need not be in the market list. It only needs to be present in the directory. The format of this command is as follows:

```
READ MARKET HISTORIES  "AnyMarketFile"  ;
```

The file name is enclosed in double quotes ("). The file name may not include blanks or special characters other than the underscore (_). If a full path name is not specified, *TSSB* will assume that the file is in the current working directory as defined by the Windows operating system.

Why name a single file instead of simply naming the directory where the files are located? There are several reasons. First, this is a convenient way to simultaneously specify the extension. Otherwise, the user would have to use a separate command to tell *TSSB* what extension to use. So naming a file simplifies things for the user and makes the script file one line shorter. Also, it is a bit of insurance against carelessness by the user. By imposing the requirement that the user name a file that exists in the directory, *TSSB* is able to generate a clear error message for the user in case the directory or extension is not correct. In other words, it can display the file name, including path and extension, so that it's obvious where the program is looking and what it's looking for.

TSSB keeps the same path and extension for all market files, and just substitutes the market names as listed in the market list file. For example, suppose the market list file contains three markets as follows:

```
IBM
T
BOL
```

Suppose also that the following READ MARKET HISTORIES command is used:

```
READ MARKET HISTORIES  "\MyMarkets\CIT.TXT"  ;
```

In this case, *TSSB* will look in the directory ***\MyMarkets*** for all market history files, and it will use the ***.TXT*** extension for all of them. In particular, it will attempt to read the following three files:

```
\MyMarkets\IBM.TXT
\MyMarkets\T.TXT
\MyMarkets\BOL.TXT
```

MARKET SCAN

This command scans one or all markets for unusual price jumps that may indicate errors in the market file. This optional command must not appear until after the market(s) have been read with the READ MARKET HISTORIES command.

If a market is named in this command, the scan will be done for only that one market. If no market is named, all markets will be scanned. For example, the following command will scan only IBM:

```
MARKET SCAN  IBM  ;
```

For each bar, the scanning algorithm computes the high minus the low and the absolute difference between the open and the prior day's close. The greater of these two quantities is divided by the lesser of today's close and yesterday's close, and the quotient is multiplied by 100 to express the difference as a percent.

If only one market is scanned, the largest 20 differences, along with their dates, are listed in descending order. If all markets are scanned simultaneously, each market's 20 worst dates are listed in descending order, as is the case for one market. However, the market results are sorted according to the magnitude of each market's worst gap. Thus, results for the market having the single worst gap appears first, followed by the market having the second-worst gap, and so forth. This command can also be accessed through the menu system.

The preceding commands are all associated with reading one or more market history files into *TSSB*. The following commands are associated with using these

market histories to compute a database of indicator and/or target variables.

RETAIN YEARS

It is often the case that the user may wish to perform a study on only a subset of the complete market history. Perhaps the user wants to implement a simple virgin-data development strategy, in which the most recent year or few years are held back from development of the trading system. Then later, after a trading system is in hand, it is tested on the recent data that was excluded from the development phase.

More commonly, the user simply wants the study to run quickly while models and options are decided upon. It can be frustrating to tweak a study when each run requires several hours of computer time! In this case, limiting the market history to a short time period can greatly speed the tweaking process. Once the models and options are decided upon, the entire historical dataset can be employed for the final study.

The format of the RETAIN YEARS command is as follows:

```
RETAIN YEARS  FirstYear THROUGH LastYear  ;
```

The specified years are inclusive. Thus, the following command would retain the years 1997, 1998, and 1999:

```
RETAIN YEARS 1997 THROUGH 1999 ;
```

The position of this command in the script file has a subtle but important effect. It must always precede the READ VARIABLE LIST command if variables are computed internally, or the READ DATABASE command (described here) if variables are read from an external file. If the RETAIN YEARS command *follows* these commands it will be ignored; it's too late, because the database is already created.

But the important distinction arises according to the position of the RETAIN YEARS command relative to the READ MARKET HISTORIES command. In vast majority of applications, the RETAIN YEARS command should *follow* the READ MARKET HISTORIES command. This way, the entire history of every market will be read, which allows the internal variables to be based on the full extent of the available data. If instead the RETAIN YEARS command *precedes* the READ MARKET HISTORIES command, then only the specified years of market history will be read. This will deny some otherwise available data to *TSSB* when it computes variables.

For example, suppose that one of the indicators the user wishes to create is a linear trend that looks back 100 days. Suppose also that the user has data back to 1990 and wants the study to run from 1992 through 1999. The following command is used:

```
RETAIN YEARS 1992 THROUGH 1999 ;
```

If this command appears *after* the READ MARKET HISTORIES command and (required) before the READ VARIABLE LIST command, when the value of th 100-day trend is computed for the first trading day of 1992, which is where the database will begin, it will have two years of prior data to work with. This is because the market histories were read *before* the RETAIN YEARS comman appeared. These two years (1990 and 1991) are obviously more than enough to look back 100 days to compute the trend.

On the other hand, suppose the RETAIN YEARS command appears *before* the READ MARKET HISTORIES command. Now, all market history prior to 1992 rejected. Thus, when *TSSB* tries to compute the value of the 100-day trend indicator for the first trading day of 1992, it will have no history to look at. It will have to let the first 100 days of 1992 pass before it can begin computing the trend indicator.

The same effect applies to the ending date and targets, which look into the future. So it should be obvious that when a variable definition list contains a variety of lookback and look-ahead distances, the results will be complex. The bottom line is that in nearly all cases, the RETAIN YEARS command should appear *after* the READ MARKET HISTORIES command so that all available history is availab to *TSSB*.

The only exception might be if one or more market history files go back much, much further than is needed, and the user wants to save computer memory by reading only part of the history. In this case, it is fine to use *two* RETAIN YEARS commands one prior to the READ MARKET HISTORIES, and one after. For exampl suppose the most distant lookback among indicators is two years, and the most distant look-ahead is one year. Then we might do this:

```
RETAIN YEARS 1992 THROUGH 2003 ;
READ MARKET HISTORIES  "\MyMarkets\CIT.TXT" ;
RETAIN YEARS 1994 THROUGH 2002 ;
```

But please note that this degree of complexity is almost never necessary, let alone desirable. Unless the market history files are gigantic and computer memory is limited, just put the RETAIN YEARS command after the READ MARKE HISTORIES command. Keep things simple.

RETAIN MOD

This command directs that the database be decimated by keeping one date, skipping several, keeping one, skipping several, and so forth. This is useful for temporarily reducing the size of the database to speed operation during early development phases. If used, this must precede the READ VARIABLE LIST or REA DATABASE (here) command. The syntax of this command is as follows:

```
RETAIN MOD  Divisor  =  Offset  ;
```

In this command, *Divisor* is the factor by which the size of the database is reduced, and *Offset* determines where the kept records begin relative to the first record in the database. *Offset* must be greater than or equal to zero and less than *Divisor*. The offset parameter facilitates testing consistency of study results.

For example, suppose we have two markets. Cases in the database may look like the following lines. In these lines, the ellipsis (...) represents the indicators and targets for the given date and market.

```
19870103  IBM   ...
19870103  BOL   ...
19870104  IBM   ...
19870104  BOL   ...
19870105  IBM   ...
19870105  BOL   ...
19870106  IBM   ...
19870106  BOL   ...
19870107  IBM   ...
19870107  BOL   ...
19870108  IBM   ...
19870108  BOL   ...
19870109  IBM   ...
19870109  BOL   ...
```

Suppose we had included the following line in the script file before the READ VARIABLE LIST command:

```
RETAIN MOD 3 = 0 ;
```

This command specifies that we will keep every third date (divisor=3) and begin with the first (offset=0). So the resultant database will look like this:

```
19870103  IBM   ...
19870103  BOL   ...
19870106  IBM   ...
19870106  BOL   ...
19870109  IBM   ...
19870109  BOL   ...
```

Now suppose we had instead included the following line in the script file before the READ VARIABLE LIST command:

```
RETAIN MOD 3 = 1 ;
```

This command specifies that we will keep every third date (divisor=3) and begin with the second (offset=1). So the resultant database will look like this:

```
19870104  IBM   ...
19870104  BOL   ...
19870107  IBM   ...
19870107  BOL   ...
```

The obvious use for the RETAIN MOD command is to shrink the database so tha studies execute quickly while models and options are being finalized by the user. But there is another interesting application of this command. Suppose we are developing models that look just one day ahead for the target, a common operation. As is well known, markets have daily changes that have very low, almost (though not quite!) negligible serial correlation. We could use the first of the following two commands in a study, and then repeat the study with the second.

```
RETAIN MOD 2 = 0 ;
RETAIN MOD 2 = 1 ;
```

The two studies would be trained and tested on datasets that cover the same total time period, and hence experience roughly the same large-scale movements. However, the targets in these two datasets would be almost independent. If the user's development strategy is stable, similar results should be obtained in both cases. If the results are wildly different, the user should be suspicious of his or her methodology.

If basing the datasets on adjacent days makes the user a little nervous because of the fear of small but significant serial correlation in the targets, the distance could easily be expanded. The following two options, one for one study and the other for a separate study, would cut the size of the usable dataset in half compared to the example just shown, but it would space the cases two days apart instead of just one.

```
RETAIN MOD 4 = 0 ;
RETAIN MOD 4 = 2 ;
```

Finally, note that there is nothing special about requiring the target to look ahead just one day, other than the fact that we can make efficient use of the historical data. Suppose the target looks ahead five days. Then we could use the following commands:

```
RETAIN MOD 10 = 0 ;
RETAIN MOD 10 = 5 ;
```

The above commands, used in separate studies, would offset the dates by five days, which is the lookahead distance, and hence make the datasets nearly independent.

CLEAN RAW DATA

Market histories are rarely perfect. Spurious price jumps are more common than most vendors are willing to admit. Also, some users may wish to disqualify price jumps that, while correct, are so unusual that they may be considered atypical and hence should be ignored. The CLEAN RAW DATA command facilitates removal of records with unusually large price changes from the market history file(s). If used, this must precede the READ VARIABLE LIST command.

The format for this command is as follows:

```
CLEAN RAW DATA  Fraction  ;
```

Each bar in each market's history is checked. If the ratio of a day's close to the prior day's close is less than the specified fraction (0-1), or greater than the specified fraction's reciprocal, the current day for the suspect market is flagged as erroneous. This date in this market will not be used to compute any variable (indicator or target) in the database.

This can have more far-reaching implications than is immediately obvious. Most indicators look back in time for a considerable distance. Even seemingly short lookback indicators may use a long stretch of history for normalization. For example, the LINEAR PER ATR indicator (discussed here) normally uses extensive history for normalization. The user may specify that this indicator reflect trend over the most recent ten days, but most users would normalize the trend with ATR (Average True Range) computed over a much longer time frame, perhaps as long as a year. In this case, if the CLEAN RAW DATA command results in a date being flagged as invalid, the indicator will be omitted for the next year! Even one piece of bad market history in the lookback period of an indicator, or the look-ahead period of a target, will suppress computation of that value. For this reason, large (strict) values of this parameter can result in ridiculously few cases in the

database of indicators and targets. Only rarely would values as large as 0.7 be appropriate, and 0.4 or so would usually be sufficient.

INDEX

The use of index markets will be discussed in detail here. Here we provide an introduction. The basic idea is that many sets of markets are officially characterized by a weighted average of their component markets, and we can make use of this fact. Perhaps the most famous index is the Dow Jones Industrial Average. Another common index 'market' is the S&P 100, with the symbol OEX. *TSSB* treats index markets in the same way as component markets. The user does not have to declare an index market as an INDEX unless any of the special uses of indices are specified in the variable list file. These special uses are briefly discussed below and will be discussed in detail here.

Up to 16 markets may be declared as index markets. However, it is very rare that the user would employ more than one index, so this discussion will focus on application of a single index. Multiple index markets will be discussed here. The syntax for declaring a market as an index is as follows:

```
INDEX IS MarketName ;
```

So, for example, we would declare OEX as an index as follows:

```
INDEX IS OEX ;
```

Any index declarations must precede the READ VARIABLE LIST command because index markets will be referenced in the definitions of indicators.

There are two uses for index markets when computing indicators from the built-in *TSSB* library:

1) An indicator can be based strictly on the index market rather than individual component markets. Thus, on any given date (and time, if intraday), the value of this indicator will be the same for all markets. This provides the predictive model with the 'average' behavior of all markets as opposed to the behavior of a specific market.

2) An indicator can be based on the departure of a specific market from the equivalent behavior of the index market. For example, we may be interested in the value of the RSI indicator in IBM relative to the RSI in the S&P 500 index. This tells the predictive model if a particular market is 'out of step'with the average behavior of the other markets in

the index.

Both of these uses will be discussed in detail here.

READ VARIABLE LIST

The commands discussed prior to this one specify options for computation of built-in indicators and targets. The READ VARIABLE LIST command does the work: reads the variable definition list file and builds the database of variables (indicators and targets) according to the definitions specified by the user in that file.

The READ VARIABLE LIST command specifies the name of the variable list fil enclosed in double quotes ("). The file name may not include blanks or special characters other than the underscore (_). If a full path name is not specified, *TSSB* will assume that the file is in the current working directory as defined by the Windows operating system. The format of this command is as follows:

```
READ VARIABLE LIST  "FileName"  ;
```

The contents of the variable list file are quite complex, so an entire chapter will be devoted to a detailed description of variable definitions. This chapter begins here. For the moment, understand that each line of this text file defines a single variable that is available in *TSSB*'s built-in library. Each defined variable may then be used as an indicator or target in subsequent studies.

OUTLIER SCAN

After the database of variables has been computed by means of the READ VARIABLE LIST command, or read in via the READ DATABASE comma (here), the user may be interested in assessing the validity of the cases in this database. The OUTLIER SCAN command scans one or all variables in th database for outliers that may indicate errors in the market file or unstable market conditions.

If the user names a single variable in this command, only the named variable will be scanned. If no variable is named, all variables are scanned. The format for these two options is as follows:

```
OUTLIER SCAN ;
OUTLIER SCAN FOR VariableName ;
```

The following information is printed to the AUDIT.LOG file (separately for eacl variable if all are scanned):

Name of the variable
Minimum value, and the market in which the minimum occurred
Maximum value, and the market in which the maximum occurred
Interquartile range
Ratio of the range (maximum minus minimum) to the interquartile range. Large values of this ratio indicate outliers.
Relative entropy, which ranges from zero (worthless) to one (max possible)

If all database variables are scanned, at the end of the report they will be listed sorted from worst to best, once by ratio and once by relative entropy.

The ratio of the range to the interquartile range is an extremely useful measure of the degree to which a variable contains extreme values. The interquartile range is the difference between the 75'th percentile and the 25'th percentile. It is a stable and accurate measure of the 'spread' of the variable because it examines only the central (least extreme) fifty percent of the distribution. The range is the difference between the largest and the smallest values of the variable. If a case contains an unusually large or unusually small value of this variable, the range will be large, and hence the ratio of the range to the interquartile range will also be large.

One nice property of this ratio as a measure of the degree to which one or more extreme values are present is that it is immune to scaling and offset. In other words, if one were to rescale a variable by multiplying it by a constant, and/or by adding a constant, the ratio of the range to the interquartile range will not change. This scaling immunity is an excellent property.

Entropy is much more difficult to explain and justify intuitively. Roughly speaking, entropy measures how well a variable is 'spread around' its range. Still roughly speaking, entropy is an upper bound on the amount of information that a variable can contain. An ideal indicator will usually have its values scattered equally throughout its range, resulting in high entropy. An indicator whose values lie in one or a few narrow clumps will have low entropy and probably contain little useful predictive information. Raw entropy is not immune to transformations, so the value printed here is relative entropy, which ranges from zero to one.

Entropy does not paint a complete picture, because it may be that some variable contains little information, but all of the information it does contain is useful for predicting a target. Conversely, another variable may contain an enormous amount of information, but this abundant information may be useless for predicting the particular target we are interested in. Nonetheless, entropy is a decent tool for assessing the potential utility of an indicator. At a minimum, unusually low entropy

may be used as a red flag that a variable needs refinement.

DESCRIBE

The DESCRIBE command can be used to obtain numerous statistics about a singl variable. If multiple markets are present, the user can choose whether to separate results by market or pool all markets into a single collection. The format of the command for these two options is as follows:

```
DESCRIBE VariableName IN MarketName ;
DESCRIBE VariableName ;
```

The following statistics are printed:

Mean
Standard error of the mean
t-score for the mean
Number of cases
Variance
Standard deviation
Skewness
Standard error of the skewness
Kurtosis
Standard error of the kurtosis
Median
Interquartile range
25'th and 75' percentiles
Range
Minimum and maximum values
Range divided by interquartile range
Relative entropy and number of bins used to compute this entropy

CROSS MARKET AD

This test, officially called the *Anderson-Darling test*, would be employed only when the application is using more than one market. When the user is trading multiple markets with a single model-based trading system, it is crucial that every indicator and target have at least approximately the same distribution in every market. Suppose some indicator ranges from -40 to 10 in one market, and this same variable ranges from 15 to 70 in another market. This can easily happen if variables do not properly compensate for natural variations in market behavior

such as extended trends and volatility that impact markets differently. In such a situation, no model can do a good job of predicting both markets. Sometimes the model will be good in one market and poor in the other. More often, the model will be useless in both. The CROSS MARKET AD command performs a sophisticate test of how well variables have similar distributions in all markets.

The user may choose to perform this test on just one variable, or perform it on all variables in the database. The format of this command for these two options is as follows:

```
CROSS MARKET AD FOR VariableName ;
CROSS MARKET AD ;
```

All markets are pooled to produce a single 'generic' distribution of a variable. Then the distribution of each individual market is tested for equality with the pooled distribution. Anderson-Darling statistics and their associated p-values are printed twice, once ordered by the appearance of markets in the Market List File, and a second time sorted from best fit to worst.

These p-values are for testing the null hypothesis that the distribution of a variable in a given market is equal to the pooled distribution of this variable. Thus, low p-values indicate the distribution for the market in question does not conform to the pooled distribution. This is undesirable. However, when numerous cases are present (the usual situation), these p-values, as well as the A-D statistic itself, can be excessively sensitive to differences in the distributions. A tiny p-value does not necessarily imply that discrepancies will be problematic, only that it is unlikely that if the distributions were identical we would have seen a p-value as small as that observed. When there are numerous cases, all it takes is a tiny, harmless discrepancy in the distributions to produce a very significant p-value. Thus, the best use of this test is for revealing the *worst* performers (lowest p-values). These variables/markets should be subjected to visual examination by histograms or other tests.

Note that the number of lines printed by this test in the audit log is proportional to the product of the number of variables and the number of markets. This output can become large quickly.

CROSS MARKET KL

This test, officially known as the *Kullback-Liebler* test, is similar to the Anderson-Darling test in that it tests the degree to which the distribution of a variable is similar across multiple markets. However, unlike the A-D test, the number of cases

in the dataset has no impact on the test statistic. On the other hand, the K-L test is no more intuitive in its result than the A-D test. Therefore, one should not judge the values of the K-L test statistic in isolation. Instead, one should use this statistic to identify the most problematic variables and markets so that they can be given additional attention. As with the *AD* test, small p-values indicate potential problems, and the variables and markets having the smallest p-values should be given close inspection.

The user may choose to perform this test on just one variable, or perform it on all variables in the database. The format of this command for these two options is as follows:

```
CROSS MARKET KL FOR VariableName ;
CROSS MARKET KL ;
```

CROSS MARKET IQ

Like the CROSS MARKET AD and CROSS MARKET KL tests, the CROSS MARKET IQ test checks the degree to which the distribution of a variable is similar across multiple markets. The user may choose to perform this test on just one variable, or perform it on all variables in the database. The format of this command for these two options is as follows:

```
CROSS MARKET IQ FOR VariableName ;
CROSS MARKET IQ ;
```

In this test, IQ stands for interquartile range. The IQ test roughly measures the degree to which the interquartile range of each tested market overlaps the interquartile range of the pooled data. It ranges from zero, meaning that the interquartile ranges are completely disjoint (no overlap at all) to one, meaning that the interquartile ranges overlap completely.

The IQ test has two nice properties. Like the KL test but unlike the AD test, the IQ test is not sensitive to the number of cases in the dataset. Unlike both of these earlier tests, the IQ test is strongly intuitive. The test statistic, defined above, has a meaning that is easy to grasp. Thus, one can judge each individual result (for a variable and a market) by the numeric value of the test statistic.

Unfortunately, the IQ test does have one disadvantage compared to the A-D and K-L tests: it is not very sensitive to mismatches of distributions in the tails. The IQ test looks only at the central half of the data. It is conceivable that a market will closely match the pooled distribution in this central half, resulting in an excellent

IQ score, while being very different in one or both tails. Nonetheless, this phenomenon is not common. For this reason, the IQ test is probably the best test to perform if one is going to rely on just one cross-market conformity test. On the other hand, it is advisable to perform all three tests whenever possible, because each has its own strengths and weaknesses. A variable/market pair that scores relatively poorly on even one of the three tests should invite close scrutiny.

STATIONARITY

A time series is *stationary* if and only if its statistical properties do not change as time passes. The expected value (mean) of a stationary series will not change, nor will its variance, nor its serial correlation, nor any of an infinite number of other properties. Since variation in *any* of an infinite number of properties destroys stationarity, strict testing for stationarity is impossible. Such a test would require an infinite number of component tests. Luckily, in practice it is only the grossest, usually most visible aspects of stationarity (a stable mean and variance) that are important. A suite of tests that covers the largest issues is usually sufficient.

It is at least desirable, and perhaps crucial, that all variables used in a trading system be reasonably stationary across the time period of interest. It is vital that the statistical properties of the variables will remain the same when the system is placed in operation as when it was trained and tested. Suppose, for example, you define a crude trend variable as a five-day moving average of closing price minus a ten-day moving average, and attempt to use this variable for predicting the S&P500. Decades ago, when SP was trading at prices less than 100, a one-point difference was significant. But today, with SP well over 1000, a one-point move is trivial. This sort of nonstationarity would be devastating.

The situation goes beyond just conformity between variables in the training period and the testing or implementation period. If the statistical properties of a variable change significantly during the training period, models will find it difficult to identify the subtle bits of predictive information buried in the data. It's like looking for a needle in a haystack when someone keeps taking hay out from one place and adding new hay to another place.

Because of the importance of stationarity, *TSSB* contains an extensive suite of stationarity tests. These tests can be invoked in several ways. The most versatile method is to use the menu system, because this way the user can select which tests to perform. A simpler but less versatile method is to invoke the test suite from within the script file. In this case, a default set of tests is performed. If a variable is specified, the tests are performed on that variable only. If no variable is specified, the tests are performed on all variables. The syntax for these two options is as

follows:

```
STATIONARITY OF VariableName IN MarketName ;
STATIONARITY IN MarketName ;
```

The simplest and fastest stationarity tests are based on crude cell counts. Imagine plotting a time series of the variable, using a single dot to represent the variable's value at each bar. Place a rectangular grid over the plot and count the number of cases that fall into each section of the grid. For example, suppose we use a 5 by 3 grid, dividing the time extent into five periods and dividing the range of the variable into three sections: large, medium, and small. If the series is stationary, the vertical distribution of counts should be similar across time (columns of the grid). If, on the other hand, we see an unusually large count of the 'large' variable category in one time period, but an unusually small count of the 'large' category in another time period, we suspect that the series is nonstationary. This decision can be made rigorous by using an ordinary chi-square test.

Another family of tests searches for a single point in time at which the nature of the series changes dramatically, a *structural break*. This might happen, for example, if a market moves from open-outcry trading to electronic trading, or if a new government regulation changes the nature of trading. A significant market change of this sort can be devastating to an automated trading system. *TSSB* contains several algorithms that attempt to determine if such a changeover point exists. The program identifies that point in time at which the largest such change in statistical properties occurs, and when possible attempts to assign a p-value to the change. This p-value is the probability that, if there is truly no structural break, we would have seen an apparent break as large as that found. Thus, very small p-values are bad, indicating a likely structural break.

