TSSB

(*T*rading *S*ystem *S*ynthesis and *B*oosting)

User's Manual 1.17

Table of Contents

General Operation

TSSB is a program that reads historical data from multiple markets, computes user-definable predictors and performance variables, and tests the ability of models to obtain useful predictions. The following features are noteworthy:

- Most operations are automated by means of ASCII text files that define variables, specify markets and models, and so forth. This minimizes tedious interactions with complex menus, making unattended operation of complex operations feasible. These self-documenting control files provide a rigorous description of test conditions and ensure exact repeatability of tests.

- Every stage of operation occurs within this one program. There is no need to write data files for import into other programs. Every step, from computation of predictor variables to final statistical analysis, is self contained.

- The user can employ the menu system to display a variety of useful charts and graphs, including various statistics, time series plots, and histograms. These capabilities facilitate visual inspection of variables and their relationships.

- Numerous statistical tests are provided to assess the degree to which variables are in conformity across markets. This property is vital if data is to be pooled for the purpose of creating trading systems that are applicable to many markets.

- The program also provides a wide variety of tests for stationarity of variables. If the goal is to generate many short-term trades across a long extent of time, reasonable stationarity is valuable, or perhaps even mandatory.

Files

Several files and families of files are required for operation. The use of control files rather than menu selection is important when operating in a strict environment when precise documentation and exact repeatability are necessary. The files are now described.

Market histories

Every market that will participate in the study must have its own history file. These files must all reside in the same directory. Each file should be an ASCII text file and have the extension TXT. The first part of the file name should be descriptive of the market and may be at most six characters. The market symbol is ideal. The name must contain only letters, numbers, and the underscore character (_). Notably, a period is not allowed, since a period is also used as part of the complete file name.

By default, the following file format is used:

YYYYMMDD Open High Low Close Volume (Additional data is ignored)

A space, comma, or tab may be used to separate fields. For example, two lines of a market history file may look like this:

```
19840104 2.69 2.78 2.68 2.78 28448 ABT
19840105 2.78 2.82 2.78 2.81 40416 ABT
```

These two lines are for January 4 and 5, 1984. The symbol ABT at the end is ignored.

Tick data is also legal. In this case each record would have only two fields: the date and the price. This format is also useful for reading non-price data as if it were a market, such as the VIX sentiment indicator. For example:

```
19840104 2.69
19840105 2.78
```

Intraday market data has the time (in 24-hour format) after the date, separated from it by a space, comma, or tab. The time may be HHMM or HHMMSS:

```
19840104 1603 2.69 2.78 2.68 2.78 28448 ABT
```

Several commands are available for altering the default date format. These commands, if used, must appear before the READ MARKET HISTORIES command, and they will apply to all market history files. These are:

MARKET DATE FORMAT YYYYMMDD ;

> *This command is never needed because this is the default date format. However, it is included for the sake of completeness. Some users may wish to use this command to document the date format.*

MARKET DATE FORMAT YYMMDD ;

> *This specifies that dates appear as YYMMDD. The year must be 1920-2019 inclusive.*

MARKET DATE FORMAT M_D_YYYY ;

> *This specifies that the date will appear as Month/Day/YYYY where the month and the day can be one or two digits. The intervening slashes must be included. For example: 4/29/2004.*

MARKET DATE FORMAT AUTOMATIC ;

> *The prior commands require that the same format be used for all market files. It may be that some market files have different date formats. Using the AUTOMATIC format causes the program to examine the date in each file and determine which format is used. At this time only two formats are allowed: YYYYMMDD and M_D_YYYY. Other formats may be added later.*

Removing Market Prices with 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 (Page 178). 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 HISTORIES command. This command is required if a TRAIN PERMUTED command appears in the script file, even if the market(s) read do not contain any zero-volume records.

Market List

The market list file is a simple ASCII text file that names the markets to be included, one per line. The entries in this file must be the root name of the market files. Thus, this is actually a list of files names, not markets. However, if you follow the convention of naming the market file by the symbol, everything will be more transparent. For example, the following market list file says that five markets will be included in the study:

```
ABT
C
IBM
MER
T
```

This assumes that we have market history files named ABT.TXT, C.TXT, and so forth. The great advantage of having a market list file separate from other control files is that we can quickly change the included markets without having to change any other aspect of the experiment. The market list file may be referenced by either a READ MARKET LIST command if you are computing variables from raw market data, or a RETAIN MARKET LIST command if you wish to retain only a subset of the markets in a database file.

Variable List

The Variable List File is an ASCII text file that defines all variables that may be used in the experiment. These variables need not be used in subsequent models. The user may include some variables simply for the purpose of studying their properties via statistics or graphs. The file contains one line per variable. Any text after a semicolon is ignored to facilitate comments. Details concerning the Variable List File can be found on Page 25.

The Script File

Most operation of the program is controlled by an ASCII text file called a *Script File*. By default, this file will have the extension SCR, although the user is free to employ any extension. Using SCR rather than TXT is useful for easily identifying script files from amongst a variety of .TXT files for variable and market definitions.

Initialization Commands

The following three initialization commands, which always end in a semicolon, will typically appear in a script file and in this order:

RETAIN YEARS FirstYear THROUGH LastYear ;

This optional command directs that only the years *FirstYear* through *LastYear* (as YYYY) will be kept when market histories are read. If this command is used, it may appear before the *READ MARKET HISTORIES* command to limit the reading of histories, or before the READ VARIABLE LIST command to limit the database. It may also precede a READ DATABASE command, in which case it limits the cases read from the database..

RETAIN MOD Divisor = Remainder ;

This optional command keeps every *Divisor*'th date, with an offset of *Remainder*.
For example, *RETAIN MOD 4 = 0* keeps every fourth date, those that are evenly divisible by four, while *RETAIN MOD 4 = 1* also keeps every fourth date, but the resulting dataset is offset one day later than the prior example. This command must precede the READ VARIABLE LIST or READ DATABASE command.

READ MARKET LIST "FileName" ;

The named market list file, described on Page 4, is read and processed. This command does not cause any market histories to be read. It simply reads the list of market files that will be read by the next command. This command would be used when you will be computing variables from raw market data.

RETAIN MARKET LIST "FileName" ;

This reads a market list file (described on Page 4). The next command would be READ DATABASE (Page 8). Only the markets named in the RETAIN MARKET LIST file will be kept from the database. This command will have no effect if you don't read a database.

READ MARKET HISTORIES "FullPathNameOfAnyHistoryFile" ;

This command reads all of the market history files (see Page 2) that were named in a previously read market list (see the command above). This command must name *any* of the market history files. It does not matter which. In fact, the named file does not even have to be a member of the market list already read. The only purpose of naming the file is to provide the program with the directory in which the file is found. It is highly recommended that the full path name of the history directory be specified. If omitted, the default directory supplied by Windows is likely to be wrong.

INDEX IS MarketName ;

Sometimes the user will have a 'market' that is really an index (such as the S&P 500) and will want to use this index as a component of computing variables. For example, the MINUS INDEX keyword that can appear as part of some variable definitions compares the value of the variable in the target market with its value in the index. The INDEX IS *MarketName* script command names the index market. If no variable references an index, this command may be omitted.

INDEX Integer IS MarketName ;

This is a generalization of the INDEX command, allowing more than one index market. You may specify any integer from 1 through 16. Variable definitions may later reference a specific index via INDEXn where n is the number corresponding to that used here. Note that the *INDEX Integer IS MarketName* command requires a space between the INDEX keyword and the number, while later variable references require that no space separate them. Also note that INDEX 1 is synonymous with INDEX.

READ VARIABLE LIST   "FileName" ;

This command reads the file that contains all variable definitions, as described on Page 25. Market histories must be read before the variable list file is read.

VariableName IS PROFIT ;

This optional command declares that the named variable is interpretable as a profit. If used, this command must come after any READ VARIABLE LIST or READ DATABASE commands. You would use this command if you are reading a database produced by another program, hence lacking a FAMILY file, and one or more of the variables in the file will be used as a profit variable in a model. You would also need it if you were deliberately subverting the internal variable computations and you want to declare as a profit a variable that the program considers to not be a profit. I see no rational reason you would ever want to do this.

Intraday Trading Commands

By default, all bars are day bars. Market and database file contain only the date. Intraday files can be processed by means of the following commands:

INTRADAY BY MINUTE ;

This command, which must precede any command involved in reading markets or a database, specifies that the time of each bar, as HHMM in 24-hour time, follows the date (and separated by a space).

INTRADAY BY SECOND ;

This is as above, except that the time is specified as HHMMSS.

Saving and Restoring the Database

Computing a large number of variables for a large number of markets can be enormously time consuming. Therefore, the program includes the ability to save a database in a portable ASCII format. This database can then be read by other statistical packages, and it can be read by this program to rapidly restore the database. These commands are:

WRITE DATABASE "DataName" "OptionalFamilyName";

The entire database is saved to the *DataName* file. Like all file names in TSSB, spaces are not allowed. The first line contains all of the variable names, beginning with the date and market. Successive lines contain cases, one per case. The first field is the date as YYYYMMDD. If the file is intraday, the time as HHMM or HHMMSS follows. The next field is the name of the market. The remaining fields are the values of the variables. One or more spaces are used to delimit fields.

If a second file name is provided, this command also writes a file called the family descriptor. This file must be read by the READ DATABASE command if a family specifier (Page 115) is used in any model definition.

VARIABLE VarName IS TEXT ;

If a database will be read (see next) and if any variables are text instead of numeric, the name of each text variable must be specified with this command. This command must precede the READ DATABASE command.

READ DATABASE "FileName" "OptionalFamilyName" ;

The named file is read. Its format must be as described above in *WRITE DATABASE*, except that spaces, tabs, or commas may be used as delimiters. They may be freely mixed; consistency is not required.

READ UNORDERED DATABASE "FileName" "OptionalFamilyName" ;

This is identical to the preceding command except that the database need not be in chronological order. The ordinary READ DATABASE command assumes chronological order and verifies it. It is faster to execute than the READ UNORDERED DATABASE command.

If a second file name is provided, this command also reads a file called the family descriptor. This file must be read if a family specifier (Page 115) is used in any model definition.

If you intend to read a database, the READ DATABASE command (preceded by any IS TEXT commands) must be first in the script file, unless you also have a RETAIN MARKET LIST command. In this case, the RETAIN MARKET LIST must be first, and READ DATABASE follow.

APPEND DATABASE "FileName" ;

This is identical to READ DATABASE except that it is used to append additional variables to an existing database.

Here is a line from a typical database file. This line says that at 1405 (2:05 PM) on April 21, 1971, in the SP market, we have two variables. These variables have the values 4.7 and -2.3.

```
19710421  1405  SP  4.7 -2.3
```

CLONE TO DATABASE "FileName" ;

This is similar to APPEND DATABASE in that it appends one or more new variables to an existing database. Like the READ DATABASE and APPEND DATABASE commands, the file to be read must contain a header that identifies the variable(s) to be appended. However, unlike those commands, neither the header nor the records in the file specify a market. Instead, the value(s) at each date are cloned onto every market in the database for that date.

Filtering a Haugen Alpha File

A Haugen Predicted Alpha file may be read and used to filter the database to include only select records. The format of the command is as follows:

READ HAUGEN FILE "FileName" LONG/SHORT FractionKept ;
READ HAUGEN FILE "FileName" ALL ;

The named Haugen file is read. *FractionKept* must be between zero and one, and is typically around 0.1 to 0.2 or so. If the type is LONG, then this fraction of the largest predicted alphas is kept. If the type is SHORT, then this fraction of the smallest predicted alphas is kept. If *ALL* is specified, all records are kept.

The order of commands in the script file is crucial. It must be:

READ MARKET LIST...
INDEX IS...   (if used)
READ HAUGEN FILE...
READ MARKET HISTORIES...
CLEAN RAW DATA...   (if used)
READ VARIABLE LIST...

More information on trade filtering can be found on Page 140.

Reading an HMB file

A file from the HMB study can be read with the following command:

READ HMB "FileName" ;

This file should contain either all long or all short trades. Mixing positions will give unsatisfactory results.

Statistical Test Commands

This section describes statistical tests that can be ordered from lines in the script file.

DESCRIBE VariableName   [IN MarketName] ;

A wide variety of basic statistics (mean, variance, range, et cetera) are computed and printed. If a market is specified, computations are limited to cases from that market. Otherwise all markets are pooled into a single database. This command can also be accessed from the menu.

MARKET SCAN [MarketName] ;

One or all markets are scanned for unusual price gaps. For each bar, it computes the greater of 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 it as a percent.

If only one market is scanned, the largest 20 gaps, 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 from the menu.

OUTLIER SCAN   [FOR VariableName] ;

One or more variables (all variables in the database if none specified) are scanned for outliers and poor entropy. The following information is printed to the AUDIT.LOG and REPORT.LOG files:

      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 to the interquartile range (large values 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 entropy.

CROSS MARKET AD   [FOR VariableName] ;

A series of Anderson-Darling tests for equality of distribution are performed. 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. If no variable is specified, the test is repeated for every variable present.

If multiple markets are pooled to form a common database, it is vital that all predictor and predicted variables have distributions that are similar across all markets. This test, which is especially sensitive to discrepancies in the tails, is a reasonable way to evaluate this conformity.

P-values are printed for testing the null hypothesis that the distribution of a variable in a given market is equal to the pooled distribution. However, especially if numerous cases are present (the usual situation) these p-values can be excessively sensitive to differences in the distributions. A tiny p-value does not necessarily imply that discrepancies will be problematic. The best use of this test is revealing the *worst* performers. These variables/markets should be subjected to visual examination by histograms or other tests.

The number of lines output in the audit log is proportional to the product of the number of variables and the number of markets. This outputs can become large quickly. The report log contains only summary information.

CROSS MARKET IQ   [FOR VariableName] ;

This is similar to the Anderson-Darling test described above. However, the interquartile-range-overlap statistic that it computes is not sensitive to the number of cases, which is a problem with the Anderson-Darling statistic. This roughly measures the degree to which the interquartile range of the 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.

Note that this test reveals to a considerable degree a necessary but not sufficient condition. A market that scores high on this test has an interquartile range that is very similar to that of the pooled distribution. However, this test reveals nothing about behavior outside the interquartile range. And of course, a market that scores very low on this test is in big trouble.   The main advantage of the IQ test is that its results are easily interpretable.

CROSS MARKET KL   [FOR VariableName] ;

This is similar to the Anderson-Darling test described above. However, the Kullback-Liebler statistic that it computes is not sensitive to the number of cases, which is a problem with the Anderson-Darling statistic. The printed value ranges from zero, meaning severe misconformity, to one, meaning perfect conformity. This test divides the range of the pooled data into five bins, so unlike the IQ test it takes tail behavior into account. Unfortunately, the actual values do not have simple interpretability like the IQ values. They should be used only to rank conformity.

STATIONARITY [OF VariableName] IN MarketName ;

Tests for statistical stationarity within a given market are performed. If no variable is specified, the tests are repeated for all variables in the Variable List File. At this time there is no way to select individual tests or parameters for tests from within the script file. All tests will be performed using arbitrary defaults. Because a large number of tests are available, and the specified parameters can have a great impact on the nature of the tests, it is strongly suggested that stationarity tests be performed from the menu system, where the user will be given full control over all aspects of the tests. Details concerning the available tests are discussed on Page 63.

Model, Committee, and Oracle Commands

All models must be defined in a script file. No menu commands exist for defining models because it would be too complex for practical use.

More than one model may be defined. If multiple model definitions appear, they will all be handled simultaneously.

A model definition includes the following information:

- A unique name by which the user will identify the model

- A list of all candidates for input variables

- The output (predicted) variable

- Information about how the model is to be trained. This includes things like the maximum number of inputs allowed, the criterion to be optimized, and so forth.

Details about models can be found on Page 115. Committees are discussed on Page 136, and oracles are discussed on Page 137.

By default, only one pair of classification thresholds, called the OUTER UPPER and OUTER LOWER, are computed and tested. If the command USE INNER THRESHOLD appears in the script file, a second, inner pair of thresholds is also computed. These inner thresholds attempt to increase the number of trades while sacrificing as little quality as possible. This command, if used, must precede all MODEL, COMMITTEE, and ORACLE definitions.

Find Groups Command

This command instructs the program to find sets of predictor variables that are reasonably independent. It cannot be invoked from the menu system, only via this command in the script file. A linear model is found that optimally predicts the target variable. Then, the remaining predictor candidates are tested for predictability from the prediction set just found. Any candidates that have significant predictability are considered to be redundant and eliminated from future contention. This process is repeated a specified number of times to produce groups of predictors that are reasonably independent of one another. The syntax of this command is as follows:

FIND GROUPS [Specifications] ;

The following specifications are available:

INPUT = [Var1 Var2 ...]
> *(Mandatory) This lists the input variables that are candidates for inclusion. Range and family options are also available. See Pages 115 and 132 for details.*

OUTPUT = VarName
> *(Mandatory) This names the variable that is to be predicted.*

PROFIT = VarName
> *(Optional) If PROFIT is not specified, the OUTPUT variable, which is the target being predicted, is also used for computing profit factor performance measures. However, sometimes the target variable is not interpretable as a profit. For example, a future return that has been cross-sectionally normalized may have questionable value in computing profits. A filter residual (Page 140) is certainly not a measure of profit. In such cases, the user can specify a variable that will be used for computing profit-based performance measures. These measures will be used for profit-factor-based stepwise predictor selection as well as in the display of all profit-factor performance measures.*

MAX STEPWISE = Number
> *(Mandatory) This is the maximum number of predictor variables that may be included in each group. Stepwise inclusion will be terminated early if inclusion of a new variable results in deterioration of performance.*

STEPWISE RETENTION = Number
> *(Optional) By default, this value is one, meaning that ordinary forward stepwise selection is performed. If this is greater than one, this many of the best variable sets are retained at each step. For example, if this is five, the best five single predictors are found. Then, for each of these five, the next best predictor, conditional on the first, is found. The best five pairs are retained, and a third variable is tried for each*

pair, and so forth. This tremendously slows training, but by testing more combinations of variables, we increase the chance of finding an effective set.

MAX GROUPS = Number

(Mandatory) This is the maximum number of groups that can be found. This maximum may not always be reached.

CRITERION = RSQUARE

(Some criterion must be specified.) The target criterion optimized is R-Square, the fraction of the predicted variable's output that is explained by the model. Note that R-square will be negative in the unusual situation that the model's predictions are, on average, worse than guessing. If this criterion is optimized, the threshold used for computing threshold-based performance criteria is arbitrarily set at zero.

CRITERION = LONG PROFIT FACTOR

(Some criterion must be specified.) The target criterion optimized is an analog of the common profit factor, under the assumption that only long (or neutral) positions are taken. A threshold is simultaneously optimized. Only cases whose predicted value equals or exceeds the threshold enter into the calculation. Sum all such cases for which the true value of the predicted variable is positive, and also sum all such cases for which the true value of the predicted variable is negative. Divide the former by the latter and flip its sign to make it positive. This is the profit factor criterion. If the predicted or 'PROFIT' variable is actual wins and losses, this criterion will be the traditional profit factor for a system that takes a long position whenever the predicted value exceeds the optimized threshold.

CRITERION = SHORT PROFIT FACTOR

(Some criterion must be specified.) This is identical to LONG PROFIT FACTOR above except that only short (and neutral) positions are taken. A threshold is optimized, and only cases whose predicted values are less than or equal to the threshold enter into the calculation. Since this criterion assumes short trades only, a positive value of the predicted variable implies a loss, and conversely.

CRITERION = PROFIT FACTOR

(Some criterion must be specified.) This combines the LONG and SHORT profit factor criteria above. Two thresholds are simultaneously optimized. If a case's predicted value equals or exceeds the upper threshold, it is assumed that a long position is taken, meaning that a positive value in the predicted variable implies a win. If a case's predicted value is less than or equal to the lower threshold, it is assumed that a short position is taken, meaning that a positive value in the predicted variable implies a loss.

CRITERION = ROC AREA

(Some criterion must be specified.) The criterion optimized is the area under the profit/loss ROC curve. This criterion considers the distribution of actual profits and losses relative to predictions made by the model. A random model will have a value of about 0.5, a perfect model will have a ROC area of 1.0, and a model that is exactly incorrect (the opposite of perfect) will have a value of 0.0.

MIN CRITERION CASES = Integer

(At least one minimum must be specified) If the criterion is one that requires an optimized threshold (such as PROFIT FACTOR), this specifies the minimum number of cases that must meet the threshold. If no minimum were specified, the optimizer might choose a threshold so extreme that only a very few cases meet it, resulting in statistical instability.

MIN CRITERION FRACTION = RealNumber

(At least one minimum must be specified) If the criterion is one that requires an optimized threshold (such as PROFIT FACTOR), this specifies the minimum fraction (0-1) of training cases that must meet the threshold. If no minimum were specified, the optimizer might choose a threshold so extreme that only a very few cases meet it, resulting in statistical instability.

RESTRAIN PREDICTED

(Optional) This causes the true values of the predicted variable to be compressed at the extreme values before training. This is almost always very useful, perhaps even mandatory. If the predicted variable contains outliers, many models will expend considerable effort accommodating these extreme values, at the expense of the more common cases. When this option is invoked, most results will be printed twice, once for the restrained values, and again for the original values.

GROUP RSQUARE CUTOFF = RealNumber

This number in the range 0-1 specifies the threshold for removing candidates from future consideration. If any candidate can be predicted from any prior group with R-square at least equal to this value, the candidate is eliminated due to redundancy.

Both AUDIT.LOG and REPORT.LOG contain detailed descriptions of each group as it is defined. In addition, AUDIT.LOG lists the R-square of each candidate with each prior group. Since this listing can be extensive, it is not included in REPORT.LOG.

After all groups are found, a summary of the chosen variables is printed. This summary also considers the best predictors and predictor sets examined as part of the stepwise selection procedure. The printout shows which variables were selected most often. The summary contains not only values for individual predictor candidates, but it is also broken down by family, lookback, index usage, historical normalization, and cross-sectional normalization. Columns list the observed percentage selection rate, the expected rate if all variables were equally valuable, the ratio of observed to expected, and a rough estimate of the probability of this extreme occurring if all variables were equally effective. The list is sorted in decreasing order of the selection ratio, because large values of this ratio imply that the variable was selected more often than would normally be expected.

When the user is employing the *FIND GROUPS* command to judge the relative importance of predictors, it is suggested that *MAX GROUPS* be set to the number of models that will later be used in committee generation. Setting the *MAX GROUPS* parameter to one will judge the candidates as if only a single model were being used, which is appropriate if that is to be the case. However, if the user will later define multiple models having mutually exclusive predictors, it is probably best to generate the same number of groups here. Of course, if the goal is to winnow down the candidates, eliminating those that are almost certainly worthless, it is probably best to set MAX GROUPS to one. Probably.

The most accurate estimation of importance will be obtained if *STEPWISE RETENTION* is set to a small fraction of the total number of candidates. Small values of this parameter will lead to identification of relatively few of the very best predictors, while large values will produce a larger, more comprehensive list of good predictors, less focused on the very best. Some experimentation may be needed to find the best compromise. In the extreme case of setting *STEPWISE RETENTION* to an effectively infinite value, all candidates will be judged to be equally important, an obviously useless judgement.

Regression Classes

It is too much to expect that a single prediction model will be valid across all markets in our universe. It may be that one model performs well for interest-rate issues, while another does well for consumer commodities. We can group our universe of markets into subgroups that have similar relationships between predictors and the target. This is done with the following command:

REGRESSION CLASS [Specifications] ;

The following specifications are available:

METHOD = LEUNG / HIERARCHICAL / SEQUENTIAL
> *(Mandatory) The LEUNG method uses the algorithm of Leung, Ma, and Zhang (2001), slightly modified, to find classes. Each class is guaranteed to have a ROC area greater than 0.5 and a positive Spearman Rho between the predicted and the target. The HIERARCHICAL method uses straightforward hierarchical clustering to produce classes that maximize the minimum ROC area in component markets. This has many wonderful optimality properties but is much too slow unless the number of cases is small. The SEQUENTIAL method is a modified hierarchical method that is sub-optimal but runs at a reasonable rate.*

INPUT = [Var1 Var2 ...]

(Mandatory) This lists the input variables that are candidates for inclusion. Range and family options are also available. See Pages 115 and 132 for details.

OUTPUT = VarName

(Mandatory) This names the variable that is to be predicted.

RESTRAIN PREDICTED

(Optional) This causes the true values of the predicted variable to be compressed at the extreme values before training. This is almost always very useful, perhaps even mandatory. If the predicted variable contains outliers, many models will expend considerable effort accommodating these extreme values, at the expense of the more common cases.

MIN CASES = Integer

(Mandatory) This specifies the minimum number of cases that must be in a market in order for the market to enter into computations. Markets that have fewer cases will not join any class.

MAX GROUPS = Integer

(Mandatory) this does not affect computation. The algorithm begins by considering each market to be its own class. As time passes, the number of classes is gradually reduced. When the number of classes drops below the MAX GROUPS threshold, detailed group membership information will be printed to the log files. If you set this to a very large number, the file may be huge because it contains vast detail.

INITIALIZE = Integer

(Mandatory for the SEQUENTIAL method, ignored for the other methods) This specifies the number of class pairs that are kept for merge testing. Execution time is a roughly linear function of this value, and larger values result in higher quality. Values around 1000 are probably a reasonable compromise between time and quality in most cases.