It is well known that variables associated with financial markets often have distributions that have heavy tails (a high probability of extreme values), are skewed (more likely to have unusually large than small values, or the opposite), or have other properties that violate assumptions of a normal distribution. For this reason, *TSSB* contains many stationarity tests that are based on order statistics and hence do not assume normality. These latter tests are vastly preferable because they are robust against outliers, which can destroy conventional tests.

Theories abound that markets tend to behave differently in some months of the year than in other months, *seasonality patterns*. If any variable has such a property to a significant extent, this inconsistent behavior will hamper a model's ability to find predictive information in the data. Month-to-month changes may swamp out more subtle but important behavior. Thus, *TSSB* contains numerous tests that attempt to determine if some months behave differently from other months.

TSSB provides the user with a unique and valuable feature for any of the tests. If the stationarity test is invoked through the menu system (GUI), a Monte-Carlo Permutation Test (MCPT) can be performed on the test statistic. These MCPTs provide two valuable pieces of information. First, they provide an alternative estimate of the p-value of the test. The ***Single pval*** is the estimated probability that the test statistic would be as large as or larger than the obtained value if the distribution of the values of the variable was independent of time (stationary). The Monte-Carlo ***Single pval*** statistic is, in many cases, more accurate than p-values computed with conventional methods.

The Monte-Carlo Permutation Test provides a second statistic that can be enormously helpful in interpreting results. In nearly all applications, the user will be simultaneously testing a (possibly large) set of candidate indicator and target variables. Those having a small p-value will be singled out for more detailed study. However, even if all of the variables happen to be nicely stationary, the luck of the draw will practically guarantee that one or more variables will have an unjustifiably small p-value, especially if a large number of variables are present. The *Grand pval* is the probability that, if *all* of the variables happen to be truly stationary, the obtained value of the test statistic that is the greatest among the variables would equal or exceed the obtained value. This allows us to account for the selection bias inherent in focusing our attention on only those variables that are most suspicious. If we see that the *p-value* or *Single pval* is suspiciously small, but the *Grand pval* is relatively large, our degree of suspicion would be lessened. Of course, failure to reject a null hypothesis does not mean that we can safely accept it. Hence, relatively large values of the *Grand pval* do not mean that we can conclude that the variable is stationary. On the other hand, tiny values of the *Grand pval* should raise a big red flag.

Understand that although tiny values of the various p-values indicate nonstationarity, this does not automatically imply that the degree of nonstationarity is serious enough to degrade performance of the trading system or signal filter. If the dataset contains a large number of cases, even trivially small amounts of nonstationarity may result in small p-values. Many of the tests provide an additional statistic that indicates, at least roughly, the practical extent of any nonstationarity. These other statistics can be quite heuristic, and should be interpreted with great caution. Still, they are useful.

This section has provided an overview of the stationarity tests provided by *TSSB*. Because the program contains a large number of tests, some of which are quite complex and sophisticated, the subject merits an entire chapter of its own. This chapter will be provided in a later edition of the tutorial, and a summary is currently in the manual.

A Final Example

This chapter has discussed the commands that are employed to read market price history files and create a database of variables that may be used as indicators and targets for a subsequent study. We end this discussion with an 'all the bells and whistles' example that combines the bare essentials with as many options as possible. The following commands read a set of market files, compute a database of variables, and perform several optional statistical tests.

```
READ MARKET LIST "SYMBOLS.TXT" ;
READ MARKET HISTORIES "E:\SP100\IBM.TXT" ;
MARKET SCAN ;
RETAIN YEARS 2001 THROUGH 2006 ;
CLEAN RAW DATA 0.65 ;
READ VARIABLE LIST "TREND_VOLATILITY.TXT" ;
OUTLIER SCAN ;
DESCRIBE P_TREND ;
CROSS MARKET AD ;
CROSS MARKET KL ;
CROSS MARKET IQ ;
STATIONARITY OF P_TREND IN IBM ;
```

The market scan produced the following results (the first few lines from the audit log):

```
-------------------------------------------------------
Market scan for large percentage price changes
-------------------------------------------------------

Market HAL largest percent differences...
    Date    Time     Difference
  20011207 000000       82.500
  20020724 000000       31.648
  19871019 000000       25.955
  19871020 000000       22.930
```

The market HAL is listed first in the log because its maximum percent price jump, 82.5 percent, was the largest among the markets. This jump occurred on December 7, 2001. The other markets also appear, in decreasing order of worst jump size. They are not listed here.

The outlier scan produced the following output in the log file, shown here in its entirety:

```
------------------------------
Outlier / Entropy scan
------------------------------

    Variable    Min  Mkt   Max  Mkt IQ Rng Ratio Entropy
     P_TREND -49.989 IBM 49.336 JNJ 23.508   4.2  0.888
P_VOLATILITY -49.507 HNZ 49.726 JPM 49.364   2.0  0.992
  DAY_RETURN  -5.226 JNJ  3.974  AA  0.775  11.9  0.593


Sorted by ratio, worst to best

       Variable     Ratio  Entropy
     DAY_RETURN     11.9    0.593
        P_TREND      4.2    0.888
   P_VOLATILITY      2.0    0.992


Sorted by entropy, worst to best

       Variable     Ratio  Entropy
     DAY_RETURN     11.9    0.593
        P_TREND      4.2    0.888
   P_VOLATILITY      2.0    0.992
```

We see that the variable P_TREND had a minimum value of -49.989, which occurred in the market IBM. Its maximum of 49.336 happened in JNJ. Its interquartile range was 23.508, the ratio of its range to its interquartile range was 4.2, and its relative entropy was 0.888. When the two measures of distribution quality are sorted from worst to best, we see that the three variables end up in the same order, a fairly common occurrence.

The DESCRIBE P_TREND command produced the following self-explanatory output:

```
Descriptive statistics for P_TREND in Pooled data
  Mean = 0.460128  (std err = 0.129125  t = 3.563)
  N = 17834  Variance = 297.336151
  Standard deviation = 17.243438
  Skewness = -0.085221  (std err = 0.029002)
  Kurtosis = -0.191218  (std err = 0.073369)
  Median = 0.558860
  Interquartile range = 23.5057  (-11.0866 to 12.4191)
  Range = 99.3247 (-49.9888 to 49.3359) Range/IQ = 4.226
  20-bin relative entropy = 0.888071
```

The CROSS MARKET AD test produced extensive output. Here are the first two sections:

```
Cross market Anderson-Darling test for variable P_TREND

     AA      2.415   (p=0.05489)
   DELL      1.555   (p=0.16390)
    DOW      0.913   (p=0.40617)
     GE      2.858   (p=0.03233)
    HAL      4.462   (p=0.00520)
    HNZ      2.124   (p=0.07860)
    IBM      1.303   (p=0.23153)
    JNJ      0.555   (p=0.69166)
    JPM      1.332   (p=0.22217)
    WMT      3.454   (p=0.01619)
    XOM      4.298   (p=0.00625)
    XRX      1.425   (p=0.19535)

Cross market Anderson-Darling test for variable P_TREND,
sorted worst to best

    HAL    0.00520
    XOM    0.00625
    WMT    0.01619
     GE    0.03233
     AA    0.05489
    HNZ    0.07860
   DELL    0.16390
    XRX    0.19535
    JPM    0.22217
    IBM    0.23153
    DOW    0.40617
    JNJ    0.69166
```

The first table lists the markets in the order in which they appeared in the market list file. It shows the raw Anderson-Darling statistic, which has no simple intuitive meaning, along with the associated p-value. Because small p-values indicate a low probability of obtaining a score this extreme if the market has good conformity, small p-values are bad. Thus, it makes sense to focus on the p-values and sort based on them. The second table does this. We see that HAL is a serious oddball market when it comes to conformity of P_TREND among this set of markets, while JNJ is a solid conformist.

These pairs of tables are repeated for the other two variables. Finally, these two summary tables appear:

```
Median Anderson-Darling across markets, worst to best...

          P_TREND  0.12125
       DAY_RETURN  0.19107
     P_VOLATILITY  0.25621

Median Anderson-Darling across variables, worst to best...

     XOM   0.00065
     HAL   0.00520
      GE   0.04263
     WMT   0.07537
      AA   0.12825
    DELL   0.16390
     JPM   0.22217
     IBM   0.23153
     XRX   0.26103
     HNZ   0.27253
     DOW   0.56027
     JNJ   0.84289
```

The first table shows that when one considers the median A-D p-value across markets, P_TREND has the worst cross-market conformity, while P_VOLATILIT has the best. If we instead consider the median across variables and compare markets, XOM is the worst non-conformer, with a median p-value of 0.00065. A result this significant indicates that we might be wise to take a closer look at XOM to see why it behaves so differently from other markets, at least as far as these variables are concerned.

The CROSS MARKET KL and CROSS MARKET IQ tests produced tables simi to those from the A-D test. However, these two statistics do not have p-values, so the sorting is based on the test statistics themselves. It is worth looking at the IQ test summary tables:

```
Median IQ range overlap across markets, worst to best...

          P_TREND  0.96019
        DAY_RETURN  0.96218
      P_VOLATILITY  0.98540

Median IQ range overlap across variables, worst to best...

      XOM   0.91561
      HAL   0.92772
      JPM   0.92911
      IBM   0.93044
       GE   0.94769
     DELL   0.95565
       AA   0.95676
      JNJ   0.97401
      HNZ   0.98087
      WMT   0.98220
      DOW   0.98352
      XRX   0.98485
```

These results are quite remarkable in how much cross-market conformity is obtained. When we look at the median IQ overlap across markets, once agair P_TREND is the worst variable, but it has more than 96 percent overlap o interquartile ranges! If we look at the median across variables, once again XOM is the worst, but it still manages to have a median overlap of almost 92 percent.

So... the A-D test gave a worst p-value of 0.00065 for XOM, which invited a warning to check on this market. Yet we see that its median interquartile range overlap with the pooled distribution was almost 92 percent, which is respectable. One or both of two things are responsible for this conflict:

1) There is some true non-conformity, but the large number of cases in the dataset (17,834) exaggerated the statistical significance of the difference, which in practice may be negligible.

2) The nonconformity lies mostly in one or both tails of the distribution. The A-D test is extremely sensitive to tail behavior, while the IQ test practically ignores tails.

Finally, the stationarity test produced voluminous output in the log file. However, this output is so complex that it is best deferred to the chapter on stationarity tests, to appear in a later edition of the tutorial.

Reading and Writing Databases

The prior chapter discussed how to read one or more market history files and create a database of variables that may be used as indicators and targets. In most cases, computing variables requires a significant amount of computer time; repeating this computation for every study would be an exercise in frustration. For this reason, *TSSB* has the ability to write a database to the hard drive, and later read it back in a very fast and efficient operation.

Besides speed, another reason for reading a database file is to bring into *TSSB* data that has been computed by other programs. It may be that the user needs indicators or targets that are not available in *TSSB*'s built-in library. We also saw in in Chapter 3, which dealt with developing signal filters for existing trading systems, that trade information for the system being filtered is provided to *TSSB* by means of a database. Finally, the user may sometimes want to use *TSSB*'s built-in library to generate variables, but then perform studies of these variables with another program. For all of these reasons, it is handy to be able to read and write database files.

Because we will often be interfacing with other programs, either to read database files produced by them, or to write database files that will be read by them, *TSSB* does not use a proprietary database format. Rather, it uses a subset of internationally recognized database standards. The exact nature of this format will be described here. For now, understand that database files created by *TSSB* can be read by all popular spreadsheet and statistical analysis programs. Also, most if not all such programs are capable of writing database files that can be read by *TSSB*.

Quick Reference to Database Commands

This section contains a list of all of the commands related to reading and writing databases, along with a brief description of each. Further details are presented in other sections.

RETAIN YEARS - Specifies that only a range of years be kept from the database file. This command was discussed in detail here. If used, this option must precede any command that reads a database.

RETAIN MOD - Specifies that every n'th date be kept from the database file. This command, which is useful for temporarily shrinking the database during initial development, was discussed in detail here. If used, this option must precede any command that reads a database.

RETAIN MARKET LIST - Specifies that only records from certain markets be read from the database. If used, this option must precede any command that reads a database. See here.

VARIABLE IS TEXT - Specifies that a particular variable is text as opposed to numeric. If used, this option must precede any command that reads a database. See here.

WRITE DATABASE - Writes the database to a disk file. See here.

READ DATABASE - Reads a database that is in chronological order. See here.

READ UNORDERED DATABASE - Reads a database that need not be in chronological order. See here.

APPEND DATABASE - Reads a database (which must be in chronological order) and appends its variables to an existing database. If used, this command must not appear until a database already exists, either from internal generation (READ VARIABLE LIST) or reading from a dis file (READ DATABASE). See here.

IS PROFIT - Specifies that a variable is a measure of profit. If used, this command must not appear until the variable is present in an existing database, so in general it would appear *after* a READ VARIABLE LIST or REA DATABASE command. See here.

Detailed Descriptions

This section provides more detailed descriptions of the commands associated with reading and writing a database. Several commands that were discussed in the prior chapter (RETAIN YEARS, RETAIN MOD) will not be repeated in detail here.

RETAIN MARKET LIST

This command is the database analogue of the READ MARKET LIST commar that named the markets for which variables are to be computed. The syntax is similar:

```
RETAIN MARKET LIST  "FileName"  ;
```

FileName specifies the name of the market list file, enclosed in double quotes ("). The file name may not include blanks or special characters other than the underscore (_). If a full path name is not specified, *TSSB* will assume that the file is in the current working directory as defined by the Windows operating system. See here for an example of a market list file.

This command, which if used must appear before a database is read, causes only a subset of markets to be read from the database file. The primary use for this command is to allow the user to keep one master database of all markets of interest, but select only a subset of the master database for a particular study. For example, we may use the following two commands to read a huge database but keep only data from certain markets:

```
RETAIN MARKET LIST  "EastCoastUtilities"  ;
READ DATABASE  "AllSP500Equities"  ;
```

VARIABLE IS TEXT

By default, all variables in a database file except the market name are numeric. However, *TSSB* can occasionally make use of category variables that are identified by names. For example, we may have a season variable with values ‘planting’, ‘growing’, and ‘harvest’. More often, the user will be presented with a database produced by another program, and this database will contain one or more variables that are text. We will probably want to ignore these variables in the study, but they are nonetheless present in the file, and hence must be accommodated.

The following syntax is used to declare a variable as text:

```
VARIABLE  VarName  IS TEXT  ;
```

This command, if used, must precede any command that reads the database. The values of this text variable must not contain a space or any special character except the underscore (_). Text variables are almost never used in *TSSB*, so we will defer further explanation until a later edition of the tutorial.

WRITE DATABASE

Any time a database is present, it can be written to a disk file. A database will be present as a result of internal variable computation (READ VARIABLE LIST) or reading a database. The database file is an ordinary ASCII text file which is readable not only by the *TSSB* program, but by most spreadsheet and statistical programs.

There are two commands for writing a database. The first command shown below writes only the database file, and the second also writes a *family file*. A family file contains supplementary information that may be useful to the *TSSB* program and will be briefly discussed later in this section.

```
WRITE DATABASE  "DatabaseName"  ;
WRITE DATABASE  "DatabaseName"  "FamilyFileName"  ;
```

The file names are enclosed in double quotes ("). The names may not include blanks or special characters other than the underscore (_). If a full path name is not specified, *TSSB* will write the file into the current working directory as defined by the Windows operating system.

In order to conform to recognized database standards, the first record written in the file will list the fields in the order that they appear on subsequent lines. The first item is the date as YYYYMMDD. If the application is intraday, the next item is the time as HHMM or HHMMSS. The next item is the name of the market. Variable follow. Blanks are used as delimiters when the file is written by *TSSB*, although tabs and commas are also legal. Here are the first few records in a typical database file:

```
Date Market LIN_ATR_5 LIN_ATR_15 RETURN
19621228     IBM    4.89092588  -1.36788785  -0.31357768
19621231     IBM   -0.92078519   1.70236468  -0.47036651
19621231   SP500    5.16105413   4.43815613  -0.38055974
19630102     IBM  -10.57968712   1.92275798   1.25470793
19630102   SP500    0.11760279   5.16873121   0.95639825
19630103     IBM   -5.70287132   1.73097610   0.00000000
```

The first line says that on subsequent lines, the first field will be the date, then the

market, then three variables. The database file is always written in chronological order.

The user may optionally specify that a *family file* is to be written to accompany the database file. This file contains information about any variables that were computed from *TSSB*'s internal library. This information concerns things such as the name of the family (discussed in the *Variables* chapter), how far the variable looks back or ahead in history, whether it involves an index, and whether it is an indicator (looks back in history) or a target (looks forward in history). Here are the first few lines of a typical family file:

```
LIN_ATR_5 "LINEAR PER ATR" 5 0 0 0 0 0 0 0 0 0
LIN_ATR_15 "LINEAR PER ATR" 15 0 0 0 0 0 0 0 0 0
RETURN "NEXT DAY ATR RETURN" -1 0 0 0 0 0 0 0 0 1
```

The first line says that the internally computed variable "LIN_ATR_5" is a membe of the "LINEAR PER ATR" family and looks back five days in history. The othe items are very advanced, and the user need not be concerned with them. here we will see how it can be useful to write a family file and then read the family file later, when the database is read. However, there is no reason why the average user would ever need to examine a family file. This file is intended strictly for internal use by the *TSSB* program, so we will not pursue the meaning of individual components of this file.

READ DATABASE

Computing variables from *TSSB*'s internal library can require a large amount of computer time. For this reason, it is recommended that whenever possible, all potential indicators and targets be computed just once and saved as a database file using the WRITE DATABASE command just presented. This database can then b rapidly read back using the READ DATABASE command.

This command can also be used to read a file produced by another program. The database file is an ordinary ASCII text file. The first line must specify the names o all fields. The first field must be 'Date' and, if the data is intraday, the next field must be 'Time'. The next field must be 'Market', with market names limited to a maximum of five characters. Subsequent fields must be variable names. These names must not contain spaces or special characters other than the underscore (_).

The maximum length of a variable name is 15 characters. See here for a sample database file. Note that spaces, commas, or tabs may be used as delimiters.

The records in a database read with the READ DATABASE command must be i

chronological order. If the records are not in order, the READ UNORDERE DATABASE command (described in the next section) must be used.

Just as there are two commands for writing a database, there are also two commands for reading it back. The first command shown below reads only the database file, and the second also reads a *family file*. A family file contains supplementary information that may be useful to the *TSSB* program. Refer back to here for a discussion of the family file. See here for an example application in which the family file plays a vital role.

```
READ DATABASE  "DatabaseName"  ;
READ DATABASE  "DatabaseName"  "FamilyFileName"  ;
```

The file names are enclosed in double quotes ("). The names may not include blanks or special characters other than the underscore (_). If a full path name is not specified, *TSSB* will read the file from the current working directory as defined by the Windows operating system.

READ UNORDERED DATABASE

The READ DATABASE command described in the prior section assumes that th records in the file are in chronological order. This allows reading to operate with great efficiency. If the database file is not chronological, then it must be read with the READ UNORDERED DATABASE command. Note that this command requir much more computer time, disk space, and memory than the READ DATABASI command, especially if the file is large. Therefore, it should be used only if necessary.

Like the WRITE DATABASE and READ DATABASE commands, this has tv forms, depending on if a family file is to be read in addition to the database file. The syntax of these two forms is as follows. See here for a discussion of family files.

```
READ UNORDERED DATABASE "DatabaseName" ;
READ UNORDERED DATABASE "DatabaseName" "FamilyFileName" ;
```

APPEND DATABASE

The user may wish to compute some variables using *TSSB*'s internal library, and then read others from an external database file. Or there may be a need to merge several external database files. These operations can be accomplished with the

APPEND DATABASE command.

The syntax of this command is as follows:

```
APPEND DATABASE  "FileName"  ;
```

The file name is enclosed in double quotes ("). The name may not include blanks or special characters other than the underscore (_). If a full path name is not specified, *TSSB* will read the file from the current working directory as defined by the Windows operating system.

The appended file must be in chronological order. At this time there is no UNORDERED option for appending files. However, there is a simple trick fo working around this limitation if the file to be appended is not ordered: Just write a two line script that reads the unordered file and then writes it back out again. Writing a database always outputs the file in chronological order. Such a script might look like this:

```
READ UNORDERED DATABASE  "UnorderedFile"  ;
WRITE DATABASE  "OrderedFile"  ;
```

Then, subsequent scripts would append "OrderedFile" instead of the original unordered file.

Also note that at this time the APPEND DATABASE command is not able to read family file. This is almost never a problem, because appended databases are usually created by other programs. The ability to append a family file may be added to a future version of *TSSB*.

The APPEND DATABASE command operates by merging records in the appende database with records already present. A *record* is the set of all variables for a particular market at a single date (and time, if intraday). Thus, for each record in the appended database, *TSSB* searches its existing database for the same market and date/time and, if it finds a corresponding record, appends the new variable(s) to the existing variable(s) to create a new record.

We'll look at a simple example using just one variable in each database. Suppose the following database exists at the time the APPEND DATABASE comman appears:

```
Date  Market  X1
20010601  SP  17.2
20010601 IBM  22.5
20010602  SP  44.1
20010603  SP  37.8
20010603 IBM  51.2
20010604 IBM  46.6
```

Now suppose that the appended database looks like this:

```
Date  Market  X2
20010601  SP  42.7
20010601 IBM  66.7
20010602 IBM  10.4
20010603  SP  13.7
20010603 IBM  48.9
20010604  SP  62.4
20010604 IBM  68.4
```

Then the merged database, which contains only those records which have the same date and market in both the original and appended databases, looks like this:

```
Date  Market  X1    X2
20010601  SP  17.2  42.7
20010601 IBM  22.5  66.7
20010603  SP  37.8  13.7
20010603 IBM  51.2  48.9
20010604 IBM  46.6  68.4
```

Notice in this example that no record remains for the date 20010602 because on that date the existing database contained only a record for SP, while the appended database has only a record for IBM on that date. On 20010604, the appended database has records for both SP and IBM, but the existing database has a record for only IBM. Thus, no record for SP remains for this date after appending.

One implication of this merging algorithm is that if the existing and the appended databases have few markets in common, the resulting merged database could be very sparse. Similarly, if the two databases are highly disjoint in time (their time periods have little overlap), the resulting merged database would cover only the short interval in which their time periods overlap. Finally, if the appended database is for a single market that does not exist in the current database, the result will be an empty database, one with no records at all! The most common use for the APPEND DATABASE command is when the current and the appended database cover approximately the same time period and have most or all of their markets in common.

IS PROFIT

TSSB contains a safety feature to prevent the user from accidentally misinterpreting results. This safety feature suppresses printing of any performance measure that involves profit (such as profit factors) when the target variable is not a profit. Such targets do exist. For example, the trend of prices in the near future can be used as a target being predicted, but it is certainly not a profit. Profit factors based on slope would be meaningless. When targets are computed from the internal library,

TSSB knows whether the target is a profit. But when it's read from a database file the program has no way of knowing whether the target is a profit unless the user informs it that this is the case.

The following syntax is used to declare a variable as profit:

```
VarName IS PROFIT  ;
```

The IS PROFIT command will usually be needed when the target variable of a model, committee, or oracle is read from a database file, as opposed to being computed internally. Of course, if the target variable is not a measure of profit, you would not want to use this command. But most targets are measures of profit, so this command will then be needed so that profit-based performance measures will be printed.