Preserving Predicted Values for Trade Simulator

NOTE... The command discussed in this section is deprecated except for one uncommon use: Data preparation for the TRADE SIMULATOR. This command has been preserved in the latest version of *TSSB* for the sake of backward compatibility with existing script files and invocation of the TRADE SIMULATOR. However, all of its former actions are now accomplished automatically, without the necessity of using this command. In fact, using it in the latest version of *TTSB* may occasionally introduce anomalous behavior and should be avoided except when it is used in conjunction with the TRADE SIMULATOR.

If the TRADE SIMULATOR (Page 191) is invoked, it requires a special form of the predictions made by models, committees, and oracles. This is accomplished with the following command:

PRESERVE PREDICTIONS

This command may appear only once in the script file, and it makes sense only if at least one TRAIN, CROSS VALIDATE, or WALK FORWARD command appears before it. The most recent of these three commands affects the nature of the preserved predictions, as shown here:

TRAIN
> The predictions cover the entire time period of the database. They are all in-sample.

CROSS VALIDATE
> The predictions cover the entire time period of the database. They are all out-of-sample, being derived from the hold-out periods of the folds.

WALK FORWARD
> The predictions cover only the beginning of the test period through the end of the database. These predictions are all out-of-sample, being derived from the walk-forward test periods. Values prior to the beginning of the test period are set to 0.0.

The TRADE SIMULATOR may be invoked any time after a PRESERVE PREDICTIONS command has appeared.

Output Files

Several ASCII text files are produced. These are as follows:

AUDIT.LOG

This is a long, detailed file that documents all major operations of the program. In particular, the following information is printed to the AUDIT.LOG file:

```
COMMAND ---> Command
```

Every command that appears in the control script file is echoed to the audit log as the command is read and processed.

```
User Specified N Markets
      Markets...
```

When the READ MARKET LIST command is processed, the audit log records the number of specified markets and lists their names.

```
Reading market histories (*.TXT) from path MarketPath
      Market had  N cases (StartDate through EndDate) ...
```

When the READ MARKET HISTORIES command is processed, the audit log records the number of days in each market, as well as the starting and ending dates. It also names the path from which the market histories were read.

```
Temporary work file is WorkFileName
Database file is DatabaseFileName
```

When the READ VARIABLE LIST command is processed, the audit log records the file names of two generally large files that will be used by the program for scratch work. Normally, these files are deleted after the program is finished. The names of these files are of no great interest to most users, but they may be handy in diagnostic situations. If the program generates an error message saying something about not being able to open or write a work file, chances are the file grew larger than the capacity of the hard drive. These files can become enormous in some conditions.

```
Summary information for variable VarName:
```

As variables are computed, summary statistics for them in each market are computed and printed in the audit log. A typical line from this section might look like the following:

```
Mkt Fbad Bbad Nvalid   First     Last      Min     Max     Mean
WFT  10    0    5976   19840118  20070924  -0.287  0.331   0.004
```

Fbad is the number of undefined cases at the start of the series, typically due to computing predictors requiring historical information.

Bbad is the number of undefined cases at the end of the series, typically due to computing predicted values requiring future information.

Nvalid is the number of valid cases

First is the date of the first valid case

Last is the date of the last valid case

Min is the minimum value of the variable in this market

Max is the maximum value of the variable in this market

Mean is the mean value of the variable in this market

```
User defined n variables
Wrote m records to the database
```

After all variable processing is complete, the audit log records the number of variables defined and the total number of records (having all variables defined) in the database.

REPORT.LOG

The report log is similar to the audit log in that it records important results from the program. However, it is much more concise. It records only the information that is likely to be crucial to understanding the results of experiments.

Variables

The Variable List File contains a list of all variables that may be used. It is an ASCII text file containing one variable definition per line. The syntax is as follows:

UserName : Definition [Parameters] [: Normalization] [! MinFraction]

The first item on the line is a name chosen by the user. This is the name that will appear in all studies. It should be reasonably short, yet descriptive to the user. It may contain only letters, numbers, 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.

References to an Index Market

Some variables reference a 'market' that is actually an index, such as the S&P 500. When this is done, the user must specify the index market using the *INDEX IS* script file entry described on Page 6. Most variables can be modified by placing the keyword IS INDEX or MINUS INDEX after the definition. The former causes the value of the variable to be computed using the index market rather than the current market. The latter causes the variable to be computed for both the current market and the index market. The final result is the value for the current market minus that for the index market. Thus, the MINUS INDEX variation describes the behavior of the current market relative to a broad market index.

For example, the PRICE MOMENTUM variable described on Page 36 measures how rapidly the market price is changing. When this variable is specified, its value will be computed for each market on each trading day. If the variable is specified as PRICE MOMENTUM IS INDEX, then for each day, the value computed for *every* market will be that for the index market rather than that for the market itself. If the variable is PRICE MOMENTUM MINUS INDEX, each day's value for each market will be that market's price momentum minus the index market's price momentum. Thus, on a day when a given market's momentum is flat but the index market has strong downward momentum, the value of this variable will be strongly positive.

If the user employs more than one index, the specific index can be specified by appending the corresponding integer to the INDEX keyword. No space should separate them. For example, INDEX5 stands for index number 5.

Historical Normalization

The only required fields are the two just described. However, one or both of two optional fields may appear as well. The family and any parameters may be followed by another colon, one of the words, *CENTER*, *SCALE*, or *NORMALIZE*, and an integer greater than one. In this case, as each value of the variable is computed, the program will look back at the history of this variable for the specified number of cases. The *CENTER* option causes the historical median to be subtracted from the computed value. The *SCALE* option causes the computed value to be divided by the historical interquartile range. The *NORMALIZE* option causes both operations to be performed (*CENTER* first, and then *SCALE*). One more stage of transformation, described at the end of this section, is also performed for SCALE and NORMALIZE. These options are an excellent way of forcing a great degree of stationarity on the data. As long as the historical lookback period is made long relative to the trading rate, important information is almost never lost, and the improvement in stationarity can be enormous.

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 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 down periods. By subtracting a historical median, 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.

Scaling is often useful for ensuring cross-market conformity. If we want to pool predictor data from several different markets, it is vital that the statistical properties of the predictors be similar for all markets. Otherwise, some markets will dominate models, while other markets may be essentially ignored. For example, suppose we measure recent price movement as an indicator. Markets having high volatility will produce values of this indicator that have much more variability than low-volatility markets. By scaling the indicator according to its historical volatility, we produce a variable that is likely to have similar variability across markets.

Since centering and scaling are often both valuable, it is useful to combine them into a single process called *normalization*. In most cases, applying this normalization using a historical window of approximately one year can greatly improve the utility of a variable compared to using it in its raw form.

When historical scaling or normalization (centering plus scaling) is performed, an additional step of transformation follows. Let F_{25}, F_{50}, and F_{75} be the 25'th, 50'th, and 75'th percentiles, respectively. Let $\Phi(\bullet)$ be the standard normal CDF. Then the actual normalized variable is defined by Equation 1, where $c=0.25$ for SCALING and 0.5 for NORMALIZATION.

$$V \; = \; 100 \; * \; \Phi\left(c * \frac{X - F_{50}}{F_{75} - F_{25}}\right) \; - \; 50 \qquad\qquad (1)$$

This variable is strictly bounded in the range -50 to 50, although the vast majority of cases will lie well within these bounds and little compression will occur.

Cross-Market Normalization

In multiple-market scenarios there is a type of normalization that is frequently useful. *Cross-Market Normalization* is applied to a variable by including an exclamation point (!) followed by a fraction 0-1 after the variable definition and any normalization.

When this is done, the variable (which may or may not have centering, scaling, or combined normalization) is computed for every market for which data is available. 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 max market gets 50. Markets near the middle of the range will obtain values near zero.

The fraction 0-1 in this command specifies 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. If too few markets are present on a given date, this variable would not have much meaning. Thus, if the minimum fraction is not met, the variable is not computed and is recorded as missing.

For example, suppose we have 20 markets, and we end a variable definition (maybe having historical normalization, or maybe not) with ! 0.6. For every day in which 06.*20=12 markets have history, the variable will be cross-market normalized.

Pooled Variables

We've already seen that an *Index Market* (Page 25) is a way of importing summary information for all markets in a multiple-market situation. Also, *Cross-Market Normalization* (Page 28) is a method for computing within the program new variables that take into account all markets on each day. Another technique for extracting information on the behavior of multiple markets is to use *Pooled Variables*. This is done by including 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. Pooling and cross-market normalization may not be performed simultaneously. The following pooling types are available:

MEDIAN - The median

IQRANGE - The interquartile range

SCALED MEDIAN - The median divided by the interquartile range

SKEWNESS - A nonparametric measure of skewness

KURTOSIS - A nonparametric measure of kurtosis

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

Note that all of these variables are scaled and slightly transformed so that in most situations their natural range will be approximately -50 to 50.

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 jump 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 prices in 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 computing the mean and covariance. In order to compute the value for daily price changes, which is the most usual situation, you would use the CLOSE TO CLOSE variable. So, for example, suppose you are working with daily data, and you want means and covariances to be computed for a window of approximately one year. The variable definition line might look like 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.

Absorption Ratio

Many experts believe that financial markets are most stable when the individual components are moving relatively independently. When the components become locked together in consistent patterns, a situation called *coherence*, the market becomes unstable; a small shock to the system can spread chaos throughout its components as they disentangle.

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. 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 us to an intuitive and effective way to measure coherence: compute the fraction of the total variance that is contained in the few largest eigenvalues. This ratio will range from a minimum of *number kept / number of markets* when all components are independent, to one when all components move in perfect lockstep. ABSRATIO (absorption ratio) is a family of variables to accomplish this. The general format is as follows:

UserName **:** *Variable* **ABSRATIO** *Lookback Fraction ShortMA LongMA*

This command follows the same pattern as other cross-market variables: The user names and defines a built-in variable, and then the keyword ABSRATIO follows, along with its four parameters.

Lookback is the number of bars to include in the moving window used to compute the covariance matrix. *Fraction* is the fraction (0-1) of the eigenvalues to use for computing the absorption ratio. Kritzman et al use 0.2. If *LongMA* and *ShortMA* are zero, the value computed is the absorption ratio, the fraction of the total variance accounted for by the *Fraction* largest eigenvalues. If *LongMA* and *ShortMA* are nonzero, the variable is what Kritzman et al call the *Shift*. It 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.

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 window of 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 (Page 33) 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.

Residuals

If the user is performing trade filtering (Page 140) then the keyword RESIDUAL may appear as the final text on a variable definition for a *predicted* variable. The result is that the filter value or fractile is subtracted from the target value. See Page 140 for more details.

Errors in the Raw Market Data

Historical data files can occasionally contain data that is unusually extreme. Such extremes may be due to real events that are so rare that they should be ignored, or they may be due to actual errors. In order to prevent this data from generating outlier cases that may adversely affect models, the user can specify the following command *before* the variable list file is read. 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 its reciprocal, no case that references this day will be generated. About 0.6 is reasonable.

```
CLEAN RAW DATA  Fraction ;
```

Families of Variables

This section discusses the various families of predictor and predicted variables that may be computed.

CLOSE TO CLOSE

This trivial indicator is 100 times the log ratio of today's close to yesterday's close.

CLOSE MINUS MOVING AVERAGE HistLength ATRlength

This predictor variable measures today's market position relative to its recent history, normalized by recent *Logarithmic Average True Range*. Let C_i be the close at a historical distance of i (where $i{=}0$ is today after the market has closed), let n be the specified historical length, and let m be the historical extent over which logarithmic ATR is computed. This variable is defined as follows, with a modest compressing transform applied to limit it to a range of -50 to 50:

$$V = \frac{1}{\sqrt{n}\ ATR\,(m)}\ \ln\left(\frac{C_0}{\dfrac{1}{n}\displaystyle\sum_{i=1}^{n} C_i}\right) \tag{2}$$

If today's close exactly equals the recent moving average, this variable will have a value of zero. If the price is higher, the variable will be positive, and if lower, negative. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength*.

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. 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. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *LongLength + Lag*.

ABS PRICE CHANGE OSCILLATOR ShortLength Multiplier

A *ShortLength* moving average of absolute log daily price changes is computed. The same is done for a length of *ShortLength* 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 slightly compressed to a range of -50 to 50. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *LongLength*.

LINEAR PER ATR   HistLength   ATRlength

This predictor variable 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. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength*.

QUADRATIC PER ATR   HistLength   ATRlength

This predictor variable computes a least-squares fit of a second-order Legendre polynomial (an orthogonal family) over the specified length of historical data. It also computes the Average True Range over the specified ATR history length. The returned value is the 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 IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength*.

CUBIC PER ATR   HistLength   ATRlength

This predictor variable computes a least-squares fit of a third-order Legendre polynomial (an orthogonal family) over the specified length of historical data. It also computes the Average True Range over the specified ATR history length. The returned value is the polynomial coefficient divided by the ATR. A small amount of compression and rescaling is applied to restrain values to the interval -50 to 50. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength*.

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 in most applications.

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.

PRICE MOMENTUM   HistLength   StdDevLength

This measures the price today relative to the price *HistLength* days ago. It is normalized by the standard deviation of daily price changes, and also normalized to account for the lookback distance. Let P_i be the mean of the open, high, low, and close i days ago, where $i=0$ is today after the market is closed. Let n be the lookback distance *HistLength*, and let m be the number of days used to compute the standard deviation, *StdDevLength*. This variable is defined by Equation 3. A small amount of compression and recaling is applied to restrain values to the interval -50 to 50. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength*.

$$V = \frac{\log\left(P_0 \, / \, P_n\right)}{\sqrt{\dfrac{n}{m} \sum_{i=0}^{m-1}\left[\log\left(\dfrac{P_i}{P_{i+1}}\right) \, - \, \overline{\Delta}\right]^2}} \tag{3}$$

$$\overline{\Delta} = \frac{1}{m} \sum_{i=0}^{m-1} \log\left(\frac{P_i}{P_{i+1}}\right) \tag{4}$$

LINEAR DEVIATION HistLength

A least-squares line is fit to the most recent *HistLength* log prices, including that of today. 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. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength*.

QUADRATIC DEVIATION HistLength

A least-squares quadratic polynomial is fit to the most recent *HistLength* log prices, including that of today. 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. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength*.

CUBIC DEVIATION HistLength

A least-squares cubic polynomial is fit to the most recent *HistLength* log prices, including that of today. 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. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength*.

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 current market from the log price of the index market. (The index market must be specified using the INDEX IS command discussed on Page 6.) The fit is over the specified history length, including today. This variable is today'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. For input list families (see Page 115), the *LOOKBACK* is *HistLength*. 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.

INDEX CORRELATION HistLength

The ordinary correlation coefficient is computed between the log price (mean of open, high, low, and close) of the current market and the log price of the index market. (The index market must be specified using the INDEX IS command discussed on Page 6.) The correlation is over the specified history length, including today. This variable is the correlation times 50, which provides a range of -50 to 50. For input list families (see Page 115), the *LOOKBACK* is *HistLength*. 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.

DELTA INDEX CORRELATION HistLength DeltaLength

This is the INDEX CORRELATION today minus that *DeltaLength* days ago.

ADX   HistLength

The ADX trend indicator for the specified history length is computed. Unlike most other indicators, ADX is neither scaled nor transformed. It retains its defined range of 0-100. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength*.

MIN ADX   HistLength   MinLength

The ADX trend indicator is computed for the current day as well as prior days, for a total of *MinLength* times. The minimum value across these times is found. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength*.

RESIDUAL MIN ADX   HistLength   MinLength

The ADX trend indicator is computed for the current day as well as prior days, for a total of *MinLength* times. The minimum value across these times is found. The final variable is today's value minus the minimum. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength*.

MAX ADX   HistLength   MaxLength

The ADX trend indicator 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. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength*.

RESIDUAL MAX ADX   HistLength   MaxLength

The ADX trend indicator 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. The final variable is the maximum value minus today's value. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength*.

DELTA ADX   HistLength   DeltaLength

The ADX trend indicator is computed for today and for the day *DeltaLength* ago. The final variable is today's minus the lagged value, slightly transformed and scaled to a range of -50 to 50. This measures the rate of change of ADX, its velocity. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength*.

ACCEL ADX   HistLength   DeltaLength

The ADX trend indicator 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. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength*.

INTRADAY INTENSITY   HistLength

The smoothed intraday intensity statistic is returned. 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). 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. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength*.

DELTA INTRADAY INTENSITY   HistLength DeltaLength

This is the difference between today's *INTRADAY INTENSITY* and that the specified number of days ago. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength*.

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*. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength* times *Multiplier*.

MIN/MAX PRICE VARIANCE RATIO HistLen Mult Mlen

This is the minimum or maximum of the *PRICE VARIANCE RATIO* over the prior *Mlength* observations. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength* times *Multiplier*.

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. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength* times *Multiplier*.

MIN/MAX CHANGE VARIANCE RATIO HistLen Mult Mlen

This is the minimum or maximum of the *CHANGE VARIANCE RATIO* over the prior *Mlength* observations. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength* times *Multiplier*.

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*. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength*.

DELTA PRICE VARIANCE RATIO   HistLength   Multiplier

This is the difference between the *PRICE VARIANCE RATIO* today minus that *HistLength* times *Multiplier* days ago. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength* times *Multiplier*.

DELTA CHANGE VARIANCE RATIO   HistLength   Multiplier

This is the difference between the *CHANGE VARIANCE RATIO* today minus that *HistLength* times *Multiplier* days ago. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength* times *Multiplier*.

DELTA ATR RATIO   HistLength   Multiplier

This is the difference between the *ATR RATIO* today minus that *HistLength* times *Multiplier* days ago. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength*.

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 quantity (mean minus twice standard deviation). Note that this variable is extremely poorly behaved in nearly every regard. Temporal normalization (Page 26) is mandatory. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength*.

DELTA BOLLINGER WIDTH   HistLength   DeltaLength

This is the difference between the BOLLINGER WIDTH today minus that *DeltaLength* days earlier. Temporal normalization (Page 26) is mandatory. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength*.

PRICE SKEWNESS   HistLength   Multiplier

This computes a quantile-based 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. 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. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength* times *Multiplier*.

CHANGE SKEWNESS HistLength Multiplier

This is identical to *PRICE SKEWNESS* except that the quantity evaluated is the daily price changes (today's close relative to yesterday's close) rather than the prices themselves. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength* times *Multiplier*.

PRICE KURTOSIS HistLength Multiplier

This is identical to *PRICE SKEWNESS* except that the kurtosis (tail weight) rather than the skewness is measured. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength* times *Multiplier*.

CHANGE KURTOSIS HistLength Multiplier

This is identical to *CHANGE SKEWNESS* except that the kurtosis (tail weight) rather than the skewness is measured. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength* times *Multiplier*.

DELTA PRICE SKEWNESS HistLength Multiplier DeltaLength
DELTA CHANGE SKEWNESS HistLength Multiplier DeltaLength
DELTA PRICE KURTOSIS HistLength Multiplier DeltaLength
DELTA CHANGE KURTOSIS HistLength Multiplier DeltaLength

These compute the difference between the current value of the specified variable and its value *DeltaLength* days ago. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength* times *Multiplier*.

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 to a range of -50 to 50. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength* times *Multiplier*.

DELTA VOLUME MOMENTUM HistLen Multiplier DeltaLen

This is the VOLUME MOMENTUM today minus that *DeltaLen* days ago, transformed to a range of -50 to 50. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is *HistLength* times *Multiplier*.

PRICE ENTROPY   WordLength

This computes the binary entropy of price changes, transformed and slightly compressed to the range -50 to 50. The number of days used in the computation is ten times two to the power *WordLength*. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is ten times two to the power of *WordLength*.

VOLUME ENTROPY   WordLength

This computes the binary entropy of volume changes, transformed and slightly compressed to the range -50 to 50. The number of days used in the computation is ten times two to the power *WordLength*. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is ten times two to the power of *WordLength*.

PRICE MUTUAL INFORMATION   WordLength

This computes the binary mutual information between today's price change relative to yesterday's, and *WordLength* prior days' price changes, transformed and slightly compressed to the range -50 to 50. The number of days used in the computation is ten times two to the power (one plus *WordLength*). The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is ten times two to the power of (one plus *WordLength*).

VOLUME MUTUAL INFORMATION   WordLength

This is identical to PRICE MUTUAL INFORMATION except that volume is used instead of price.

REAL MORLET   Period

The real component of a Morlet wavelet, slightly transformed to a range of -50 to 50, is computed for the log of closing prices. The wavelet is centered 2 * *Period* days prior to the current day. This roughly measures the position of the price within a periodic waveform of approximately the specified period. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is four times the *Period*.

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* days prior to the current day. 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. This roughly measures whether the fast motion is pushing beyond or lagging behind the slow motion. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is four times the *Period*.

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* days prior to the current day. 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 fast motion and the slow motion are in agreement, with the sign telling the direction of the common motion. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is four times the *Period*.

IMAG MORLET   Period
IMAG DIFF MORLET   Period
IMAG PRODUCT MORLET   Period

These are identical to the real versions, but they compute the imaginary version. Roughly speaking, the real coefficient measures position, while the imaginary coefficient measures velocity.

PHASE MORLET   Period

This is the rate of change of the phase, transformed and scaled to a range of -50 to 50. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is four times the *Period*.

DAUB MEAN   HistLength   Level

A Daubechies wavelet is computed for the log of daily close ratios (today divided by yesterday) over *HistLength* days. *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 days 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. I recommend that we do our own experiments to choose optimal values. The *DAUB MEAN* variable is the mean of the smooth (parent wavelet) coefficients, with the detail coefficients ignored. It is also transformed to a range of -50 to 50. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is the *HistLength*.

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 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 rather than the mean is returned.

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 IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is the *HistLength*.

DIFF VOLUME WEIGHTED MA OVER MA   ShortDist LongDist

This is the log of the ratio of the volume-weighted moving average to the ordinary moving average over *ShortDist* days, minus that over *LongDist* days. It is slightly compressed and transformed to a range of -50 to 50.   The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is the *ShortDist*.

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. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is the *HistLength*.

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. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is the *ShortDist*.

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. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is the *HistLength*.

REACTIVITY   HistLength

This is the *REACTIVITY* technical indicator computed over the specified history length. It is slightly transformed to the range -50 to 50. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is the *HistLength*.

DELTA REACTIVITY   HistLength   DeltaDist

This is the *REACTIVITY* 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. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is the *HistLength*.

MIN/MAX REACTIVITY   HistLength   MinMaxDist

This is the minimum (or maximum) of *HistLength REACTIVITY* with the minimum or maximum taken over *MinMaxDist* days. It is slightly transformed to the range -50 to 50. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is the *HistLength*.

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. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is the *HistLength*.

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. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is the *HistLength*.

POSITIVE VOLUME INDICATOR   HistLength

This is the average over the specified number of days of relative price change (yesterdays' close to today's close), where only changes corresponding to increased volume are taken into consideration. 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 transformed to the range -50 to 50. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is the *HistLength*.

DELTA POSITIVE VOLUME INDICATOR HistLength DeltaDist

This is the *POSITIVE VOLUME INDICATOR* 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. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is the *HistLength*.

NEGATIVE VOLUME INDICATOR HistLength
DELTA NEGATIVE VOLUME INDICATOR HistLen DeltaDist

These are identical to the prior *POSITIVE* versions, except that price changes are considered only when they correspond to decreasing volume.

PRODUCT PRICE VOLUME   HistLength

For each day, today's volume is normalized by dividing it by the median of the prior 250 day's volumes. Thus, if today's volume is 'average' the result will be one. If today's volume is unusually small, the result will be near zero, and if unusually large, it will be much greater than one.

Today's price change (log of the ratio of today's close to yesterday's close) is normalized by subtracting the median of this quantity over the prior 250 days and dividing by the interquartile range.

The *PRODUCT PRICE VOLUME* indicator is computed as the product of the normalized price and the normalized volume. It is slightly transformed to the range -50 to 50. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is the *HistLength*.

SUM PRICE VOLUME   HistLength

Today's price change and volume are normalized as in the *PRODUCT PRICE VOLUME* indicator. The *SUM PRICE VOLUME* indicator is computed as the sum of the normalized volume and the absolute value of the normalized price. If the normalized price change is negative, the sign of the sum is flipped. It is slightly transformed to the range -50 to 50. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is the *HistLength*.

DELTA PRODUCT PRICE VOLUME   HistLength   DeltaDist
DELTA SUM PRICE VOLUME   HistLength   DeltaDist

These are the *PRODUCT PRICE VOLUME* and *SUM PRICE VOLUME* indicators 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. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is the *HistLength*.

N DAY HIGH   HistLength

Let *N* be the number of days one has to go back in order to find a price higher than today's high. Search only *HistLength* days back. If after this many days are checked, no higher high is found, set *N*=*HistLength*+1. Then this variable is defined as 100 * (*N*-1) / *HistLength* - 50. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is the *HistLength*.

N DAY LOW   HistLength

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

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 today's true range. True range is defined as the maximum of today's high minus today's low, today's high minus yesterday's close, and yesterday's close minus today's low. Search only *HistLength* days back. If after this many days are checked, no smaller true range is found, set *N*=*HistLength*+1. Then this variable is defined as 100 * (*N*-1) / *HistLength* - 50. The IS INDEX or MINUS INDEX modifiers described on Page 25 may be applied to this variable. For input list families (see Page 115), the *LOOKBACK* is the *HistLength*.

N DAY WIDER   HistLength

This is identical to N DAY NARROWER except that we search for greater true range.