A Saving/Restoring Example

here we saw an example of a simple standalone trading system. That example script computed the indicators and target, and then immediately tested a prediction model. We now show an approach that is usually more efficient: split the script into two parts. The first part computes and saves the indicators and target, and the second part reads the saved database and tests the prediction model. Here is the script that computes the indicators and targets and saves them as a database file:

```
READ MARKET LIST "SYMBOLS.TXT" ;
READ MARKET HISTORIES "E:\SP100\IBM.TXT" ;
READ VARIABLE LIST "TREND_VOLATILITY.TXT" ;
WRITE DATABASE "EXAMPLE_DBASE.DAT" "EXAMPLE_DBASE.FAM";
```

The author's personal convention is to use the .DAT extension for all database files. This makes it easy to look at a directory listing and know which file(s) are databases. Other users may wish to use .TXT because the database really is a text file. This is purely a personal preference; *TSSB* does not care what you use for an extension. Similarly, the author uses .FAM for family files. Again, the family file is

a text file, so .TXT would be fine also.

This script did not need to save the family file, because the second script, which tests a prediction model, does not make use of family information. Still, it's a good habit to save the family file whenever possible, as you never know when you might change the application in a way that needs family information. The family file is very small and fast to write to disk, so saving it costs nothing and may prove handy later.

Here is the second part of the script, which reads the previously saved database and tests the prediction model:

```
READ DATABASE "EXAMPLE_DBASE.DAT" "EXAMPLE_DBASE.FAM" ;
MODEL SIMPLE_MODEL IS LINREG [
   INPUT = [ P_TREND P_VOLATILITY ]
   OUTPUT = DAY_RETURN
   MAX STEPWISE = 0
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   ] ;

WALK FORWARD BY YEAR 10 1999 ;
```

Creating Variables

Several prior chapters provided an overview of the process of creating a database of variables (indicators and targets) that are based on the extensive library built into *TSSB*. In particular, the control script reads a list of markets with the READ MARKET LIST command, reads the market price histories with the READ MARKET HISTORIES command, and reads the list of variables to be compute with the READ VARIABLE LIST command. It is this final step that will be th focus of this chapter.

Overview and Basic Syntax

The variable definition list is an ASCII text file containing one variable definitio per line. The syntax is simple for basic variables, and it can become quite complex when more sophisticated options are invoked. Rather than trying to present the complete set of all syntax variations in one general statement, we will focus here on only the most basic form, and deal with extensions to the syntax in separate sections.

A variable definition in its simplest form consists of three items: a hopefully descriptive name chosen by the user, the family selected from the *TSSB* library, and any parameters that are needed to fully define that family member. (An indicator family refers to indicators that all have the same computational form but differ in their values of user-specified parameters.) For example, we may want to define an indicator that we choose to call MA_XOVER. For this variable, we select the MA DIFFERENCE family that is a member of the built-in library. This variable, whic will be discussed later, subtracts a long-term moving average from a short-term moving average and normalizes the result. It requires three parameters: the short-term history length, the long-term history length, and the number of bars to lag the long-term history. Members of this family differ only in these three parameters. The line in the variable definition list file to implement this variable could be as follows:

```
MA_XOVER: MA DIFFERENCE  5  20  5
```

The name chosen by the user is the name that will appear in all studies. It should be reasonably short, yet descriptive to the user. The maximum length is 15 characters. It may contain only letters, numbers (though the name cannot begin with a number), and the underscore character (_).

The user name is followed by a colon (with or without a space between) and then the definition of the family. Legal families will be discussed soon. Most but not all families require one or more parameters to immediately follow.

Blank lines, which can help separate groups of variables, are legal. Any text after a semicolon (;) is ignored and can be used for comments.

Index Markets and Derived Variables

Many index ‘markets’ are commercially available. These may be actual baskets of securities, or they may be computed from weighted averages of numerous equity prices and traded indirectly by means of options or futures. A useful property of

most indices is that because they are derived from numerous securities, they reflect an average behavior of the market. Because they in a sense summarize the state of an entire market or large segment of the universe of markets, they are often useful in the computation of indicators.

If the user wishes to employ one or more indices in the variable definition list, the index (indices) must be declared before the variable definition list file is read. If the user will employ only one index, the declaration is in the first form shown below. The second form is used to declare multiple individual indices, up to a maximum of 16.

```
INDEX  IS  MarketName  ;
INDEX  Number  IS  MarketName  ;
```

The *Number* of the index must be an integer from 1 through 16. There is no requirement that numbering start at one, although that would probably reduce confusion.

The following command declares that OEX (the S&P 100 index) will be referenced as an index in the variable definition list file:

```
INDEX IS OEX ;
```

TSSB allows the user to use indices in two ways when creating variables. The most common use is to cause a variable to be computed from the index instead of individual markets. In normal operation, the value of a variable for a particular market at a particular date/time is based on the price history of that market. Naturally. But if the variable definition is followed by the keyword IS INDEX, tha variable is computed from the price history of the index, regardless of the market.

A second use for indices is to compute the deviation of a variable for a market from the same variable for an index. For example, we may want to compute the trend of prices in a market, also compute the trend of an index, and define our variable of interest as their difference. If a market is trending the same as the index, the value of this variable would be near zero. If the market were trending upward more strongly than the index, the variable would be positive, and if the opposite were true the variable would be negative. This lets us measure how much a market is conforming to 'average' behavior. In order to do this, we place the keyword MINUS INDEX after the variable definition (but before any parameters).

An Example of IS INDEX and MINUS INDEX

Here is an example of both uses of an index. The LINEAR PER ATR family c

indicators fits a straight line to a specified length of history, computes ATR (Average True Range) over a specified length, and scales the slope of the trend line by dividing by ATR. The first of the following three variables (RAWVAR) computes this variable by fitting the trend line over 20 bars and fitting ATR over 250 bars. Because no index is referenced, the variable is computed based on the price history of each market in the market list. The second variable (INDEXVAR) bases the value on the index for all markets in the market list. The third variable (DIFFVAR) is the value for the market minus the value for the index.

```
RAWVAR:   LINEAR PER ATR 20 250
INDEXVAR: LINEAR PER ATR IS INDEX 20 250
DIFFVAR:  LINEAR PER ATR MINUS INDEX 20 250
```

Some records in the resulting database might look like the following:

Date	Market	RAWVAR	INDEXVAR	DIFFVAR
19970301	IBM	37.2	32.0	5.2
19970301	BOL	31.5	32.0	-0.5
19970302	IBM	24.6	20.1	4.5
19970302	BOL	18.5	20.1	-1.6

Observe that on any given date, the IS INDEX variable, INDEXVAR, has the same value for each market. The difference variable, DIFFVAR, is the value computed from the market price history minus that computed from the index history. So for the first record, 37.2-32.0=5.2. This tells us that on this date, IBM had a stronger upward trend than the index.

Note that this example took a simplified approach in order to provide the reader with exact numbers for the differences. If this example were run in *TSSB*, we would probably see that the difference variable DIFFVAR does not exactly equal the difference between the market and index variables. This is because nearly all variables computed in *TSSB* are subjected to a nonlinear squashing function as a final operation so as to fix them in a common range and suppress outliers. In the case of the MINUS INDEX version, the subtraction is done before the transformation so that the quantities being differenced are in conformity. The net result is that the differences may not always be exact, depending on where the various values lie.

Also note that the index market did not appear in this example database. There is a subtle reason for this. If a MINUS INDEX variable appears anywhere in the variable list file, then the index will be omitted from the database. This is an annoying consequence of the fact that values of the MINUS INDEX variable would be effectively undefined for the index. Mathematically, the value is zero, but if a variable were identically zero for all cases of a market, modeling results would be nonsensical and all sorts of problems would arise during system development. So

this quandary is resolved by simply omitting the index market when the MINUS INDEX option is used.

Multiple Indices

In the vast majority of applications, one index is sufficient. However, *TSSB* allows the user to declare up to 16 markets as indices and use them individually. We saw how to declare index markets here. Suppose we have declared two index markets as follows:

```
INDEX 1 IS OEX ;
INDEX 2 IS DJIA ;
```

We might then use them as follows:

```
OEXVAR:  LINEAR PER ATR IS INDEX1 20 250
DJIAVAR: LINEAR PER ATR IS INDEX2 20 250
```

Note in this example that the declaration of an index requires a space between the INDEX keyword and the number, while later variable references require that no space separate them. When an index is referenced, its number (1-16) is part of the name.

Also note that INDEX1 is synonymous with INDEX. Thus, the user could declare an index with the command INDEX IS OEX and later reference it as INDEX. Similarly, the user could declare an index with the command INDEX 1 IS OEX and later reference it as INDEX. However, for the sake of clarity, it is suggested that the user be consistent in the script and variable definition list in order to avoid confusion.

Historical Adjustment to Improve Stationarity

Indicators generally fall into one of two categories: those whose actual value at the moment is of primary importance in and of itself, and those whose primary importance is based on their current value relative to recent values. Many of the indicators built into *TSSB* have this relativity built into them. However, for those that do not, the program includes the ability for the user to adjust current values in several ways that normalize the value according to recent values. This capability will now be discussed.

Why would one want to adjust the current value of an indicator according to recent values? The basic reason is that such adjustment is an excellent way of forcing a great degree of stationarity on the indicator. In most cases, stationarity improves the accuracy of predictive models. (Recall that, roughly speaking, stationarity means that the statistical properties of an indicator do not change over time.) As long as the historical lookback period for the adjustment is made long relative to the frequency of trading signals, important information is almost never lost, and the improvement in stationarity can be enormous.

TSSB supports three types of historical adjustment. *Centering* subtracts the historical median from the indicator. *Scaling* divides the indicator by its historical interquartile range. Full *normalization* does both: first it centers the indicator by subtracting the median, and then it scales it by dividing by the interquartile range. This is roughly equivalent to traditional standardization, in which data is converted to Z scores by subtracting the mean and dividing by the standard deviation. However, by using the median and interquartile range instead of the mean and standard deviation, we avoid problems with extreme values. Each of these options will now be presented.

Centering

Centering can be useful for stabilizing slow-moving indicators across long periods of time. For example, suppose we have a large-scale trend indicator, and we devise a trend-following trading rule that opens long positions when this indicator is unusually positive. Also suppose we hope for trades that have a duration of a few days to perhaps a few weeks, and we would like to obtain such trades regularly. If a market spends long periods of time in an upward trend, and other long periods in a downward trend, this indicator will flag an enormous number of trades in the up periods, and then shut off in the flatter and down periods. By subtracting a historical median of this trend indicator, these long periods of alternating performance will be reduced. In a long period of upward trend, the

indicator will no longer describe the trend. Rather, it will tell us whether the current trend is up versus down relative to what it has been recently. This will more evenly distribute trades across time. Many applications find this useful.

Centering is invoked by following the family name and its parameters with a colon (:), the word CENTER, and the historical lookback period for computing the median which centers the data. Here is an example using the LINEAR PER ATR trend indicator that we saw in the prior section:

```
TREND_OSC:  LINEAR PER ATR 20 250 : CENTER 100
```

The definition above computes the LINEAR PER ATR indicator using 20 days to define the trend and 250 days for the Average True Range scaling. Then, it finds the median of the prior 100 values of this quantity and subtracts it from the current value. This provides a powerful 'return to zero' force that prevents the computed value from staying above or below zero for too long. In effect, it converts an 'absolute' measurement into a variable that oscillates about its mean with guaranteed regularity.

Scaling

Sometimes centering a variable based on its historical values destroys vital information. In such cases, the sign and magnitude of the variable at the moment is of paramount importance, and centering would destroy this information. Yet we may want to compensate for varying volatility. It may be that a value considered 'large' in a time period during which it has low variation would be 'small' in a high-variation period. Thus, we may want to divide the value of the variable by a measure of its recent variation. The interquartile range is an ideal measure of variation because it is not impacted by extremely large or small values the way a standard deviation would be.

Scaling is particularly useful for ensuring cross-market conformity of an indicator. If we want to pool indicator data from several different markets, it is vital that the statistical properties of the indicators be similar for all markets. Otherwise, marketswith large variation will dominate models, while those with small variation may be essentially ignored. By scaling the indicator according to its historical volatility, we produce a variable that is likely to have similar variability across markets. Scaling is invoked the same way as centering, except that the keyword SCALE is used instead of CENTER.

In addition to division by the interquartile range, the final value is transformed with a nonlinear function that compresses outliers and produces a variable lying in the

fixed range of -50 to 50. In the following equation, which defines the scaling option, Φ(•) is the standard normal CDF. Also, F_{25}, F_{50}, and F_{75} are the 25'th, 50'th, and 75'th percentiles, respectively, of the historical values of the indicator.

$$V = 100 * \Phi\left(0.25 * \frac{X}{F_{75} - F_{25}}\right) - 50 \quad \textbf{(1)}$$

Normalization

In situations in which centering is appropriate, it often makes sense to simultaneously scale the indicator. This is almost like converting each observation into a standard normal Z score based on recent history, except that here we use the median instead of the mean and the interquartile range instead of the standard deviation. Normalization is the ultimate in imposing stationarity on an indicator. This is invoked the same way as centering except that the keyword NORMALIZE i used instead of CENTER.

In addition to subtraction of the median and division by the interquartile range, the final value is transformed with a nonlinear function that compresses outliers and produces a variable lying in the fixed range of -50 to 50. In the following equation, which defines the normalization option, Φ(•) is the standard normal CDF. Also, F_{25}, F_{50}, and F_{75} are the 25'th, 50'th, and 75'th percentiles, respectively, of the historical values of the indicator.

$$V = 100 * \Phi\left(0.5 * \frac{X - F_{50}}{F_{75} - F_{25}}\right) - 50 \quad \textbf{(2)}$$

An Example of Centering, Scaling, and Normalization

An illustration of these historical adjustments may make them more clear. Figure 3 on the next page illustrates a 100-day trend variable LINEAR PER ATR, scale per its ATR at a lookback of 1000 days. (This scaling is part of the definition of LINEAR PER ATR, and it has nothing to do with the scaling option discussed i this section.) Notice how, when it moves far upward around the middle of the time period, it stays up there for a long time.

Figure 4 shows this trend variable centered with a lookback period of 200 days. Notice that it does not stay high for long after the rise in the middle time period. The centering pulls it back toward zero.

Figure 5 shows the trend variable with the scale option applied. It keeps the same sign, but otherwise its behavior is quite different. Beginning around 7/09, the strong positive trend takes a sharp dip because the steep rise in trend just prior to 7/09 is interpreted (rightly or wrongly) as a sudden jump in volatility.

Figure 6 shows the trend variable with the normalize option used. The influence of normalization (centering plus scaling) is prominent.

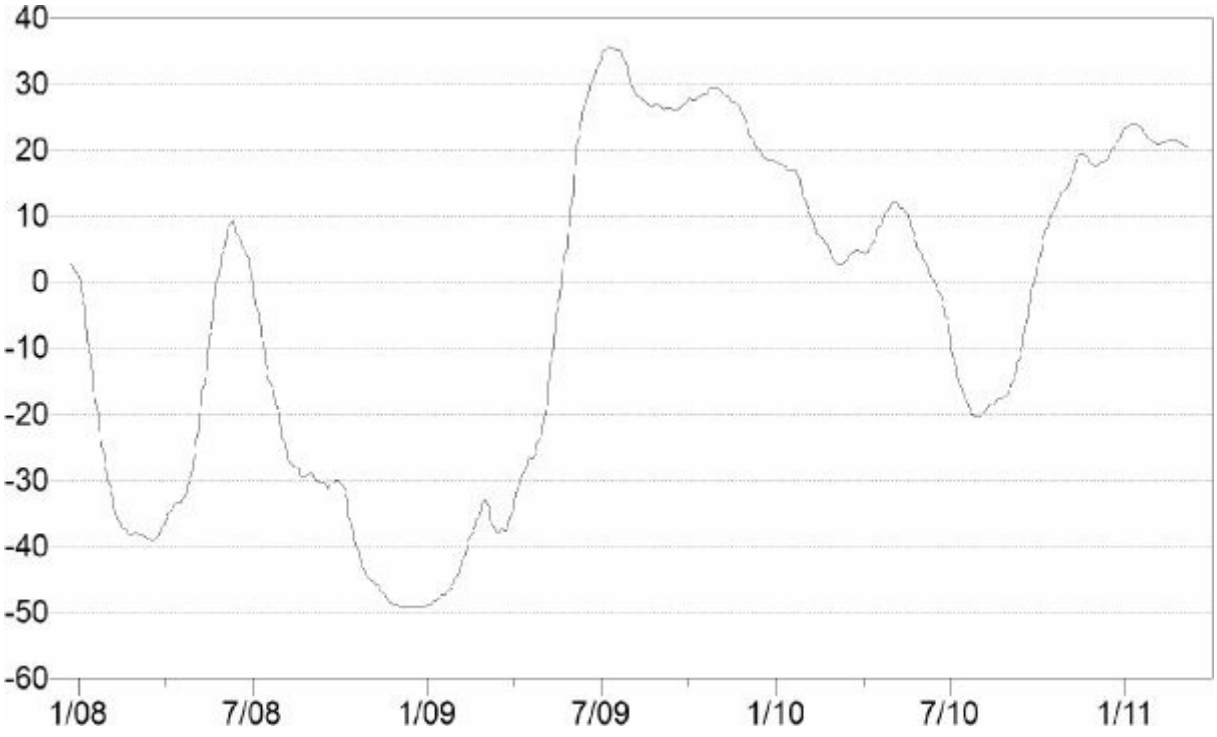


Figure 3: A trend indicator

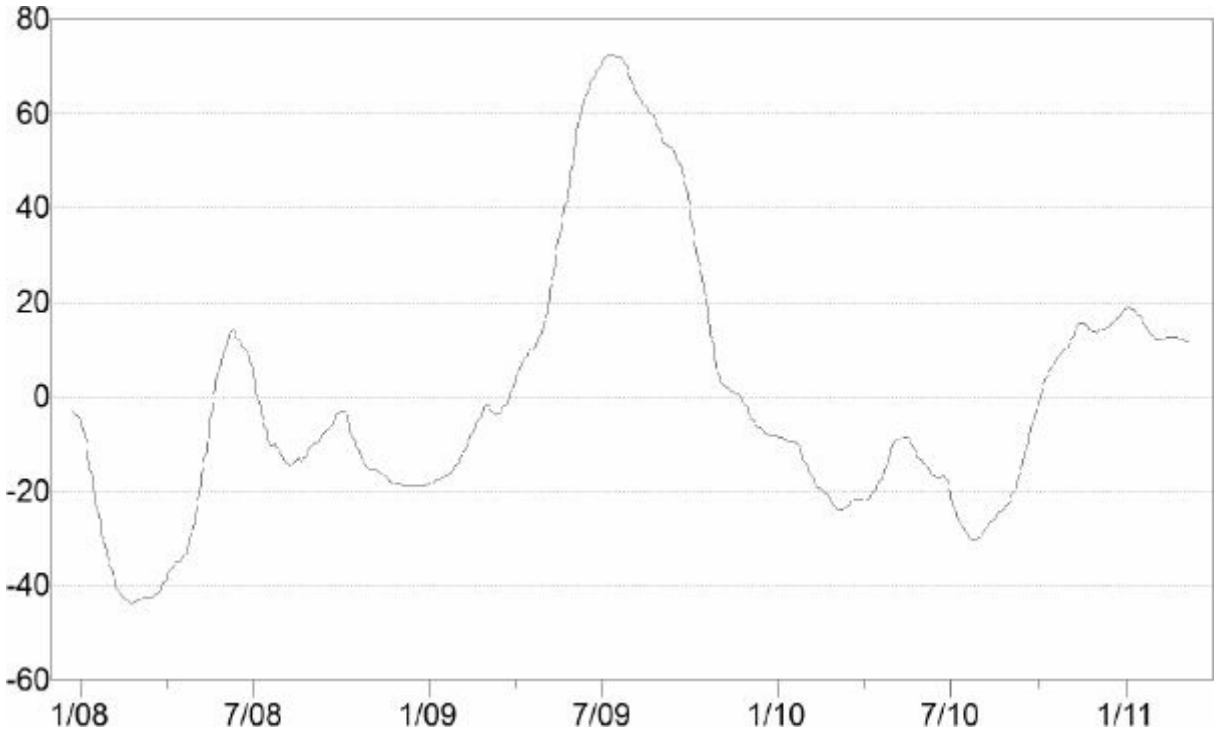


Figure 4: Centered trend

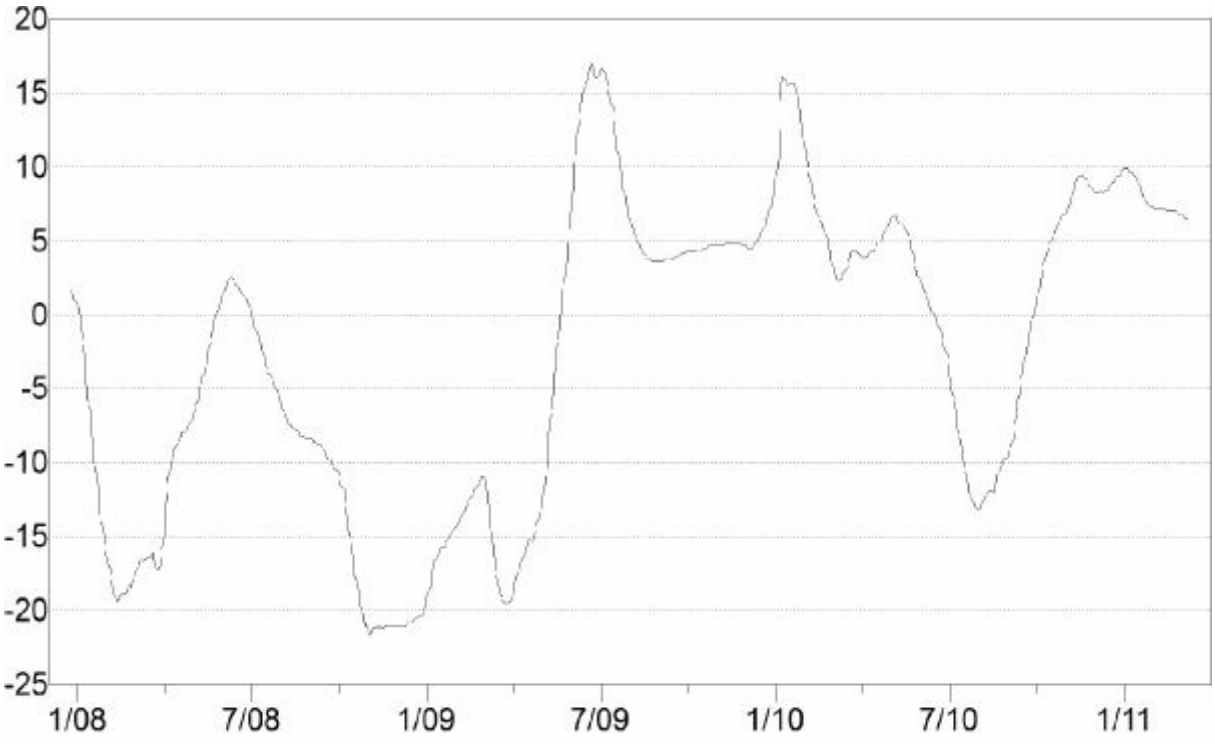


Figure 5: Scaled trend

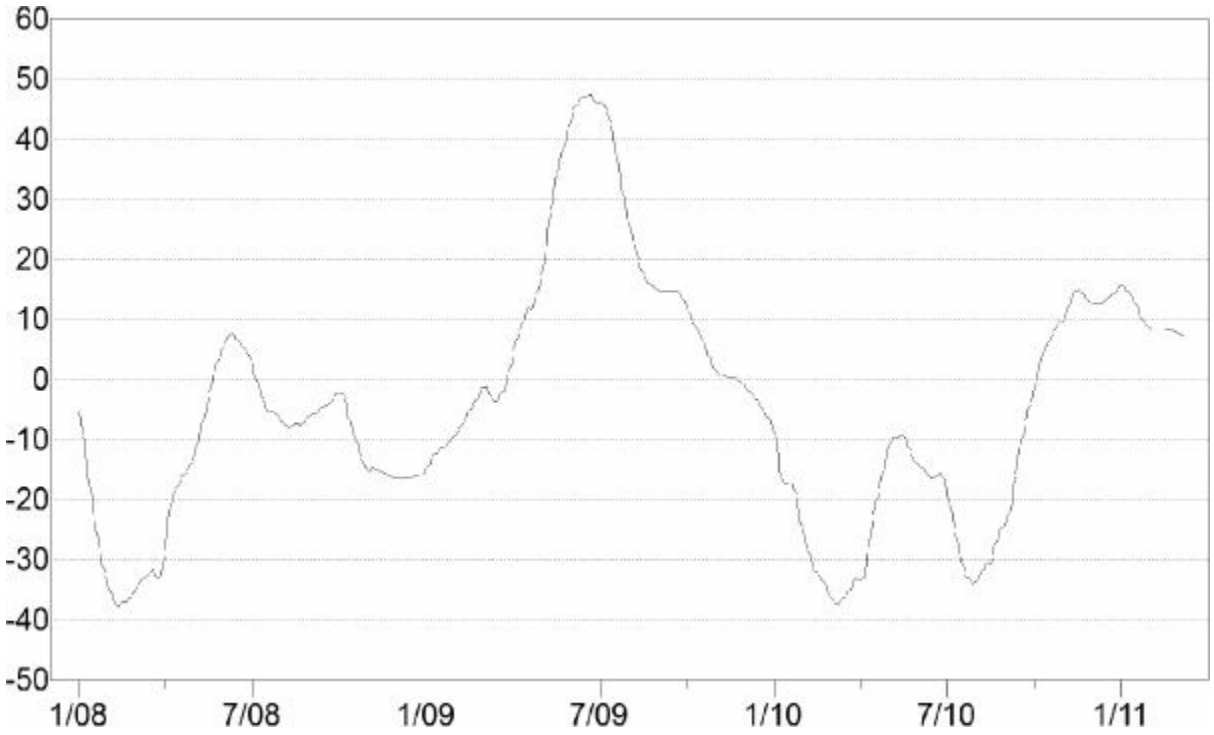


Figure 6: Normalized trend

Cross-Market Normalization

When you are trading many markets simultaneously, it can be useful to know how the various markets rank in terms of an indicator. For example, suppose we compute a trend variable for each market, and we are now considering whether to open a position in, say, IBM. If the trend in IBM is the largest among all markets that may well mean something important. Similarly, if we know that IBM has the minimum trend of all of the markets, that may mean something entirely different. Cross-market normalization ranks markets according to an indicator's value in each. This can be particularly useful in a long/short market-neutral trading system.