FTI family... BlockSize HalfLength LowPeriod HighPeriod

The next 14 variables are members of the Govinda Khalsa FTI family. They have several things in common.

- The first parameter is the length of the moving-window block (called *BlockSize* later), which for input list families (see Page 115) is the *LOOKBACK*.

- The second parameter is the half-length of the filter, the number of data points on each side of the center point from which a filtered value is computed. This value must be considerably less than the *BlockSize*. Setting the *HalfLength* equal to half of the *BlockSize* is a reasonable choice.

- The third parameter is the period of the minor (faster; shorter period) filter, or the period of the only filter for those variables that use a single filter. This is the period at which a cyclic quantity is attenuated to approximately half its value.

- The fourth parameter, if used, is the period of the major (slower; longer period) filter. Obviously, it must exceed *LowPeriod*. It must also be less than or equal to twice the *HalfLength*. For best performance, it should be quite a bit less.

- The *BlockSize* minus the *HalfLength* is the number of points in the channel for which FTI variables are computed. Obviously, this difference must be at least 2. Preferably it should be at least several dozen.

- All FTI variables are based on the log of the closing price.

FTI LOWPASS BlockSize HalfLength Period

Gavinda Khalsa's zero-lag lowpass filter is applied to the log of closing prices. The most likely use for this quantity would be as part of an oscillator crossover system.

FTI MINOR LOWPASS BlockSize HalfLength LowPeriod HighPeriod

A pair of lowpass filters is computed according to the Govinda-Khalsa rules. The periods are restricted to the specified range. This returns the minor filter value. In all likelihood, this quantity is useful for display and diagnostic purposes only.

FTI MAJOR LOWPASS BlockSize HalfLength LowPeriod HighPeriod

As above, except the major filter value is returned.

FTI FTI BlockSize HalfLength Period

This returns the FTI value at the specified period. This may be the single most useful FTI variable, because 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.

FTI LARGEST FTI BlockSize HalfLength LowPeriod HighPeriod

This returns the value of the largest FTI within the specified range of periods (inclusive).

FTI MINOR FTI BlockSize HalfLength LowPeriod HighPeriod

A pair of lowpass filters is computed according to the Govinda-Khalsa rules. The periods are restricted to the specified range. This returns the FTI for the minor (smaller period) filter.

FTI MAJOR FTI BlockSize HalfLength LowPeriod HighPeriod

As above, but returns the FTI of the major filter.

FTI LARGEST PERIOD BlockSize HalfLength LowPeriod HighPeriod

This returns the period, within the inclusive specified range, which corresponds to the largest FTI value.

FTI MINOR PERIOD BlockSize HalfLength LowPeriod HighPeriod

A pair of lowpass filters is computed according to the Govinda-Khalsa rules. The periods are restricted to the specified range. This returns the period of the minor (smaller period) filter.

FTI MAJOR PERIOD BlockSize HalfLength LowPeriod HighPeriod

As above, except the major period is returned.

FTI CRAT BlockSize HalfLength LowPeriod HighPeriod

The minor/major channel ratio for the specified pair of periods is returned. This variable nicely complements the FTI FTI variable, because the exact periods can be provided to best fit the application.

FTI MINOR BEST CRAT BlockSize HalfLength LowPeriod HighPeriod

The minor/major channel ratio is returned. The major period is fixed at *HighPeriod*, and the Govinda-Khalsa algorithm chooses the best minor period. This is what they do in their paper. They fix the major period at 65 and find the best associated minor period.

FTI MAJOR BEST CRAT BlockSize HalfLength LowPeriod HighPeriod

The minor/major channel ratio is returned. The minor period is fixed at *LowPeriod*, and the Govinda-Khalsa algorithm chooses the best major period.

FTI BOTH BEST CRAT BlockSize HalfLength LowPeriod HighPeriod

The minor/major channel ratio is returned. The Govinda-Khalsa algorithm chooses the best minor and major periods.

DETRENDED RSI DetrendedLength DetrenderLength Lookback

RSI at the two specified lengths are computed and a least-squares regression line is fit for predicting that of *DetrendedLength* from that of *DetrenderLength*. This least-squares fit is based on 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 are based on this transformed value. Note that *DetrendedLength* must be less than *DetrenderLength*.

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, normalized as discussed below. No compression is done. Typically, the index 'market' will not be an actual market. Rather, it will be a sentiment indicator such as VIX. The predictors are based on recent 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; value = 100 * (actual - predicted) / actual
> **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 %
```

NEXT DAY LOG RATIO

This predicted variable measures the one-day change of the market in the immediate future, *roughly* expressed as an annualized percent. The implied trading scenario is that after today's close we make an entry decision for the next day. We enter at the next day's open, and close the position at the following day's open. Let O_i be the open i days in the future, where $i=0$ is today (whose open has passed, since we are at the end of the day when variables for the day are computed). This variable is defined as follows:

$$V = 25000 * \ln\left(\frac{O_{i+2}}{O_{i+1}}\right) \tag{5}$$

NEXT DAY ATR RETURN Distance

This predicted variable measures the one-day change of the market in the immediate future, relative to recent *Average True Range* (*ATR*) taken over a specified distance. The implied trading scenario is that after today's close we make an entry decision for the next day. We enter at the next day's open, and close the position at the following day's open. Let O_i be the open i days in the future, where $i=0$ is today (whose open has passed, since we are at the end of the day when variables for the day are computed). This variable is defined as follows:

$$V = \frac{\left(O_{i+2} - O_{i+1}\right)}{ATR\,(Distance)} \tag{6}$$

If *Distance* is specified as zero, the denominator is one so that the value is the actual point return, not normalized in any way.

SUBSEQUENT DAY ATR RETURN Lead Distance

This is identical to *NEXT DAY ATR RETURN* except that instead of looking one day ahead, it looks *Lead* days ahead.

NEXT MONTH ATR RETURN Distance

This is identical to *NEXT DAY ATR RETURN* except that the return is computed starting from the first day of the first month following the current day, and ending the first day of the next month after that. If *Distance* is set to zero, the return is the actual point return, not normalized in any way.

HIT OR MISS Up Down Cutoff ATRdist

The ATR is computed for a history of *ATRdist* bars, and then the future price move is examined up to *Cutoff* bars ahead, beginning at tomorrow's open. If the price goes up at least *Up* times ATR before going down at least *Down* times ATR, the value of the variable is *Up*. If the price goes down at least *Down* times ATR before going up at least *Up* times ATR, the value of the variable is minus *Down*. If the price hits neither of these thresholds by the time *Cutoff* bars have passed, the value of the variable is the price change divided by ATR.

If *ATRdist* is set to zero, no normalization is done. The price movement is defined as the actual point return.

FUTURE SLOPE Ahead ATRdist

This is the slope (price change per bar) of the least-squares line *Ahead* bars, divided by ATR looking back *ATRdist* bars.

RSQ FUTURE SLOPE Ahead ATRdist

This is the slope (price change per bar) of the least-squares line *Ahead* bars, divided by ATR looking back *ATRdist* bars, multiplied by the R-square of the fit.

CURVATURE INDEX LookAhead MinSep SepInc Nseps Vdist

This target measures the change in slope over a time period that is partially or entirely in the future. A positive value means that the slope is increasing, and a negative value means that the slope is decreasing.

The most basic source of curvature measure, the fundamental building block of this target variable, is a set of three points which may be called P_{-1}, P_0, and P_1. P_{-1} precedes P_0 by a certain

number of bars, and P_1 follows P_0 by this same number of bars, called the *separation* in this discussion. Thus, P_0 is the 'center' of these three equally spaced points. The slope (except for a constant factor related to the point separation) prior to the center point is $P_0 - P_{-1}$, and the slope after the center point is $P_1 - P_0$. Subtracting the former from the latter gives the curvature, $P_{-1} + P_1 - 2 * P_0$.

This simplistic formula for curvature has a serious problem when used as a target in market prediction: It is not immune to changes in scale. If two price series are identical except that one is twice the other, then this double-value series will, at every point, have twice the curvature of the other series. This is obviously undesirable behavior. If we were to plot two series and find that they were identical except for the axis labels, we would be annoyed to find that their curvatures were different (at least in market-trading applications).

Such differences in scaling can arise in any of several ways:

- Different markets generally have different price scaling.

- A given market may experience periods of unusually high or low volatility.

- If the three price points are adjacent bars, the price differences will be much smaller than if the outer points are separated from the center by 20 bars. So the separation impacts the simple definition of curvature.

All of these sources of scaling variation can be addressed by a single expedient: look backward from the current bar over a relatively long window, and compute the volatility within this recent historical window. This allows *TSSB* to normalize the curvature measure according to the degree of price movement that would be expected in this market, during this time period, with this point separation. The distance to look back for volatility normalization is given by the final parameter, shown as *Vdist* in the syntax above.

Sometimes the user does not wish to be locked into a single separation distance. Pooling curvatures from a range of separations can provide a more global estimate of curvature. In other words, if prices have strong upward (or downward) curvature at separations of 2, 6, and 10 bars, we can be confident that this is a real turning point in the market. This pooling of multiple separations is effected by three parameters.

MinSep is the minimum separation between each of the two outer points and the center point.

SepInc is the increment in separations across the pooled set of separations.

Nseps is the number of different separations to pool. Set this to 1 for a single separation.

For example, *MinSep=2*, *SepInc=4*, and *Nseps=3* would provide the separations of 2, 6, and 10 bars mentioned earlier.

Finally, the user needs to decide where the center P_0 is located relative to the current bar. All separations use the same center. The *LookAhead* parameter is the number of bars past the current bar at which the center is located.

A strong argument can be made that the best value for *LookAhead* is zero, as this centers the curvature measure(s) at the current bar. In other words, the value of this target would reveal whether the slope of the price is about to change relative to what the slope has been recently. This would be a valuable piece of information for the model to know. If a model can predict that the price slope is about to increase or decrease dramatically starting at the next bar, it is likely that this prediction can be used profitably.

However, it must be emphasized that if the backward-looking point, P_{-1}, is at or prior to the current bar, there will be overlap between the market segment used to compute indicators and that used to compute the target. This means that performance results related to this target will be optimistic, even out-of-sample. Also, CURVATURE INDEX is not a profit-based target, so profit factors and the like would not be computed for it. For these reasons, it is virtually mandatory that the user employ the PROFIT option to declare a secondary profit-based target that will be used to compute legitimate profitability performance measures.

If for some reason the user wants to push the center of curvature into the future far enough that the indicator and target price histories do not overlap, this can be accomplished with the following reasoning: The maximum separation between the center point (P_0) and P_{-1} will be *MinSep + (Nseps-1) * SepInc*. Thus, *LookAhead* must be at least one greater than this quantity if the P_{-1} at maximum separation is to lie at least one bar past the current bar. However, be advised that the further the center is pushed ahead of the current point, the further the target's price data will be from the indicators' price data. This will almost certainly deprive the model of valuable predictive information. It is difficult to imagine a scenario in which having a *LookAhead* greater than zero would be advisable.

The CURVATURE INDEX target was inspired by the paper "Measuring the Statistical Significance of Financial Trend Change Forecasts" by S. Kris Kaufman.

Examples of Variables

This section presents some examples of predictor and predicted variables.


```
CMMA: CLOSE MINUS MOVING AVERAGE 10
```

A variable that the user calls *CMMA* is computed using Equation 2, 3, 4 with a historical moving-average length of 10 days.


```
CMMA_N: CLOSE MINUS MOVING AVERAGE 10 : NORMALIZE 250
```

The same variable as above is computed. It is then normalized (center and scale) going back 250 days (approximately a year).


```
CMMA_C: CLOSE MINUS MOVING AVERAGE 10 ! 0.5
```

The variable shown first in this section is computed. It is then subjected to cross-market normalization, provided that at least half of the specified markets have valid data for the day.


```
CMMA_N_C: CLOSE MINUS MOVING AVERAGE 10: NORMALIZE 250 ! 0.5
```

The variable shown first in this section is computed. It is then normalized (center and scale), and finally subjected to cross-market normalization as above.


```
LOGRET: NEXT DAY LOG RATIO
```

This predicted variable is computed using Equation 5.

Stationarity Tests

This section discusses the stationarity tests that are available from the menu (the recommended method) or from a command in the script file (not recommended at this time).

Roughly speaking, a time series is stationary if its statistical properties do not change as time passes. The expected value 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. Luckily, in practice it is only the grossest, usually most visible aspects of stationarity 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. This is because variables will be pooled into a common dataset from which models will be trained, and it is expected 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 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 *TSSB* program contains several families of stationarity tests. The simplest and fastest tests are based on the chi-square distribution. Imagine plotting a time series of the variable and placing a rectangular grid over the plot. Count the number of cases that fall into each section of the grid. If the series is stationary, the vertical distribution of counts should be roughly equal across time (columns of the grid.

Another family of tests searches for a single point in time at which the nature of the series changes dramatically. This might happen, for example, if a market moves from open-outcry trading to electronic trading. These algorithms attempt to determine if such a changeover point exists.

Some tests assume at least approximate normality of the series, while other tests 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 parametric tests. However, the nonparametric tests are much slower to compute, and may be impractical in some situations.

Most of the tests segment time linearly, looking for changes as time passes from the distant past forward. However, a variety of tests are available based on the month of the year. These tests attempt to determine if some months behave differently from other months.

Finally, the user can request an additional interesting feature for any of the tests. A Monte-Carlo Permutation Test can be performed on the test statistic with a specified number of replications. This feature tremendously slows the test, increasing the required time by a factor roughly equal to the number of replications. As few as 100 replications may suffice to provide a crude estimate of the enhanced performance figure, although at least 1000 are required for any degree of certainty.

These Monte-Carlo tests 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. This is of little or no use for the ordinary chi-square tests when the expected cell count is large, nor is it useful for the monthly tests. For these tests, the p-value computed in the ordinary way and listed as *p-value* is usually accurate enough for most work. However, the Monte-Carlo *Single pval* is immensely useful under the following circumstances:

- An ordinary chi-square test that employs a large number of bins relative to the number of cases results in the test statistic poorly approximating the chi-square distribution.

- The *mutual information with time* test statistic has an unknown distribution, so the Monte-Carlo test is the only way to approximate its p-value.

- All tests that find a maximally-separating boundary have a test statistic with unknown distribution, so the Monte-Carlo test is the only way to approximate its p-value.

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 predictor and predicted 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 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 safely conclude that the variable is stationary. On the other hand, tiny values of the *Grand pval* should raise a big red flag.

Finally, understand that although tiny values of the various p-values strongly indicate nonstationarity, this does not automatically imply that the degree of nonstationarity is serious enough to degrade performance of the trading system. 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.

For example, all of the chi-square tests list the *contingency coefficient*. This value ranges from zero for perfect stationarity (as far as this particular test goes!) to one for enormous nonstationarity. My own rule of thumb is to treat values less than 0.1 as unimportant, and values greater than 0.3 or so as deadly, with intermediate values in a gray area.

All t-tests and U-tests for a single changeover boundary compute an *Adjusted* p-value. This provides a heuristic compensation for the sample size. It is virtually always much larger than a raw p-value, and should not be taken as a probability. However, the adjusted p-value is a good means of judging relative degrees of nonstationarity.

Linear Chi-Square Tests

These two tests divide the time extent into a user-specified number of intervals (five is often reasonable), and they also divide the values of the variable into a user-specified number of bins. Again, five or so is usually reasonable. The null hypothesis tested is that the distribution of values into bins is independent of the time period. The only difference between the tests is how the value bins are defined.

Equal-Spacing Chi-Square

The range of the variable is divided into segments of equal length. For example, suppose the variable ranges from 10 through 15, and three bins are specified. The bins will each be two units wide. This choice is usually poor if outliers are expected, because one or more extreme bins can be very sparsely populated.

Equal-Count Chi-Square

The value bins are defined in such a way that each row (sum of all bins across time) contains an approximately equal number of cases. This method is almost always superior to the equal-spacing method, and is virtually mandatory if outliers are expected. The only (rare) instance in which equal spacing might be preferred to equal count is if certain specific boundaries are meaningful.

Mutual Information With Time

This test computes the mutual information shared by the variable and the date. In other words, the question answered is the degree to which knowledge of one of these quantities provides information about the other quantity. This is vaguely similar to the linear chi-square tests in that the mutual information is based on counts within value and time cells. However, mutual information cell boundaries, in both the time and value dimensions, adapt to the data density rather than being arbitrary. This test statistic is less likely than chi-squared to be fooled by unusual data patterns. The associated R-square is a good heuristic for judging the degree of nonstationarity, with values under 0.1 probably being safe.

Boundary T-Tests

These two tests move a boundary point across the series, searching for the division that maximally separates the two sections. The user must specify a minimum distance from the edge. In other words, each of the two final sections must contain at least the specified number of cases. This prevents the test from splitting near one end of the series, resulting in one section being so small that its measured property is statistically unstable. This test is most appropriate when the user suspects that a single event, such as a change in rules affecting the market, might cause a rapid and permanent shift in the properties of the variable.

Because the test must be repeated for each candidate boundary point, computation time can be slow. Also, because the test statistic is based on the maximum from among all candidate boundaries, the p-value (as well as its adjusted value) associated with the t-score will be wildly pessimistic and hence nearly useless. For this reason, the Monte-Carlo Permutation Test extension, with at least 1000 replications, should be used if at all possible. This will provide a decent approximation to the correct p-value.

Note that these tests assume that the data follows a normal distribution. In particular, even one wild outlier can render the test practically useless. For this reason, unless you are certain that the data is well behaved, the U-tests of the next section should be employed. Unfortunately, the U-tests are even slower than these t-tests. If the data definitely does not contain outliers, and computer resources are limited, the t-tests may be the only possibility.

T-Test for Mean

The series is split in such a way that the two sides have maximum difference in mean.

T-Test for Mean Absolute Value

This is identical to the prior test except that the absolute value of the series is taken.

Boundary U-Tests

The t-tests described in the prior section suffer from the flaw that outliers can and usually will have an adverse impact on results. As long as outliers are definitely not present, the t-tests are fast and effective. However, if there is any possibility of outliers, the U-tests of this section must be performed instead. These tests are fabulous in that they have nearly the same power as a t-test (asymptotic efficiency of about 0.95), yet because they are based on order statistics rather than raw data, they are robust against outliers. Their only disadvantage is that they are a *lot* slower than t-tests.

U-Test for Mean

The series is split in such a way that the higher-ranked cases tend to lie on one side of the boundary while the lower-ranked cases lie on the other side, with maximum separation.

U-Test for Mean Absolute Value

This is identical to the prior test except that absolute values of the series are taken prior to performing the test.

U-Test for Deviation from Median

This is an interesting test that captures an element of volatility for data that is not centered near zero. The median of the earlier section of the data is computed and subtracted from each case in that section. The absolute values of these differences are taken. The same is done for the later section. The boundary is chosen such that the difference in ranked departures from the median is maximized. In other words, the cases with greatest (after ranking) departure from the section median lie on one side of the boundary, and those with least departure lie on the other side.

Monthly Tests

In some markets, such as those related to agriculture or weather, nonstationarity can take the form of monthly variation. Tests in this group test for differences in behavior in different months.

Monthly Equal-Spacing Chi-Square

The range of the variable is divided into bins (five is often reasonable) having equal width by value, and also into twelve columns, one for each month. An ordinary chi-square test is performed. If outliers are present, this test will be severely degraded and the equal-count method of the next section should be employed.

Monthly Equal-Count Chi-Square

This is identical to the prior test except that the bin boundaries are defined so that each row (summed across months) has an approximately equal number of cases. This is usually preferred to equal spacing, especially if outliers are present.

Monthly ANOVA for Location

An ordinary analysis of variance is performed on the variable, using the twelve months as the category. This is a very fast test to compute, but if outliers are present its results will be distorted, perhaps to the point of being meaningless. In this case, a Kruskal-Wallis test should be used.

Monthly Kruskal-Wallis for Location

An ordinary Kruskal-Wallis analysis of variance is performed on the variable, using the twelve months as the category. This is much slower to compute than ANOVA, but if outliers are present it is virtually mandatory. This tests the null hypothesis that ranked values are equally distributed across all twelve months.

Monthly Kruskal-Wallis for Absolute Location

This is identical to the prior test except that the absolute values of all cases are computed before the test is performed.

Monthly Kruskal-Wallis for Deviation from Median

The median for each month is computed. Each case has its month's median subtracted, and the absolute value of each difference is taken. The test is performed on these quantities. This is an effective test for differences of volatility among months when the data is not centered around zero.

Menu Commands

Although most operation is controlled by a script file, the user has the ability to perform some operations from the menu system. This section describes the commands that are available.

File Menu Commands

The following actions are available from the *File* menu:

Read Script - A script file, as described on Page 5, is read. This menu selection will be commonly used, as it is the means by which you read the script control file that controls nearly all operations.

Read Market List - a Market List File, as described on Page 4, is read. This selection will not normally be used, as it is far easier to read the market list file via the READ MARKET LIST command in a script file. It is included only because in some experimental situations it is easier to omit this command from the script file and try different market lists with a common script file.

Read Market History - All Market History files, as described on Page 2, are read. This command will almost never be used because it is generally much easier to read the market histories via a READ MARKET HISTORY command in a script file.

Read Variable List - The Variable List File, as described on Page 4, is read. This command will almost never be used because it is generally much easier to read the variable list via a READ VARIABLE LIST command in a script file.

Print - If a graphic box (described later) is displayed and selected by clicking on it, this option will send the graphic to a printer.

Exit - The program is terminated.

Job Continuation with Multiple Scripts

If a script file has been read and processed, and then *Read Script* is selected again, the file that is read will be appended to the current script file. *This action does not start a new job; it continues the current job.* If you wish to discard all aspects of the current job and begin a new job, you must exit TSSB and restart it.

The ability to continue an existing job can be very handy in some situations. The user may have just run a very time-consuming job but suddenly wish that the script file had employed a slightly different command or an additional command. Making this change or addition to the script file and then re-running the entire job would waste a lot of time. Instead, the user can read an additional script file, often just one line long, and achieve the desired result.

One common use for this capability occurs in the early development stages of a trading system, especially if the script file contains one or more transforms. The user may run the job, display results (such as transform outputs or model predictions), and become suspicious that something may not have gone as intended. In such situations it can be enlightening to examine the actual numbers produced by the transform or model. This can be accomplished by reading a one-line script file such as the following:

```
WRITE DATABASE "Debug.txt" ;
```

This will write the database as a text file that the user can then study.

Another example of a use of this capability might be if the user wants to change a parameter in a walkforward operation. Suppose, for example, that the user's script file has begun a walkforward in 2005 using a 5-year training period. Now the user wants to back up to begin one year earlier, in 2004. This is done by reading the following one-line script file:

```
WALK FORWARD BY YEAR 5 2004 ;
```

All results are appended to the existing audit log, which conveniently keeps everything together in one file. So in the walkforward example just shown, the results of starting at 2004 would appear in the audit log immediately after the results of starting at 2005, which was the command in the original script file.

Note that because this continues an existing job as opposed to starting a new job, all commands must be legal in the context of the existing job. For example, READ MARKET LIST, READ DATABASE, and READ VARIABLE LIST cannot appear if the current script file has already read and computed these items, which will nearly always be the case. Similarly, if the user defines a new model or other entity, the name must not conflict with one that already exists. In any case, an appropriate error message will be issued if such a mistake is made.

Describe Menu Commands

Commands in the *Describe* menu cause statistics to be computed and printed to the screen, the audit log, and the report log.

Univariate

Basic univariate statistics are computed. This command can also be issued from the DESCRIBE command in a script file. See Page 11.

Chi-Square

A chi-square test between a set of predictor candidates and a single target is performed. See Page 152 for details of this test. A block of predictor candidates is selected by clicking on the first, holding down the *Shift* key, and clicking on the last. Individual variables can be toggled between selected and not selected, without impacting existing selections, by holding the *Ctrl* key while clicking on the variable.

Nonredundant Predictor Screening

A set of indicators is selected from a list of candidates. The selection criterion is such that the set has maximum power to predict the target while simultaneously having minimal redundancy among themselves. See Page 158 for details of this test. A block of predictor candidates is selected by clicking on the first, holding down the *Shift* key, and clicking on the last. Individual variables can be toggled between selected and not selected, without impacting existing selections, by holding the *Ctrl* key while clicking on the variable.

Market Scan

One or all markets are scanned for unusually large gaps. See Page 11.

Outlier Scan

One variable or all variables in the database are scanned for outliers and poor entropy. This command can also be issued in a script file. See Page 12.

Cross Market AD

Tests for cross-market conformity are performed with the Anderson-Darling statistic. This test can also be issued from the CROSS MARKET AD command in a script file. See pages 12, 13.

Stationarity

Tests for stationarity of one or all variables in a single market. This command can also be executed in a script file via the STATIONARITY command. However, the script file does not currently have any way of selecting tests or specifying parameters. Therefore, it is preferable to execute this command from the menu. See Page 63 for details on the stationarity tests.

Plot Menu Commands

Selections in this menu cause graphics to be displayed on the screen. Nothing is written to a log file. These graphics can be printed by clicking the desired window and then clicking *File/Print*.

Series

This selection causes a time series to be displayed. The user must specify a variable and a market. The display can be magnified by positioning the mouse at the left side of the desired area of magnification, pressing the left mouse button, dragging to the right as desired, and releasing the button. Once magnified, the enhanced area can be shifted left or right by dragging the horizontal scroll at the bottom of the screen, or clicking on the scroll bar in the usual Windows fashion. At this time, the program does not allow expanding the image back to its original size, although the same effect can be had by plotting the series again.

Series + Market

This is identical to the *Series* option above with two exceptions. First, it overlays the log of the market close on top of the variable. This overlay is colored red and it is automatically scaled to fill the entire plot, top to bottom. Second, the user has the option of specifying a threshold for display. This threshold, if specified, is plotted as a horizontal green line.

Market

The log (base 10) of the market close is plotted. Up to four colors may be used for the plot, as determined by the values of two variables relative to thresholds. If one or both of the color-determining variables is a predicted value from a model, committee, or oracle, and the predictions were derived from walkforward testing, then the plot begins with the first out-of-sample date. Otherwise the plot covers the entire date range of the market in the database.

The user must specify the market to be plotted and two color-determining variables, as well as a threshold for each variable. This gives two binary conditions: the first variable may be greater than its threshold versus less than or equal to its threshold, and ditto for the second variable. Thus, we have four possible conditions, above/below for the first variable and above/below for the second. The user must choose a color for each of these four conditions. The user may also choose a width (in pixels) for the plotted market prices, although the default value of 3 should be good for most cases. The color of the line connecting adjacent prices is determined by the values of the variables as of the date at the start of the line.

Histogram

A histogram of a selected variable is displayed. By default, the entire range of the variable is displayed. The user can impose a lower and/or upper limit for display by clicking the corresponding checkbox and entering the desired limit(s). This is handy for variables that have one or a few extreme values that cause the graph to become unnaturally compressed. Cases outside the specified limits accumulate in a bar at the lower (and/or upper) edge of the display.

The user also specifies the number of bins to use. Making this an odd number causes the graph to be centered at zero if the upper and lower limits are equal, which is good in many cases.

Optionally, two histograms can be overlayed. The primary market is displayed as solid black bars, and the secondary market is displayed as cross-hatched bars. Instead of choosing a market, the user can select 'Pooled' to plot the entire dataset for this variable, including all markets. Checking the 'Normalize' box causes the distributions to be scaled to have equal area, which greatly improves the display when the two markets have unequal numbers of cases.

Thresholded Histogram

One or two histograms of a selected variable are displayed. In the dialog box, this is called the *Plotted Variable*. By default, the entire range of the variable is displayed. The user can impose a lower and/or upper limit for display by clicking the corresponding checkbox and entering the desired limit(s). This is handy for variables that have one or a few extreme values that cause the graph to become unnaturally compressed. Cases outside the specified limits accumulate in a bar at the lower (and/or upper) edge of the display.

The user also specifies the number of bins to use. Making this an odd number causes the graph to be centered at zero if the upper and lower limits are equal, which is good in many cases.

What makes the thresholded histogram interesting is that the user also selects a *Thresholded Variable*. In an ordinary histogram, all cases are plotted. But in a thresholded histogram, only those cases which lie in the lower and/or upper specified percent of the thresholded variable are plotted. In other words, the plot contains only a subset of the cases, and the particular subset is determined by the value of the thresholded variable for each case.

Two check boxes are available, one for a lower percent of the thresholded variable, and one for an upper. The user may check either of these boxes, or both. If both are checked, pairs of histogram bars are displayed, with the solid (or red if color is selected) bar corresponding to the lower values, and the hatched (or blue, if color is selected) bar corresponding to the upper values. In this case, it is usually best to also check the *Normalize Areas* box. This will cause the two overlayed histograms to be displayed with equal area, making interpretation easier in most cases.

The thresholded histogram is most useful when we are filtering trades with a single variable. For example, suppose that large values of the filter (thresholded) variable correspond to large expected profits, and suppose we want to keep just the largest ten percent cases. We would enter '10' in the *Upper Percent* box. This will cause the cases kept by the filter to be displayed as hatched bars. We have two reasonable choices for the *Lower Percent*. If we enter 100, the solid bars will display a complete histogram of the target variable. If we enter 90, the solid bars will show the cases that were rejected by the filter.

Density Map

A density plot is a sophisticated version of a scatterplot for showing the relationship between two variables. A scatterplot displays individual cases on a map that uses the horizontal axis for the value of one variable and the vertical axis for the value of the other variable. Each case has a unique position on this map and this position is typically marked with a small dot.

The problem with this approach is that if there are numerous cases, so many that limited display or visual resolution results in multiple cases being overlaid on the same spot, it is impossible to see the quantity of cases at that spot. Whether one case lies there, or a thousand, it's still a single dot.

Another problem with scatterplots is that they unavoidably include marginal densities in the display. If one of the variables clusters at one extreme, most of the cases will be plotted there. Even if the two variables are unrelated, the map will show a gradient, when what most people are most interested in is departures (inconsistencies) in the actual bivariate density after taking marginal distributions into account.

A density plot overcomes these problems by using the gray level or color of the display to portray the log ratio of the actual density to the theoretical density (product of the marginals) of cases in every area of the map. It does this by fitting a Parzen smoothing window to the data in order to produce an estimated bivariate density as well as both marginal densities. Optionally, it converts these figures to an estimate of the contribution of that locality to the mutual information contained in the pair of variables.

The following parameters may be set by the user:

Horizontal variable - Selects the variable plotted along the horizontal axis. By checking the *Upper Limit* and/or *Lower Limit* box above the list of variables and filling in a numeric value, the user can limit the range plotted. Cases outside this range still take part in computations, but the plot does not extend to include them.

Vertical variable - Selects the variable plotted along the vertical axis. The same plot limit options as above apply.

Resolution - This is the number of grid locations along both axes that are used for computation. The plot interpolates between the grid points. Larger values result in more accuracy but longer run time.

Relative width - This is the width of the Parzen smoothing window. The user should set this in accord with the quantity of noise in the data. Smaller values will produce very precise density estimates, but if the data contains a large amount of noise, these noise points will be considered to be valid data and thus make an undue contribution to the display. Larger width values smooth out noise, at the price of less precision in the density estimates.

Tone shift - This has no effect on computation, but it controls how the density estimates map to grey levels or color on the display. Positive values will push the display in one direction, and negative values in the opposite direction. Use this option to highlight desired features.

Tone spread - Like *Tone shift*, this has no effect on computation, but it controls how the density estimates map to grey levels or colors on the display. Negative values are legal but rarely useful. Positive values act to sharpen contrast, emphasizing boundaries near the middle of the range of densities at the expense of blurring detail at the extremes. Use this option to highlight desired features.

Inconsistency / Mutual information - The default *Inconsistency* plots the bivariate density relative to the theoretical density. This is often called density inconsistency because variables that are unrelated will produce a uniform map, regardless of their marginal distributions. To the degree that the density is inconsistent (higher than expected in some regions and lower in others), the gray level or color of the display in that area will shift in one direction or the other. By selecting *Mutual information*, the display will show the contribution of each region to the mutual information between the variables. This is rarely of much value.

Plot in color - By default the display is black and white. Checking this box causes the display to be shades of red and blue.

Equity

If the rulebase contains one or more Models, Committees, Oracles, or Portfolios, and a TRAIN, WALK FORWARD, or CROSS VALIDATE command has been executed, the equity can be graphed. If a WALK FORWARD or CROSS VALIDATE command has just been executed, the equity is the out-of-sample performance.

The user must choose whether the equity curve portrays the performance of only long trades, only short trades, or the net performance of all trades. Because Portfolios are by nature long-only, short-only, or net long+short, the choice of which to display in a plotted equity curve is ignored when a Portfolio is selected for plotting.

The user must select exactly what is plotted from among the following choices:

Individual trades - The equity of each individual trade is plotted. This plot is not usually very informative, but it can be handy for locating time periods of unusually fast or slow trading.

Equity only - The cumulative equity is plotted.

Market overlaid - The cumulative equity is plotted in black, and the log of the market price is overlaid in red. If multiple markets are present, the user must select the market to be overlaid.

Benchmark overlaid - The cumulative equity is plotted in black, and a benchmark cumulative equity curve is overlaid in red. *Benchmark overlaid* is probably the most informative plot in the equity curve family, as the benchmark shows the average equity that would be obtained by a trading system that makes random guesses for its trade decisions, but that makes the same number of long and short trades as the system whose equity is selected. This red benchmark curve shows how much the development process has used an imbalance in long and short positions to take advantage of any long-term trend in the market. The degree to which the actual equity curve exceeds the benchmark shows the degree to which the trading system is exhibiting intelligence in its choice of trades.

Hold overlaid - The cumulative equity is plotted in black, and the cumulative equity curve of a buy-and-hold system for a long-only plot, or a sell-short-and-hold system for a short-only plot, is overlaid in red. If the user selected a 'net long+short' plot, *Hold overlaid* makes no sense, as the red overlay will just be the equity curve of an always-neutral system, not very interesting! In fact, even if the user is plotting long-only or short-only equity, the *Hold overlaid* option is of questionable value, because the different amount of time spent in the market for the trading system and for the 'hold' system causes a serious difference in scaling, making visual interpretation difficult. The *Benchmark overlaid* option is far better than *Hold overlaid* in most situations.

Benchmark spread - The difference between the cumulative equity of the system being selected and that of the benchmark described under *Benchmark overlaid*, is plotted. This is the degree to which the performance of the trading system exceeds the average performance of a system that takes equal advantage of any trend in the market but that makes random guesses for its trade decisions. The primary advantage of this plot is to see if there are meaningful epochs when the spread expanded (a good thing) or contracted (a bad thing).

Hold spread - The difference between the cumulative equity of the system being studied and that of a buy-and-hold strategy (for a long-only plot) or a sell-short-and-hold strategy (for a short-only plot) is shown. Because comparing the equity of a trading system to that of a hold strategy is not generally useful due to scaling issues, the *Hold spread* plot is not recommended.

There are several issues to consider when plotting equity curves:

- The date corresponding to each position on the plot is that of the trade decision which will cause the equity change. This is obviously prior to the actual change in equity. This may be annoying to some users, although it can also be argued that flagging an equity change when it is 'in the box' even if not yet realized is the better approach. In any case, it is unavoidable, because equity curves are based on targets which can have distant look-aheads. In fact, some targets (such as the TSSB library's HIT OR MISS family) have an indefinite lookahead. Moreover, if the user imports targets from an external source, the lookahead is undefined. Thus, the only possibility is to plot an equity change at the moment that it is known to be in progress and guaranteed, rather than waiting until it has been attained.

- Benchmarks, which may optionally be overlaid with the equity curve, are computed differently for Models (including Committees and Oracles) versus Portfolios. For models, benchmark normalization is based on the number of long and short trades across the entire out-of-sample period, with all folds pooled. This is probably the best

approach, as pooling across what may be unusually active and inactive years yields stability. However, such pooling is not possible for Portfolios, because in general different Models will be used as Portfolio members in each fold. These Models may have very different target variances (which impacts their contribution to the benchmark) and they may have very different trading frequencies. Thus, there is no way to pool benchmarks across all folds. The benchmark must be evaluated separately for each fold, based on the Models present and the trades they make. This is not a disaster; in fact, some might argue that this approach should be used for Models as well as Portfolios. However, it does sometimes lead to seemingly anomalous behavior. For example, suppose some Model has no out-of-sample trades in a given year. The benchmark curve for this Model will nonetheless change during that year, while the contribution of this Model to a Portfolio's benchmark will be flat for this year. Reasonable users may argue whether this is a good or a bad thing, but the fact remains that for Portfolios there is no alternative if testing is to be honest.

- If multiple markets are present, equity curves and benchmarks may show sudden sharp jumps up or down. This effect, which is ugly and may be mistaken for erroneous operation, is really due to markets appearing and disappearing in the market universe. Some markets (i.e. Google) are young, appearing in trading in the last few years. Other markets may disappear when a company merges with another company or ceases doing business. As a result, major contributions to equity curves and benchmarks may come and go, resulting in large discontinuities in the curves.

Prediction Map

It can be informative to examine a map of how a pair of input variables maps to a prediction in a trained model. Obviously, this helps in understanding how a model operates, what it sees in the data that leads it to make decisions.

Beyond this, it can help the user distinguish between models that are acting on real information versus those that are modeling noise. A valid model will tend to have predictions that vary smoothly across the range of predictors, while those that model noise will tend to have predictions that clump into numerous small groups with irregular boundaries.

The following things should be kept in mind:
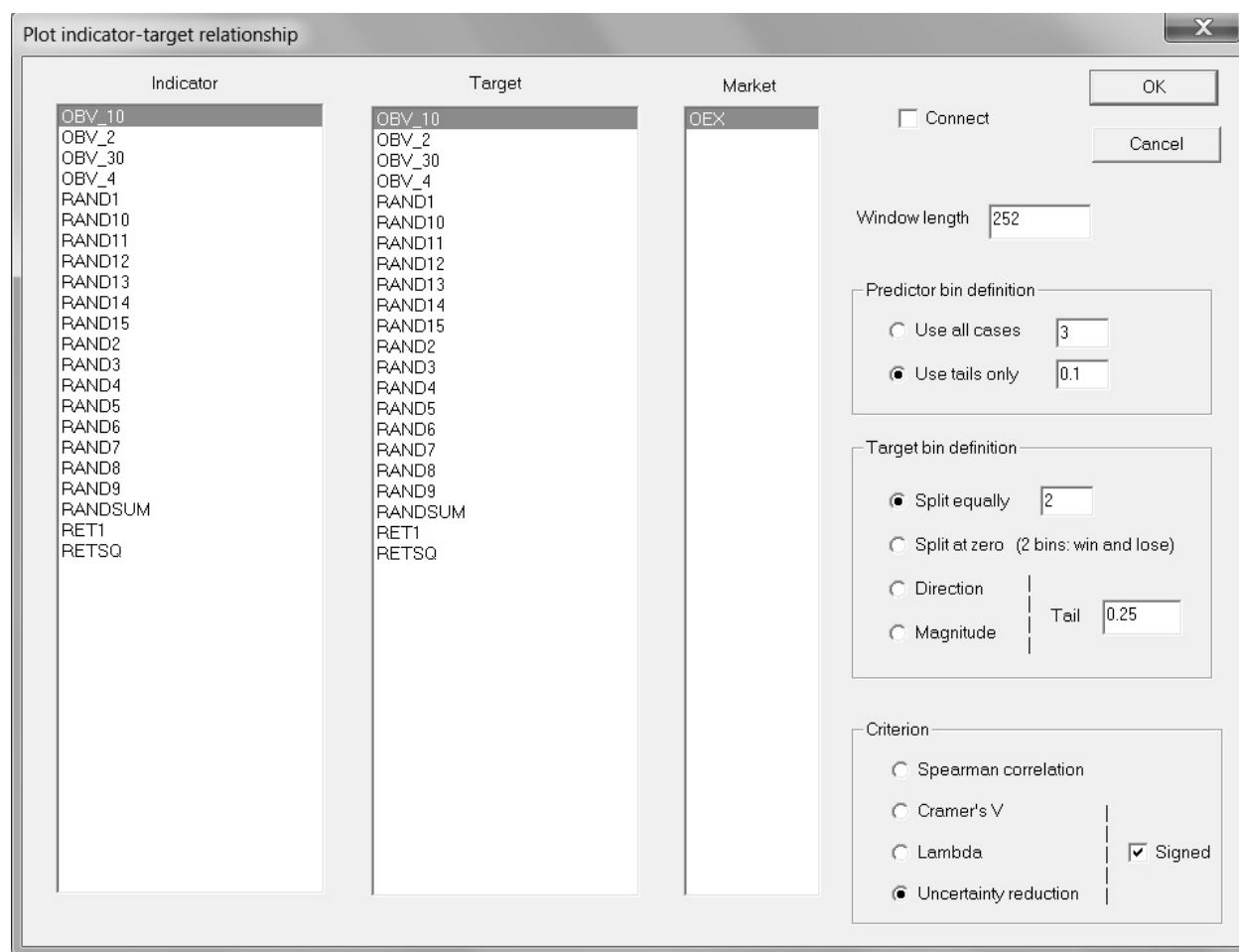
- Because the map is two dimensional, it can be applied only to models that use two predictors. Only two-input models are shown in the list of available models, and if no two-input models exist, the *Prediction Map* option is grayed out.

- It makes no sense to consider a prediction map for a committee or oracle, which uses model outputs as its inputs.

- The model displayed will be the most recently trained model. Thus, it is nearly always best to use a TRAIN command as the last command in the script file.

The user can set many options for the prediction map:

- By default, the full range of both predictors is used. The user may optionally limit the displayed range.

- The user can check a box that causes training cases to be displayed on the map. Those with a positive target value are printed as X, and those with a negative or zero target are printed with the letter O.

- By default, the map will be computed as a 51 by 51 grid. This should be fine for most applications. If you wish to print a very high resolution plot for publication, you can set a higher resolution, at the price of slower display time.

- By default, two colors will be used: red for positive predictions and blue for zero or negative predictions. By checking the "Multiple Tones" box, shading can be added. Red will still be used for positive predictions, and blue for negative and zero. But the brightest red and blue will be reserved for the most extreme ten percent of predictions of that sign. A dim brightness will be used for predictions below the median (in absolute value). Intermediate brightness will be used for the remainder. But see the next option, which impacts color assignment.

- The default color assignment mentioned above (red for positive and blue for negative) can be overridden by checking the "Split at median" box. This option is handy in filtering applications in which so many cases are a win that all predictions have the same sign. When this box is checked, the split for color assignment will occur at the median of the target rather than zero. If this box is not checked and all predictions have the same sign, a single color will be used even if the "Multiple Tones" box is checked, making the plot worthless. Thus, if all predictions have the same sign, the "Split at median" box should be checked in order to have a meaningful plot.

Indicator-Target Relationship

TSSB offers several screening algorithms for relatively quickly screening batches of indicators to assess their degree of relationship to a target. The chi-square test (Page 152) ranks indicators based on their individual relationships to the target, and the nonredundant predictor screening test (Page 158) ranks indicators in terms of their predictive power that is not redundant with other predictors. Both tests also offer statistical assessment of the probability that relationships are legitimate as opposed to good luck. But both of these tests benefit from a supplement, the ability to examine the consistency of a relationship across time. The *Indicator-Target Relationship Plot* allows the user to view a historical plot of the strength and nature of the relationship between an indicator and a target. The dialog for performing this test is as shown below:


Figure 1: Dialog box for Indicator-Target Relationship

The user inputs the following information:

Indicator - The independent variable for relationship computations

Target - The dependent variable for relationship computations

Market - If multiple markets are present, the user must specify the market to be plotted.

Connect - If this box is checked, the plot will be a single line consisting of individual line segments connecting the points. If it is not checked, the plot will be a separate vertical line segment connecting each point to the horizontal axis. For this application, this latter option (box checked) is generally more effective.

Window length - The length of the moving window used to compute the relationships. If the user specifies a length less than 2 it will be set to 2. If the length exceeds the number of data points it will be reduced to the number of data points. The plot is technically undefined for the first *window*-1 points, so these early points are all set equal to the value at the *window* point.

Predictor bin definition - The user can either select 'use all cases' and specify the number of bins to employ, or select 'use tails only' and specify the tail fraction, a number greater than zero and less than 0.5. In the former case, the range of the predictor is split in such a way that all bins contain the same number of data points. In the latter case, the specified fraction of extreme values of the indicator are kept in each tail. Thus, the total number of cases kept is twice the tail fraction. Choosing 'use tails only' is usually most appropriate because the majority of predictive information tends to be in the tails. A tail fraction of 0.05 to 0.1 is typical, as this will examine the cases having the 5 to 10 percent largest and 5 to 10 percent smallest values of the indicator.

Target bin definition -The user can select 'use all cases' and specify the number of bins to employ, select 'split at zero', select 'direction' and specify a tail fraction, or select 'magnitude' and specify a tail fraction. In the first case, the range of the target is split in such a way that all bins contain the same number of data points. In the second case, there are two bins, and the bin membership of each case is defined by the sign (win/lose) of the target. In many applications it is most effective to split the target into two or three equal bins. Three will correspond to a big win, a big loss, and an intermediate return that is relatively inconsequential. The third option (Direction) compares the left and right tails to determine the direction of the relationship based on tails only. The fourth option (Magnitude) compares the tails to the center of the distribution to assess the relationship between the indicator and the magnitude (absolute value) of the target).

Criterion - The user may choose from the following:

> *Spearman correlation* - A rank based (and hence immune to outliers) measure of monotonic correlation. This ranges from -1 (perfect inverse correlation) to +1 (perfect correlation).

> *Cramer's V* - A contingency-table-based measure of nominal (category) correlation which ranges form 0 (no correlation) to 1 (perfect correlation). See Page 159 for more information.

> *Lambda* - A contingency-table-based measure of nominal (category) correlation which ranges form 0 (no correlation) to 1 (perfect correlation). See Page 159 for more information.

> *Uncertainty redunction* - A contingency-table-based measure of nominal (category) correlation which ranges form 0 (no correlation) to 1 (perfect correlation). See Page 159 for more information. This, the default, is best in most applications.

> *Signed* - This is valid only for Cramer's V, Lambda, and Uncertainty reduction, which are, by definition unsigned (never negative). They measure the degree of correlation, but not its nature (positive or inverse). If the 'Signed' box is checked, the nature of the relationship between the indicator and the target is assessed in each window. If it is determined that the relationship is primarily inverted (large values of the indicator correspond to small values of the target) then the sign of the correlation measure is made negative. This checkbox is ignored if the user chooses 'Spearman correlation' because that measure is inherently signed.

Transforms

A *Transform* is an operation that results in the generation of one or more new variables that do not exist before the transform operation. Usually, but not always, the new variable is based on values of existing variables. Some transforms are flexible in that they need to be trained on a dataset before they can be invoked. Other transforms are deterministic, meaning that they are fully specified in the script file.

Transforms are integrated into the training/testing cycle system so that if a transform is not deterministic, but rather needs to be trained on existing data, training will be done without snooping into an out-of-sample fold. (But as always, the user is responsible for setting the optional OVERLAP parameter to account for overlap from historical indicators and/or future gaps.) When a transform is invoked as part of a walkforward or cross validation procedure, it will first be trained on the current training fold (if such training is required), and then all values of the output variable(s) will be computed for all cases in the dataset, including both the training fold and the upcoming test set fold.

The transform name will be the name used for the output variable(s) generated. If the transform produces more than one output variable, the integers 1, 2, 3, ..., will be appended to the transform name to define the names of the output variables.

Because transforms are integrated into the training/testing cycle system, The OVERLAP command (Page 123) is a legal option for all transforms except the RANDOM transform, which takes no options at all.

The RANDOM Transform

This transform generates random numbers in the range -1 to 1. It is invoked with the following command:

`TRANSFORM  TransformName  IS RANDOM ;`

At this time, no options are available. Options may be added in the future.

The NOMINAL MAPPING Transform

If a nominal predictor is suspected to actually be an ordinal variable but with unknown ordering, this mapping can be used to compute a function that maps class membership to an ordinal output. The syntax is as follows:

```
Transform TransformName IS NOMINAL MAPPING [ Specs ] ;
```

The model name may consist of letters, numerals, and the underscore character (_). The following specs (in addition to the usual OVERLAP option) are available:

```
FLAG INPUT  =  [ Var1 Var2 ... ]
```
(Optional, but some INPUT statement must appear) This lists two or more input variables that will be used as class flags. In typical operation, all of these variables will have the value zero except for one, which will have the value one. This identifies the nominal class membership of the case. However, the program actually defines class membership more generally by saying that the class is determined by whichever variable has the largest value.

```
NOMINAL INPUT  =  Variable
```
(Optional, but some INPUT statement must appear) This names a single variable whose value defines the nominal class membership. This must be a TEXT variable (Page 8).

```
TARGETVAR  =  Variable
```
(Mandatory) This names a variable, usually a future variable, that will be used to train the mapping. The nominal-to-ordinal mapping will be defined so as to maximize a measure of correlation between the mapped values and the target variable.

```
GATEVAR  =  Variable
GATEVARS  =  Variable1 Variable2
```
(Optional) It may be the case that a single mapping does not suffice for all possibilities in the application. Specifying a single gate variable allows for two separate mappings, one for positive values of the gate variable, and another for nonpositive values. Specifying two gate variables allows for four mappings as defined by the four possible combinations of positive versus nonpositive for the first and second gates.

IGNORE = (Value1 Value2 ...)

> *(Optional) It may be that one or more values of the first gate variable indicate an 'undefined' condition. These values can be listed here. For cases whose value of the first gate variable equals a value in this list, the output variable is set equal to a neutral 'median' value. Values of the second gate variable, if used, cannot be ignored.*

MAPPING = MEAN PERCENTILE

> *(Optional, but a mapping method must be specified) For each category (nominal class combined with any gates) the output value generated is equal to the mean percentile rank of all target values in that category. This is usually the preferred mapping, although it does assume that the target mapping carries some interval information. This is usually a valid assumption.*

MAPPING = RANKED PERCENTILE

> *(Optional, but a mapping method must be specified) The MEAN PERCENTILE mapping is computed, but the category values are then ranked across all categories. The output value is the percentile rank across categories. This is less powerful than the MEAN PERCENTILE method, but it should be used if no interval information in the target can be assumed.*

KEEP Number PERCENT

> *It may be that only categories that produce extreme responses are considered useful. Those the produce only a small deviation of the target from its median should be considered noise. This number, less than 50 percent, keeps as few classes as possible at each extreme such that at least this percent of cases at each extreme are represented.*

The audit log will contain a table showing the mean target percentile for each category. If a gate is used, the name of the class will be followed by a plus sign to indicate the class corresponding to positive values of the gate, and a negative sign for nonpositive values. If two gates are used, the first sign applies to the first gate, and the second sign applies to the second gate.

The EXPRESSION Transform

It is possible to use simple arithmetic operations to combine existing variables and/or market prices to produce new variables. These variables are entered into the database as soon as they are defined, so it is legal to reference prior computed values in an expression transform. The syntax is as follows:

Transform TransformName IS EXPRESSION (MaxLag) [Specs] ;

MaxLag will be discussed later. There is only one mandatory specification. This is:

TransformName   =   NumberExpression

This specification must appear exactly once, and it must be the last specification if more than one is used. The expression defines the arithmetic operations. The name of the computed variable is the name of the transform.

Here is a silly example:

TRANSFORM DUMDUM IS EXPRESSION (0) [DUMDUM = 5] ;

This will generate a new database variable called DUMDUM which has the value 5 for every case.

The only other specification available is used to define a temporary variable (not permanently saved in the database) that may then appear in the output specification. This is:

TempVarName = Expression

Output or temporary expressions may reference any of the following quantities:

Existing VarName - The name of a variable already in the database, or a previously defined expression, or a temporary variable name

@OPEN : Market - This is the opening price of a bar. *Market* names a market, and it (as well as the colon) may be omitted if there is only one market. Note that the market must have been read for this to make sense. *@HIGH*, *@LOW*, *@CLOSE*, and *@VOLUME* are also legal. For tick data, all four of the prices will be equal and hence interchangeable.

@OPEN : THIS - As above, except that in a multi-market situation, the market whose price is used for each record is that record's market as opposed to being fixed.

Many operations are available. Logical operations always return 1.0 if true and 0.0 if false. Values not equal to zero are treated as true, and zero is false. Common priorities of evaluation are observed, with priorities as shown below. Operations are grouped with dashes (------) from last evaluated to first evaluated. However, in case of doubt, use of parentheses is advised.

|| Logical: Are either of the quantities true?
&&   Logical: Are both of the quantities true?

==   Logical: Are the quantities equal?
!=   Logical: Are the quantities not equal?
<   Logical: Is the left quantity less than the right?
<=   Logical: Is the left quantity less than or equal to the right?
>   Logical: Is the left quantity greater than the right?
>=   Logical: Is the left quantity great than or equal to the right?

+   Sum of the two quantities
-   The left quantity minus the right quantity
----
*   Product of the two quantities
/   The left quantity divided by the right quantity
%   Both quantities are truncated to integers; the remainder of the left divided by the right
**   The left quantity raised to the power of the right quantity
----
-   Negative of the quantity
!   Logical: reverses the truth of the quantity
ABS( ) Absolute value of the quantity
SQRT( ) Square root of the quantity
CUBERT( ) Cube root of the quantity
LOG10( ) Log base ten of the quantity
LOG( ) Natural log of the quantity
POWER( ) Ten raised to the power of the quantity
EXP( ) Exponentiation (base e) of the quantity
ATAN( ) Arc-tangent of the quantity
LOGISTIC( ) Logistic transform of the quantity
HTAN( ) Hyperbolic tangent of the quantity

Here are a few examples of EXPRESSION transforms:

Compute the difference between two existing predictors:

```
TRANSFORM DIFF12 IS EXPRESSION (0) [ DIFF12 = X1-X2 ] ;
```

Compute the high minus the low for IBM:

```
TRANSFORM HIGHLOW IS EXPRESSION (0) [
    HIGHLOW = @HIGH:IBM - @LOW:IBM
    ] ;
```

Compute *BIZARRE* as X3 if X1>X2, or X4 otherwise:

```
TRANSFORM BIZARRE IS EXPRESSION (0) [
    X1BIGGER = X1 > x2
    X2BIGGER = ! X1BIGGER
    BIZARRE = X1BIGGER * X3 + X2BIGGER * X4
    ] ;
```

Here is a pair of expressions from my test suite. The first just sets *Exp5* equal to the open of IBM. The second subtracts the open of each record's market from *Exp5*. The resulting quantity *Exp6* is zero for IBM and more or less random for all other markets.

```
TRANSFORM Exp5 IS EXPRESSION (0) [
    Exp5 = @OPEN:IBM
    ] ;
```

```
TRANSFORM Exp6 IS EXPRESSION (0) [
    Exp6 = Exp5 - @OPEN:THIS
    ] ;
```

Vector Operations in Expression Transforms

Database variables and market variables may be referenced with a lag, and they may also be used to generate vectors. To lag a variable, append the lag in parentheses. To generate a vector, append the lag and the vector length in parentheses. For example:

SLOPEVAR (5) is the variable *SLOPEVAR* five bars ago in this market.

SLOPEVAR (0 , 20) is a vector of the 20 most recent values of *SLOPEVAR* in this market.

If a lag and/or vector length results in references to values prior to the first value in the database or market, this first value is duplicated as needed to play the role of nonexistent prior values.

When the EXPRESSION transform is declared, the maximum historical lookback (taking into account lags and vector lengths) must be specified. This is an annoyance to the user, but it enables some valuable optimization that saves both memory and execution time. Larger values than required are legal but will cause excessive memory use and slower operation.

When a monadic operator such as SQRT() is applied to a vector, the result is a vector of the same length in which the operator is applied to each component individually.

When a dyadic operator such as + is applied to a pair of vectors, these vectors should be the same length. Results are undefined if the lengths are different. The result is a vector of this same length in which the operator is applied to corresponding elements pairwise.

When a dyadic operator is applied to a vector and a scalar, the result is a vector of the same length in which the scalar is paired with each element of the vector individually.

The following operations take a vector as an argument and return a scalar:

@MIN (vec) - The minimum of all values in the vector
@MAX (vec) - The maximum of all values in the vector
@RANGE (vec) - The range (max minus min) of the values in the vector
@MEAN (vec) - The mean of the values in the vector
@STD (vec) - The standard deviation of the values in the vector
@MEDIAN (vec) - The median of the vector
@IQRANGE (vec) - The interquartile range of the vector
@SIGN_AGE (vec) - The number of elements in the vector for which the sign remains the same as that of the first (historically most recent) element. Counting starts at zero. The value zero is treated as negative.
@SLOPE (vec) - The slope per point of a least-squares line fit to the vector
@LIN_PROJ (vec) - The predicted value of the current point based on a linear fit

Here is an example of using the *@SIGN_AGE* function in a way that seems reasonable but is not what is intended. This is followed by a demonstration of how to do it correctly.

Suppose you want to count the number of bars that the variable X is on the same side of its 50-bar moving average. When the current value crosses its moving average, this variable is to have the value zero. If it then stays on this side of its moving average for the next bar, the value increments to one. If it stays there yet again, the value is two. This is to continue up to a maximum count of 20, where it remains if the current value continues on the same side of the moving average. (This sort of truncation is vital to preventing heavy tails.)

On first thought, the following transform may seem reasonable. Note that we look back at 50 historical values, including the current. Thus, the maximum lag is 49.

```
TRANSFORM AGE_A IS EXPRESSION ( 49 ) [
   AGE_A = @SIGN_AGE ( X(0,20) - @MEAN(X(0,50)) )
   ] ;
```

Look first at the term **@MEAN(X(0,50))**. The **@MEAN** function takes a vector as its argument. In this case, the vector argument is **X(0,50)**, which is the most recent 50 values of X. The **@MEAN** function returns the scalar mean of the vector, which in this case provides the 50-bar moving average.

This scalar quantity is subtracted from each element of *X(0,20)*, the vector of the 20 most recent values of X. Finally, the @SIGN_AGE function counts back from the most recent element of this vector, seeing how far it can get before the sign changes (the value crosses the moving average). The argument to this function is the value of X minus its 50-bar mean. The value returned by this function is the count of how many historical differencess have the same sign as the most recent difference.

What's wrong with this transform? It's perfectly legal, and it returns a result that may look reasonable. However, it does not do exactly what you want. This transform computes its count by examining the value of each of the differences between the most recent 20 values of X and the 50-bar moving average based on a window ending *on the current bar*. Your goal, as stated at the beginning of this example, is to examine the difference between a historical case and the 50-bar moving average *behind the case being examined* as opposed to the moving average *behind the current bar*.

In order to do this the way you wish, which is also the most sensible way, you need to perform two transforms in sequence. The first computes the difference between each case and its trailing 50-bar moving average. The second counts the sign-change age. This is performed as shown below:

```
TRANSFORM PRICE_DIFF IS EXPRESSION ( 49 ) [
   PRICE_DIFF = X - @MEAN(X(0,50))
   ] ;

TRANSFORM AGE_B IS EXPRESSION ( 19 ) [
   AGE_B = @SIGN_AGE ( PRICE_DIFF(0,20) )
   ] ;
```

Consider the **PRICE_DIFF** transform. For each case, it computes the trailing 50-bar moving average and subtracts this from the current value of X. Note that if you do not want the current case to be included in the moving average, you would use **X(1,50)** instead of **X(0,50)**. In this situation, you must bump up the maximum lag from 49 to 50.

The **AGE_B** transform then examines the series of differences and counts the age of signs.

Logical Evaluation in Expression Transforms

It is possible to evaluate the truth of a logical expression and choose the execution path accordingly. This is done with the following expressions:

IF (NumberExp) {
   Commands executed if NumberExp is nonzero
   }

ELSE IF (NumberExp) {
   Commands executed if NumberExp is nonzero and all prior if expressions are zero
   }

ELSE {
   Commands executed if all prior if expressions are zero
   }

Note:

1) curly braces {} are required for all logical and looping operations, even if the operation encloses just one command.

2) Logical expressions may be nested.

3) Remember that the final output statement must be the *last* statement in the transform, so it cannot occur inside a logical expression.

Here is an example of the use of logical expressions. If long and short trends are both positive, the computed value is +2. If the longer is positive but the short is not, the returned value is +1. The same rules apply for negative trends. Finally, if the longer trend happens to be zero, the returned value is zero.

```
TRANSFORM DoubleTrend IS EXPRESSION ( 0 ) [
   IF (LIN_ATR_15 > 0) {
      IF (LIN_ATR_5 > 0) {
         RETVAL = 2
         }
      ELSE {
         RETVAL = 1
         }
      }
   ELSE IF (LIN_ATR_15 < 0) {
      IF (LIN_ATR_5 < 0) {
         RETVAL = -2
         }
      ELSE {
         RETVAL = -1
         }
      }
   ELSE {
      RETVAL = 0
      }
   DoubleTrend = RETVAL
   ] ;
```

The LINEAR REGRESSION Transform

It can be useful to use linear regression to combine two or more existing predictors into a single new predictor which carries the combined predictive information from the input predictors. For example, we might have several measures of volatility. Rather than feeding all of them to a model, it might be better to find a linear combination of them that has maximum correlation with a target variable, and feed only this single weighted average to the model. The command for computing a linear regression transform is as follows:

`Transform TransformName IS LINEAR REGRESSION [Specs] ;`

This transform has two mandatory specifications:

`INPUT  =  [ VarName1  VarName2 ... ] ;`
> *This names two or more variables that will take part in a weighted average to compute a new variable that has a least-squares fit to the target variable.*

`TARGETVAR  =  VariableName ;`
> *This names a single variable that will be the target for the least-squares fit. This target variable will frequently be the same target that will be used for training subsequent models, but it need not be. It doesn't even need to be a profit. It can be, for example, a future slope.*

The QUADRATIC REGRESSION Transform

This transform is identical to the LINEAR REGRESSION transform except that in addition to the linear terms, squares and all possible cross products are included in the fit. Since the number of cross products blows up rapidly as the number of inputs increases, the number of inputs should be kept small, perhaps even as small as just two. Syntax is as follows:

`Transform TransformName IS QUADRATIC REGRESSION [Specs] ;`

Note that all linear, square, and cross-product terms take part in the regression. This is in contrast to the QUADRATIC model, in which the terms retained are selected by internal cross validation.

The PRINCIPAL COMPONENTS Transform

This transform is used to perform either principal components analysis or factor analysis by means of Varimax rotation of a subset of the principal components.

Suppose we measure two or more indicator variables. In a great many applications, the bulk of the variation across cases occurs in a space of dimensionality lower than the space defined by all of the indicators. Also, it is usually this reduced space that contains the majority of the useful information (although notable exceptions do occur).

The more indicators we throw at a model, the more likely it is that we will suffer overfitting. Thus, it may be in our best interest to compute new indicators from a space of reduced dimensionality and use only these indicators in the model. For example, we may measure three different forms of volatility at four different history lengths. This generates twelve indicators. Principal component analysis may let us distill the information contained in these twelve indicators down to just a few new indicators.

Once we've defined a reduced dimensionality space, we have a choice for computing new indicators from within that space: rotate or do not rotate. If the indicators will all be fed to a linear model, the choice is unimportant because they will give identical results (except for trivial rounding errors in computation). However, if the reduced set of indicators is given to a linear model via stepwise selection, or if they are given to a nonlinear model, the choice can have an impact on performance.

The straightforward and default choice is to stick with principal components. In this case, the first new indicator will define the single dimension along which the majority of the variation occurs across all cases. The second indicator will capture the second-most amount of variation which is orthogonal to (uncorrelated with) the first. This repeats for as many dimensions as you desire.

Optionally, you can request Varimax rotation of the reduced space. You must have at least two dimensions to do this, and it usually makes little sense (although it is legal) to rotate if you have kept all of the original dimensions. This rotation scheme preserves exactly the same information as is contained in the reduced space before rotation. However, the ordering of most-to-least variation is sacrificed in order to (usually) obtain factors that are easier to interpret than principal components. These rotated factors tend to have correlations with the original indicators that are either near +/- one, or near zero. In other words, rotation tends to produce factors that represent subsets of the original indicators.

This transform is invoked with the following command:

Transform TransformName IS PRINCIPAL COMPONENTS [Specs] ;

The following specifications may be included:

INPUT = [VarName1 VarName2 ...]
> *(Mandatory) This names two or more variables that will take part in the dimension reduction.*

N KEPT = Integer
> or
N KEPT = ALL
> *(One of these is mandatory) This specifies how many dimensions to keep.*

ROTATION = VARIMAX
> *(Optional) If specified, Varimax rotation will be applied to the space in order to drive weights toward zero or extremes. If this is not specified, principal components will be output.*

Here is a small example from some EWAVES program output. I used four indicators: the closely related D_COUNT and D_WAVEDAYS variables at degrees 3 and 4. Here is the script file command to do the transform:

```
TRANSFORM TPC IS PRINCIPAL COMPONENTS [
   INPUT = [D_COUNT_3 D_WAVEDAYS_3 D_COUNT_4 D_WAVEDAYS_4]
   N KEPT = 3
   ROTATION = VARIMAX
   ] ;
```

This states that we will reduce the dimensionality from four to three, and then perform Varimax rotation. The following table of factor loadings (correlations between the factors and the original variables) is produced:

	1	2	3	Comm
Percent	60.5	34.0	3.5	
Cumulative	60.5	94.5	98.0	
D_COUNT_3	0.811	-0.550	0.005	96.0
D_WAVEDAYS_3	0.789	-0.582	0.000	96.1
D_COUNT_4	0.750	0.606	0.264	100.0
D_WAVEDAYS_4	0.761	0.592	-0.266	100.0

The following items should be noted in the table just shown:

- The first factor alone accounts for over 60 percent of the variation in the dataset. This factor is essentially a pooled measure of time in a wave, as it has high correlation (around 0.8) with all four original indicators.

- The second factor accounts for another 34 percent of the variation. These first two new indicators together account for almost 95 percent of the total variation! This factor has positive correlation with the two Degree-4 indicators, and negative with the two at Degree 3. In other words, it is a measure of the contrast between what's happening at Degree 4 and what at Degree 3.

- The third factor accounts for only 3.5 percent of the total variation. It contrasts the two different measures of wave time at Degree 4. It essentially ignores Degree 3.

- The last column in a loading chart is always Communality. This is the percent of variation of each original indicator that is captured in the reduced space. Note that this reduced space captures all of the Degree-4 variation, and almost all of that at Degree 3.

Because the ROTATION = VARIMAX option appeared, the program also displays the rotated loading matrix:

	1	2	3	Comm
Percent	48.1	46.4	3.5	
Cumulative	48.1	94.5	98.0	
D_COUNT_3	0.968	0.153	0.001	96.0
D_WAVEDAYS_3	0.973	0.115	0.006	96.1
D_COUNT_4	0.134	0.957	-0.256	100.0
D_WAVEDAYS_4	0.149	0.950	0.274	100.0

Note the following aspects of the rotated loadings:

● After rotation, we still capture 98 percent of the total variation. Rotation does not change the total, although the individual factors need not remain ordered largest to smallest.

● The communalities always remain the same after rotation. This is because it's still the subspace. Only the axes have rotated.

● After rotation, the first factor (the largest here only by chance; order is lost after rotation) essentially tells us what's going on at Degree 3. The second factor captures information from Degree 4. The third is pretty much unchanged by the rotation.

Remember that rotation does not alter the information content of the reduced set. It only changes how we look at it and how we compute the new indicators.

Finally, it may (rarely) be interesting to see the actual coefficients that are used to transform the standardized original indicators into the new factors. Note that these will often be quite different from the loadings, which are correlations, not weights. Here is the score coefficient matrix:

	1	2	3
D_COUNT_3	0.521	-0.066	-0.034
D_WAVEDAYS_3	0.530	-0.090	0.001
D_COUNT_4	-0.069	0.552	-1.877
D_WAVEDAYS_4	-0.078	0.518	1.897

The ARMA Transform

An ARMA (autoregressive, moving average) model can be fit to a moving window in the database or a market, and any of several quantities can be computed to generate a new database variable. The ARMA transform is invoked as follows:

```
Transform TransformName IS ARMA [Specs] ;
```

The following specifications may be included:

SERIES = Source

This specifies the series which will be fit with an ARMA model. The following sources are available:
VarName - The name of a variable in the database. If multiple markets are present, each market will be processed independently.
VarName:Market - The name of a variable in the database, using values in the specified market. If multiple markets are present, the specified market's computed values will be cloned onto each market.
@CLOSE - The close of the market will be used. Legal only if there is just one market.
@CLOSE:Market - The close of the named market will be used.
For the above market series, @OPEN, @HIGH, @LOW, and @VOLUME are also legal.

ARMA WINDOW = Integer

This specifies gives the length of the data window. If the ARMA model has only one parameter, a window length of as little as 20 is acceptable, although larger windows, such as 100, are more stable. If two parameters are estimated, a window length of 100 should be considered a minimum.

AR = Integer

This specifies the number of AR parameters.

MA = Integer

This specifies the number of MA parameters.

ARMA OUTPUT = Type

This specifies the value that will be generated for the database. The following types are available:
PREDICTED - The predicted value of the last case in the window
SHOCK - The shock experienced by the last case in the window
STANDARDIZED SHOCK - The shock divided by the standard deviation of shocks
MSE - The mean squared error (variance of the shocks) in the window
RSQUARE - The fraction of series variance that is predictable by the ARMA model
AR PARAMETER Integer - The value of the trained AR parameter at the integer lag
MA PARAMETER Integer - The value of the trained MA parameter at the integer lag

ARMA TRANSFORM = LOG

> *This optional specification decrees that the source series will be transformed by taking natural logs before the ARMA model is fit. The source series must be strictly positive.*

ARMA TRANSFORM = DIFFERENCE

> *This optional specification decrees that the source series will be differenced before the ARMA model is fit. Differencing will often remove or greatly decrease nonstationarity of the mean of a series.*

ARMA UNDO TRANSFORM

> *This option makes sense only if at least one of the above transforms is performed and the output is either PREDICTED or a SHOCK. If this option is not used, the predicted value (and hence the derived shock) remains in the transformed domain. If this option is used, the transform is undone for the prediction, and the shock computed accordingly. Note that if the only transform is a difference, the shock will be the same regardless of whether or not the transform is undone.*

For example, the following transform will fit an ARMA model containing just one AR parameter (a common, stable, and usually effective model; an excellent first choice) and create a new indicator whose value is the AR parameter, which indicates the degree and sign of lag-one serial correlation in the closing value of OEX. This may be useful predictive information because it indicates whether the market is in a trending or mean-reversion mode. Since this is a market, it makes sense to first do a log transform and then difference so that the ARMA model is operating on changes, not actual values.

```
TRANSFORM OEX10_LD_AR IS ARMA [
   SERIES = @CLOSE:OEX
   ARMA WINDOW = 100
   ARMA TRANSFORM = LOG
   ARMA TRANSFORM = DIFFERENCE
   ARMA OUTPUT = AR PARAMETER 1
   ARMA AR = 1
   ARMA MA = 0
   ] ;
```

The PURIFY Transform

It is a universal fact of life in predictive modeling that useful predictive information in an indicator is diluted by variability attributed to other, less relevant phenomena. Such phenomena may contribute little or no useful predictive information, and hence be essentially noise. If the developer is able to identify a potential source of the noise pollution, it may be possible to remove it and thereby purify the original indicator, increasing its predictive power.

The method for doing this is straightforward: first one must identify a series, measured concurrently with the 'polluted' indicator, which is believed to be a source of pollution. This series, called the *purifier*, may be prices of a market, or an indicator itself, or some other market measure such as volume or open interest. Then apply one or more moving-window functions to the purifier series. Find a model (linear is fast and easy) which does a decent job of using these functions of the purifier to predict the indicator being purified. Finally, subtract the predicted values of the indicator from the actual values of the indicator. This predicted value is, to at least some degree, the pollutant. Thus, subtracting it from the series to be purified has the effect of reducing the pollution in the indicator, leaving behind a higher concentration of useful information.

The following command is used to purify a series:

```
TRANSFORM TransformName IS PURIFY [Specs] ;
```

Some of the specifications are mandatory, and others are optional. All possible specifications are now discussed, grouped by their uses.

Defining the Purified and Purifier Series

The user must specify two series. These are the series to be purified, and the purifier series. As with all transforms, the generated output series is given the name of the transform, *TransformName*, in the definition. The purified and purifier series may be any of the following:

- **The open, high, low, close, or volume of a market**. The purified series is invariably an indicator, not actual trading prices of a market. Nonetheless, it is often advantageous to either read these two series as TSSB markets (READ MARKET HISTORIES) or let them be 'markets' that have been operated on by a prior transform. Computing them from the built-in library, or reading them in from an external database, can, in some instances, deprive the user of valuable early history information. This will be discussed in detail later in this section.

- **An indicator computed from the built-in library**. If this is done, the length of early history available to the transform will be reduced by the maximum lookback in the variable definition list (READ VARIABLE LIST). If any internally computed indicator has a longer lookback than the indicator(s) needed for the purified or purifier series, data will be wasted. Thus, use of the internal library may better be done in a separate script file if available history is limited. The separately computed indicators and the PURIFY transform values can be merged for training and testing of trading systems by means of an APPEND DATABASE command.

- **A variable read in from an external data file with the APPEND DATABASE command.** If this is done, the user should ensure that the external data file starts at as early a date as possible in order to maximize the length of history available to the transform. The appending is done *before* transforms are computed, so even if extensive historical data is available prior to appending, if the appended data file starts at a later date, the previously available data will become unavailable. Also, realize that the PURIFY transform, like all transforms, preserves market associations. Thus, the file read should contain data for any market(s) that will ultimately be traded.

Exactly one PURIFIED and one PURIFIER specification are required. The following commands are used to define the purified and purifier series:

PURIFIED = @close:MarketName

This names a market whose close will be purified. It is also legal to use the 'open', 'high', 'low', or 'volume' of a market. Because the purified series is virtually always an indicator, in practice this series will not actually be trading prices of a market. Rather, it will be an indicator that is read in as if it were a market by means of the READ MARKET HISTORIES command. Usually this is the best approach because this method provides the most history to the transform, and also because data read in as a TSSB market is automatically cloned to all markets named in the READ MARKET LIST file, which simplifies model training and testing.

PURIFIED = VariableName

This names an indicator that will be purified. This indicator may have been computed from the built-in library, or computed from a prior transform in the script file, or read in from an external database. One powerful technique which gives the user great versatility but does not generally cost availability of historical data is to read in a variable as a TSSB market, apply an expression transform, and then use the result of this expression transform as the purified or purifier series.

PURIFIER = @close:MarketName
PURIFIER = VariableName

The same considerations just discussed for the purified series apply to the purifier series, with one small exception. The only difference is that although the purified series will nearly always be an indicator, the purifier series will often be actual market price history.

PURIFY USE LOG PURIFIER

> *This optional command decrees that the natural log of the purifier series will be taken before predictor functions are computed from the series. This is usually appropriate when the purifier series is market price histories, and usually not needed when the purifier series is an indicator.*

Specifying the Predictor Functions

The user must specify at least one predictor function and at least one lookback length for the predictor function(s). The user must also specify whether one or two predictors are to be used in the linear model that uses the functions to predict the purified series. The relevant specifications are the following:

PURIFY N PREDICTORS = Integer

> *This is the number of predictors (functions of the purifier series) that the purification model will use to predict the purified series. It must be 1 or 2. If 2 is specified, all possible pairs of predictors will be tested, so execution time could be slow if the user employs numerous predictor candidates.*

PURIFY LOOKBACKS = (Integer Integer ...)

> *This list of one or more integers specifies the lookbacks (window lengths) in the purifier series to use when computing the predictor functions. Each must be at least 3. This specification must appear if any predictor function other than VALUE (defined below) is used.*

PURIFY PREDICTOR FAMILY = TREND

> *This makes available to the purification model the linear trend of the purifier series, measured over each lookback distance given in the PURIFY LOOKBACKS specification.*

PURIFY PREDICTOR FAMILY = ACCELERATION

> *This makes available to the purification model the quadratic component (rate of change in trend) of the purifier series, measured over each lookback distance given in the PURIFY LOOKBACKS specification.*

PURIFY PREDICTOR FAMILY = ABS VOLATILITY

> *This makes available to the purification model the volatility of the purifier series, measured over each lookback distance given in the PURIFY LOOKBACKS specification. The volatility here is the exponentially smoothed absolute change.*

PURIFY PREDICTOR FAMILY = STD VOLATILITY

> *This makes available to the purification model the volatility of the purifier series, measured over each lookback distance given in the PURIFY LOOKBACKS specification. The volatility here is the standard deviation of the changes in the purifier series, annualized by multiplying by sqrt(252).*

PURIFY PREDICTOR FAMILY = Z SCORE

This makes available to the purification model the moving z-score of the purifier series, measured over each lookback distance given in the PURIFY LOOKBACKS specification. The z-score is found by computing the mean and standard deviation of the purifier series across the lookback window, subtracting the mean from the current value of the series, and dividing by the standard deviation. Nonlinear compression combined with hard limiting is used to prevent extreme values due to tiny standard deviations.

PURIFY PREDICTOR FAMILY = SKEW

This makes available to the purification model an order-statistic-based measure of skewness.

PURIFY PREDICTOR FAMILY = VALUE

This predictor specification is different from all of the others. The VALUE predictor makes available to the purification model the current value of the purifier series. It is not a function, and hence lookback distance is irrelevant. The VALUE option would almost never make sense when the purifier series is market price history. However, it may be that the user has contrived a special predictor for the purification model, a predictor that is designed to be used itself, as opposed to functions of it being used. In this case, the VALUE predictor would be specified. The PURIFY USE LOG PURIFIER option is ignored for the VALUE predictor; the actual value is always used.

Miscellaneous Specifications

This section discusses the PURIFY transform specifications that do not fit in the prior categories. These are as follows:

PURIFY WINDOW = Integer

This mandatory line specifies the number of cases (the lookback length) that are used to train the purification model. It must be at least 3, and is usually much larger, perhaps as much as several hundred.

PURIFY NORMALIZE = PERCENT

By default, the output of the PURIFY transform is the actual value of the purified series minus the predicted value. But if the PURIFY NORMALIZE = PERCENT option appears, this difference is divided by the absolute value of the purified series and then multiplied by 100. This expresses the difference as a percent of the actual value. If, as is often the case, the local standard deviation of the purified series is proportional to its absolute value, the PERCENT normalization can stabilize the variance of the transform output. Note that purifying the log of such a series also produces variance stabilization, and usually does so better than the PERCENT option.

PURIFY NORMALIZE = STDERR

> *By default, the output of the PURIFY transform is the actual value of the purified series minus the predicted value. But if the PURIFY NORMALIZE = STDERR option appears, the difference is divided by the standard error of the estimate produced by the purification model. Note that near the beginning of the available history, when the model has incomplete information, the standard error can drop to near zero, producing huge z-scores. For this reason, the output of the PURIFY transform is strongly constrained near the beginning of the data series, and moderately constrained thereafter to prevent extreme values that can hinder subsequent model training.*

PURIFY OUTPUT = FIRST PREDICTOR

> *This option is for diagnostic purposes only. It disables purification. The output of the PURIFY transform when this option is employed is the first PURIFY PREDICTOR FAMILY variable named in the option list, computed at the first lookback in the PURIFY LOOKBACKS list. For example, suppose the user had the following option as the first predictor: PURIFY PREDICTOR FAMILY = Z SCORE. Then the output of the transform would be the exact z-score values used as predictors by the purification model. Subsequent predictors and lookbacks other than the first in the list are ignored. This option can occasionally be a handy way to inspect the predictors in order to verify visually (or even numerically) that they look like what you expect them to look like. It's unlikely that casual users would ever employ this option.*

Usage Considerations

The PURIFY transform is integrated into the train/test cycle like other transforms, and it mostly behaves the same as others. However, there are two aspects of the nature of purification itself that merit special consideration.

First, purification almost always has a very long total lookback length. It requires extensive history before it has the full amount of data needed to meet the user's specifications. This is because two windows are involved. At the outermost level we have the PURIFY WINDOW lookback which tells the transform how many cases will go into training the purification model. Then, for each case in this window, we have the windows for computing the predictor variables, the longest of which in the PURIFY LOOKBACKS list may be quite long. So when computing the purified value of any record (bar of data in a market), the oldest case in the training set for the purification model will be back in history by the PURIFY WINDOW distance, and in order to compute the predictors for this case we will need an additional historical window whose length is the longest lookback in the PURIFY LOOKBACKS list. It's not unusual to need a total lookback of several years, which is substantial if data is limited. Computation does begin immediately, without waiting for the full lookback to be reached. However, the first few values will be seriously in error due to massive lack of data, and it will probably be on the order of a hundred or so records before accuracy becomes even marginally usable. Full accuracy is not obtained until the entire total lookback distance is reached.

For this reason we are inspired to make as much history as possible available to the PURIFY transform. In most situations the easiest way to do this is to do the purification alone, in a single script file, and write the purified series as a database using the WRITE DATABASE command. Read both the purified and the purifier series as TSSB markets. Have a token variable list, probably containing just one variable definition which has an extremely short lookback. (This is needed because the current version of the program requires that either a variable list or database have been read before any transform is done. This requirement may be eliminated in a future version.) It is perfectly legitimate and often useful to then apply one or more expression transforms to the 'markets' read, and then purify the output of the transform. This does not cause the loss of history, so it is innocuous. On the other hand, early history of a length equal to the longest lookback in the variable definition list *will* be eliminated, which is why we want to keep the lookback in the variable definition list very short.

If the trading model(s) being developed require other indicators, they can be computed in a separate script file and saved to a database. Their lookbacks will cost market history, but that's not a problem, because the purification will have been done separately, and the history lost due to lookbacks from the internal indicator library will just be early 'warm-up' data in the purified series, data that is probably best discarded anyway!

Of course it's legal for the purified and/or purifier series to be computed from the internal library, or read in from an outside source with an APPEND DATABASE command. Just be aware of the potentially long lookback requirement of purification and try hard to provide historical data that begins at as early a date as possible.

The second issue of purification is that the purified and purifier series must be precisely concurrent in history if full accuracy is to be maintained. Suppose one series has missing data on a certain date. Then even though the ending dates of the two series are aligned to the 'current' date, the two series will remain aligned only as far back as the missing data. After that, the measurement dates of the two series will become misaligned and accuracy can suffer.

In order to prevent this from happening, the first time a PURIFY transform is encountered in the script file the database will be preprocessed to eliminate any dates that do not have a full complement of markets. When this happens, a line similar to the following will appear in the audit log:

```
Removing database records with incomplete markets reduced
cases from 11924 to 11918
```

In this example, the database originally contained 11924 records, but 6 of them did not have a complete set of records, leaving 11918 complete dates. This is a one-time operation.

A Simple Example

We now examine a simple two-part example. The first script file purifies the log of a 'market' called VIX, and the second file uses the purified value to make trade decisions. Here is the first script file:

```
READ MARKET LIST "OEX_VIX.TXT" ;
READ MARKET HISTORIES "E:\SP100\OEX.TXT" ;
READ VARIABLE LIST "VARS_PURIFY_DEMO1.TXT" ;

TRANSFORM LOG_VIX IS EXPRESSION ( 0 ) [
   LOG_VIX = LOG ( @CLOSE:VIX )
   ] ;

TRANSFORM PURIF_LOG IS PURIFY [
   PURIFIED = LOG_VIX
   PURIFIER = @close:OEX
   PURIFY USE LOG PURIFIER
   PURIFY WINDOW = 300
   PURIFY LOOKBACKS = ( 11 22 44 65 130 260 )
   PURIFY N PREDICTORS = 2
   PURIFY PREDICTOR FAMILY = TREND
   PURIFY PREDICTOR FAMILY = ACCELERATION
   PURIFY PREDICTOR FAMILY = VOLATILITY
   PURIFY PREDICTOR FAMILY = Z SCORE
   ] ;

TRAIN ;

WRITE DATABASE "T_PURIFY_DEMO2.DAT" ;
```

The market list OEX_VIX.TXT contains only two markets: VIX, which is the market sentiment series which will be purified, and OEX, the market index that will be the purifier series.

The variable list VARS_PURIFY_DEMO1.TXT contains only one token indicator:
DUMMY: LINEAR PER ATR 3 5

VIX tends to have a standard deviation proportional to its magnitude, so taking its log before purification stabilizes its variance, which is a crucial property of stationarity. Like all transforms (except ARMA), the expression transform used to compute the log of VIX does not cost any market history.

The purifier series is the close of OEX, and its log will be taken before any predictor functions are computed.

The purification model will be trained with a moving window of 300 cases. For each case in that window, six different lookbacks, ranging from 11 through 260, will be used to compute the predictor functions.

The model will be given the four named predictor functions, each evaluated at each of the six lookbacks, making a total of 24 predictor candidates. The two best predictors will be found.

The TRAIN command causes the purification to be performed across the entire database at once, and the results will be written to a database file.

Here is the second part of this example, in which the database just created is read, and a trading model is walked forward:

```
READ MARKET LIST "OEX.TXT" ;
READ MARKET HISTORIES "E:\SP100\OEX.TXT" ;
READ VARIABLE LIST "VARS_PURIFY_DEMO2b.TXT" ;
APPEND DATABASE "T_PURIFY_DEMO2.DAT" ;

MODEL MOD_PURIF IS LINREG [
    INPUT = [ PURIF_LOG ]
    OUTPUT = RETURN
    MAX STEPWISE = 0
    CRITERION = PROFIT FACTOR
    MIN CRITERION FRACTION = 0.1
    RESTRAIN PREDICTED
    ] ;

WALK FORWARD BY YEAR 10 2000 ;
```

This time, the market list contains only one market, OEX. This is because we want to trade only OEX in this example. If VIX, as it exists in the market history file, were tradeable and the file contained the actual trading prices, we might (or might not) be interested in also including it in the market list. If this were done, all performance results would include trades in VIX as well as OEX.

The new variable list VARS_PURIFY_DEMO2b.TXT contains just one line:

```
RETURN: NEXT DAY ATR RETURN 252
```

This variable is needed because it is the target for training and testing the model. It would have been legal to place this line in the variable definition list for the first half of this example, in which the purification was done. If so, the RETURN variable would have been computed there, written to the database, and read in this example, saving the trouble of having this separate variable list. However, the RETURN variable has a lookback of 252 days. If it had been computed in the purification script, its entire lookback of 252 days would have been unavailable to the purification transform, a significant loss.

After this one-line variable list is read, we append the database that was computed in the first half of this example. This contains the purified VIX for both the OEX and VIX markets. However, since the existing database contains only OEX, all VIX records will be eliminated during the APPEND operation, leaving the model to be trained and tested on only OEX.

The model is trained and tested using purified VIX as the sole indicator, and RETURN as the target. The CRITERION is irrelevant because it is used for stepwise selection only, and this is a single-indicator model. Nonetheless, it must be included in any model definition.

Finally, the model is walked forward from 2000 using 10-year training sets.

Models

One or more models can be defined in a script file, and the model(s) can be trained or tested by cross validation. Models cannot be defined or tested from the menu.

The format of a model definition is as follows:

MODEL   ModelName IS ModelType   [ModelSpecs   ]   ;

The model name may consist of letters, numerals, and the underscore character (_). The model type may be LINREG for ordinary linear regression, QUADRATIC for quadratic regression, OPSTRING for Operation String Prediction, GRNN for General Regression Neural Network, or MLFN for Multiple-Layer Feedforward Network. Note that training time for a GRNN can be extremely long. Also, the following tree-based models are available: TREE, FOREST, and BOOSTED TREE. Finally regime regression can be implemented with the SPLIT LINEAR model.

Some of the model specs are mandatory, and others are optional. Note that no semicolon appears, even when (as is usual) the specs are on separate lines. Model specs include the following:

INPUT   =   [Var1   Var2   ...   ]
(Mandatory) This lists the input variables that are candidates for stepwise inclusion. You may also indicate one or more contiguous ranges of variables by separating two names with a dash. In this case, all variables that lie between (and including) the variables as defined by the order in the variable definition file will be included. For example, specifying MA10-MA80 would include these two variables, as well as all variables between them. You may also specify one or more families by including one or more family specifiers enclosed in <> brackets. All conditions inside the brackets must be satisfied in order for a variable to be included. Legal conditions are:
> *NAME = "FamilyName"*
> *LOOKBACK = (LowLim HighLim)*
> *INDEX = NONE / IS / MINUS*
> *NORMALIZE = NONE / CENTER / SCALE / BOTH*
> *CROSS = YES / NO*
See Page 132 for examples. Note that if the model is a COMPLEX INPUT or COMPLEX HIDDEN MLFN, stepwise variable selection will not be done. All variables will be included.

EXCLUSION GROUP = Integer

> *(Optional) If this model will be used in a committee, and the user wishes to exclude inputs that already appear in prior models (to avoid redundancy), use this command with any positive integer. Models that have the same exclusion group number will be guaranteed to not share any predictors.*

OUTPUT = VarName

> *(Either this or the next OUTPUT command is mandatory.) This names the variable that is to be predicted.*

OUTPUT = MODEL ModelName RESIDUAL

> *(Either this or the prior OUTPUT command is mandatory.) This command specifies that this model is to predict the residual error from the named model. The named model must have appeared already in the script file. In other words, you cannot make a forward reference, naming a model that has not yet been defined. If a PRESERVE PREDICTIONS command (Page 21) appears later, the values assigned to this predicted variable will be the predicted value of the target, taking into account this and any prior residual predictions, rather than the predicted residual itself.*

PRESCREEN VarName < or > or <= or >= Number

> *(Optional) This command specifies that only cases whose value of the variable VarName is less than (or greater than, or also equal to) the specified value will take part in training. This command has no effect on prediction. The trained model is used exactly as it exists for all test cases, regardless of the value of the variable VarName. This option has little or no use for a stand-alone model. However, it can be useful for generating specialist models for an oracle.*

PROFIT = VarName

> *(Optional) If PROFIT is not specified, the OUTPUT variable, which is the target being predicted, is also used for computing profit factor performance measures. However, sometimes the target variable is not interpretable as a profit. For example, a future return that has been cross-sectionally normalized may have questionable value in computing profits. A filter residual (Page 140) is certainly not a measure of profit. In such cases, the user can specify a variable that will be used for computing profit-based performance measures. These measures will be used for stepwise predictor selection as well as in the display of all performance measures.*

LONG ONLY
SHORT ONLY

> *(Optional) This option specifies that the model will execute only long (or short) trades. Probably the only use of this option is in creating separate trading systems that specialize in only long or short trades and then combining them into a Portfolio.*

MAX STEPWISE = Number

(Mandatory) This is the maximum number of predictor variables that may be included in the model. Stepwise inclusion will be terminated early if inclusion of a new variable results in deterioration of performance. Setting this to zero forces all inputs to be used. It is recommended that stepwise selection not be used for tree-based models (TREE, FOREST, and BOOSTED TREE).

STEPWISE RETENTION = Number

(Optional) By default, this value is one, meaning that ordinary forward stepwise selection is performed. If this is greater than one, this many of the best variable sets are retained at each step. For example, if this is five, the best five single predictors are found. Then, for each of these five, the next best predictor, conditional on the first, is found. The best five pairs are retained, and a third variable is tried for each pair, and so forth. This tremendously slows training, but by testing more combinations of variables, we increase the chance of finding an effective set.

XVAL STEPWISE

(Optional) This directs that stepwise variable selection will be based on ten-fold cross validation of the training set rather than the usual direct optimization. In many cases this results in better selection of variables because the process more closely simulates out-of-sample experience. However, this option slows training by approximately a factor of ten. Beware!

FORCE IDENTITY

(Optional; for LINREG models only) No training is done. The model's input coefficients are forced to be 1.0 (usually just one input) and the constant is forced to be 0.0. This is a crude but effective way to make a single input be a 'model' as far as the program is concerned.

SHOW SELECTION COUNT

(Optional) This option causes a chart to be printed at the completion of walkforward or cross validation, immediately after the performance summary. This chart shows the percentage of times each predictor variable was selected during the run. This is especially informative when STEPWISE RETENTION is greater than one.

MODEL CRITERION =   LONG PROFIT FACTOR

(Optional; for LINREG models only) The model's coefficients are determined by maximizing the long profit factor. (By default, mean squared error is minimized. This option overrides the default.) See CRITERION = LONG PROFIT FACTOR below for more details.

MODEL CRITERION =   SHORT PROFIT FACTOR

As above except that short profit factor is maximized.

MODEL CRITERION = PROFIT FACTOR

As above except that long and short profit factors are maximized simultaneously.

MODEL CRITERION = LONG/SHORT/[none] ULCER INDEX (Equity)

As above except that long, short, or dual Ulcer Index is minimized. The single-sided options are quite slow (run time proportional to the square of the number of cases.) The dual option is extremely slow (runtime proportional to the cube of the number of cases.) The initial equity must be specified. In general, large values are best, because computation is most accurate when the equity curve never approaches ruin.

MODEL CRITERION = LONG/SHORT/[none] MARTIN RATIO (Equity)

As above except that long, short, or dual Martin Ratio is maximized. The single-sided options are quite slow (run time proportional to the square of the number of cases.) The dual option is extremely slow (runtime proportional to the cube of the number of cases.)

CRITERION = MODEL CRITERION

(Some stepwise criterion must be specified.) The stepwise criterion is the model criterion computed by one of the options above. This option is valid only if a MODEL CRITERION has been specified.

CRITERION = RSQUARE

(Some stepwise criterion must be specified.) The stepwise criterion optimized is R-Square, the fraction of the predicted variable's output that is explained by the model. Note that R-square will be negative in the unusual situation that the model's predictions are, on average, worse than guessing. If this criterion is optimized, the threshold used for computing threshold-based performance criteria is arbitrarily set at zero.

CRITERION = LONG PROFIT FACTOR

(Some stepwise criterion must be specified.) The stepwise criterion optimized is an analog of the common profit factor, under the assumption that only long (or neutral) positions are taken. A threshold is optimized. Only cases whose predicted value equals or exceeds the threshold enter into the calculation. Sum all such cases for which the true value of the predicted variable is positive, and also sum all such cases for which the true value of the predicted variable is negative. Divide the former by the latter and flip its sign to make it positive. This is the profit factor criterion. If the predicted variable is actual wins and losses, this criterion will be the traditional profit factor for a system that takes a long position whenever the predicted value exceeds the optimized threshold. CRITERION must not be used for tree-based models (TREE, FOREST, and BOOSTED TREE).

CRITERION = SHORT PROFIT FACTOR

(Some stepwise criterion must be specified.) This is identical to LONG PROFIT FACTOR above except that only short (and neutral) positions are taken. A threshold is optimized, and only cases whose predicted values are less than or equal to the threshold enter into the calculation. Since this criterion assumes short trades only, a positive value of the predicted variable implies a loss, and conversely. CRITERION must not be used for tree-based models (TREE, FOREST, and BOOSTED TREE).

CRITERION = PROFIT FACTOR

(Some stepwise criterion must be specified.) This combines the LONG and SHORT profit factor criteria above. Two thresholds are simultaneously optimized. If a case's predicted value equals or exceeds the upper threshold, it is assumed that a long position is taken, meaning that a positive value in the predicted variable implies a win. If a case's predicted value is less than or equal to the lower threshold, it is assumed that a short position is taken, meaning that a positive value in the predicted variable implies a loss. CRITERION must not be used for tree-based models (TREE, FOREST, and BOOSTED TREE).

CRITERION = ROC AREA

(Some stepwise criterion must be specified.) The criterion optimized is the area under the profit/loss ROC curve. This criterion considers the distribution of actual profits and losses relative to predictions made by the model. A random model will have a value of about 0.5, a perfect model will have a ROC area of 1.0, and a model that is exactly incorrect (the opposite of perfect) will have a value of 0.0. CRITERION must not be used for tree-based models (TREE, FOREST, and BOOSTED TREE).

```
CRITERION = MEAN_ABOVE_10
CRITERION = MEAN_ABOVE_25
CRITERION = MEAN_ABOVE_50
CRITERION = MEAN_ABOVE_75
CRITERION = MEAN_ABOVE_90
```

(Some stepwise criterion must be specified.)   The criterion is the mean target value for those cases whose predicted values are in the top specified percentage of total cases. This family is not compatible with the XVAL STEPWISE option because thresholds are not generally consistent across training folds.

```
CRITERION = MEAN_BELOW_10
CRITERION = MEAN_BELOW_25
CRITERION = MEAN_BELOW_50
CRITERION = MEAN_BELOW_75
CRITERION = MEAN_BELOW_90
```

(Some stepwise criterion must be specified.)   The criterion is the negative of the mean target value for those cases whose predicted values are in the bottom specified percentage of total cases. This family is not compatible with the XVAL STEPWISE option because thresholds are not generally consistent across training folds.

```
CRITERION = PF_ABOVE_10
CRITERION = PF_ABOVE_25
CRITERION = PF_ABOVE_50
CRITERION = PF_ABOVE_75
CRITERION = PF_ABOVE_90
```

(Some stepwise criterion must be specified.)   The criterion is the profit factor for those cases whose predicted values are in the top specified percentage of total cases. This family is not compatible with the XVAL STEPWISE option because thresholds are not generally consistent across training folds.

```
CRITERION = PF_BELOW_10
CRITERION = PF_BELOW_25
CRITERION = PF_BELOW_50
CRITERION = PF_BELOW_75
CRITERION = PF_BELOW_90
```

(Some stepwise criterion must be specified.)   The criterion is the short profit factor for those cases whose predicted values are in the bottom specified percentage of total cases. This family is not compatible with the XVAL STEPWISE option because thresholds are not generally consistent across training folds.