The indicator being cross-market normalized (which may or may not also have centering, scaling, or combined normalization as described in the prior section) is computed for every market for which data is available. Each day (or bar), the values are ranked across all of the markets, and the percentile rank for each market is computed. The final value of this variable is defined by subtracting 50 from the percentile. Thus, a cross-market-normalized variable ranges from -50 to 50. The market having minimum value of the variable is assigned a value of -50. The maximum market gets 50. Markets near the middle of the range will obtain values near zero.

Naturally, in order for this rank variable to be defined, we must have at least two markets with valid data for this indicator. Usually we would want many more than that! For this reason, the user specifies a fraction 0-1 which is the minimum fraction of the markets that must be present in order for this value to be computed. Some markets may begin their history later than other markets, or end earlier. Even in the interior, some market may have missing data. For this reason, we cannot count on every market having valid data on every bar. If too few markets are present on a given date/time, cross-market normalization would not have much meaning. Thus, if the minimum fraction is not met, the variable is not computed and is recorded as missing.

In order to invoke cross-market normalization, follow the variable definition with an exclamation point (!) and the minimum fraction of markets required. For example, the first line below computes the cross-normalized value of the LINEAR PER ATR indicator we've seen before, and the second line centers the variable before cross-market normalization. In both cases, we require that at least 60 percent (0.6) of the markets be present on a date in order to produce a valid value.

```
P_TREND:    LINEAR PER ATR 20 250 ! 0.6
P_TREND_C:  LINEAR PER ATR 20 250 : CENTER 100 ! 0.6
```

Pooled Variables

Another technique for extracting information about the behavior of an indicator across multiple markets is to use a *Pooled Variable*. This is done by appending the keyword POOLED followed by the type of pooling to be done, followed by the minimum fraction (0-1) of markets that must be present, as in computing cross-sectional normalization. These specifications must come last on a line, after the variable definition and any normalization.

All pooled variables are scaled and slightly transformed so that in most situations their natural range will be approximately -50 to 50. Pooling and cross-market normalization may not be performed simultaneously. The following pooling types are available:

MEDIAN - The median of the indicator across all markets. This provides a general 'consensus' value for the indicator in the complete set of all markets.

IQRANGE - The interquartile range of the indicator across all markets. This reveals the degree of 'spread' of values of this indicator across the universe of markets. A large IQRANGE means that the indicator has a large diversity of values within the set of markets. A small value means that the markets all tend to have about the same value for the indicator.

SCALED MEDIAN - The median divided by the interquartile range. Like the MEDIAN, this provides a consensus view of the indicator across markets. However, this consensus is scaled by the variation of the indicator across markets. Thus, if the indicator has a wide spread across markets, the actual value of the median will be scaled back. Conversely, if the indicator has similar values across the markets, the median will be accentuated in computing this variable. This often makes sense, because if a median is far from zero but the individual values for markets are all over the place, the median is less noteworthy than if all markets have about the same value for this indicator.

SKEWNESS - A nonparametric measure of skewness. If the indicator has a perfectly symmetric distribution, this value will be zero. Positive SKEWNESS means that there are a preponderance of relatively large positive values of the indicator, while negative SKEWNESS means that large negative values prevail. So, for example, suppose that on some day the SKEWNESS of a trend indicator is very large. This means that most markets had similar trends, but there were a few markets that had unusually large positive trends, and there were few or

no markets with an unusually small trend.

KURTOSIS - A nonparametric measure of kurtosis. If the indicator has a roughly normal bell-shaped curve, the value of this will be near zero. A large KURTOSIS means that one or both tails of the indicator are unusually heavy (numerous outliers). A small KURTOSIS means that the distribution of values of this indicator across markets is compact, with few or no 'oddball' markets that stand out from the crowd. KURTOSIS says nothing about the width of the distribution. It measures only the degree to which extreme values outside the majority range of the indicator are present.

CLUMP60 - If the 40'th percentile is positive (meaning that at least 60 percent of the markets have a positive value of this indicator) CLUMP60 is the 40'th percentile. If the 60'th percentile is negative, CLUMP60 is the 60'th percentile. Otherwise it is zero. This variable measures the degree to which the markets are moving in conformity. This variable is discussed in detail later in this section.

MEDIAN pooling

Here is a quick example of using the MEDIAN pooling option. With the exception of the unusual CLUMP60 pooling which will be covered separately, the other options should be clear once MEDIAN pooling is understood. Once again we will use the LINEAR PER ATR trend variable that appeared in prior examples. Suppose the following two variable definitions appear in a situation involving three markets:

```
P_TREND:    LINEAR PER ATR 20 250
P_TREND_M:  LINEAR PER ATR 20 250 POOLED MEDIAN 0.6
```

We may see that the generated database includes records that look like this:

Date	Market	P_TREND	P_TREND_M
20010601	IBM	5.2	7.3
20010601	BOL	7.3	7.3
20010601	IFF	9.1	7.3
20010602	IBM	6.2	6.2
20010602	BOL	4.4	6.2
20010602	IFF	6.5	6.2
20010603	IBM	5.5	6.8
20010603	BOL	7.2	6.8
20010603	IFF	6.8	6.8

Notice that for each date, the value of P_TREND_M is the same in all markets. It i the median of P_TREND in the three markets.

CLUMP60 Pooling

MEDIAN pooling provides a general consensus of what an indicator is doing in the universe of markets being studied. The median is the 'center' value of the indicator across these markets. Thus, if the median is positive, we know that at least half of the markets have a positive value of the indicator, and probably more than half of them do, because there likely are some markets whose value of the indicator is less than the median but greater than zero. Similarly, if the median is negative, we know that at least half of the markets have a negative value of the indicator, and probably more than half of them do.

The problem with using the median as a consensus figure for the universe is the word 'probably' in the prior paragraph. Even if we obtain a large positive median, we have no guarantee that more than half of the markets have a positive value of the indicator, and similarly for a negative median. The markets may well be split 50-50 between positive and negative. If our goal in pooling is to detect an informative consensus among the markets, MEDIAN pooling may not be adequate. A 50-50 split is hardly a consensus!

An obvious solution to this problem is to create a 'dead zone' in the area around a 50-50 split, so that if there is little or no consensus, the value of the pooled variable will be zero. This way, if we obtain a nonzero value for the pooled variable, we know that the split is outside the dead zone, well away from a tie between positive and negative values of the indicator. There is a significant degree of consensus among the markets.

CLUMP60 pooling accomplishes this. The number 60 in the name means that the dead zone is sized so that the pooled variable will have a nonzero value only if at least 60 percent of the markets have the same sign on the indicator for which CLUMP60 is being computed. This is easy to do:

If the 40'th percentile of the indicator is positive, then by definition at least 60 percent of the markets have a positive value of the indicator. So in this case we set the CLUMP60 pooled variable equal to the 40'th percentile of the indicator across markets. The positive sign signifies that a substantial majority of the markets have positive values for this indicator, and the magnitude of CLUMP60 gives an indication of the magnitude of the indicators.

Similarly, if the 60'th percentile of the indicator is negative, then by definition at

least 60 percent of the markets have a negative value of the indicator. So in this case we set the CLUMP60 pooled variable equal to the 60'th percentile of the indicator across markets. The negative sign signifies that a substantial majority of the markets have negative values for this indicator, and the magnitude of CLUMP60 gives an indication of the magnitude of the indicators.

Finally, if neither of the above conditions are true, then the distribution of indicator values across markets is in the dead zone, near a tie, with no clear consensus. In this case we set the CLUMP60 pooled variable to zero to flag the lack of consensus.

Mahalanobis Distance

A surprising world event or some other destabilizing influence can cause a sudden dramatic change in market behavior. This can be measured for an individual market by a spike in a traditional volatility indicator. But in a multiple-market scenario we can compute a more sophisticated turbulence measure by taking into account not only how much market prices change relative to their historical norm, but also by how their interrelationships change. For example, suppose two markets normally move together. If they suddenly diverge, this may indicate a major market upheaval of some sort.

A standard mathematical technique for quantifying change in terms of not only individual components but interrelationships as well is the *Mahalanobis Distance* that separates a vector (in this context, a set of market price changes as of a single bar) from its historical mean, with historical covariance taken into account. This can be accomplished for any variable by following the variable definition with the keyword MAHALANOBIS and the number of bars to look back when computin the mean and covariance.

In Kritzman and Li ("Skulls, Financial Turbulence, and Risk Management" i *Financial Analysts Journal*, September/October 2010, Vol. 66), Mahalanobis distance is based on daily price changes, which is the most intuitive choice. In this case you would use the CLOSE TO CLOSE variable. For example, suppose yc are working with daily data, and you want a mean/covariance lookback window of approximately one year. The variable definition line be this:

```
YEAR_MAHAL: CLOSE TO CLOSE  MAHALANOBIS  250
```

Don't feel limited to daily price changes, though. Other indicators, such as trend or volatility, may prove to be useful raw material for measuring turbulence. Here is a short example of how this variable might be generated. For this example, assume that these two market have been highly correlated over the window period.

Date	Market	Change	Mahal
19950601	C	23.1	0.21
19950601	JPM	21.4	0.21
19950602	C	10.3	0.32
19950602	JPM	9.2	0.32
19950603	C	-11.5	0.25
19950603	JPM	-12.7	0.25
19950604	C	-14.9	48.60
19950604	JPM	8.8	48.60

Observe that the Mahalanobis distance is small when the markets are moving

together, and it jumps when the markets move in opposite directions.

Absorption Ratio

Many experts believe that financial markets are most stable when the individual components are moving relatively independently. At such times investors tend to be making decisions independently, and market efficiency is likely to be high. However, when investors as a group become focused on the same market driving factors, independent decision-making declines and herding emerges. During such periods, when the components become locked together in consistent patterns, a situation called *coherence*, the market becomes unstable.

The degree of market coherence can be measured by examining the eigenvalues of the covariance matrix of the market components. If all components are completely independent, so that all correlations are zero, the eigenvalues are equal. As correlation among components increases, variance tends to shift so that it is concentrated in some eigenvalues and depleted in others. In the extreme, if all components move exactly together (all correlations are plus or minus one), all variance is contained in one eigenvalue; the other eigenvalues are zero. Intermediate degrees of coherence cause degrees of concentration of variance in few eigenvalues.

This leads to an intuitive and effective way to measure coherence: compute the fraction of the total variance that is contained in the few largest eigenvalues. We must specify in advance the number of largest eigenvalues that will be kept. The fraction of the total variance contained in these few kept eigenvalues will range from a minimum of *number kept*/*number of markets* when all components are independent, to 1.0 when all components move in perfect lockstep. ABSRATIO (absorption ratio) is a family of variables to accomplish this, and its close relative *Absorption Shift* may do so as well.

In order to compute the absorption ratio or absorption shift for a variable, follow the variable definition with the keyword ABSRATIO and then the following four parameters:

- The number of bars to include in the moving window used to compute the covariance matrix.

- The fraction (0-1) of the eigenvalues to use for computing the absorption ratio. Kritzman et al use 0.2.

- The short-term moving-average lookback for computing absorption shift (discussed soon), or zero to compute the absorption ratio.

- The long-term moving-average lookback for computing absorption shift (discussed soon), or zero to compute the absorption ratio.

If those last two parameters, the moving-average lookbacks, are zero, the value computed is the absorption ratio. If they are nonzero, the variable is what Kritzman et al call the *Absorption Shift*. This is the short-term moving average of the absorption ratio, minus the long-term, scaled by the standard deviation of the absorption ratio during the long-term time period. Note that it is illegal for one of the lookbacks to be zero but not the other.

Any predefined *TSSB* variable can be used to compute the absorption ratio, but in order to implement the algorithm of Kritzman et al, the CLOSE TO CLOSE variable should be used. Here are two examples. Both use a lookback windowof 250 bars and the fraction 0.2 of the eigenvalues. The first example computes the absorption ratio, and the second computes the shift using a short-term MA of 20 bars and a long-term MA of 100 bars.

```
YEAR_ABSRT: CLOSE TO CLOSE  ABSRATIO  250  0.2  0  0
YEAR_SHIFT: CLOSE TO CLOSE  ABSRATIO  250  0.2  20  100
```

Note that in order to compute the absorption ratio for a case, valid data for all markets must be present for every date in the lookback period. This is a severe restriction. There are two major ways in which the lookback period might have missing data:

1) Markets might exist for non-overlapping time periods. For example, suppose you are processing the components of an index such as the S&P 100. Many of its current components did not exist a few years ago. Thus, the first date on which all markets could have data will be the date on which the most recently 'born' market has its first price information. This could easily leave only a small subset of the data that would otherwise be available. For this reason it is best to check the starting and ending dates of all markets and process only markets that have substantial overlap. All dates outside the time period in which *all* markets have data will be excluded.

2) The CLEAN RAW DATA command (here) will eliminate data which has a suspiciously large bar-to-bar price jump. If even one market has missing data due to this command for any date in the lookback period behind the current date, the absorption ratio will not be computed for the current date. For this reason, the CLEAN RAW DATA command should use as small a parameter value as possible, perhaps 0.4 or even less.

Trend Indicators

This section discusses indicators that respond to trends in the market price. In all cases, the relationship is monotonic; large values correspond to upward trends and small values correspond to downward trends. Most but not all of these are shifted, scaled, and compressed in such a way that they have a natural range of about -50 to 50. Most of these indicators are self-explanatory. The more complex indicators are explained in detail in later supplementary sections.

MA DIFFERENCE ShortLength LongLength Lag

A short-term moving average and a long-term moving average, the latter lagged by the specified amount, are computed for closing prices. Typically, the lag will equal the short length, thus making the two windows disjoint, although many users will want to set the lag to zero so that the windows overlap. The long-term MA is subtracted from the short-term MA, and this difference is divided by the average true range measured across the *LongLength+Lag* history. Finally, this normalized difference is slightly compressed to a range of -50 to 50. For example:

```
MADIFF: MA DIFFERENCE 10 100 10
```

The preceding definition creates a moving average difference with a short-term lag of 10 bars and a long-term lag of 100 bars. The long-term moving average is lagged by 10 bars so that it does not overlap the short-term average. Thus, the total lookback period is 110 bars.

LINEAR PER ATR HistLength ATRlength

This indicator computes a least-squares straight line over the specified length of historical data. The data which is fit is the log of the mean of the open, high, low, and close. It also computes the Average True Range over the specified ATR history length. The returned value is the slope of the line divided by the ATR. This is sometimes called price velocity. A small amount of compression and rescaling is applied to restrain values to the interval -50 to 50. For example:

```
LINFIT: LINEAR PER ATR 50 250
```

The preceding definition fits a least-squares straight line to the most recent 50 bars and divides the slope by the ATR for the most recent 250 bars.

QUADRATIC PER ATR HistLength ATRlength

This indicator computes a least-squares fit of a second-order Legendre polynomial (an orthogonal family) over the specified length of historical data. The data which is fit is the log of the mean of the open, high, low, and close. It also computes the Average True Range over the specified ATRhistory length. The returned value is the (quadratic) polynomial coefficient divided by the ATR. This is sometimes called price acceleration. A small amount of compression and rescaling is applied to restrain values to the interval -50 to 50. The quadratic coefficient measures the *change in trend* of historical prices. This value will be positive if the trend is increasing and negative if the trend is decreasing. Note that this indicator is not affected by the trend itself, only the *change* in the trend.
For example:

```
QUADFIT: QUADRATIC PER ATR 50 250
```

The preceding definition fits a least-squares quadratic polynomial to the most recent 50 bars and divides the coefficient by the ATR for the most recent 250 bars.

CUBIC PER ATR HistLength ATRlength

This indicator computes a least-squares fit of a third-order Legendre polynomial (an orthogonal family) over the specified length of historical data. The data which is fit is the log of the mean of the open, high, low, and close. It also computes the Average True Range over the specified ATR history length. The returned value is the (cubic) polynomial coefficient divided by the ATR. A small amount of compression and rescaling is applied to restrain values to the interval -50 to 50. This indicator measures the rate of change in acceleration, the rate at which the quadratic term is changing. See QUADRATIC PER ATR. Note that this indicator i independent of trend (price velocity), as well as the quadratic term (acceleration, or rate of change in trend). For example:

```
CUBEFIT: CUBIC PER ATR 50 250
```

The preceding definition fits a least-squares cubic Legendre polynomial to the most recent 50 bars and divides the coefficient by the ATR for the most recent 250 bars.

RSI HistLength

This is the ordinary Relative Strength Indicator proposed by J. Welles Wilder, Jr. It is computed as described in *The Encyclopedia of Technical Market Indicators* by

Colby and Meyers, with one exception. That reference uses an ordinary moving average for smoothing, while most modern references claim that exponential smoothing give superior results, so that's what is done here. Actually, it's hard to tell the two apart, so in reality the difference is probably inconsequential inmost applications. For example:

```
RSI_WEEK: RSI 5
```

The preceding definition computes the RSI for the most recent five bars. No transformation of any sort is done; this produces the ordinary RSI which varies from zero through one hundred. Note that because exponential smoothing is done, the CLEAN RAW DATA command (here) is ignored. This is because exponential smoothing has a very long (theoretically infinite) lookback distance.

STOCHASTIC K HistLength
STOCHASTIC D HistLength

These are the first-order smoothed (K) and second-order smoothed (D) Lane Stochastics as proposed by George C. Lane. The algorithm used here is from*The Encyclopedia of Technical Market Indicators* by Colby and Meyers. For example:

```
STO_K_YEAR: STOCHASTIC K 250
```

The preceding definition computes the K stochastic looking back approximately one year (250 days, assuming we have day bars). No transformation is done; this produces the ordinary stochastic which varies from zero through one hundred. Note that because a form of exponential smoothing is done, the CLEAN RAW DATA command (here) is ignored. This is because exponential smoothing has a very long (theoretically infinite) lookback distance.

PRICE MOMENTUM HistLength StdDevLength

This measures the price today relative to the price *HistLength* days ago (or bars, for intraday data). It is normalized by the standard deviation of daily price changes. A small amount of compression and rescaling is applied to restrain values to the interval -50 to 50. For example:

```
PMOM: PRICE MOMENTUM 20 250
```

The preceding definition computes the price momentum over the most recent 20

bars and then normalizes it by the standard deviation of the most recent 250 bars.

ADX HistLength

The ADX trend indicator proposed by J. Wells Wilder is computed for the specified history length. Unlike most other indicators in the *TSSB* library, ADX is neither scaled nor transformed. It retains its defined range of 0-100. Remember that ADX does not indicate the direction of a trend, only its strength (degree of directionality or persistence). For example:

```
ADX_MONTH: ADX 21
```

The preceding definition computes the ADX trend strength indicator for the most recent 21 bars, which is approximately a month for day bars.

MIN ADX HistLength MinLength

The ADX trend indicator with a lookback of *HistLength* is computed for the current bar as well as prior bars, for a total of *MinLength* times. The minimum value across these times is found. For example:

```
MIN_ADX_M2M: MIN ADX 21 42
```

The preceding definition computes the 21-bar ADX trend strength indicator for the most recent bar, and the second-most recent, and the third-most recent, et cetera, a total of 42 times. The minimum of the current value and the 41 historic values is found.

RESIDUAL MIN ADX HistLength MinLength

The ADX trend indicator with a lookback of *HistLength* is computed for the current day (or bar for intraday data) as well as prior days, for a total of *MinLength* times. The minimum value across these times is found. This is the MIN ADX as described above. The final variable is today's ADX minus the minimum. For example:

```
RESMIN_ADX_M2M: RESIDUAL MIN ADX 21 42
```

The preceding definition first computes the MIN ADX indicator as described here.

It then computes ADX for the current bar and subtracts the former from the latter.

MAX ADX HistLength MaxLength

The ADX trend indicator with a lookback of *HistLength* is computed for the current bar as well as prior bars, for a total of *MaxLength* times. The maximum value across these times is found. For example:

```
MAX_ADX_M2M: MAX ADX 21 42
```

The preceding definition computes the 21-bar ADX trend strength indicator for the most recent bar, and the second-most recent, and the third-most recent, et cetera, a total of 42 times. The maximum of the current value and the 41 historic values is found.

RESIDUAL MAX ADX HistLength MaxLength

The ADX trend indicator with a lookback of *HistLength* is computed for the current day as well as prior days, for a total of *MaxLength* times. The maximum value across these times is found. This is the MAX ADX as described above. The final variable is the maximum minus today's ADX. For example:

```
RESMAX_ADX_M2M: RESIDUAL MAX ADX 21 42
```

The preceding definition first computes the MAX ADX indicator as described here. It then computes ADX for the current bar and subtracts the latter from the former.

DELTA ADX HistLength DeltaLength

The ADX trend indicator with a lookback of *HistLength* is computed for the current bar and for the bar *DeltaLength* ago. The final variable is the former minus the latter, slightly transformed and scaled to a range of -50 to 50. This measures the rate of change of ADX, its velocity. For example:

```
ADXVEL: DELTA ADX 21 10
```

The preceding definition computes the 21-bar ADX for today and for 10 days ago. It subtracts the lagged value from the current value and slightly transforms/compresses the result.

ACCEL ADX HistLength DeltaLength

The ADX trend indicator with a lookback of *HistLength* is computed for today and for the day *DeltaLength* ago and for the day 2**DeltaLength* ago. The final variable is today's value plus the doubly lagged value, minus twice the single-lagged value, slightly transformed and scaled to a range of -50 to 50. This measures the curvature or acceleration of the ADX function, the rate at which ADX velocity is changing For example:

```
ADXACC:  ACCEL ADX 21 8
```

The preceding definition computes the 21-bar ADX for today, for 8 days ago, and for 16 days ago. It doubles the lag-8 value and subtracts that from the sum of the current and lag-16 values. Finally, it slightly transforms/compresses the result.

INTRADAY INTENSITY HistLength

The smoothed intraday intensity statistic is returned. (The concept underlying this indicator was proposed by David Bostian. The version is TSSB is a modified version.) First, the true range is computed as the greatest of (today's high minus today's low), (today's high minus yesterday's close), and (yesterday's close minus today's low). (For intraday data, these quantities refer to bars, not days.) Then today's change is computed as today's close minus today's open. The intraday intensity is defined as the ratio of the latter to the former. This quantity is computed over the specified history length, and the moving average returned after being slightly transformed and rescaled to the range -50 to 50. For example:

```
INTENS_20: INTRADAY INTENSITY 20
```

The preceding definition computes the 20-bar-smoothed intraday intensity and returns the transformed/compressed result. Note that out of deference to tradition, this definition referred to 'day' bars, but it applies to any bar definition.

DELTA INTRADAY INTENSITY HistLength DeltaLength

This is the difference between today's *INTRADAY INTENSITY* and that of the specified number of days ago. For example:

```
D_INTENS_20_10: DELTA INTRADAY INTENSITY 20 10
```

The preceding definition computes the INTRADAY INTENSITY with a 20-ba

moving-average window, and also does the same for 20 bars ago. The latter is subtracted from the former.

REACTIVITY HistLength

This is the *REACTIVITY* technical indicator computed over the specified history length. It is slightly compressed and transformed to the range -50 to 50. Reactivity is primarily used when trading cycles. The *HistLength* lookback period should be approximately half of the period of the cycle the trader is exploiting. The reactivity is the price change over the history length, normalized by a function of the smoothed range and volume. For example:

```
REAC_20: REACTIVITY 20
```

The preceding definition computes reactivity over the most recent 20 bars, which will capture information about the status of the market in regard to a 40-bar cycle.

The Reactivity Indicator was proposed by Gietzen as part of trading approach described in Advanced Cycle Trading (Irwin Professional Publishing, 1995). It combines price rate-of-change (PROC) with price range and trading volume to overcome a deficiency associated with rate-of-change indicators based on price alone. PROC extremes (over-bought and over-sold) signal peaks and troughs in price during non-trending periods. However, during strong trends PROC extremes are premature. Gietzen's intuition is that when the reactivity indicator exceeds a critical high or low threshold, a strong up or down trend is in effect and is therefore more likely to continue than reverse. At such times PROC extremes should be ignored.

To define reactivity, a look-back window (*HistLength*) must be specified. It should be approximately one half of the duration of the cycle the trader is interested in exploiting. For example, if the trading cycle's duration is estimated to be 24 days, *HistLength* would set to 12. *HistLength* is used to compute the three factors that are combined to produce reactivity: price change, price range, and trading volume. These factors are defined as follows:

Price Change is the most recent value of the closing price minus the closing price *HistLength* bars ago.

Price Range is the maximum price bar high minus the minimum price bar low determined over *HistLength*.

Trading Volume is the total volume occurring during *HistLength*.

Price range and trading volume can be combined into a single quantity call the aspect ratio, defined as Price Range / Trading Volume. The aspect ratio is inspired by the work of Richard Arms discussed in Volume Cycles in the Stock Market (Dow Jones Irwin, 1983). However, the raw ratio is problematic for two reasons: (1) range and volume both vary considerably over time. A value that would be considered large at one time might be insignificant at another time, and (2) the quantities range and volume are denominated in different units, points and number of shares. To solve both problems, Gietzen suggests normalizing range and volume by their smoothed averages. The normalized price range is the price range divided by its smoothed value, and the normalized volume is the volume divided by its smoothed value. This modified aspect ratio is defined as follows:

$$AspectRatio = \frac{Range / SmoothedRange}{Volume / SmoothedVolume} \quad \textbf{(3)}$$

Geitzen uses exponential smoothing, and his heuristic for choosing a smoothing constant is a time period of four times the length of the trading cycle. Thus a trading cycle of 25 days calls for the exponential smoothing equivalent to a 100 day moving average. In particular, if n is the number of days of a simple moving average, the (very roughly) equivalent exponential smoothing constant is $2 / (n+1)$. In this example of a 100-day moving average, the exponential smoothing constant would be $2/101 \approx 0.02$.

The aspect ratio (which is a dimensionless constant) is multiplied by the price change (M) computed over HistLength, which is denominated in price points. This causes raw reactivity to be scaled in price points. In particular:

$$M = Price_0 - Price_{HistLength} \quad \textbf{(4)}$$

$$RawReactivity = M * AspectRatio \quad \textbf{(5)}$$

To obtain a dimensionless (pure) value for reactivity, Gietzen suggests dividing raw reactivity by the same exponentially smoothed range used above to obtain the aspect ratio. Thus Reactivity is defined as follows:

$$Reactivity = RawReactivity / SmoothedRange \quad \textbf{(6)}$$

The resulting pure number is interpreted as follows: values greater than one indicate a strong uptrend, values less than minus one indicate a strong down-trend, and intermediate values indicate a trading range. However, *TSSB* applies a compressing transform to the traditional reactivity to rescale it a range of -50 to 50 and thereby improve its compatibility with other indicators in the built-in library, so the original interpretation does not apply.

DELTA REACTIVITY HistLength DeltaDist

This is the *REACTIVITY* computed over the *HistLength* most recent bars, minus the same quantity computed for the bar *DeltaDist* bars earlier. It is slightly compressed and transformed to the range -50 to 50. For example:

```
DREACT_20_15: DELTA REACTIVITY 20 15
```

The preceding definition computes REACTIVITY using a lookback of the mos recent 20 bars, as well as the same thing at a lag of 15 bars. The latter is subtracted from the former and slightly compressed and transformed to a uniform range.

MIN REACTIVITY HistLength Dist

This is the minimum of *HistLength REACTIVITY* with the minimum taken over *Dist* days. It is slightly compressed and transformed to the range -50 to 50. For example:

```
MINREAC: MIN REACTIVITY 20 15
```

The preceding definition computes ‘REACTIVITY 20' for the most recent 20 bars 20 bars lagged one bar, 20 bars lagged two bars, and so forth. A total of 15 such computations are done. The indicator is the minimum of these 15 values.

MAX REACTIVITY HistLength Dist

This is identical to MIN REACTIVITY except that the maximum is taken instead o the minimum.

Trend-Like Indicators

The prior section presented indicators that depict multiple-bar price trends in obvious ways. This section covers indicators whose relationship to trend is close but not quite so obvious.

CLOSE TO CLOSE

This rapidly changing indicator is 100 times the log ratio of the current bar's close to the prior bar's close. It would rarely, if ever, be used as an input to a prediction model. It is extremely unstable and nonstationary. Also, it has very poor cross-market conformity (its distribution varies widely for different markets). However, it is the most common indicator used for Mahalanobis Distance (here) and Absorption Ratio (here) computation. See those sections for examples of its use.

N DAY HIGH HistLength

Let *N* be the number of bars one has to go back in order to find a price higher than the current bar's high. Search only *HistLength* bars back from the current bar. If this many bars are examined and no higher high is found, set *N*=*HistLength*+1. Then this variable is defined as 100 * (*N*-1) / *HistLength* - 50. Note that this variable is highly unstable. Suppose that over the most recent *HistLength* bars the highs have steadily increased. Then this indicator will attain its maximum value of 50. But now suppose the same steady increase is true, except that the current bar's high drops off by one tick lower than the prior bar's high. It may even be that this bar closes higher than the prior bar's close. Nevertheless, this indicator will attain its minimum value of -50!

For example:

```
NDH10: N DAY HIGH 10
```

The preceding definition examines the most recent 10 bars. It compares the high of the current bar to the highs of the prior 10 bars. If the immediately prior high is greater than the current bar's high, the value of this indicator is -50. If none of the prior highs exceed that of the current bar, the value of this indicator is 50. Intermediate locations of the most recent higher high will result in intermediate values of this indicator.

N DAY LOW HistLength

This is identical to N DAY HIGH except that we search for lower prices. Fo example:

NDL10: N DAY LOW 10

The preceding definition examines the most recent 10 bars. It compares the low of the current bar to the lows of the prior 10 bars. If the immediately prior low is less than the current bar's low, the value of this indicator is -50. If none of the prior lows is less than that of the current bar, the value of this indicator is 50. Intermediate locations of the most recent lower low will result in intermediate values of this indicator.

Deviations from Trend

The prior section discussed indicators that capture trend information. This section presents indicators that quantify the degree to which the current value of the market deviates from recent trend. We also include in this category the indicator that compares the current price to the moving average, because this is a deviation that is similar to deviations from trend.

CLOSE MINUS MOVING AVERAGE HistLen ATRlen

This indicator measures today's market price relative to its recent history, normalized by recent average true range. The close of the current bar is divided by the current moving average. The log of this ratio is divided by the average true range (based on log changes, of course). The result is transformed and slightly compressed to a range of -50 to 50. *HistLen* is the lookback period for the moving average, and *ATRlen* is the lookback period for the ATR calculation. For example:

```
CMMA_10: CLOSE MINUS MOVING AVERAGE 10 250
```

The preceding definition computes the average close of the most recent 10 bars and divides the close of the current bar by this average. The log of this ratio is found, and this is divided by the average true range of the most recent 250 bars. The result is transformed and compressed to a uniform range.

LINEAR DEVIATION HistLength

A least-squares line is fit to the most recent *HistLength* log prices, including that of today (or the current bar, if this is intraday data). The data which is fit is the log of the mean of the open, high, low, and close. The returned value is today's price minus the fitted line's value for today, divided by the standard error of the fit. A modest compressing transform is applied to limit the range to the interval -50 to 50. For example:

```
LINDEV_20: LINEAR DEVIATION 20
```

The preceding definition fits a straight line to the most recent 20 bars. This provides an estimate of what the price would be for the current bar if the trend line were exactly followed. This quantity is subtracted from the current price in order to quantify the degree to which the current price deviates from the trend line.

QUADRATIC DEVIATION HistLength

A least-squares quadratic polynomial is fit to the most recent *HistLength* log prices, including that of today. The data which is fit is the log of the mean of the open, high, low, and close. The returned value is today's price minus the fitted line's value for today, divided by the standard error of the fit. A modest compressing transform is applied to limit the range to the interval -50 to 50. This indicator is similar to LINEAR DEVIATION except that curvature (a second-orde term) is allowed in the fit. For example:

```
QUADDEV_20: QUADRATIC DEVIATION 20
```

The preceding definition does a least-squares fit of a parabola (a quadratic polynomial) to the most recent 20 bars. This provides an estimate of what the price would be for the current bar if the curving trend line were exactly followed. This quantity is subtracted from the current price in order to quantify the degree to which the current price deviates from that predicted by the parabolic fit.

CUBIC DEVIATION HistLength

A least-squares cubic (third degree) polynomial is fit to the most recent *HistLength* log prices, including that of today. The data which is fit is the log of the mean of the open, high, low, and close. The returned value is today's price minus the fitted line's value for today, divided by the standard error of the fit. A modest compressing transform is applied to limit the range to the interval -50 to 50. This indicator is similar to LINEAR DEVIATION except that curvature of second ar third order is allowed in the fit. For example:

```
CUBEDEV_20: CUBIC DEVIATION 20
```

The preceding definition does a least-squares fit of a cubic polynomial to the most recent 20 bars. This provides an estimate of what the price would be for the current bar if the curving trend line were exactly followed. This quantity is subtracted from the current price in order to quantify the degree to which the current price deviates from that predicted by the cubic fit.

DETRENDED RSI DetrendedLength DetrenderLength Lookback

Ordinary RSI at the two specified lengths are computed and a least-squares regression line is fit for predicting RSI at*DetrendedLength* from RSI at

DetrenderLength. This least-squares fit is based on these pairs of RSIs over the specified *Lookback* period. The value returned is the *DetrendedLength* RSI minus its predicted value. The only exception is that when *DetrendedLength* is 2, an inverse logistic function is applied to that RSI, and all computations (linear fit and computation of residual) are based on this transformed value. This function approximately linearizes what is otherwise a highly nonlinear relationship. No other compression or transformation is done. Note that *DetrendedLength* must be less than *DetrenderLength*. For example:

```
DETRSI: DETRENDED RSI 4 20 50
```

The preceding definition causes RSI at lengths of 4 and 20 to be computed for the 50 most recent bars. A least-squares line is computed for predicting 'RSI 4' from 'RSI 20' using these 50 pairs of RSI values. The current value of 'RSI 20' is plugged into this linear equation to predict the value of 'RSI 4' at the current bar. This indicator is then defined as the actual value of 'RSI 4' at the current bar minus its predicted value. In this way we can assess the direction and degree to which short-term RSI is currently deviating from its longer-term trend.

The intuition behind this indicator is the widely recognized tendency for RSI to oscillate in a range that reflects the longer-term trend. During non-trending periods the typical extremes of RSI are 30 and 70. However, in a strong long-term up-trend RSI tends to be contained in the range 40 to 80 while in strong long-term down-trends the fluctuation bounds are 20 to 60.

Volatility Indicators

An enormous number of popular measures of market volatility exist. This section presents many of them, as well as a few relatively obscure indicators. There is a common thread running through most of them: they are not absolute quantities. Rather, they measure recent volatility relative to 'historical norms' of volatility. There are at least two reasons for this:

1) Markets often experience very long-term, slow evolution in volatility. Thus, the same absolute volatility may be 'high' when it occurs in a low-volatility period of history, and 'low' when it occurs in a high-volatility period of history.

2) *TSSB* is often called upon to work with datasets that pool multiple markets. Different markets often have inherently different volatilities. If we measured absolute volatility, we would not have conformity of meaning across markets, which would make it impossible to find universally effective prediction models. By measuring current volatility in the context of 'normal' volatility for each market, we can achieve good conformity across markets.

ABS PRICE CHANGE OSCILLATOR ShortLen Multiplier

A *ShortLen* moving average of absolute log daily price changes is computed. The same is done for a length of *ShortLen* times *Multiplier*. The long-term MA is subtracted from the short-term MA, and this difference is divided by the average true range measured across the longer history. Finally, this normalized difference is transformed and slightly compressed to a range of -50 to 50. For example:

```
PCO_10_20: ABS PRICE CHANGE OSCILLATOR 10 20
```

The preceding definition computes the mean of the absolute values of log of closing price changes (ratio of current to prior) over the most recent 10 bars and the most recent 200 bars. The latter is subtracted from the former, divided by the average true range across the most recent 200 bars, and then transformed and scaled to a uniform range.

PRICE VARIANCE RATIO HistLength Multiplier

This is the ratio of the variance of the log of closing prices over a short time period to that over a long time period. It is transformed and scaled to a range of -50 to 50. The short time period is specified as *HistLength*, and the long time period is

HistLength times *Multiplier*. For example:

```
PVR_10_20: PRICE VARIANCE RATIO 10 20
```

The preceding definition computes the variance of the log of closing prices over the most recent 10 bars and the most recent 200 bars. The former is divided by the latter, and then transformed and scaled to a uniform range.

MIN PRICE VARIANCE RATIO HistLen Mult Mlength

This is the minimum of the *PRICE VARIANCE RATIO* over the prior *Mlength* observations. For example:

```
MNPVR: MIN PRICE VARIANCE RATIO 10 20 50
```

The preceding definition computes " PRICE VARIANCE RATIO 10 20" for tl most recent bar, the second-most recent, third-most recent, and so forth, for a total of 50 bars back in history. The value produced is the minimum of these 50 quantities.

MAX PRICE VARIANCE RATIO HistLen Mult Mlength

This is identical to MIN PRICE VARIANCE RATIO except that the maximum taken.

CHANGE VARIANCE RATIO HistLength Multiplier

This is identical to *PRICE VARIANCE RATIO* except that the quantity whose variance is computed is the log ratio of each day's close to that of the prior day. In other words, this is operating on changes rather than prices. For example:

```
CVR_10_20: CHANGE VARIANCE RATIO 10 20
```

The preceding definition computes the variance of the log of closing price changes over the most recent 10 bars and the most recent 200 bars. The former is divided by the latter, and then transformed and scaled to a uniform range.

MIN CHANGE VARIANCE RATIO HistLen Mult Mlen

This is the minimum of the *CHANGE VARIANCE RATIO* over the prior *Mlength* observations.

```
MNCVR: MIN PRICE CHANGE RATIO 10 20 50
```

The preceding definition computes "CHANGE VARIANCE RATIO 10 20" for tl most recent bar, the second-most recent, third-most recent, and so forth, for a total of 50 bars back in history. The value produced is the minimum of these 50 quantities.

MAX CHANGE VARIANCE RATIO HistLen Mult Mlength

This is identical to MIN CHANGE VARIANCE RATIO except that the maximum is taken.

ATR RATIO HistLength Multiplier

This is the ratio of the Average True Range over a short time period to that over a long time period. It is transformed and scaled to a range of -50 to 50. The short time period is specified as *HistLength*, and the long time period is *HistLength* times *Multiplier*. For example:

```
ATRRAT_10_20: ATR RATIO 10 20
```

The preceding definition computes the Average True Range over the most recent 10 bars and the most recent 200 bars. The former is divided by the latter, and then transformed and scaled to a uniform range.

DELTA PRICE VARIANCE RATIO HistLength Multiplier

This is the difference between the *PRICE VARIANCE RATIO* for the current bar minus that *HistLength* times *Multiplier* bars ago. For example:

```
DPVR: DELTA PRICE VARIANCE RATIO 10 20
```

The preceding definition computes "PRICE VARIANCE RATIO 10 20" for tl current bar as well as for that 200 bars earlier in history. The latter is subtracted from the former.

DELTA CHANGE VARIANCE RATIO HistLength Multiplier

This is the difference between the *CHANGE VARIANCE RATIO* for the current bar minus that *HistLength* times *Multiplier* days ago. For example:

```
DCVR: DELTA CHANGE VARIANCE RATIO 10 20
```

The preceding definition computes "CHANGE VARIANCE RATIO 10 20" for tl current bar as well as for that 200 bars earlier in history. The latter is subtracted from the former.

DELTA ATR RATIO HistLength Multiplier

This is the difference between the *ATR RATIO* for the current bar minus that *HistLength* times *Multiplier* days ago. For example:

```
DARRAT: DELTA ATR RATIO 10 20
```

The preceding definition computes "ATR RATIO 10 20" for the current bar as wel as for that 200 bars earlier in history. The latter is subtracted from the former.

BOLLINGER WIDTH HistLength

The mean and standard deviation of closing prices is computed for the specified history length. The value of this variable is the log of the ratio of the standard deviation to the mean. The effect of this division is to make the standard deviation be relative to the moving average. Note that this variable is extremely poorly behaved in nearly every regard. For this reason, historical normalization (full normalization, including both centering and scaling) as discussed beginning here is strongly recommended. For example:

```
BOLLWIDTH: BOLLINGER WIDTH 40 : NORMALIZE 200
```

The preceding command computes the BOLLINGER WIDTH variable for tl current bar, the prior bar, and so forth, for a total of 200 bars back in history. It computes the median and interquartile range of these 200 values, subtracts this median from the current value, and divides by the interquartile range. A transformation is also applied, as described in the section beginning here. This normalization converts the highly unstable BOLLINGER WIDTH to a nicely stab oscillator.

DELTA BOLLINGER WIDTH HistLength DeltaLength

This is the difference between the BOLLINGER WIDTH for the current bar min that *DeltaLength* bars earlier. Again, historical normalization (full normalization, including both centering and scaling) as discussed beginning here is strongly recommended. For example:

```
DBOLLWIDTH: DELTA BOLLINGER WIDTH 40 20 : NORMALIZE 200
```

The preceding command computes the DELTA BOLLINGER WIDTH variable f the current bar, the prior bar, and so forth, for a total of 200 bars back in history. It computes the median and interquartile range of these 200 values, subtracts this median from the current value, and divides by the interquartile range. A transformation is also applied, as described in the section beginning here. This normalization converts the highly unstable DELTA BOLLINGER WIDTH to nicely stable oscillator.

N DAY NARROWER HistLength

Let *N* be the number of days one has to go back in order to find a true range less than the current bar's true range. True range is defined as the maximum of the current bar's high minus its low, the current bar's high minus the prior bar's close, and the prior bar's close minus the current bar's low. Search only *HistLength* bars back. If after this many bars are checked, no smaller true range is found, set *N*=*HistLength*+1. Then this variable is defined as 100 * (*N*-1) / *HistLength* - 50. Note that this variable is highly unstable. Suppose that over the most recent *HistLength* bars the true range has steadily decreased. Then this indicator will attain its maximum value of 50. But now suppose the same steady decrease is true, except that the current bar's true range is one tick greater than that of the prior bar. This indicator will attain its minimum value of -50!

For example:

```
NDN10: N DAY NARROWER 10
```

The preceding definition examines the most recent 10 bars. It compares the true range of the current bar to that of the prior 10 bars. If the immediately prior bar's true range is less than that of the current bar, the value of this indicator is -50. If none of the prior true ranges are less than that of the current bar, the value of this indicator is 50. Intermediate locations of the most recent lesser true range will result in intermediate values of this indicator.

N DAY WIDER HistLength

This is identical to N DAY NARROWER except that we search for greater tru range. For example:

```
NDW10: N DAY WIDER 10
```

The preceding definition examines the most recent 10 bars. It compares the true range of the current bar to that of the prior 10 bars. If the immediately prior bar's true range is great than that of the current bar, the value of this indicator is -50. If none of the prior true ranges are greater than that of the current bar, the value of this indicator is 50. Intermediate locations of the most recent greater true range will result in intermediate values of this indicator.

Indicators Involving Indices

We saw here that one or more markets can be declared as *index* markets. Usually, an index market is a 'summary' market, an average of the markets in a defined set. We also saw that the IS INDEX modifier can be used to clone a variable for an index to all markets, and the MINUS INDEX modifier can be used to compute the deviation of a variable for a certain market from its value in an index market.

The IS INDEX and MINUS INDEX modifiers are general-purpose tools that can be applied to nearly all indicators in *TSSB*'s built-in library. In this section we will explore several special-purpose indicators that involve an index market.

INDEX CORRELATION HistLength

The ordinary correlation coefficient is computed between the log price (mean of open, high, low, and close) of the market under consideration and the log price of the index market. The correlation is over the specified history length, including the current bar. This variable is the correlation times 50, which provides a range of -50 to 50. If the user employs more than one index market, INDEX1 is used for this variable. Currently there is no way to specify any other index, although this feature could be added later if it becomes necessary. For example:

```
INDCORR: INDEX CORRELATION 20
```

The preceding definition computes the correlation between the market under consideration and the index market (such as OEX or the S&P500 index) over the most recent 20 bars. If the two markets (that under consideration and the index) happen to be perfectly correlated, the value of this variable will be 50. If they are perfectly negatively correlated (highly unlikely!), the value will be -50. The value will be zero if the two markets are totally uncorrelated.