```
CRITERION = BALANCED_01
CRITERION = BALANCED_05
CRITERION = BALANCED_10
CRITERION = BALANCED_25
CRITERION = BALANCED_50
```

(Some stepwise criterion must be specified.) These criteria are valid only if the FRACTILE THRESHOLD option is also used. They are similar to the PROFIT FACTOR criterion in that the quantity optimized is the profit factor obtained by taking a long position for cases above the upper threshold and a short position for cases below the lower threshold. The difference is that the PROFIT FACTOR computes optimal upper and lower thresholds separately, meaning that positions will usually be unbalanced. These balanced criteria set upper and lower thresholds that are equal. BALANCED_01 decrees that the highest one percent of market predictions in each time slice will take a long position, and the lowest one percent of market predictions will take a short position. The other variations cover five, ten, 25, and 50 percent thresholds. For all thresholds except 50, the position is inclusive. In other words, BALANCED_10 will take a long position for any market whose prediction is at the ten percent or higher threshold. However, this is obviously not possible for the 50 percent threshold, as inclusion would result in markets at exactly the 50 percent fractile being both long and short! Thus, markets that happen to land at exactly 50 percent will remain neutral.

MIN CRITERION CASES = Integer

(At least one minimum must be specified) If the criterion is one that requires an optimized threshold (such as PROFIT FACTOR), this specifies the minimum number of cases that must meet the threshold. If no minimum were specified, the optimizer might choose a threshold so extreme that only a very few cases meet it, resulting in statistical instability. At this time, a minimum must be specified even if the criterion does not require a threshold.

MIN CRITERION FRACTION = RealNumber

(At least one minimum must be specified) If the criterion is one that requires an optimized threshold (such as PROFIT FACTOR), this specifies the minimum fraction (0-1) of training cases that must meet the threshold. If no minimum were specified, the optimizer might choose a threshold so extreme that only a very few cases meet it, resulting in statistical instability. A minimum must be specified even if the criterion does not require a threshold, because thresholded statistics are computed and printed even if they are not used for variable selection.

FRACTILE THRESHOLD

(Optional, makes sense only if multiple markets are present) By default, all predictions in the training set are pooled into a single group in order to determine the optimal threshold(s). Similarly, when the trained model is used on out-of-sample data, the prediction for each market is compared to the threshold in order to make a trade decision. The implication is that trades may cluster, occurring rapidly in times when predictions run high, and slowing or even stopping in times when predictions run low. The FRACTILE THRESHOLD option provides an alternative. Instead of individual market predictions being used for training and testing, the rank order of predictions across all markets for each time slice are treated as predictions. The implication is that the trade rate stabilizes, at the cost of trades being done at times when the original prediction might argue against it. The range is from -1.0 to 1.0, being two times the actual fractile, minus one.

RESTRAIN PREDICTED

(Optional) This causes the true values of the predicted variable to be compressed at the extreme values before training. This is almost always very useful, perhaps even mandatory. If the predicted variable contains outliers, many models will expend considerable effort accommodating these extreme values, at the expense of the more common cases. When this option is invoked, most results will be printed twice, once for the restrained values, and again for the original values.

MCP TEST = Iterations

(Optional) This is relevant only if the user performs cross validation or walkforward testing. If so, a Monte-Carlo Permutation Test is performed on the out-of-sample cases. This computes the approximate probability that results as good as or better than those obtained could be obtained purely by luck from a worthless model.

ORDINARY BOOTSTRAP = Integer

(Optional) This is relevant only if the user performs cross validation or walkforward testing. If so, an ordinary bootstrap test is performed on the out-of-sample cases. This computes the approximate probability that a mean return at least as great as that obtained could be obtained purely by luck from a worthless model. The user specifies the number of replications, which should be at least 1000 for a reasonably accurate estimate.

STATIONARY BOOTSTRAP = Integer

As above, except that a stationary bootstrap test is performed. This test is reasonably effective for situations in which the sample has serial correlation. The block size is automatically computed and printed for the user.

TAPERED BLOCK BOOTSTRAP = Integer

> *As above, except that a tapered block bootstrap test is performed. This test is reasonably effective for situations in which the sample has serial correlation. The block size is automatically computed and printed for the user. Many experts claim that the tapered block bootstrap is more accurate than the stationary bootstrap, although this is not a universal rule.*

STATIONARY BOOTSTRAP (BlockSize) = Integer
TAPERED BLOCK BOOTSTRAP (BlockSize) = Integer

> *The user can override the automatic blocksize computation and specify a blocksize for the two dependent bootstrap tests.*

OVERLAP = Integer

> *(Optional) This should be used if the target variable looks ahead in the data more than one time interval and a walkforward or normal cross validation is performed. The lookahead implies that there is some information overlap at the boundaries between training and test periods. Specifying an overlap here causes the training set to shrink away from this boundary by the specified number of time intervals. This value should be one less than the lookahead period.*

RESAMPLE

> *(Optional) Instead of training with the training set, cases are randomly sampled from it, with replacement. The number of cases is equal to the original number of cases. Some original cases may be omitted from the resampled set, and some may be repeated. This can be useful for creating models that will be part of a committee.*

SUBSAMPLE RealNumber PERCENT

> *(Optional) Instead of training with the training set, cases are randomly sampled from it. The number of cases is equal to the specified percent (0-100) of the original number of cases. No cases are repeated. This can be useful for creating models that will be part of a committee.*

MLFN Models

The MLFN model has some options that are not available to other models. These are the following:

FIRST HIDDEN = Integer

    (Mandatory) This specifies the number of hidden neurons in the first layer. It should be kept small, though greater than one. As few as two hidden neurons, and three at most, can be enormously powerful. Once you reach four or more hidden neurons, overfitting becomes likely.

SECOND HIDDEN = Integer

    (Optional) If this option appears, a second hidden layer will be included. I have never seen an application in which this is useful, although the literature claims that second hidden layers are often valuable.

DOMAIN REAL

    (One DOMAIN specification is required) This is by far the most common type of MLFN. Processing is entirely in the real domain, beginning to end.

DOMAIN COMPLEX INPUT

    (One DOMAIN specification is required) This option specifies that the inputs are complex-valued. The first input is treated as real, the second imaginary, the third (if any) real, the fourth imaginary, et cetera. The user must specify an even number of inputs. Stepwise selection is not performed; all inputs are used. This option should be used only if the inputs truly have a complex interpretation, such as real and imaginary Morlet wavelets. There is no reason to believe that truly real inputs will ever be appropriate in a complex MLFN.

DOMAIN COMPLEX HIDDEN

    (One DOMAIN specification is required) This option is identical to that above, except that the hidden neurons also operate in the complex domain. This is almost always more useful than a strictly complex input.

OUTPUT SQUASHED

(Either OUTPUT SQUASHED or OUTPUT LINEAR must be specified.) This option specifies that the output is squashed to a range of -1 to 1 via the hyperbolic tangent function. The is occasionally useful for classification, but it should never be used for profit prediction unless the profits are similarly scaled. I have never found any value in squashing the output, although this is traditional for neural networks.

OUTPUT LINEAR

(Either OUTPUT SQUASHED or OUTPUT LINEAR must be specified.) This option specifies that outputs are linear, meaning that the range of outputs is unlimited. I have always found this to be preferable to squashing outputs.

Tree-Based Models

The tree-based models (TREE, FOREST, and BOOSTED TREE) have some options that are not available to other models:

`TREE MAX DEPTH  =  Integer`

> *(Mandatory) For any tree-based model, this specifies the maximum depth of the tree(s). Note that pruning may result in shallower trees. A depth of one means that a single decision is made, splitting the root into two nodes. A depth of two means that these two nodes may themselves be split, and so forth.*

`MINIMUM NODE SIZE  =  Integer`

> *(Optional; default=1) For any tree-based model, this specifies that a node will not be split if it contains this many cases or fewer. Note that some nodes may be smaller than this if a split occurs near a boundary, so this is not really the minimum size of a node. It just prevents small nodes from being split.*

`TREES  =  Integer`

> *(Mandatory for FOREST, illegal for others) This specifies the number of trees in the forest.*

`MIN TREES  =  Integer`

> *(Mandatory for BOOSTED TREE, illegal for others) This specifies the minimum number of trees in the boosting set. The optimum number will be determined by cross validation.*

`MAX TREES  =  Integer`

> *(Mandatory for BOOSTED TREE, illegal for others) This specifies the maximum number of trees in the boosting set. The optimum number will be determined by cross validation.*

`ALPHA  =  Real`

> *(Mandatory for BOOSTED TREE, illegal for others) This specifies the Huber Loss value. Its maximum value of one results in no Huber loss, and is best for clean data. Smaller values (zero is the minimum) cause extremes to be ignored. Values less than 0.9 or so are only rarely appropriate.*

FRACTION OF CASES = Real

> *(Optional; default=0.5) For a BOOSTED TREE model, this specifies the fraction of cases, zero to one, that are used to train each component tree. This parameter is not used for other models.*

SHRINKAGE = Real

> *(Optional; default=0.01) For a BOOSTED TREE model, this specifies the learning factor for each component tree. In general, smaller values produce better generalization but require more trees to learn the data pattern.*

Operation String (OPSTRING) Models

Sometimes prediction can be done effectively by combining two or more predictors via one or more simple binary operations, and following the operation string with simple linear regression to map the result to the target. For example, suppose A, B, C, and D are predictors. We might add A and B to get A+B, subtract C from D to get D-C, and divide the difference by the sum, giving (D-C)/(A+B). Finally, we compute the least-squares fit of this quantity to the target. This is an Operation String (OPSTRING) model.

OPSTRING models are trained by genetic optimization. A large initial population is generated by randomly choosing predictor candidates, constants, and operations. Then succeeding generations are created by combining parents via crossover and mutation. Training ceases when a few generations in a row fail to provide improved performance as measured by the dual profit factor, that obtained by taking both long and short trades. A future release of TSSB may allow the user to specify different optimization criteria, such as long-only, short-only, or balanced market-neutral performance.

Let A and B be predictors. The following operations are used in this implementation of Operation Strings:

ADD: A + B
SUBTRACT: A - B
MULTIPLY: A * B
DIVIDE: A / B
GREATER: 1.0 if A > B, else 0.0
LESS: 1.0 if A < B, else 0.0
AND: 1.0 if A > 0.0 and B > 0.0, else 0.0
OR: 1.0 if A > 0.0 or B > 0.0, else 0.0
NAND: 0.0 if A > 0.0 and B > 0.0, else 1.0
NOR: 0.0 if A > 0.0 or B > 0.0, else 1.0
XOR: 1.0 if (A > 0.0 and B <= 0.0) or (A <= 0.0 and B > 0.0), else 0.0

In addition to the usual model options, the following items may (but do not have to) be specified:

MAX LENGTH = Integer

The maximum length of the operation string. Smaller values force simpler operations. The actual string length may be less than this quantity. This must be odd. (It will be decremented to an odd number if specified even.) The default is 7.

POPULATION = Integer

The number of individuals in the population. Generally, this should be at least several times the number of candidate predictors in order to provide sufficient genetic diversity. The default is 100.

CROSSOVER = RealNumber

This is the probability (0-1) of crossover. The default is 0.3.

MUTATION = RealNumber

This is the probability (0-1) of mutation. It should be small, because mutation is usually destructive. The default is 0.02.

REPEATS = Integer

If this many generations in a row fail to improve the best child, training terminates. The default is 10.

CONSTANTS (Probability) = ListOfConstants

By default, no constants are allowed in an operation string. This command lets the user request that specific constants or random constants within a range may be used. A range is specified by surrounding a pair of numbers with angle brackets, as in <-40 40>. The Probability (0-1) specifies the probability that a constant will be used rather than a variable.

CONSTANTS IN COMPARE

If the user specifies a CONSTANTS list, then by default constants may be used in any operations. This option restricts constants to being used in compare operations (greater/less than).

CONSTANTS IN COMPARE TO VARIABLE

This is even more restrictive than the prior command. It restricts constants to use in comparisons to variables.

Regime Regression with SPLIT LINEAR Models

It is often the case that we cannot effectively model a process with a single model that is expected to cover all states of the process. For example, one model may do a good job of predicting market movement when we are in a high-volatility regime, while a different model may be best in a low-volatility regime. These models may employ identical predictors but use them differently. Or they may employ different predictors. A straightforward way of performing regime regression is with the SPLIT LINEAR model.

Selection of the predictors in a SPLIT LINEAR model takes place in two steps. The stepwise process used for all models produces a pool of variables from which predictors for all regimes will be chosen. At each step, the predictors in the current common pool will be optimally assigned to the individual regime models.

In addition to selection of predictors, a gate variable will be chosen from a list supplied by the user. An optimal split point (or two points) based on the gate variable will be computed in order to define the regimes.

By default, every regime will have its own model. However, the SPLIT LINEAR SPLITS = NOISE option produces a different configuration. This option results in only one of two regimes being modeled. The other regime is considered to be noise, and predictions made within this noise regime are always set equal to the training set target mean. As a result, trades will generally be rare within the noise regime.

The following options specially apply to SPLIT LINEAR models:

SPLIT VARIABLE = [Var1 Var2 ...]

This lists one or more candidates for the variable that will define regimes. As in an INPUT list, a range may be specified with the dash (-).

SPLIT LINEAR SPLITS = Integer

This specifies the number of split points, and it must be one or two. The number of regimes is one more than the number of split points. If omitted, the default is 1.

SPLIT LINEAR SPLITS = NOISE

The domain is split into two regimes, but only one of the two regimes is modeled. The other regime is considered to be noise. Any prediction made within the noise regime will be set equal to the mean of the target variable within the training set. Note that it will not always be possible to meet the specified SPLIT LINEAR MINIMUM FRACTION specification when the NOISE option is invoked.

SPLIT LINEAR MAXIMUM INDICATORS = Integer

This specifies the maximum number of predictors that will be used in each regime. These predictors will be optimally chosen from the pool of candidates provided by the model's stepwise selection process. If you want to guarantee that the regimes can have completely disjoint predictor sets, the model's MAX STEPWISE must be equal to SPLIT LINEAR MAXIMUM INDICATORS times the number of regimes. It makes no sense to make it larger. If omitted, the default is 2.

SPLIT LINEAR MINIMUM FRACTION = RealNumber

This specifies the minimum fraction of the training cases that must fall into each regime for the ordinary version of the model, and the minimum for the non-noise regime in the NOISE version. If the SPLIT LINEAR SPLITS = NOISE option is used, it may not always be possible to satisfy this minimum fraction, especially if the value is 0.5 or greater. In this case, the program will come as close as it can. Also, note that if the NOISE option is not used, the maximum value for this MINIMUM FRACTION that the user may specify is 0.4. This ensures that degenerate conditions are avoided. Realize that in the ordinary version, a minimum exceeding 0.5 would be illegal because it is not possible for all regimes to contain more than half of the cases! If omitted, the default is 0.1.

SPLIT LINEAR RESOLUTION = Integer

This is the number of split points that will be tested in order to define the regimes. Larger values require more run time but produce higher accuracy. If omitted, the default is 10. If there is one split point, this quantity will determine how many splits are tested to find an approximate split, and then the exact optimum will be found by further iterations. If there are two split points, no further refinement is done.

Input List Examples

Here are some examples of an input list:

INPUT = [CMMA_5   CMMA_10   CMMA_20]
These three variables are specified.

INPUT = [CMMA_5 – CMMA_20]
These two variables, as well as any that lie between them in the Variable List File, are specified.

INPUT = [<NAME = "CLOSE MINUS MOVING AVERAGE" >]
All variables from this family are specified.

INPUT = [
    < NAME = "DELTA PRICE SKEWNESS"
      INDEX = MINUS >
    ]
All variables from the DELTA PRICE SKEWNESS family that are also of the MINUS INDEX version are specified.

INPUT = [
    < NAME = PRICE SKEWNESS >
    < NAME = CHANGE SKEWNESS >
    ]
All variables from the PRICE SKEWNESS as well as CHANGE SKEWNESS families are specified.

Training and Testing Models

The following commands are available for training and testing models:

TRAIN ;

> *All models are trained using all available cases in the database. Specifications of the model as well as its in-sample performance will be printed in both the AUDIT.LOG and REPORT.LOG files.*

CROSS VALIDATE BY YEAR ;

> *The model is cross validated using the year of each case as the fold variable. Specifications of each model as well as its in-sample performance measures are printed to the AUDIT.LOG file but not the REPORT.LOG file. Out-of-sample results are printed to both files.*

CROSS VALIDATE BY MONTH ;

> *As above, except that hold-out blocks are one month long.*

CROSS VALIDATE BY DAY ;

> *As above, except that hold-out blocks are one day long.*

CROSS VALIDATE BY VariableName ;

> *As above, except that the named variable is used to control folds. Cases having exactly the same value of this variable occupy the same fold.*

CROSS VALIDATE BY VariableName (Thresholds) ;

> *As above, except that folds are determined by where the value of the named variable lies among the one or more specified thresholds. For example, suppose we have two thresholds: (0.3 0.7). Values less than 0.3 go into one fold. Values greater than or equal to 0.3 but less than 0.7 go into a second fold. Values of 0.7 or more go into the third fold. There will of course be one more fold than number of thresholds.*

CROSS VALIDATE Nfolds RANDOM BLOCKS ;

> *As above, except that the specified number of folds are created by random selection of cases.*

WALK FORWARD BY YEAR TrainYears StartYear ;

> *The model is walked forward using the year of each case as the control variable. The user must specify the number of years to include in each training period, and the starting year for testing. Specifications of each model as well as its in-sample performance measures are printed to the AUDIT.LOG file but not the REPORT.LOG file. Out-of-sample results are printed to both files.*

WALK FORWARD BY MONTH TrainMonths StartDate ;

> *As above, except that walking advances one month at a time. The start date must be expressed as YYYYMM.*

WALK FORWARD BY DAY TrainDays StartDate ;

> *As above, except that walking advances one day at a time. The start date must be expressed as YYYYMMDD.*

Results include the number of cases tested and the number of those cases that equaled or exceeded the threshold. For criteria that are based on a threshold, the *naive* version of the criterion is its value ignoring the model, using all cases in the fold. The thresholded version uses only those cases whose predicted value meets the threshold.

If the user specified MCP TEST or a bootstrap test (Page 122 and following) in any model definition, and if cross validation or walkforward testing is done, the test is performed on the out-of-sample results. This test computes the approximate probability that results as good as or better than those observed could have been obtained by pure luck from a worthless model.

When WALK FORWARD testing is done for a script file that includes one or more transforms, the transforms are evaluated for only the portion of the dataset needed at any time. This is the training set and the test set. The transform values are set to zero outside this area. The implication is that if prescreening is done based on a value produced by a transform, the number of cases in the dataset passing the prescreen, a value reported in the audit log, may change as the walkforward progresses. This is information only, and has no bearing on results.

Model Examples

Here is an example of a simple model definition and its evaluation. It should be self-explanatory.

```
MODEL mod1 IS LINREG [
   INPUT = [ CMMA_5_S CMMA_5_N CMMA_5_S_C CMMA_5_N_C
             CMMA_50_S CMMA_50_N CMMA_50_S_C CMMA_50_N_C ]
   OUTPUT = LOGRET
   MAX STEPWISE = 3
   CRITERION = PROFIT FACTOR
   MIN CRITERION CASES = 3000
   RESTRAIN PREDICTED
   MCP TEST = 1000
   ] ;
CROSS VALIDATE BY YEAR ;
```

Committees

A committee is a lot like a model. In fact, the following MODEL options (see Page 115) can be used in a committee:

    INPUT =
    OUTPUT =
    PROFIT =
    LONG ONLY
    SHORT ONLY
    MAX STEPWISE =
    STEPWISE RETENTION =
    CRITERION =
    MIN CRITERION CASES =
    MIN CRITERION FRACTION =
    RESTRAIN PREDICTED
    MCP TEST =

A committee is specified with the following command:

COMMITTEE   CommName IS CommType   [CommSpecs   ]   ;

At this time, these committee types are available:

    AVERAGE - The output is the average (mean) of the inputs
    LINREG - Ordinary linear regression is used to model the target
    CONSTRAINED - A linear combination constrained to positive weights that sum to
        one
    GRNN - General Regression Neural Network
    MLFN - Multiple-Layer Feedforward Network
    TREE - Simple binary tree
    FOREST - Forest
    BOOSTED TREE - Boosted tree

The greatest difference between a model and a committee is that a model's input list (usually) names predictor variables, while a committee's input list (always) names models. The named model outputs are chosen by forward stepwise selection unless MAX STEPWISE is specified as zero, in which case all models are used.

Oracles

An *oracle* is an intelligent committee. Like a committee, it combines predictions from multiple models in order to produce a single consensus prediction. However, an oracle uses one or more *gate variables* to control the relative weighting of the component models. A committee uses a single predefined methodology for all cases processed. An oracle changes its combining method according to the value of the gate variable(s). This is extremely powerful, because it allows different models to be differently weighted in different conditions. For example, some models might be favored in a high-volatility market, while others might dominate in a low-volatility market.

The following MODEL options (see Page 115) can be used in an oracle:
> INPUT =
> OUTPUT =
> PROFIT =
> LONG ONLY
> SHORT ONLY
> CRITERION =
> MIN CRITERION CASES =
> MIN CRITERION FRACTION =
> RESTRAIN PREDICTED
> MCP TEST =

Note that stepwise options are not allowed, because all component models are always considered. The following options are specific to an oracle:

GATE = [Var1 Var2 ...] ;
> *(Mandatory) This lists the variables that will be used as gates. It is strongly recommended that only one variable be used. Training time and the risk of overfitting both explode if even two variables are used.*

HONOR PRESCREEN ;
> *(Optional) If one or more of the component models has a PRESCREEN (Page 116) that is also a gate, and if this option is specified, training and execution of the oracle will, for any given case, consider only models whose value of the gate variable lies within the prescreen range. If any component model has a prescreen, it is almost always sensible (though not required) to make this prescreen a gate and specify the HONOR PRESCREEN option.*

An oracle is specified with the following command:

ORACLE OracleName [CommSpecs] ;

Oracles and Prescreening

There are two quite different philosophies for using an oracle. Neither is inherently superior to the other. Different applications may favor different alternatives.

The simplest method is to train two or more models on the entire dataset, and intelligently combine their predictions based on values of a gate variable. The gate may be one of the predictors, although this would almost never be the case. A model output cannot be used as a gate. The component models would typically be different in fundamental design. For example, they may use different predictor sets, different targets, or different optimization criteria. The key is that they all share the same (complete) training set. Their differences come from fundamental design changes.

An alternative oracle technique is to train specialist models via the PRESCREEN option (Page 116) and use the oracle to oversee their decisions. For example, one might train three models based on the value of a trend variable. One model might specialize in down markets, and second in flat markets, and the third in up markets. The oracle would use the prescreen variable as its gate. If the HONOR PRESCREEN option is specified for the oracle and a component model's prescreen variable is also a gate, training and execution cases will be processed for the component model if and only if and the value of this variable is within the prescreen range of the model. Note that when a prescreened model is used in an oracle, in the vast majority of applications one should make the prescreen variable a gate and specify the HONOR PRESCREEN option. Otherwise the model will be trained as a specialist, but the specialization will be ignored when the model is used by the oracle. This is silly.

In a prescreened specialist situation, it is perfectly legal to use one or more gate variables that are not prescreen variables (keeping in mind that the prescreen variables should also be gates!). Such a gate will impact all cases in all models. However, note that when more than one gate is employed, training time may become excessive.

Here is an example of using three specialist models in conjunction with an oracle that combines them according to their specialization.

```
MODEL MOD_DOWN IS LINREG [
  INPUT = [ X1 - X10 ]
  OUTPUT = DAY_RETURN
  PRESCREEN    TREND < -10
  MAX STEPWISE = 2
  CRITERION = PROFIT FACTOR
  MIN CRITERION FRACTION = 0.1
  ] ;

MODEL MOD_FLAT IS LINREG [
  INPUT = [ X1 - X10 ]
  OUTPUT = DAY_RETURN
  PRESCREEN   TREND > -10
  PRESCREEN   TREND < 10
  MAX STEPWISE = 2
  CRITERION = PROFIT FACTOR
  MIN CRITERION FRACTION = 0.1
  ] ;

MODEL MOD_UP IS LINREG [
  INPUT = [ X1 - X10 ]
  OUTPUT = DAY_RETURN
  PRESCREEN   TREND > 10
  MAX STEPWISE = 2
  CRITERION = PROFIT FACTOR
  MIN CRITERION FRACTION = 0.1
  ] ;

ORACLE ORACLE_TREND [
  INPUT = [ MOD_DOWN  MOD_FLAT  MOD_UP ]
  OUTPUT = DAY_RETURN
  GATE = [ TREND ]
  HONOR PRESCREEN
  MIN CRITERION FRACTION = 0.1
  ] ;
```

Trade Filtering

It is possible to read a file containing trade decisions produced by another program and then use the *TSSB* program to attempt to improve performance by choosing to enhance or abort certain trades. The format for reading a Haugen file is described on Page 10. Here, we describe how trade filtering can be performed.

When a trade file is read, two new predictor variables will be automatically added to the variable list. They may be plotted through the menu system, and one or both may optionally be placed in an INPUT list for a model. These variables are:

FILTER VALUE - This is the value of the variable that determines trades in the external system. In the case of a *Haugen* file, this is his predicted *alpha*.

FILTER FRACTILE - For each date, the filter values are ranked across all markets that have records on this date. The market having minimum value will be given a fractile of -50. The largest will have a fractile of 50. Other markets will have equally spaced intermediate values.

Residuals

There are two main approaches to using models and committees to filter trades generated by an external system. One method is to use an actual future return as a target, and supply the FILTER VALUE and/or FILTER FRACTILE as supplementary inputs to a model or models. The idea is that the combination of the external prediction with *TSSB* variables will prove superior to one or the other.

A frequently useful alternative is to compute the difference between the actual future return and the prediction from the external model. This difference is called the *residual*, and it can be an excellent prediction target. The idea is that if we can devise a means of estimating the external system's error, we can add our predicted error to the external system's prediction in order to generate a more accurate prediction of future return. Thresholds would be computed for this sum, rather than for the predicted residual itself.

Residual variables are computed by defining a target variable in the usual way, except that the keyword *RESIDUAL* appears last on the definition line.

It is vital that the future variable and the external system's prediction be measuring the same thing in the same way, so that the difference between these two quantities measures only

prediction error of the external system. For example, if one of the quantities is annualized and the other is not, the difference will be nearly meaningless.

Because of this rather stringent requirement of commensurability, it is usually best to perform cross-market normalization on both variables. You need specify this only on the target variable. If done, the external system's prediction is also cross-market normalized. For example, consider this line in a definition file:

```
RESID_C: NEXT MONTH ATR RETURN 250  ! 0.3  RESIDUAL
```

For each date in the filter file, this will compute the *NEXT MONTH ATR RETURN* for all markets on that date, rank the returns for all markets, and compute the variable rescaled to a range of -50 (for the market having least return) to 50 (for the marking having maximum return). The external system's predictions for all markets on that date will be similarly ranked. Then the difference for each market will be computed as the predicted variable *RESID_C*. An implication is that this variable will have theoretical limits of -100 to 100. Markets for which the actual return greatly exceeds the external system's prediction will have a large value for this variable, while small values (very negative) will be obtained when the external system's prediction greatly exceeds the actual future return.

Note that if a residual variable is used as the model's output, you must also supply a PROFIT variable (Page 15, 116) to use for computing profit-based performance measures. Residuals make no sense as profits.

Committees, Oracles, and Trade Filtering

A committee or oracle can be used in trade filtering. As was the case with models, you can either predict the actual future return, or you can predict the residual difference between the future return and the prediction made by the external system.

Because a committee attempts to combine multiple predictions of a quantity into a single superior prediction, it is vital that the quantities being combined (the model outputs) all measure the same thing. Thus it makes no sense to use a committee to combine the output of a model predicting a cross-sectionally normalized return with that of a model predicting an actual return. Also, it probably makes no sense to combine a one-day-ahead prediction with a one-month-ahead prediction.

Moreover, one cannot in principle combine a predicted residual with a predicted actual value, or use one of these as an input to a committee predicting the other. However, the *TSSB* program allows you do get away with this by performing an internal fix. If the committee is predicting an actual value, and a component model is predicting a residual, the external system's

prediction will be added to the model's prediction before presentation to the committee. Similarly, if the committee is predicting a residual, and a component model is predicting an actual value, the external system's prediction will be subtracted from the model's prediction before presentation to the committee.

Comparing Our System with the External System

If the external system makes only binary trade decisions, it is perfectly reasonable to judge the quality of our system by comparing the profit factor of our system above its threshold with that of all cases. If our system makes uselessly random predictions, these two profit factors will be nearly equal. But if our system is good, the cases above (or below, for shorts) the threshold will have a profit factor that exceeds that of the general population (the trades indicated by the external system).

However, if the external system makes continuous predictions, with the ultimate trade decisions being based on the value of the prediction relative to a threshold determined by value or quantile, the comparison is not straightforward. Suppose that on each date we choose the top ten percent of the external system's predictions as our population, and we in turn decide to keep the top ten percent of our predictions within that select population. Thus, we would be ultimately be keeping one percent of the original system's predictions (ten percent of ten percent). It would not be fair to compare the performance of our top ten percent with that of our population (the external system's top ten percent). Rather, we would need to compare our top ten percent with the top one percent of the original system's predictions.

There is a reasonable way to make fair comparisons. For each date in the external system's prediction database, we rank its predictions across all markets for which predictions were made on that date. We similarly rank our predictions for that date. Then, for select fractiles, we compare the profit factor of the filter's predictions in that fractile with our model's predictions in the same fractile. The superior system will have greater profit factors across most or all fractiles.

A Useless but Instructive Script File

The following is one of the many script files that I used to verify (hopefully!) correct operation of the program. First, however, note the variable definition file that accompanies this script:

```
LIN_ATR_5: LINEAR PER ATR 5 100
RETURN: NEXT MONTH ATR RETURN 250
RETURN_CHEAT: NEXT MONTH ATR RETURN 250
RESID: NEXT MONTH ATR RETURN 250 RESIDUAL
```

RESID_CHEAT: NEXT MONTH ATR RETURN 250 RESIDUAL
RETURN_C: NEXT MONTH ATR RETURN 250 ! 0.3
RETURN_C_CHEAT: NEXT MONTH ATR RETURN 250 ! 0.3
RESID_C: NEXT MONTH ATR RETURN 250 ! 0.3 RESIDUAL
RESID_C_CHEAT: NEXT MONTH ATR RETURN 250 ! 0.3 RESIDUAL

Observe that every target variable appears twice, once with a descriptive name and again with the same name followed by *CHEAT*. This lets us train perfect models. The main models in this script file use the same predictor and target. The idea is that all results should be perfect to the extent that is theoretically possible. My hope is that any bugs in the program will manifest themselves as less-than-perfect results for this script.

This script appears here because it is instructive in that it demonstrates the many ways we can train models and committees in the task of filtering an existing trading system. The comments inserted in the listing should explain its operation.