DELTA INDEX CORRELATION HistLength DeltaLength

This is the INDEX CORRELATION of the current bar minus that *DeltaLength* bars ago. For example:

```
DINDCORR: DELTA INDEX CORRELATION 20 10
```

The preceding definition computes INDEX CORRELATION for the current bar and for the bar 10 bars ago. It subtracts the latter from the former.

DEVIATION FROM INDEX FIT HistLength MovAvgLength

A least-squares line is computed for predicting the log price (mean of open, high, low, and close) of the market under consideration from the log price of the index market. The fit is over the specified history length, including the current bar. This variable is the current bar's log price minus its predicted value, normalized by the standard error of the fit and slightly compressed to the range -50 to 50. If *MovAvgLength* is greater than one, a moving average of the variable is taken before compression. To avoid a moving average being taken, specify *MovAvgLength* as 0 or 1. If the user employs more than one index market, INDEX1 is used for this variable. Currently there is no way to specify any other index, although this feature could be added later if it becomes necessary. For example:

```
DEVFIT: DEVIATION FROM INDEX FIT 50 0
```

The preceding definition finds a least-squares linear equation for predicting the log price of the market under consideration from the log price of the index market. It uses the most recent 50 bars to fit this line. It then uses this equation to predict the log price of the current bar in the market under consideration, and subtracts this predicted value from the actual value. This deviation from the prediction is normalized and compressed.

PURIFIED INDEX Norm HistLen Npred Nfam Nlooks Look1 ...

A linear model is used to predict values of an index variable, and these predictions are subtracted from the true value of the index. This difference is the value of the PURIFIED INDEX indicator. No compression is done. Typically, the inde 'market' will not be an actual market. Rather, it will be a sentiment indicator such as VIX (or Investors Intelligence Survey). The predictors are based on recen market dynamics. In particular, linear trend, quadratic trend, and volatility of the market may be used as predictors in the linear model that predicts the sentiment index.

The user can specify up to ten lookback distances for computing the market-based predictors. For each lookback, linear trend, quadratic trend, and volatility may be measured. Thus, we may have as many as 10*3=30 predictors available for the linear model. The user must specify whether the model will use one or two of these candidates. If the user specifies that one be used, the best predictor will be chosen. If two, all possible pairs of predictors will be examined and the best pair chosen. This process of choosing the best predictor(s) and training the linear model to predict the sentiment index is repeated for every bar.

The following parameters must be specified by the user:

Normalization - Either or both of two normalization schemes can be used. One scheme is to multiply the predicted value by R-square before it is subtracted from the index. This has the effect of de-emphasizing the prediction when the model does a poor job of predicting the index. The price paid for this R-square normalization is less centering of the purified value. The other scheme is to divide the purified index by the standard error of the prediction. This stabilizes the variance of the purified index and has the effect of shrinking the value toward zero when the model does a poor job of prediction. The normalization parameter has the value 0, 1, 2, or 3:

0 - No normalization

1 - R-square normalization

2 - Standard error normalization (This is the method in David Aronson's paper)

3 - Both normalizations

HistLen - The number of recent observations that will be used to train the predictive model.

Npred - The number of predictors that will be used by the model. This must be 1 or 2.

Nfam - The number of families of predictors for each lookback. This must be 1, 2, or 3.

1 - Use linear trend only

2 - Use linear and quadratic trend

3 - Use linear and quadratic trend, and volatility

Nlookbacks - The number of lookback distances to use for the trends and volatility. This must be at least one, and at most ten.

Lookback1 - The first lookback distance

Lookback2 - The second lookback distance, if used. A total of *Nlookbacks* of these appear.

For example, the following definition would employ Type 2 normalization and use the 50 most recent values of linear trend, quadratic trend, and volatility to train the model. These 50 do not include the current bar. The linear model that predicts the sentiment index would employ two predictors. Four different lookbacks will be used for the trend and volatility indicators: 10, 20, 40, and 80 bars. Thus, the model

will have 3*4=12 candidate predictors.

```
PURE1: PURIFIED INDEX  2  50  2  3  4  10  20  40  80
```

Because the optimal model is recomputed for every bar, it is not practical to print for the user the predictors chosen for the model. They can and do change frequently. However, *TSSB* does print a summary of how often each candidate is chosen, which can be interesting. It also prints the mean R-square. Here is a sample such output, showing the use of all three families for lookbacks of 10 and 50 bars:

```
PURIFIED INDEX results for PURIF232 in OEX
Mean R-square = 0.927
  10 Linear      28.7 %
  10 Quadratic    8.4 %
  10 Volatility  10.7 %
  50 Linear      25.2 %
  50 Quadratic   13.2 %
  50 Volatility  13.8 %
```

Note that this indicator, while still valid, has been deprecated by the PURIFY Transform.

Basic Price Distribution Statistics

Sometimes it can be useful to know whether recent prices or price changes have an 'unusual' statistical distribution. Here, 'unusual' means that the prices or changes are unusually skewed (with extremely large or small values), or they have unusually heavy tails (outliers). The problem with using the traditional measures of skewness and kurtosis is that with financial data, a certain number of outliers are the norm. If one were to use the ordinary moment-based measures, skewness and kurtosis would be inflated and unstable. Instead, the indicators presented in this section use rank-based statistics so as to minimize this problem.

PRICE SKEWNESS HistLength Multiplier

This computes a measure of the skewness of the price distribution with a lookback period of *HistLength* days. This variable ranges from -50 to 50, with a value of zero implying symmetry. Positive values imply right skewness (some unusually larger prices) and negative values imply left skewness (some unusually smaller prices). If a *Multiplier* greater than one is specified, the skewness with a lookback period of *HistLength* times *Multiplier* is also computed, and the variable is the skewness of the shorter period relative to that of the longer period. A recent increase in skewness results in a positive value for this variable. To compute just the current skewness, specify a *Multiplier* of 0 or 1. Consider the following two examples:

```
PSKEW:  PRICE SKEWNESS 50 0
PRSKEW: PRICE SKEWNESS 50 3
```

The first of the two preceding commands computes the price skewness for the most recent 50 bars. The second example computes the price skewness for the most recent 50 and the most recent 150 bars. It then compares the two skewness values and returns a positive value if skewness is increasing and a negative value if skewness is decreasing.

CHANGE SKEWNESS HistLength Multiplier

This is identical to *PRICE SKEWNESS* except that the quantity evaluated is the daily price changes (one bar's close relative to the prior bar's close) rather than the prices themselves.

PRICE KURTOSIS HistLength Multiplier

This is identical to *PRICE SKEWNESS* except that the kurtosis (tail weight) rather than the skewness is measured.

CHANGE KURTOSIS HistLength Multiplier

This is identical to *CHANGE SKEWNESS* except that the kurtosis (tail weight) rather than the skewness is measured.

DELTA PRICE SKEWNESS HistLen Multiplier DeltaLen

This computes the difference between the current value of the PRICE SKEWNES and its value *DeltaLength* days ago. For example:

```
DPSKEW: DELTA PRICE SKEWNESS 50 3 200
```

The preceding definition computes ‘PRICE SKEWNESS 50 3' for the current ba and also for the bar 200 bars earlier. It then subtracts the latter from the former. So, for example, a positive value for this variable implies that skewness is increasing faster right now than it did 200 bars ago.

DELTA CHANGE SKEWNESS HistLen Multiplier DeltaLen

This is identical to DELTA PRICE SKEWNESS except that it is based on the dai price changes (one bar’s close relative to the prior bar’s close) rather than the prices themselves.

DELTA PRICE KURTOSIS HistLen Multiplier DeltaLen

This is identical to DELTA PRICE SKEWNESS except that the kurtosis (ta weight) is measured, instead of skewness.

DELTA CHANGE KURTOSIS HistLen Multiplier DeltaLen

This is identical to DELTA PRICE KURTOSIS except that it is based on the dail price changes (one bar’s close relative to the prior bar’s close) rather than the

prices themselves.

Indicators That Significantly Involve Volume

All of the indicators seen so far are based on price information only. The indicators presented in this section also incorporate volume as a major component of their information conveyance.

VOLUME MOMENTUM HistLength Multiplier

This computes the *Histlength* moving average of volume, as well as that over a length of *HistLength* times *Multipler*. The variable is the ratio of the former (short term) to the latter (long term), transformed and compressed to a range of -50 to 50. Thus, this variable measures the degree to which volume is increasing or decreasing. For example:

```
VMOM: VOLUME MOMENTUM 20 4
```

The preceding definition compares the average volume over the most recent 20 bars to the average volume over the most recent 80 bars. If they are equal, the result is zero. If the former exceeds the latter, the result is positive, and vice versa.

DELTA VOLUME MOMENTUM HistLen Multiplier DeltaLen

This is the VOLUME MOMENTUM for the current bar minus that *DeltaLen* bars ago, transformed to a range of -50 to 50. For example:

```
DVMOM: DELTA VOLUME MOMENTUM 20 4 100
```

The preceding definition computes ‘VOLUME MOMENTUM 20 4' for the curre bar as well as for the bar 100 bars ago. It subtracts the latter from the former and then transforms/compresses it to a uniform range.

VOLUME WEIGHTED MA OVER MA HistLength

This is the log of the ratio of the volume-weighted moving average to the ordinary moving average. It is slightly compressed and transformed to a range of -50 to 50. The volume-weighted moving average is computed by multiplying each closing price in the *HistLength* lookback period by the volume for that bar, summing, and then dividing that sum by the total volume for the lookback period. In this way, bars with high volume are given more emphasis than bars with low volume. If the

volume is equal for all bars, the ratio will be 1.0 and the value of this variable will be zero. A positive value of this variable implies that high-volume bars tended to have higher closing prices than low-volume bars. A negative value means the opposite is true. For example:

```
VWMAMA: VOLUME WEIGHTED MA OVER MA 50
```

In the preceding definition, the volume-weighted moving average of the most recent 50 bars is computed, as well as the ordinary moving average. The former is divided by the latter, the log is taken, and the result is transformed and compressed to a uniform range.

DIFF VOLUME WEIGHTED MA OVER MA ShortDist LongDist

This is 'VOLUME WEIGHTED MA OVER MA *ShortDist*', minus that over *LongDist*. It is slightly compressed and transformed to a range of -50 to 50. For example:

```
DVWMAMA: DIFF VOLUME WEIGHTED MA OVER MA 20 100
```

The preceding definition computes VOLUME WEIGHTED MA OVER MA w lookbacks of 20 and 100 bars. The latter is subtracted from the former, and the result is transformed and compressed to a uniform range.

PRICE VOLUME FIT HistLength

This is the slope of the least-squares regression line for predicting log closing price from log volume. It is slightly transformed to the range -50 to 50. If there is no relationship between volume and price, this variable is zero. A positive value implies that higher prices and higher volumes tend to occur together, while a negative value implies that higher volumes are associated with lower prices. For example:

```
PVF_50: PRICE VOLUME FIT 50
```

The preceding definition examines the most recent 50 bars and fits a regression line for predicting the closing price of each bar from the volume of that bar. The slope of this line is compressed and transformed to a uniform range.

DIFF PRICE VOLUME FIT ShortDist LongDist

This is the *PRICE VOLUME FIT* computed over *ShortDist* days, mius that over *LongDist* days. It is slightly transformed to the range -50 to 50. For example:

```
DIFPVF_20_100: DIFF PRICE VOLUME FIT 20 100
```

The preceding definition computes PRICE VOLUME FIT with lookbacks of 20 ar 100 bars. The latter is subtracted from the former, and the result is transformed and compressed to a uniform range.

DELTA PRICE VOLUME FIT HistLength DeltaDist

This is the *PRICE VOLUME FIT* computed over *HistLength* days minus the same quantity computed at a lag of *DeltaDist* days. It is slightly transformed to the range -50 to 50. For example:

```
DELPVF_20_30: DELTA PRICE VOLUME FIT 20 30
```

The preceding definition computes PRICE VOLUME FIT for the current bar with lookback of 20 bars. It then computes PRICE VOLUME FIT with a lookback of 2 bars for the bar that is 30 bars ago. The latter is subtracted from the former, and the result is transformed and compressed to a uniform range.

ON BALANCE VOLUME HistLength

This is the *ON BALANCE VOLUME* technical indicator computed over *HistLength* days. It is slightly transformed to the range -50 to 50. In order to compute this indicator, two quantities are cumulated. The total volume is the sum of the volumes of the *HistLength* most recent bars. The signed volume is the sum of the volumes of all bars whose close exceeds the close of the prior bar, minus the sum of the volume of all bars whose close is less than that of the prior bar. Thus, high volumes on bars having increasing price will push up the signed volume, while high volumes on bars having decreasing price will push down the signed volume. Bars with low volume will have relatively little impact on the signed volume. The ON BALANCE VOLUME indicator is the ratio of the signed volume to the tota volume, compressed and transformed to a range of -50 to 50. For example:

```
OBV50: ON BALANCE VOLUME 50
```

The preceding definition computes the total and the signed volumes over the most

recent 50 bars. The latter is divided by the former, and the ratio is compressed and transformed to a uniform range.

DELTA ON BALANCE VOLUME HistLength DeltaDist

This is the *ON BALANCE VOLUME* computed over *HistLength* days, minus the same quantity computed at a lag of *DeltaDist* days. It is slightly transformed to the range -50 to 50. For example:

```
DOBV50: DELTA ON BALANCE VOLUME 50 45
```

The preceding definition computes the ON BALANCE VOLUME for the mc recent 50 bars, as well as for the bar 45 bars earlier. The latter is subtracted from the former, and the difference is compressed and scaled to a uniform range.

POSITIVE VOLUME INDICATOR HistLength

This is the average relative price change (the difference between the current bar's close and the prior bar's close, divided by the prior bar's close) over the specified number of bars. (Actually, it examines *HistLength*+1 bars, because it considers *HistLength* changes.) Only changes corresponding to increased volume are cumulated into the sum for finding the mean. Price changes that correspond to constant or decreased volume are treated as zero for the purposes of finding the average, thus reducing the average. In order to provide cross-market conformity (which is terrible in its raw form), this average is normalized by dividing by the standard deviation of price changes taken over a history of 2**HistLength* or 250 days, whichever is longer. It is slightly compressed and transformed to the range -50 to 50. For example:

```
POSVOL: POSITIVE VOLUME INDICATOR 40
```

The preceding definition examines the most recent 41 bars, which implies examination of the 40 most recent relative price changes. Those changes that correspond to increasing volume are summed, and the sum is divided by 40 to find the mean. This mean is then divided by the standard deviation of price changes over the most recent 250 bars. This normalized quantity is compressed and transformed to a uniform range.

DELTA POSITIVE VOLUME INDICATOR HistLen DeltaDist

This is the *POSITIVE VOLUME INDICATOR* computed over *HistLen* changes minus the same quantity computed at a lag of *DeltaDist* bars. It is slightly transformed to the range -50 to 50. For example:

```
DPOSVOL: DELTA POSITIVE VOLUME INDICATOR 40 35
```

The preceding definition computes 'POSITIVE VOLUME INDICATOR 40' for t current bar as well as for the bar 35 bars earlier. The latter is subtracted from the former, and the difference is compressed/transformed to a uniform range.

NEGATIVE VOLUME INDICATOR HistLength

This is identical to *POSITIVE VOLUME INDICATOR* except that only bars having decreasing volume are considered.

DELTA NEGATIVE VOLUME INDICATOR HistLen DeltaDist

This is identical to *DELTA POSITIVE VOLUME INDICATOR*, except that price changes are considered only when they correspond to decreasing volume.

PRODUCT PRICE VOLUME HistLength

For each bar in the history, the 'precursor' to this indicator is computed in three steps:

1) The current bar's volume is normalized by dividing it by the median of the prior 250 bar's volumes. Thus, if the current bar's volume is 'average' the result will be one. If the current bar's volume is unusually small, the result will be near zero, and if unusually large, the result will be much greater than one.

2) The current bar's price change (log of the ratio of the current close to the prior close) is normalized by subtracting the median of this quantity over the prior 250 bars and dividing by the interquartile range.

3) The precursor to the *PRODUCT PRICE VOLUME* indicator is computed as the product of the normalized price of the current bar times the normalized volume of the current.

The net result of these three steps is that bar price changes that correspond to unusually large relative volume will be amplified, while price changes

corresponding to relatively small volumes will be diminished. However, this precursor value varies greatly from bar to bar. It needs smoothing. So...

The final value of the PRODUCT PRICE VOLUME indicator is computed means of a moving average of the precursor described above. The moving average is taken over *HistLength* bars. It is slightly compressed and transformed to the range -50 to 50.

For example:

```
PPV: PRODUCT PRICE VOLUME 25
```

For each bar, the preceding definition evaluates the normalized volume and price changes over the most recent 25 bars, computes their average product, and compresses/transforms the result to a uniform range.

SUM PRICE VOLUME HistLength

Each bar's price change and volume are normalized as in the *PRODUCT PRIC VOLUME* indicator. The precursor to the *SUM PRICE VOLUME* indicator is computed as the sum of the normalized volume and the absolute value of the normalized price change. If the normalized price change is negative, the sign of the sum is flipped. This produces a result that is vaguely similar to what was had in the PRODUCT PRICE VOLUME case, in that extreme results are obtained wh volume is relatively high and the price change is extreme. However, in the case of SUM PRICE VOLUME, an extreme value of either volume or price change alo can cause the result to be extreme, while for PRODUCT PRICE VOLUME th both must be extreme. The final value is computed by averaging this sum over the most recent *HistLength* bars. For example:

```
SPV: SUM PRICE VOLUME 25
```

For each bar, the preceding definition evaluates the normalized volume and price changes over the most recent 25 bars, computes their average signed sum, and compresses/transforms the result to a uniform range.

DELTA PRODUCT PRICE VOLUME HistLen DeltaDist

This is the *PRODUCT PRICE VOLUME* indicator computed over *HistLen* bars minus the same quantity computed at a lag of *DeltaDist* bars. It is slightly compressed and transformed to the range -50 to 50. For example:

DPPV: DELTA PRODUCT PRICE VOLUME 40 35

The preceding definition computes ‘PRODUCT PRICE VOLUME 40' for t current bar as well as for the bar 35 bars earlier. The latter is subtracted from the former, and the difference is compressed/transformed to a uniform range.

DELTA SUM PRICE VOLUME HistLen DeltaDist

This is the *SUM PRICE VOLUME* indicator computed over *HistLen* bars minus the same quantity computed at a lag of *DeltaDist* bars. It is slightly compressed and transformed to the range -50 to 50. For example:

DSPV: DELTA SUM PRICE VOLUME 40 35

The preceding definition computes ‘SUM PRICE VOLUME 40' for the current b as well as for the bar 35 bars earlier. The latter is subtracted from the former, and the difference is compressed/transformed to a uniform range.

Entropy and Mutual Information Indicators

The exact mathematical definition of information is beyond the scope of this text. However, it is not too different from the intuitive meaning of the term. When a variable carries information, this variable tells us something about the state of the process on which it is based.

Two variables may share some information. A simple example might be two measures of volatility. Each of them might contain information about some general aspect of volatility, while at the same time each might also respond to some aspect of volatility that is specific to that variable. Shared information is called *mutual information*. The mutual information indicators available in *TSSB* do not involve pairing of variables. Rather, they relate to information on price changes at various short lags. This will be made explicit in the definitions of these indicators.

On a simple level, *entropy* is a measure of disorder. A variable with high entropy appears to be highly disordered. It contains a large number of states. Conversely, a variable with low entropy is very ordered. It contains relatively few distinct states.

This section describes indicators that are based on entropy and mutual information. They are all based on a simple partitioning of the historical data. Two specifications are important. The *word length* is a small number, typically 1-5 or so, which is the number of contiguous bars considered together to define a single unit of relationship. Each bar in a word defines a binary quantity: either a bar closes higher than the prior bar, or it does not. Thus, the total number of possible patterns is two to the power of the number of comparisons made. This will become more clear when illustrated in the context of specific indicators.

The other important specification is the *window length*. This is the number of historical bars that are examined in order to compute the value of an indicator for a single date/time. The user cannot set the window length. It is predefined to be ten times the number of possible patterns discussed above. As a result, the average number of cases that fit each binary pattern is ten.

PRICE ENTROPY WordLength

This computes the binary entropy of price changes, transformed and slightly compressed to the range -50 to 50. The number of historical bars used in the computation (window length) is ten times two to the power *WordLength*. For example:

PENT: PRICE ENTROPY 2

The preceding definition compares the closing price of the current bar with the closing price of the prior bar. Either the price increased or it did not, a binary outcome. Then it compares the closing price of the prior bar with the closing price of the bar just before it. Again, either the price increased or it did not. This give us four possible relationships which can be visualized as a 2 by 2 arrangement of four cells. Whichever cell corresponds to the pattern of these two price changes has its count incremented. Then this block of adjacent comparisons is moved back in time by one bar. The same pair of comparisons is made, and the appropriate cell count is incremented. This is repeated 10*4=40 times. Finally, the entropy of this 2 by 2 set of cells is computed and transformed/compressed to a uniform range. If the patterns vary widely in this set of 40 historical bars, the entropy will be high. Conversely, if a relatively small set of patterns predominate, the entropy will be low.

VOLUME ENTROPY WordLength

This computes the binary entropy of volume changes, transformed and slightly compressed to the range -50 to 50. Computation is identical to PRICE ENTROPY except that the volume of each bar is used instead of closing price.

PRICE MUTUAL INFORMATION WordLength

This computes the binary mutual information between the current bar's price change (prior close to current close), and the *WordLength* prior bars' price changes. This quantity is transformed and slightly compressed to the range -50 to 50. The number of bars used in the computation is ten times two to the power [one plus *WordLength*]. Note that for ENTROPY, the unit of relationship is *WordLength* bars, but for MUTUAL INFORMATION the unit of relationship is *WordLength*+1 bars. This is because the mutual information is between a set of *WordLength* contiguous bars (lagged one bar behind the current bar) and the current bar. For example:

PMI: PRICE MUTUAL INFORMATION 2

The preceding definition compares the closing price of the current bar with the closing price of the prior bar. Either the price increased or it did not, a binary decision. This is one of the two variables whose mutual information is computed. The other variable has four states (two to the power *WordLength*). These states are defined exactly as was done for entropy: The close of the lag-1 bar is compared to the close of the lag-2 bar. Either the price increased or it did not. Similar, the close

of the lag-2 bar is compared to the close of the lag-3 bar. Either it increased or it did not. Recalling that the current bar also has two states, we thus have a total of 2*2*2=8 cells. The cell count corresponding to the combined state is incremented. Then this block of 1+2=3 contiguous bars is moved back one bar and the operation is repeated. This counting operation is done a total of 10*2^3=10*8=80 times. The mutual information between the binary current bar price change and the 4-state prior two-bar price changes is computed and transformed/compressed to a uniform range. If over this 80-bar window there is a substantial relationship between the 4-state price pattern of the prior two bars and the binary price pattern of the current bar, the mutual information indicator will be large, toward 50. Conversely, if there is little such relationship within this 80-bar window, the mutual information indicator will be small, toward -50. One would tend to see large mutual information when the market behavior is organized, with good short-term predictability. Conversely, periods of highly random market behavior will result in small mutual information.

VOLUME MUTUAL INFORMATION WordLength

This computes the mutual information of volume changes, transformed and slightly compressed to the range -50 to 50. Computation is identical to PRICE MUTUA INFORMATION except that the volume of each bar is used instead of closin price.

Indicators Based on Wavelets

A wavelet decomposition analyzes a time series in two dimensions: location and frequency (or, equivalently, period). Features in the time series are identified by their location in time and their approximate frequency. So, for example, we may (very roughly speaking) say that on a certain date the time series contained an event whose dominant frequency/period was a certain value.

Note that frequency and period have an inverse relationship and are equivalent ways of specifying the repetition rate of a periodic event. An event with a period of n bars has a frequency of $1/n$ cycles per bar. Because of a mathematical limitation called the Nyquist limit, the maximum frequency we can measure is 0.5 cycles per bar. Equivalently, the minimum period we can measure is 2 bars.

Although wavelets are new to many market analysts they are similar to indicators that are commonly used: moving average oscillators. An n-period simple moving average is derived by adding up n prices and dividing by n. An alternative way of looking at it is to multiply each of n prices by a weight of $1/n$ and then sum the products. Thinking of a moving average in these terms makes clear the idea of a weight function, a set of weights that are multiplied against a set of prices. If one visualizes a set of equally valued weights as a sequence of numbers along the time axis, the weights have a rectangular shape. This is the shape of the weight function used to produce a simple moving average. Other types of moving averages use weight functions that have other shapes, such as a triangle or weights that decrease in value exponentially. So long as the sum of the weights equals 1.0 you obtain some form of moving average.

A common indicator based on subtracting a short term moving average from a long-term moving average is a price oscillator. For example, we may have the 3-day moving average minus the 10-day moving average. Price oscillators vary around a mean value of zero. This is because the weights sum to zero. Thus, the moving average oscillator is the result of applying a weight function to prices. In this example, three weights, each having a value of 0.3333, form the 3-day average, while 10 weights each having a value of 0.10 form the 10-day moving average. When the weights for the 10-day MA are subtracted from the weights for the 3-day MA, the resulting weights are pictured in Figure 7 below. Note that it has both positive and negative weights, which sum to zero.

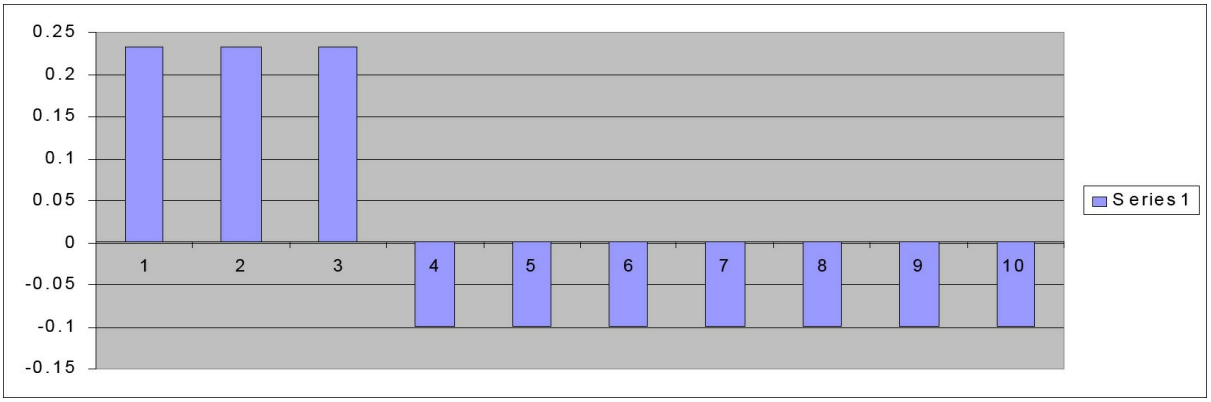


Figure 7: Weights for a moving-average oscillator

Wavelets can be thought of as a sophisticated type of price oscillator that uses a more intelligently designed weight function. By altering the shape of the weight function we can obtain an oscillator that more precisely responds to the price events in which we are most interested. Figure 8 below is a weight function for a particularly useful wavelet, the Morlet Wavelet. Note that it has both positive and negative values, and they sum to zero.

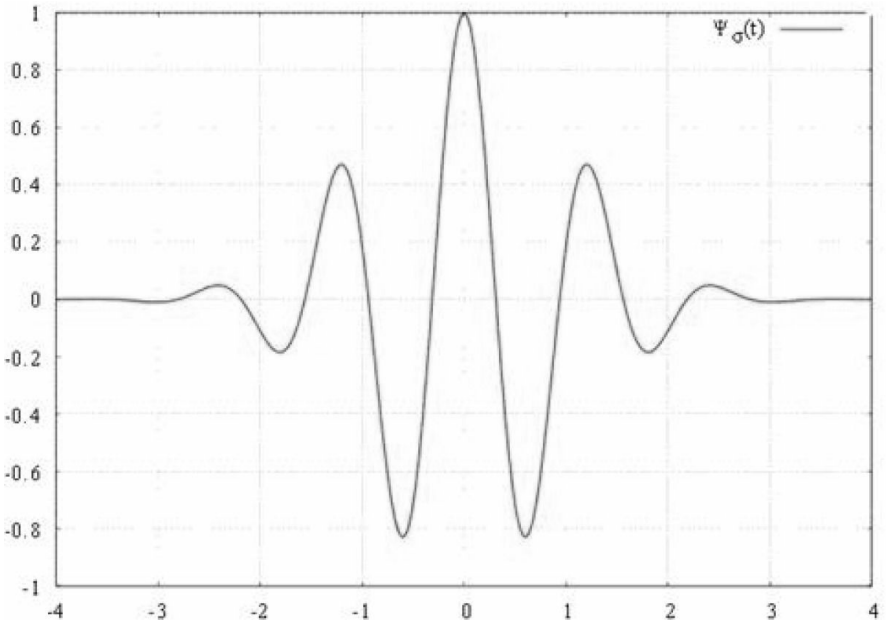


Figure 8: A Morlet Wavelet

There are an infinite number of wavelet decompositions (types of wavelet families) that are theoretically possible. Several tradeoffs are involved in choosing the best wavelet for a particular application. One of the two primary considerations is the Heisenberg Uncertainty Principle. This says that we cannot accurately identify ar event in terms of both its period and its time of occurrence. If we demand high accuracy in finding the time, we will have to settle for low accuracy in its period. Conversely, if we need to locate an event of a certain narrowly specified period,

we will not be able to pinpoint when it occurred with high accuracy. It is usually in our best interest to use a wavelet that does the best possible job of dealing with this unfortunate compromise.

There is a second consideration for our choice of wavelet type. This issue is usually of lesser importance than the Heisenberg time/period compromise, but not always. The issue is redundancy. In some applications we will want to detect wavelet events at a wide range periods and at frequent intervals, probably every bar. This can be a lot of information for a prediction model to process, so we want to use as few indicators as possible to satisfy our goal of being thorough in our capture wavelet information. Ideally, we want each wavelet indicator to carry its own unique set of information so that we convey the maximum possible amount of information in the minimum possible number of indicators in order to avoid overfitting due to the so-called *curse of dimensionality*. The best possible wavelet (in terms of redundancy) would be able to (theoretically, at least) perfectly reproduce the original time series with fewer indicators than any other wavelet.

Unfortunately, these two criteria (simultaneous location in period and time, and redundancy) are in direct conflict. The best 'simultaneous locators' are terribly redundant, and the least redundant wavelets do an abysmal job of simultaneously locating an event in terms of period and time of occurrence.

Because of this conflict, *TSSB* provides two wavelet families that are at opposite ends of this continuum. *Morlet wavelets* perfectly attain the Heisenberg Uncertainty Principle limit; no other wavelet does a better job of simultaneously locating events in terms of period and time. But members of the Morlet wavelet family are seriously redundant. A huge number of Morlet wavelets would be required to come even close to encapsulating all of the information in a time series. Supplying enough Morlet wavelet wavelets to predictive models so as to capture all of the information in a price series would create massive overfitting. Thus, the user must be judicious in choosing Morlet wavelets as indicators.

At the opposite extreme, *Daubechies wavelets* have zero redundancy. No other wavelet captures as much information about the time series in as few indicators. The price paid is exceptionally poor localization in terms of period and time. In most financial applications, we want to know the time of an event with maximum precision. For this reason, Morlet wavelets are almost always preferred to Daubechies wavelets. However, there are some exceptions, so both types are available.

Keep in mind that because future leak must be strictly avoided, all wavelet indicators operate at a lag. For Morlet wavelets, this lag is exactly twice the user-specified period. In other words, any time we compute a Morlet wavelet indicator, we are actually measuring (and hence providing to the prediction model) the value

of the indicator two periods ago. There is no way to compute Morlet wavelets of less lag without making major sacrifices in frequency response, although the DIFF and PRODUCT indicators discussed soon do so as part of their operation. For Daubechies wavelets, the exact lag is not so easily specified because of their very poor time localization.

REAL MORLET Period

The real (in-phase) component of a Morlet wavelet is computed for the log of closing prices It is slightly compressed and transformed to a range of -50 to 50. The wavelet is centered 2 * *Period* bars prior to the current bar. REAL MORLET roughly measures the position of the price within a periodic waveform of approximately the specified period. The period must be greater than or equal to 2.0. For example:

```
RMORLET: REAL MORLET 5
```

The preceding definition computes the real component of a wavelet having a repetition period of 5 bars. The indicator computed for a given bar is the value of the real component 2*5=10 bars earlier. Roughly speaking, this means that if we were to isolate only the component of the market series that repeats with a period of 5 bars, and ignore all other repetitive components and noise, this value is the price due to that component at that time.

REAL DIFF MORLET Period

The real component of a Morlet wavelet is computed for the log of closing prices. The wavelet is centered 2 * *Period* bars prior to the current bar. The same quantity is computed for twice the period, though the lag is the same. The long-period quantity is subtracted from the short-period quantity. This roughly measures whether the price value due to fast motion is above or below that due to the slow motion. The period must be greater than or equal to 2.0. For example:

```
RDMORLET: REAL DIFF MORLET 5
```

The preceding definition computes the real component of a wavelet having a repetition period of 5 bars, and also the real component of a wavelet having a period of 10 bars. The pair of indicators computed for a given bar are the values of the real components 2*5=10 bars earlier. The 10-bar wavelet is subtracted from the 5-bar wavelet, and the difference is transformed/compressed to a uniform range.

REAL PRODUCT MORLET Period

The real component of a Morlet wavelet is computed for the log of closing prices. The wavelet is centered 2 * *Period* bars prior to the current bar. The same quantity is computed for twice the period, though the lag is the same. If the two quantities have opposite signs, the value of the variable is zero. Otherwise, the value is their product with their sign preserved. This roughly measures whether the price value due to fast motion and that due to slow motion are in agreement, with the sign telling their position within the common motion. The period must be greater than or equal to 2.0. For example:

```
RPMORLET: REAL PRODUCT MORLET 5
```

The preceding definition computes the real component of a wavelet having a repetition period of 5 bars, and also the real component of a wavelet having a period of 10 bars. The pair of indicators computed for a given bar are the values of the real components 2*5=10 bars earlier. If the 10-bar and 5-bar wavelets have opposite signs, the value of this indicator is set to zero because the price positions of the two periodic components are in conflict. Otherwise, the 10-bar wavelet is multiplied by the 5-bar wavelet. The result is given the sign of the wavelets, and the difference is transformed/compressed to a uniform range.

IMAG MORLET Period

The imaginary (in-quadrature) component of a Morlet wavelet is computed for the log of closing prices It is slightly compressed and transformed to a range of -50 to 50. The wavelet is centered 2 * *Period* bars prior to the current bar. IMAG MORLET roughly measures the velocity of the price within a periodic waveform of approximately the specified period. The period must be greater than or equal to 2.0. For example:

```
IMORLET: IMAG MORLET 5
```

The preceding definition computes the imaginary component of a wavelet having a repetition period of 5 bars. The indicator computed for a given bar is the value of the imaginary component 2*5=10 bars earlier. Roughly speaking, this means that if we were to isolate only the component of the market series that repeats with a period of 5 bars, and ignore all other repetitive components and noise, this value is the price velocity (signed rate of change) due to that component at that time.

IMAG DIFF MORLET Period

The imaginary component of a Morlet wavelet is computed for the log of closing prices. The wavelet is centered 2 * *Period* bars prior to the current bar. The same quantity is computed for twice the period, though the lag is the same. The long-period quantity is subtracted from the short-period quantity. This roughly measures whether the velocity of the fast motion is above or below the velocity of the slow motion. The period must be greater than or equal to 2.0. For example:

```
IDMORLET: IMAG DIFF MORLET 5
```

The preceding definition computes the imaginary component of a wavelet having a repetition period of 5 bars, and also the imaginary component of a wavelet having a period of 10 bars. The pair of indicators computed for a given bar are the values of the imaginary components 2*5=10 bars earlier. The 10-bar wavelet is subtracted from the 5-bar wavelet, and the difference is transformed/compressed to a uniform range.

IMAG PRODUCT MORLET Period

The imaginary component of a Morlet wavelet is computed for the log of closing prices. The wavelet is centered 2 * *Period* bars prior to the current bar. The same quantity is computed for twice the period, though the lag is the same. If the two quantities have opposite signs, the value of the variable is zero. Otherwise, the value is their product with their sign preserved. This roughly measures whether the velocities of the fast motion and the slow motion are in agreement, with the sign telling the direction of the common motion. The period must be greater than or equal to 2.0. For example:

```
IPMORLET: IMAG PRODUCT MORLET 5
```

The preceding definition computes the imaginary component of a wavelet having a repetition period of 5 bars, and also the imaginary component of a wavelet having a period of 10 bars. The pair of indicators computed for a given bar are the values of the imaginary components 2*5=10 bars earlier. If the 10-bar and 5-bar wavelets have opposite signs, the value of this indicator is set to zero because the price velocities of the two periodic components are in conflict. Otherwise, the 10-bar wavelet is multiplied by the 5-bar wavelet. The result is given the sign of the wavelets, and the difference is transformed/compressed to a uniform range.

PHASE MORLET Period

This is the *rate of change* of the phase (not the phase itself) of a wavelet having the

specified period, transformed and scaled to a range of -50 to 50. The wavelet is centered 2 * *Period* bars prior to the current bar. Roughly speaking, this tells us how rapidly the actual phase of the wavelet is changing relative to what one would expect from a pure (noiseless) wave. For example:

```
PHMORLET: PHASE MORLET 5
```

The preceding definition computes a Morlet wavelet with a repetition period of 5 bars, centered 2*5=10 bars prior to the current bar. The phase at that bar ten bars prior to the current bar is computed, as well as the phase at the bar just prior to it (11 bars before the current bar). The phase difference is computed and transformed/compressed to a uniform range.

DAUB MEAN HistLength Level

A Daubechies wavelet is computed for the log of bar close ratios (current bar's close divided by the prior bar's close) over the most recent *HistLength* bars. *HistLength* must be a power of two. If not, it is increased to the next power of two. The level must be 1, 2, 3, or 4, with larger values resulting in more smoothing and noise elimination. Two to the power of (*Level*+1) must be less than or equal to *HistLength*. Li, Shi, and Li recommend that *HistLength* be approximately three times the number of bars we will predict into the future. They also recommend that *Level* be two, because a value of one results in too much noise being retained, while values larger than two result in loss of useful information. The user may wish to do his/her own experiments to choose optimal values. The *DAUB MEAN* variable is the mean of the smooth (parent wavelet) coefficients, with the detail coefficients ignored. This mean is compressed and transformed to a range of -50 to 50. For example:

```
DAUBMEAN: DAUB MEAN 16 2
```

The preceding definition computes a level-2 Daubechies wavelet over the most recent 16 bars. The mean of the parent wavelet's coefficient is found and transformed/compressed to a uniform range.

DAUB MIN HistLength Level

This is identical to *DAUB MEAN* except that the minimum rather than the mean is returned.

DAUB MAX HistLength Level

This is identical to *DAUB MEAN* except that the maximum rather than the mean is returned.

DAUB STD HistLength Level

This is identical to *DAUB MEAN* except that the standard deviation rather than the mean is returned.

DAUB ENERGY HistLength Level

This is identical to *DAUB MEAN* except that the sum of squared coefficients rather than the mean is returned.

DAUB NL ENERGY HistLength Level

This is identical to *DAUB MEAN* except that the sum of squared differences between neighbors (adjacent coefficients of the parent wavelet) rather than the mean is returned.

DAUB CURVE HistLength Level

This is identical to *DAUB MEAN* except that the sum of absolute differences between neighbors (adjacent coefficients of the parent wavelet) rather than the mean is returned.

Follow-Through-Index (FTI) Indicators

Look at the hypothetical market price graphs shown in Figures 9 and 10 at the bottom of this page. Which market do you think would be easier for a prediction model to handle? In fact, even if you were just sitting at a terminal, watching the market, and trading by the seat of your pants, which market do you think would be easier for you to successfully trade?

Surely, most people would prefer a market that looks like that in Figure 10. The dominant movement of the market stands out well above the noise. One might say that prices *follow through* on their motion better in Figure 10 than they do in Figure 9. In that latter figure, prices keep bouncing back and forth in wild swings.

Thus, it is useful to be able to measure the degree to which longer-term price movement continues marching onward, overshadowing smaller local movement that is usually just noise. When a market is in a period of high follow-through we would probably be inclined to trade it, while if the follow-through is low we might want to sit out for a while.

There are many simple ways to measure long-term net change relative to noise. The ordinary Sharpe Ratio comes to mind, as well as ADX and many variance-adjusted trend indicators. However, G. S. Khalsa, in "New Concepts in Identifying Trends and Categorizing Market Conditions" proposes an extremely sophisticated measure that he calls the *Follow Through Index* or *FTI*. The exact computations are far too complex to present in full detail here. However, the basic idea is discussed starting on the here.

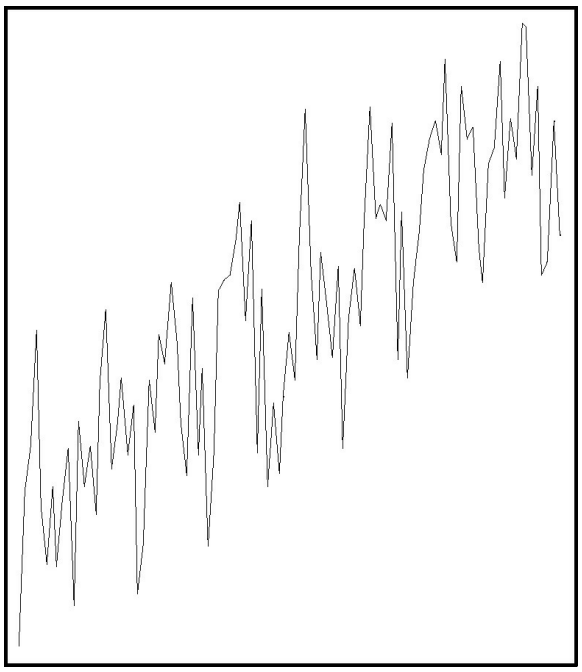


Figure 9: Market with small follow-through

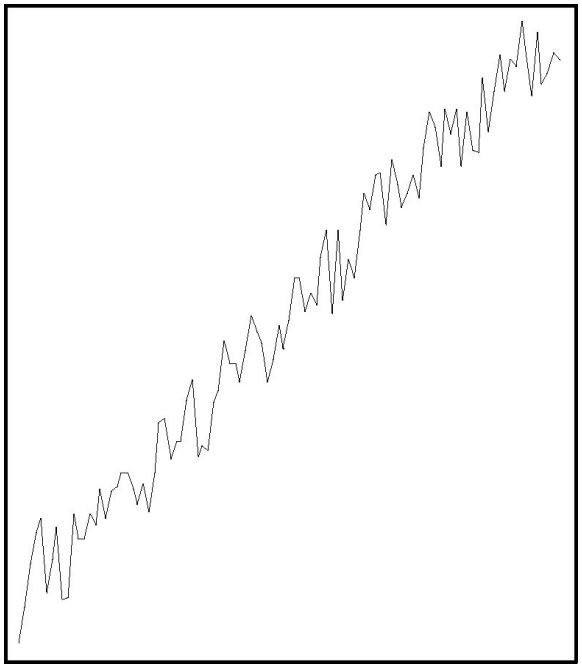


Figure 10: Market with large follow-through

Low-Pass Filtering and FTI Computation

The concept of a low-pass filter is central to FTI computation, so it is important that the reader be clear on this topic. An example of a simple low-pass filter is the ordinary moving average. The term *low* refers to the frequency of the oscillations that remain (i.e., are passed) by the filter. In the time domain, with which many market technicians are more familiar, low-frequencyoscillations are long-duration changes in trend. Applying a moving average to price data results in a smooth curve that retains the long-term trend changes but removes the shorter-term movements around the trend. The term *cutoff* refers to the frequency of the shortest-duration trend change that will show up in the filtered (smoothed) data. Trend changes of shorter duration than the cutoff will not show up in the behavior of the filtered data.

Block Size and Channels

The developer of the FTI indicator family comes from a signal-processing background, so his terminology is somewhat foreign to many market traders. We will stay with his terminology. However, as an aid to translating terminology from the signal-processing domain to the market-trading domain, please keep these two issues in mind:

1) Computation of the FTI indicators is done by moving a window across the market history. This is similar to most or all of the indicators previously discussed

in this document. In order to compute FTI indicators at a point in time, we examine a block of history extending backwards from that point in time. The length (lookback) of this moving-window block is the *Block Size* that will be frequently referred to here.

2) Due to the mechanical aspects of filtering, the entire block of *Block Size* historical bars cannot be used for the most critical aspects of FTI indicator computation. Only the most recent subset of the total block size may be used. This subset of the entire block (the moving window) is called the *Channel*. Only bars in the channel are used for most aspects of FTI indicator computation. Later, when we see illustrations of FTI operations, the reader will see how the concept of an FTI channel is similar to channels of other types that are already familiar to market technicians. In particular, FTI channels often appear strikingly similar to Bollinger Bands.

Essential Parameters for FTI calculation

The user must specify three parameters in order to compute the FTI measure of price follow-through:

BlockSize is the length of the moving-window block that marches through the market history. This is exactly the same concept as in virtually all indicators: when we compute the value of an indicator at the current point, we examine a block of the most recent points in order to do this calculation. For most other indicators described in this document, the length of this block is called *HistLength*. However, we call this quantity *BlockSize* here in order to conform to convention in the FTI literature.

Period is the number of bars for the cutoff of a lowpass filter that is applied to the data for much of the FTI calculation. Consider that any measure of follow-through is highly dependent on where we draw the line between noise and valid price movement. Ifwe are liberal and label all but the smallest, most frantic market moves to be valid, we will probably get a very different measure of follow-through than if we are conservative and label every move as noise unless it is long and large. In the Khalsa algorithm, price changes that remain after lowpass filtering are used to compute net price movement, and the differences between the unfiltered and the filtered (smoothed) prices are used for noise calculation. Our choice of a period for the lowpass filter is application dependent and may require some experimentation. Khalsa also has a method available in *TSSB*, described later, for automatically

computing an 'optimal' filter period based on the characteristics of the block of market prices being analyzed. This is a convenient way to avoid the need for specifying a period, but the utility of this algorithm is open to debate.

HalfLength is more a computational issue than an experimental issue. This is the number of points on each side of the center point that are used to compute the lowpass filter. Choosing a reasonable value for the *HalfLength* is discussed soon.

These three quantities interact in various ways, so careful choice of their values is critical. Here is a summary of the relevant issues and rules:

- In most cases, it is best to begin by choosing a *Period* for the filter. Khalsa processes day bars, and he uses periods ranging from 5 days up to 65 days in his demonstrations. Try different values, or use the automated selection algorithm described later.

- *HalfLength* must be greater than or equal to half of the *Period*. In most cases it is best to make it somewhat greater, as strict equality produces a lowpass filter of marginal quality. Larger values of *HalfLength* produce better filters and hence more accurate FTI indicators. On the other hand, larger values of *HalfLength* also 'waste' data, as described in the next bullet point.

- The FTI indicator is based on filtered and unfiltered prices in a window (called the *channel* in FTI nomenclature) extending back from the current bar a total of *BlockSize* - *HalfLength* bars. The more bars in the channel, the more accurate the measure of follow-through. This inspires one to make *HalfLength* as small as possible, in direct conflict with the prior bullet point. The solution is to make *BlockSize* as large as possible, remembering that excessive lengths will look so far back in history that response to recent changes in market behavior may be masked. This is the universal indicator-definition conflict between wanting to use long lookback windows for stability, while wanting to use short lookback windows for rapid response to current market conditions. Unfortunately, for FTI calculation we have one more monkey wrench thrown into the mix, the need for a filter *HalfLength* that is as large as possible! In many cases, setting the *BlockSize* equal to twice the *HalfLength* is a reasonable choice. The channel length (BlockSize minus HalfLength) certainly should be at least 20 or so. An error will be generated if the channel length is less than 2, an absurdly small (but legal) amount.

- In summary, a reasonable procedure is to choose the *Period* first. Then choose *HalfLength* to be somewhat greater than half of *Period*. Finally, set

BlockSize to twice *HalfLength*, larger if needed to make *BlockSize* - *HalfLength* at least 20, or smaller if that difference is at least 20 and you are worried about looking too far back in history.

- Of interest only to readers familiar with signal processing: You are probably thinking that the channel length is not *BlockSize* - *HalfLength* but rather is *BlockSize* - *2* * *HalfLength*. The reason it works out this way is that Khalsa has a clever method for using just half of the complete filter for the most recent data point, half of the filter plus one for the second-most recent, and so forth. The result is a correct and realizable filter with zero lag for all data points up to the *HalfLength* bar, and slowly deteriorating quality as it approaches the most recent bar. The filter at these recent points still has zero lag, but its frequency response suffers (unavoidably, of course). As long as the channel is reasonably long compared to the *HalfLength*, this error is of little consequence, and the fact that the filter has zero lag is immensely useful.

Lag can be a real problem in market analysis. Lowpass filters such as moving averages are known to lag the price data. Troughs and peaks in the moving average show up later than troughs and peaks in the price. The lag for a simple moving average is the moving average span minus 1 divided by 2 (e.g., 11 day simple MA has a lag of 5). Some applications of smoothing require that the smoothed day be plotted so that its peaks and trough line up with those in the raw price data. The disadvantage however, is that there will be no moving average values for the most recent (*n*-1) /2 days where *n* is the span of the moving average. The lowpass filter used in generating this family of indicators is superior to a moving average but it too has a lag in its basic form. Khalsa developed a clever way to eliminate the lag and thereby obtain a current value for the lowpass smoother. The price is suboptimal performance of the filter (a considerable quantity of undesirable high-frequency components leak through), but it is often a price worth paying.

Computing FTI

The procedure for computing FTI is quite complex and will not be presented ir detail. However, for those interested in the basics, here is a rough overview of the process:

1) For each bar in the channel (the most recent *BlockSize* - *HalfLength* bars) compute the lowpass filtered version of the log of the closing prices. Only the bars in the channel will take part in subsequent calculations. The *HalfLength* oldest bars in the block are ignored from now on. Their only function was to take part in the lowpass filtering.

2) Partition the filtered price moves into up legs and down legs. Examine the distribution of absolute leg lengths and use a rule to set a threshold for distinguishing between legitimate legs and noise legs. Compute the mean length of the legitimate legs.

3) For each bar in the channel, compute the difference between the filtered (smoothed) log price and the actual log price. Examine the distribution of these absolute differences and thereby compute the *channel width*, a measure which defines a pair of boundaries above and below each bar such that these boundaries enclose the majority (though not necessarily all) of the price moves within the channel. An example of such a channel is shown bounding the prices in Figure 11 below.

4) Divide the mean length of the legitimate legs found in Step 2 by the channel width found in Step 3. This is the FTI measure for the current bar. InFigure 11 below, FTI is graphed in green below the bounded market prices. Notice that the FTI value is declining over time. This is telling us that the follow-through of the price trend in question is decreasing.

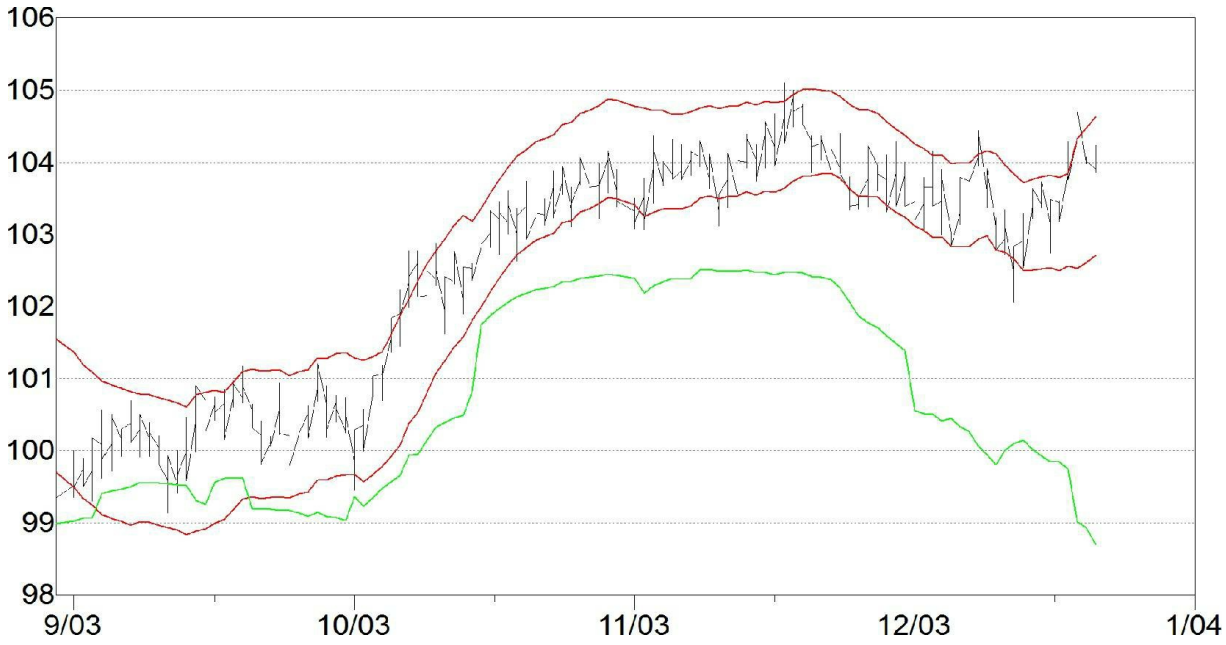


Figure 11: Noise bounds around a market, and FTI

Automated Choice of Filter Period

Khalsa states an important point about his Follow-Through Index, somewhat paraphrased here: *If a trend exists, then the FTI for some filter period will be significantly greater than the FTI for other nearby filter periods.* Two things to note about this statement are:

1) His rule begins *if a trend exists*. Markets do not always trend.

2) The dominant FTI should achieve a significant local peak for some filter period.

What does 'significant' mean here? Khalsa does not address this issue at all, leaving it to a human to examine a table of FTI values for various filter periods and make the decision. Later, we will present some *TSSB* FTI indicators that automatically choose the 'optimal' filter period. They do this in a very naive manner: the user specifies a range of periods to try, and the program picks the period that has the largest FTI.

Note that the chosen period is highly data dependent, with the result that as we move from bar to bar through the historical dataset, the chosen period can vary widely, often jumping by a large amount even from one bar to the next. Thus, if this automated choice is used, it is probably best for the user to specify a relatively narrow range of periods to try in order to avoid dramatic changes in indicator behavior from bar to bar.

What about the decision of whether there even is a trend? Actually, experience

indicates that the value of FTI obtained is itself a decent indicator of the existence of a trend, regardless of how 'significantly' it stands out from its neighbors. The larger the FTI, the stronger is the trend.

Trends Within Trends

The Follow-Through Index is the most important and useful aspect of Khalsa's work. However, he does present a second measure of market state that is closely related to FTI. This is the *Channel Width Ratio*.

The idea behind the channel width ratio is that we find two different filter periods, each of which produces a locally maximum, large FTI. In other words, we find the two trends with the highest FTI values within the span searched. The FTI having the larger period is called the *major trend* and the one having the smaller period is called the *minor trend*.

To be specific, *TSSB* finds every period within a user-specified range whose FTI is greater than or equal to the FTI of its two neighbors (periods one greater and one less). These are the locally maximum FTIs. The two largest such FTIs are found. The one having the greater period defines the major trend within the channel, while the one having the lesser period defines the minor trend within the channel. The actual values of the FTIs are ignored.

For each of these two periods, we compute the channel width as described here. The ratio of the width of the minor trend channel to that of the major trend channel is called the *Channel Width Ratio*, and it can be a useful indicator of the current market state. To see this double channel in action, look at Figure 12 below. The (usually) wider channel is for the major trend.

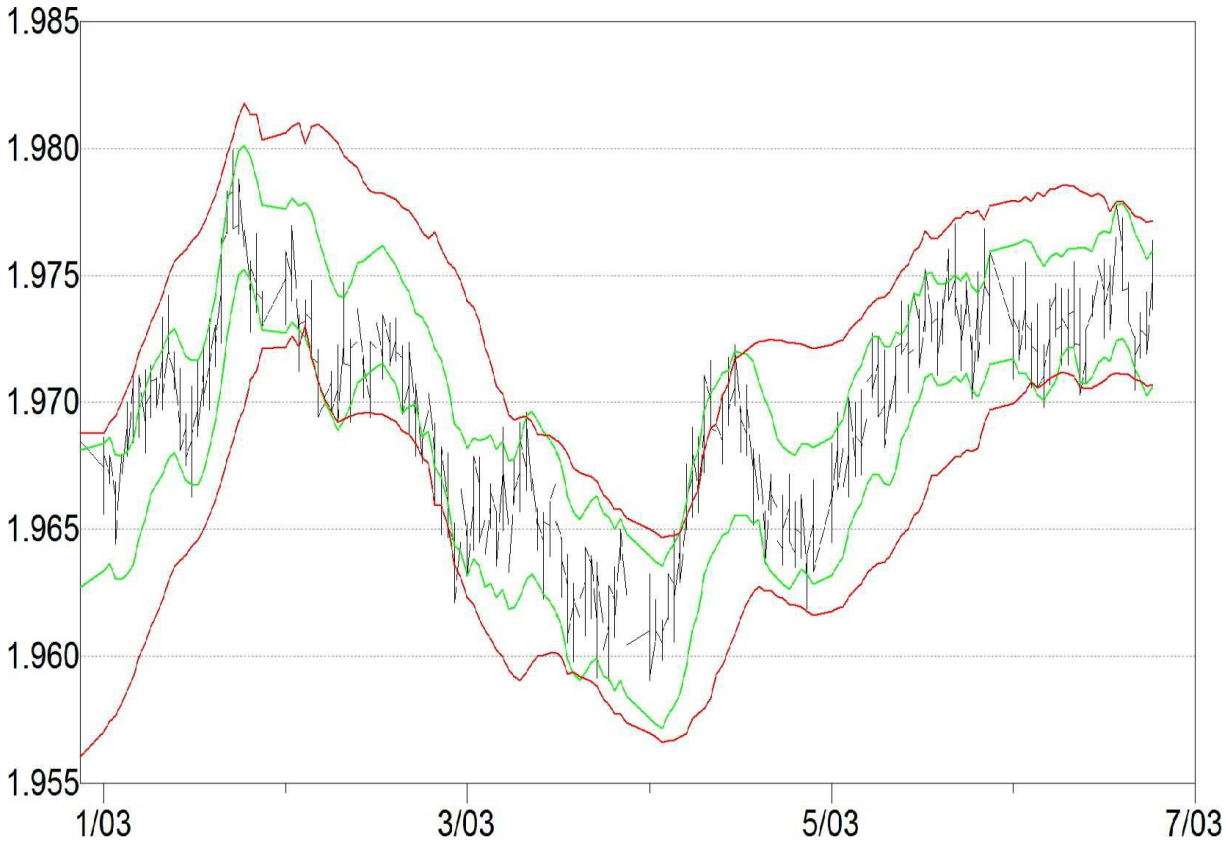


Figure 12:FTI Major and Minor Trend channels

FTI Indicators Available in *TSSB*

We now list the FTI indicators that exist in the *TSSB* library. These all begin with 'FTI' for clarity, and they all require the parameters that have been discussed in the prior sections. Recall that in most situations, the easiest approach to setting parameters is to decide on the desired *Period* (or pair of periods for those that require two periods or a range) first. Then choose the filter *HalfLength* to be at least half of the *Period* (at least half of the larger period, if two periods are specified). Finally, decide on how many bars you want in the channel and add this quantity to the *HalfLength* in order to get the *BlockSize*.

FTI LOWPASS BlockSize HalfLength Period

Gavinda Khalsa's zero-lag lowpass filter is applied to the log of closing prices. In other words, this produces the filtered (smoothed) log prices that form the basis of subsequent FTI computations. This can make for an interesting and informative plot. One possible application for this quantity would be as part of an oscillator crossover system as a substitute for a moving average. Note, however, that the price paid for zero lag is poor frequency response for recent bars. Because this

indicator is the filtered value at the current bar, this deterioration in frequency response will be at its maximum. Therefore, this quantity is probably inferior to an ordinary moving average for use in crossover systems. It's zero-lag property is not important enough to overshadow its poor frequency response. In all likelihood, the only legitimate use of this indicator is for visual examination of its plot. For example:

```
FTILOW: FTI LOWPASS 6 4 6
```

The preceding definition applies Khalsa's zero-lag lowpass filter to the log close of each bar. The *Period* is 6 bars. The filter *HalfLength* is 4, which is legal because it is at least half of the *Period*. The *Blocksize* is 6, which is the minimum legal amount. Recall from the rules here that the *BlockSize* must be at least 2 more than the *HalfLength*. There is no point in making the channel any longer, because this indicator returns only the current value. Earlier values in the channel would be used for FTI calculations, but this indicator does not do any! It just returns the filtered log closes, so all channel entries except the most recent are ignored.

FTI MINOR LOWPASS BlockSize HalfLength LowPeriod HighPeriod

This is similar to the FIT LOWPASS indicator just described in that it returns the zero-lag lowpass filtered value of the log prices. The only difference is in how the period of the filter is chosen. In the prior indicator, FTI LOWPASS, the user explicitly specifies the filter period. In this indicator, FTI MINOR LOWPASS, the user instead specifies an inclusive range of possible periods, *LowPeriod* through *HighPeriod*. The program then uses the algorithm described here to find the period that characterizes the *minor trend*. This is the period chosen for the lowpass filter. In all likelihood, this quantity is useful for display and diagnostic purposes only. As explained earlier, the automated selection of the *minor period* and *major period* is highly unstable and of limited utility. For example:

```
FTIMINLP: FTI MINOR LOWPASS 26 6 5 10
```

The preceding definition applies lowpass filters having periods ranging from 5 through 10 bars. The *HalfLength* of each filter is 6. This is legal because it exceeds half the period of the maximum (half of 10 is 5, and 6 equals or exceeds 5). Because FTI values will be computed, we make the channel contain 20 points by adding 20 to 6, giving us a *BlockSize* of 26. In accordance with the algorithm described here, the program computes FTI for every period from 5 through 10. It chooses the two highest FTIs that are local maxima and chooses the smaller of the two associated periods (the minor trend). This period is used for the lowpass filter whose output defines this indicator.

FTI MAJOR LOWPASS BlockSize HalfLength LowPeriod HighPeriod

This is identical to FTI MINOR LOWPASS except that the major trend's period selected to define the lowpass filter whose output defines this indicator. In all likelihood, this quantity is useful for display and diagnostic purposes only.

FTI FTI BlockSize HalfLength Period

This returns the FTI value for the specified parameters. This may be the single mos useful FTI indicator. It allows the user to bypass the unstable and often unpredictable automated period selection algorithm and directly specify a period that is tailored to the application. For example:

```
FTI10: FTI FTI 36 6 10
```

The preceding definition computes FTI for a filter period of 10 bars. The *HalfLength* of the lowpass filter is 6, which satisfies the requirement of being at least half of the *Period*. A *BlockSize* of 36 provides a *channel length* of 36-6=30 bars, which is decently long.

FTI LARGEST FTI BlockSize HalfLength LowPeriod HighPeriod

This returns the value of the largest FTI within the specified range of periods (inclusive). Note that this does not use the automated period selection algorithm. That algorithm finds *two* local maxima and defines the major and minor trends according to the size of the associated periods. FTI LARGEST FTI finds the *single* global maximum. If the user feels uncomfortable choosing a period, then FTI LARGEST FTI may be a useful alternative to FTI FTI because it gives the progra the freedom to search a range of periods of the largest FTI, but it does not have suffer the instability due to choosing a *pair* of local maxima. For example:

```
FTIBIG: FTI LARGEST FTI 36 6 5 10
```

The preceding definition computes FTI for periods from 5 through 10 bars. The *HalfLength* of the lowpass filter is 6, which satisfies the requirement of being at least half of the *Period* for all trial periods. A *BlockSize* of 36 provides a *channel length* of 36-6=30 bars, which is decently long. The largest FTI found among these trial periods is returned as the computed indicator.

FTI MINOR FTI BlockSize HalfLength LowPeriod HighPeriod

The user specifies an inclusive range of possible periods, *LowPeriod* through *HighPeriod*. The program uses the algorithm described here to find the period that characterizes the *minor trend*. This returns the FTI for the minor (smaller period) filter. For example:

```
FTIMIN: FTI MINOR FTI 36 6 5 10
```

The preceding definition computes FTIs using lowpass filters having periods ranging from 5 through 10 bars. The *HalfLength* of each filter is 6. This is legal because it exceeds half the period of the maximum (half of 10 is 5, and 6 equals or exceeds 5). We make the channel contain 30 points by adding 30 to 6, giving us a *BlockSize* of 36. In accordance with the algorithm described here, the program computes FTI for every period from 5 through 10. It chooses the two highest FTI that are local maxima and chooses the smaller of the two associated periods (the minor trend). This period is used to compute FTI, which is this indicator.

FTI MAJOR FTI BlockSize HalfLength LowPeriod HighPeriod

This is identical to FTI MINOR FTI as described above, except that it uses th major trend, not the minor.

FTI LARGEST PERIOD BlockSize HalfLength LowPeriod HighPeriod

This returns the period corresponding to the largest FTI within the specified range of periods (inclusive). Note that this does not use the automated period selection algorithm. That algorithm finds *two* local maxima and defines the major and minor trends according to the size of the associated periods. FTI LARGEST PERIO finds the *single* global maximum FTI and returns its filter period as the value of this indicator. For example:

```
FTIBIGPER: FTI LARGEST PERIOD 36 6 5 10
```

The preceding definition computes FTI for periods from 5 through 10 bars. The *HalfLength* of the lowpass filter is 6, which satisfies the requirement of being at least half of the *Period* for all trial periods. A *BlockSize* of 36 provides a *channel length* of 36-6=30 bars, which is long enough to get stable FTI values. The period corresponding to the largest FTI found among these trial periods is returned as the computed indicator. It is unlikely that this would ever be of any use as an indicator for a prediction model. However, series and histogram plots of this variable can

sometimes be informative.

FTI MINOR PERIOD BlockSize HalfLength LowPeriod HighPeriod

The user specifies an inclusive range of possible periods, *LowPeriod* through *HighPeriod*. The program uses the algorithm described here to find the period that characterizes the *minor trend*. This period is the value of the indicator. For example:

```
FTIMINPER: FTI MINOR PERIOD 36 6 5 10
```

The preceding definition computes FTIs using lowpass filters having periods ranging from 5 through 10 bars. The *HalfLength* of each filter is 6. This is legal because it exceeds half the period of the maximum (half of 10 is 5, and 6 equals or exceeds 5). We make the channel contain 30 points by adding 30 to 6, giving us a *BlockSize* of 36. In accordance with the algorithm described here, the program examines FTI for each period. It chooses the two highest FTIs that are local maxima and chooses the smaller of the two associated periods (the minor trend). This smaller period is the indicator. It is unlikely that this would ever be of any use as an indicator for a prediction model. However, series and histogram plots of this variable can sometimes be informative.

FTI MAJOR PERIOD BlockSize HalfLength LowPeriod HighPeriod

This is identical to FTI MINOR PERIOD, except that the major period is returned.

FTI CRAT BlockSize HalfLength LowPeriod HighPeriod

The minor/major *channel width ratio* for the specified pair of periods is returned. (See here for a description of the *channel width ratio*.) Note that unlike the indicators that contain the keywords MAJOR or MINOR, this indicator does not use the major/minor definition algorithm described here. Rather, the user explicitly specifies the periods for the two channels, thus avoiding the instability of the automated algorithm. *LowPeriod* is the filter period for the minor channel, and *HighPeriod* is that for the major channel. This indicator nicely complements the FTI FTI variable, because the exact periods can be provided to best fit the application. In most cases, if the user is employing both FTI FTI and FTI CRAT as indicators, the same low and high periods would be used in both definitions. For example:

```
CRAT_5_10: FTI CRAT 36 6 5 10
```

The preceding definition computes the channel width for a filter period of 5 bars as well as for a period of 10 bars. The *HalfLength* of the lowpass filter is 6, which satisfies the requirement of being at least half of the longest *Period*. A *BlockSize* of 36 provides a *channel length* of 36-6=30 bars, which is decently long. The channel width at a period of 5 bars is divided by the channel width at a period of 10 bars in order to compute this indicator.

FTI MINOR BEST CRAT BlockSize HalfLength LowPeriod HighPeriod

The minor/major *channel width ratio* is returned. (See here for a description of the *channel width ratio*.) The major period is fixed at *HighPeriod*. All periods from *LowPeriod* up to (but of course not including) *HighPeriod* are searched for local maxima in FTI. The largest local maximum in this range is used for the minor period. This is what Khalsa does in the FTI paper. He fixes the major period at 65 and finds the best associated minor period. The keywords MINOR BEST are used in this definition because we are searching for the best minor trend, with the major trend's filter period fixed by the user. For example:

```
MINBESTCRAT: FTI MINOR BEST CRAT 120 35 5 65
```

The preceding definition computes the channel width using a filter period of 65 bars. The *HalfLength* of this filter is 35, which is legal because it exceeds half of 65. The channel length is a generous 120-35=85 bars. Then it tries all smaller periods, down to and including 5. It chooses the one having the largest local maximum FTI and computes its channel width. This quantity is divided by the channel width at a period of 65 in order to compute the indicator.

FTI MAJOR BEST CRAT BlockSize HalfLength LowPeriod HighPeriod

This is identical to the FTI MINOR BEST CRAT described above, except that the minor period is fixed at *LowPeriod*. Periods up to and including *HighPeriod* are tried, and the one having the largest local maximum is used for the major trend period.

FTI BOTH BEST CRAT BlockSize HalfLength LowPeriod HighPeriod

The minor/major *channel width ratio* is returned. (See here for a description of the *channel width ratio*.) The program uses the algorithm described here to find the periods that characterize the minor trend as well as the major trend. Compare this with the two prior indicators. FTI MINOR BEST lets the user fix the major tren period, and it finds the best minor trend period. FTI MAJOR BEST lets the user fi the minor trend period, and it finds the best major trend period. But this indicator, FTI BOTH BEST CRAT, uses the automatic algorithm to find both periods restricting the search to the inclusive range *LowPeriod* through *HighPeriod*. For example:

```
FTIBOTHCRAT: FTI BOTH BEST CRAT 120 35 5 65
```

The preceding definition computes FTIs using lowpass filters having periods ranging from 5 through 65 bars. The *HalfLength* of each filter is 35, which is legal because it exceeds half of 65. The channel length is 120-35=85 bars. In accordance with the algorithm described here, the program examines FTI for each period. I chooses the two highest FTIs that are local maxima and lets the smaller of the two associated periods define the minor trend. The larger period defines the major trend. The minor/major *channel width ratio* is returned as the indicator.