```
//****************************************************************************
//  This script file demonstrates the wide variety of possibilities in
//  filtering an external model.
//  In all cases, appending _C to a model or variable means that it uses
//  cross-sectionally normalized values.
//  Appending _CHEAT to a variable name means that it is just another name
//  for the same variable.  This lets us create perfect models.
//****************************************************************************



//****************************************************************************
//  Models DUMMY and DUMMY_C are deliberately weak models intended only to
//  serve as 'second models' in committees, where they will be rejected.
//****************************************************************************

MODEL DUMMY IS LINREG [
  INPUT = [ LIN_ATR_5 ]
  OUTPUT = RETURN
  MAX STEPWISE = 1
  CRITERION = PROFIT FACTOR
  MIN CRITERION FRACTION = 0.1
  MCP TEST = 100
  ] ;

MODEL DUMMY_C IS LINREG [
  INPUT = [ LIN_ATR_5 ]
  OUTPUT = RETURN_C
  MAX STEPWISE = 1
  CRITERION = PROFIT FACTOR
  MIN CRITERION FRACTION = 0.1
  MCP TEST = 100
  ] ;
```

```
//***************************************************************************
//  Model TWEAK is a simple model that predicts actual return using itself.
//  Naturally, it will do a perfect job!
//***************************************************************************

MODEL TWEAK IS LINREG [
  INPUT = [ RETURN_CHEAT ]
  OUTPUT = RETURN
  MAX STEPWISE = 1
  CRITERION = PROFIT FACTOR
  MIN CRITERION FRACTION = 0.1
  MCP TEST = 100
  ] ;


//***************************************************************************
//  This model does the same thing, except that the predicted value (as well
//  as the identical predictor) are cross-sectionally normalized.
//  Since the predictor and predicted are the same values, we would expect
//  perfection.  And we get it in MSE and R-Square.
//  But profit factors are not perfect, just excellent.  Why?
//  We would get perfect profit factors if the ranking were done across the
//  entire dataset.  But we don't.  We normalize separately for each day.
//  Some days are overall up, and others down.  So some days a high rank may
//  still correspond to a loss, and other days a low rank may still be a win.
//***************************************************************************

MODEL TWEAK_C IS LINREG [
  INPUT = [ RETURN_C_CHEAT ]
  OUTPUT = RETURN_C
  PROFIT = RETURN
  MAX STEPWISE = 1
  CRITERION = PROFIT FACTOR
  MIN CRITERION FRACTION = 0.1
  MCP TEST = 100
  ] ;
```

```
//**************************************************************************
//  This model predicts the residual, the actual return minus the FILTER_VALUE.
//  When thresholds are computed, the FILTER_VALUE is added to the prediction
//  to get the value that is thresholded.
//**************************************************************************

MODEL RESIDUAL IS LINREG [
   INPUT = [ RESID_CHEAT ]
   OUTPUT = RESID
   PROFIT = RETURN
   MAX STEPWISE = 1
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   MCP TEST = 100
   ] ;


//**************************************************************************
//  This is identical to the prior model except that the predicted value is
//  the cross-sectional rank of the actual return minus the cross-sectional
//  rank of the FILTER_VALUE, which is known as FILTER_FRACTILE.
//**************************************************************************

MODEL RESIDUAL_C IS LINREG [
   INPUT = [ RESID_C_CHEAT ]
   OUTPUT = RESID_C
   PROFIT = RETURN
   MAX STEPWISE = 1
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   MCP TEST = 100
   ] ;
```

```
//***************************************************************************
//  This is a simple committee.
//  It predicts actual return from the output of a model that also predicts
//  actual return.
//***************************************************************************

COMMITTEE HYBRID1 IS AVERAGE [
  INPUT = [ TWEAK DUMMY ]
  OUTPUT = RETURN
  MAX STEPWISE = 1
  CRITERION = PROFIT FACTOR
  MIN CRITERION FRACTION = 0.1
  MCP TEST = 100
  ] ;


//***************************************************************************
//  This is a more complex committee.  It predicts the residual by using the
//  output of a model that predicts actual values.  Thus, before the model's
//  output is presented to the committee, it is reduced by FILTER_VALUE.
//  After the committee makes its prediction, FILTER_VALUE is added back
//  to the prediction prior to computing or using a threshold.
//***************************************************************************

COMMITTEE HYBRID2 IS AVERAGE [
  INPUT = [ TWEAK DUMMY ]
  OUTPUT = RESID
  PROFIT = RETURN
  MAX STEPWISE = 1
  CRITERION = PROFIT FACTOR
  MIN CRITERION FRACTION = 0.1
  MCP TEST = 100
  ] ;
```

```
//**************************************************************************
//  This is the opposite of the prior committee.
//  It predicts an actual return using the output of a model that predicts
//  the residual.  Thus, before the output of the model is presented to the
//  committee, FILTER_VALUE is added to it.
//**************************************************************************

COMMITTEE HYBRID3 IS AVERAGE [
  INPUT = [ RESIDUAL DUMMY ]
  OUTPUT = RETURN
  MAX STEPWISE = 1
  CRITERION = PROFIT FACTOR
  MIN CRITERION FRACTION = 0.1
  MCP TEST = 100
  ] ;


//**************************************************************************
//  This is a consistent model in that it predicts the residual using the
//  output of a model that also predicts the residual.  Thus, the model's
//  output is presented to the committee unchanged.  The committee's output
//  is increased by FILTER_VALUE before thresholds are computed or used.
//**************************************************************************

COMMITTEE HYBRID4 IS AVERAGE [
  INPUT = [ RESIDUAL DUMMY ]
  OUTPUT = RESID
  PROFIT = RETURN
  MAX STEPWISE = 1
  CRITERION = PROFIT FACTOR
  MIN CRITERION FRACTION = 0.1
  MCP TEST = 100
  ] ;
```

```
//***************************************************************************
//  These four committees are identical to the prior four except that they
//  work with cross-sectionally normalized values instead of actual values.
//***************************************************************************

COMMITTEE HYBRID1_C IS AVERAGE [
  INPUT = [ TWEAK_C DUMMY_C ]
  OUTPUT = RETURN_C
  PROFIT = RETURN
  MAX STEPWISE = 1
  CRITERION = PROFIT FACTOR
  MIN CRITERION FRACTION = 0.1
  MCP TEST = 100
  ] ;

COMMITTEE HYBRID2_C IS AVERAGE [
  INPUT = [ TWEAK_C DUMMY_C ]
  OUTPUT = RESID_C
  PROFIT = RETURN
  MAX STEPWISE = 1
  CRITERION = PROFIT FACTOR
  MIN CRITERION FRACTION = 0.1
  MCP TEST = 100
  ] ;

COMMITTEE HYBRID3_C IS AVERAGE [
  INPUT = [ RESIDUAL_C DUMMY_C ]
  OUTPUT = RETURN_C
  PROFIT = RETURN
  MAX STEPWISE = 1
  CRITERION = PROFIT FACTOR
  MIN CRITERION FRACTION = 0.1
  MCP TEST = 100
  ] ;
```

```
COMMITTEE HYBRID4_C IS AVERAGE [
  INPUT = [ RESIDUAL_C DUMMY_C ]
  OUTPUT = RESID_C
  PROFIT = RETURN
  MAX STEPWISE = 1
  CRITERION = PROFIT FACTOR
  MIN CRITERION FRACTION = 0.1
  MCP TEST = 100
  ] ;
```

Miscellaneous Commands

This section discusses commands that do not readily fit into any other category.

Clumped trades

Some analysts evaluate the performance of a trading system by calculating trade profits from the opening of the trade to the closing of the trade. Others, including myself, prefer to mark positions to market every day (or bar if intraday). This is done for nearly all profit calculations in this program. This fine granularity is especially important for profit factor calculations. The following commands can be used to combine a position vector with a return vector and compute some statistics based on clumped trades (trades defined as open-to-close of position).

CLUMPED PROFIT FACTOR PosVar RetVar ;
CLUMPED PROFIT FACTOR PosVar RetVar MCreps ;

The position vector must be integers, typical zero and plus or minus one. If *MCreps* is specified, crude Monte-Carlo tests are performed by shuffling and rotating the returns.

Chi-Square Tests

It can be interesting to test a set of individual predictor candidates to discover the degree to which they are related to a target. The following command does this in the most basic version. Several variations are available and will be described later. A mouse-based GUI interface is also available and will be discussed later.

CHI SQUARE [Pred1 Pred2 ...] (Nbins) WITH Target (Nbins) ;

The user specifies a list of predictor candidates, the number of bins for predictors (at least 2), a single target, and the number of bins for this target (at least 2).

The program will print to AUDIT.LOG a list of the predictor candidates, sorted from maximum relationship (Cramer's V) to minimum. In addition to printing the chi-square value, it will also print the contingency coefficient, Cramer's V, and a p-value.

The contingency coefficient is a nominal form of correlation coefficient. Note, however, that unlike ordinary correlation, its maximum value is less than one. If the table is square with k bins in each dimension, the maximum value of the contingency coefficient is sqrt $((k\text{-}1)/k)$.

Cramer's V is a slightly better nominal correlation coefficient. It ranges from zero (no relationship) to one (perfect relationship).

The p-value does not take into account the fact that we are doing multiple tests. It is the p-value associated with a single test of the given predictor with the target. An option for taking selection bias (choosing the best indicators from a set of many candidates) into account will be discussed later.

The advantage of a chi-square test over ordinary correlation is that it is sensitive to nonlinear relationships. An interaction that results in certain regions of the predictor being associated with unexpected values of the target can be detected.

Larger numbers of bins increase the sensitivity of the test because more subtle relationships can be detected. However, p-values become less reliable when bin counts are small. As a general rule of thumb, most cells should have an expected frequency of at least five, and no cell should have an expected frequency less than one. But keep in mind that because we are doing multiple comparisons, individual p-values don't mean much anyway. The primary value of this test is the ordering of predictor candidates from maximum target relationship to minimum.

Options for the Chi-Square Test

The prior section discussed the most basic version of the chi-square test. Any or all of several options may be added to the command to extend the test. Suppose we have the following basic chi-square test:

CHI SQUARE [X1 X2 X3] (3) WITH RET (3) ;

The test shown above splits each indicator candidate (X1, X2, X3) into three equal bins, and it does the same for the target, RET. We can replace the number of bins for the indicators with a fraction ranging from 0.0 to 0.5 (typically 0.05 or 0.1) and the word TAILS to specify that the test employ just two bins for the indicator, the most extreme values. The following command says that cases with indicator values in the highest 0.1 (ten percent) of all cases go into one bin, cases whose indicator is in the lowest 0.1 go into the other bin, and the 80 percent middle values of the indicator will be ignored.

CHI SQUARE [X1 X2 X3] 0.1 TAILS WITH RET ;

Another option is to specify that instead of using equal-count bins for the target, the sign of the target (win/loss) is used. This implies that there are only two target bins: win and lose. (Note that using three bins is often better than splitting at zero, because this splits the target into big win, big loss, and near zero.) We split at zero by replacing the bin count with the word SIGN. The following command does this:

CHI SQUARE [X1 X2 X3] (3) WITH RET SIGN ;

Finally, it was already mentioned that the p-values printed with the basic test are individual, appropriate only if you test exactly one indicator. When you test many indicators and look for the best, lucky indicators will be favored. This is called selection bias, and it can be severe, causing huge underestimation of the correct p-value. A Monte-Carlo Permutation Test (MCPT) can be employed to approximately control for selection bias. This is done by appending MCPT = Nreps at the end of the command, as is shown in the following example:

CHI SQUARE [X1 X2 X3] (3) WITH RET (3) MCPT=100 ;

Finally, we can combine any or all of these options as shown below:

CHI SQUARE [X1 X2 X3] 0.1 TAILS WITH RET SIGN MCPT=100 ;

See *Running Chi-Square Tests from the Menu* on Page 156, 162 for more discussion of these options.

Output of the Chi-Square Test

The following table is a typical output from the chi-square test:

Chi-square test between predictors (2 bins) and DAY_RETURN_1 (3 bins) (Sorted by Cramer's V)
  Tails only, each tail fraction = 0.050

Variable	Chi-Square	Contingency	Cramer's V	Solo pval	Unbiased p
INT_5	30.02	0.225	0.231	0.0000	0.0010
DPPV_10	13.43	0.152	0.154	0.0012	0.2370
DINT_10	12.70	0.148	0.150	0.0017	0.3270
PENT_2	10.59	0.104	0.105	0.0050	1.0000

The heading for this table shows that two bins were used for the predictor (indicator here, though it can be interesting to use targets as predictors!) and three for the target. Only the tails were used for the predictors, the upper and lower five percent.

The indicators are listed in descending order of Cramer's V. This is a better quantity to sort than either chi-square or the contingency coefficient, because it is the best at measuring the practical relationship between the indicator and the target. Chi-square and the contingency coefficient are especially problematic for sorting when the indicator contains numerous ties. Also, the real-life meaning of chi-square and the contingency coefficient is vague, while Cramer's V is easily interpretable because it lies in the range of zero (no relationship) to one (perfect relationship).

The solo pval column is the probability associated with the indicator's chi-square, *when that indicator is tested in isolation*. This would be a valid p-value to examine if the user picked a single indicator in advance of the test. But because we will typically be examining a (frequently large) collection of candidate indicators, and focusing on only the best from among them, there is a high probability that indicators that were simply lucky hold our attention. Their luck will produce a highly significant (very small) p-value and we will be fooled. Thus, it is important that we compensate for the impact of luck.

A Monte-Carlo Permutation Test with 1000 replications was performed, and its results are shown in the last column labeled 'unbiased pval'. The smallest p-value possible is the reciprocal of the number of replications, so the minimum here is 0.001. This unbiased p-value will be correct (within the limitations of a MCPT) for only the indicator having largest Cramer's V. All others will somewhat over-estimate the correct p-value. But that's fine, because we at least know that the true p-value does not exceed that shown. There does not appear to be any way to compute MCPT p-values that are correct for all indicators.

The most important thing to note is that the unbiased p-value, which takes selection bias into account, can (and usually does) greatly exceed the solo p-value.

It the user specified that the predictors are to be tails only, the table shown above is followed by a smaller table showing the value of the lower and upper tail thresholds. This table looks like the following:

Variable	Lower thresh	Upper thresh
CMMA_20	-31.6299	35.5345
CMMA_5	-35.1684	37.1704

Running Chi-Square Tests from the Menu

As an alternative to issuing commands in a script file, chi-square tests can be run from the menu by clicking Describe / Chi-Square. The following dialog box will appear:
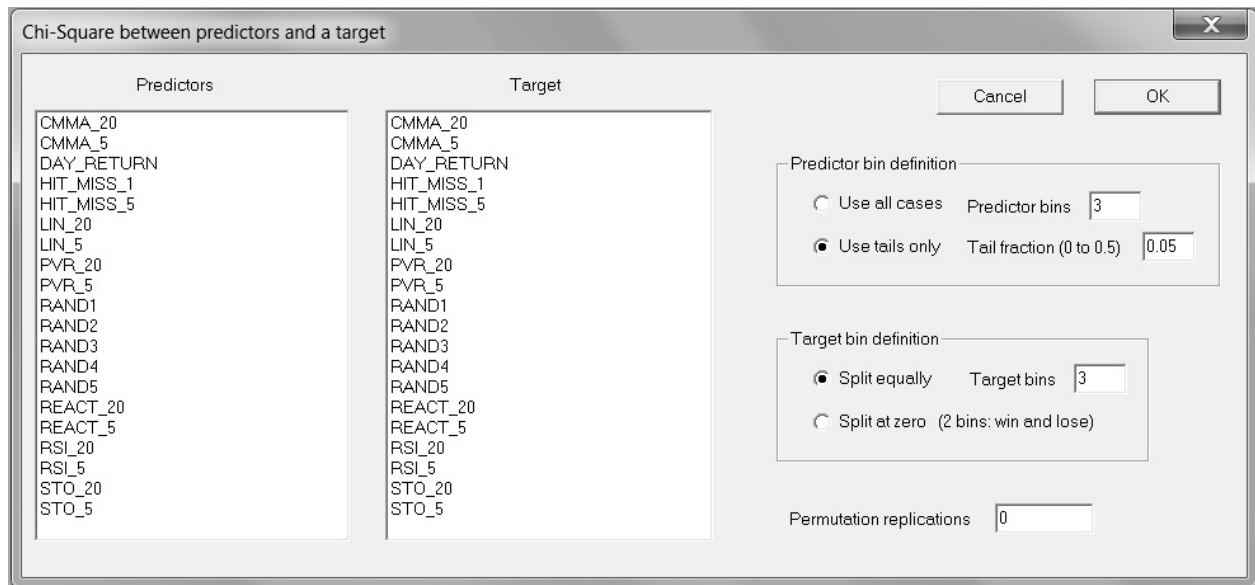
Figure 2: Dialog box for chi-square tests

The user selects one or more predictors (usually indicators, but target variabless can be selected as well) from the left-hand list. Selection can be done in any of three ways:

1) Drag across a range of predictors to select all of them.

2) Select a predictor, hold down the Shift key, and select another predictor to select all predictors that lie between them.

3) Hold the *Ctrl* key while clicking any predictor to toggle it between selected and not selected, without impacting existing selections.

A single target must be selected.

The user must specify how the predictor bins are defined. There are two choices:

1) Select 'Use all cases' and enter the number of equal-count bins to employ.

2) Select 'Use tails only' and enter the tail fraction on each side (greater than zero and less than 0.5) to employ two bins, one for each tail, with interior values ignored. Values of 0.05 or 0.1 are typical. Keeping more than ten percent of each tail usually results in significant loss of predictive power. The majority of predictive power in most indicators lies in the most extreme values.

The user must specify how the target bins are defined. There are two choices:

1) Select 'Split equally' and specify the number of equal-count bins to employ. In most cases this is the best choice, with three bins used. Three equal-count bins split the target into 'big win', 'big loss', and 'fairly inconsequential' trade outcomes. (This labeling assumes that the target distribution is fairly symmetric, the usual situation.)

2) Select 'Split at Zero', in which case there are two bins, with bin membership defined by the sign of the target. A target value of zero is considered to be a win.

Note that if the user selects 'Use tails only' for the predictor and 'Split equally' for the target, the target bin thresholds are defined by the entire target distribution, not the distribution in the predictor tails only. In most cases this results in a more sensitive test than determining the split points using predictor tails only.

Finally, the user can specify 'Permutation replications' to be a number greater than zero in order to perform a Monte-Carlo Permutation Test of the null hypothesis that all predictors are worthless for predicting the target (at least in terms of the chi-square test performed). If this is done, at least 100 replications should be used, and 1000 is not unreasonable. The smallest computed p-value possible is the reciprocal of the number of replications. The MCPT will produce a column labeled 'Unbiased p' which contains an upper bound (exact for the first predictor listed, increasingly conservative for subsequent predictors) on the probability that a set of entirely worthless predictors could have produced a best predictor as effective as the one observed. It would be nice to have exact unbiased p-values for predictors below the first (best), but there does not appear to be a practical way of computing this figure. Still, having the p-values be conservative (too large) is better than having them anti-conservative (too small). At least you know that the true p-value is no worse than that shown for each predictor.

Nonredundant Predictor Screening

The chi-square test described in the prior section is an effective way to rapidly screen a set of candidate indicators and rank them according to their power to predict a target variable. However, it has one disadvantage: indicators can be seriously redundant. This is especially true in regard to the information in the indicators which is related to the target. A statistical study of indicators alone (ignoring a target) may show that they have little correlation. However, if one devises a test which considers only that information component of the indicators that is related to a target variable, the degree of redundancy of this predictive information can be high. Thus we are inspired to consider an alternative test.

The method employed in *TSSB* operates by first selecting from a list of candidates the indicator that has the most predictive power. Then the candidate list is searched to find the indicator which adds the most predictive power to that obtained from the first selection. When a second indicator has been found, a third is chosen such that it adds the most predictive power to that already obtained from the first two. This repeats as desired.

The algorithm just described is in essence ordinary stepwise selection, similar to that obtainable from building predictive models. However, it is different in several ways:

- Because the algorithm is based on segregating the data into bins, it can detect nonlinear relationships without imposing the restrictions of a formally defined model.

- Unlike the models in *TSSB* (and those available in any other statistics packages known to the author), this algorithm allows the option of examining only extreme values of the candidate indicators. It is well known that in most situations, the majority of useful predictive information is found in the tails of indicators.

- The relative simplicity and speed of this algorithm makes it possible to perform Monte-Carlo Permutation Tests of the indicator sets to help decide which sets have legitimate predictive power and which were just lucky.

The following command does this in the most basic version. Note that it is identical to the syntax of the chi-square test of the prior section, except for the beginning of the command. Several variations are available and will be described later. A mouse-based GUI interface is also available and will be discussed later.

```
NONREDUNDANT PREDICTOR SCREENING
    [Pred1 Pred2 ...] (Nbins) WITH Target (Nbins) ;
```

The user specifies a list of predictor candidates, the number of bins for predictors (at least 2), a single target, and the number of bins for this target (at least 2).

The program will print to AUDIT.LOG a list of the predictor candidates, sorted into the order in which the predictors were selected. This order can be interpreted as saying that the first predictor listed is the single most important, the second predictor is the one that contributed the most additional predictive information, and so forth.

Several columns of useful data are printed. These are:

Mean Count is the mean number of cases expected in each cell. This is the total number of cases going into the test, divided by the number of cells. The number of cells for the first (best) predictor is the number of predictor bins times the number of target bins. As each additional predictor is added to the set, the number of cells is multiplied by the number of predictor bins. Obviously, if numerous predictor bins are specified, the number of cells will increase rapidly, and hence the mean per-cell count will drop rapidly. Larger counts produce a better test. If the count drops below five, a warning message will be printed.

100*Cramer's V is a nominal (category-based) correlation coefficient. It ranges from zero (no relationship) to 100 (perfect relationship). The primary disadvantage of Cramer's V is that it is symmetric: the ability of the target to predict the predictor is given the same weight as the ability of the predictor to predict the target. This is, of course, counterproductive in financial prediction.

100*Lambda is another nominal measure of predictive power which ranges from zero (no relationship) to 100 (perfect relationship). Unlike Cramer's V, it is one-sided: it measures only the ability of the predictor to predict the target. Also unlike Cramer's V, it is proportional in the sense that a value which is twice another value can be interpreted as implying twice the predictive power. However, it has the property that it is based on only the most heavily populated cells, those that occur most frequently. If more thinly populated cells are just noise, this can be good. But in most cases it is best to consider all cells in computing predictive power.

100*UncertReduc is an excellent measure of predictive power which is based on information theory. The label *UncertReduc* employed in the table is an abbreviation of *Uncertainty Reduction*. If nothing is known about any predictors, there is a certain amount of uncertainty about the likely values of the target. But if a predictor has predictive power, knowledge of the value of this predictor will reduce our uncertainty about the target. The *Uncertainty Reduction* is the amount by which our uncertainty about the target is reduced by gaining knowledge of the value of the predictor. This measure is excellent. Like Lambda, it is one-sided, as well as proportional. But like Cramer's V and unlike Lambda, it is based on all cells, not just those most heavily populated. So it has all of the advantages and none of the disadvantages of the other two measures.

Inclusion p-value is printed if the user specified that a Monte-Carlo Permutation Test be performed. For each predictor, this p-value is the probability that, if the predictor truly has no predictive power beyond that already contained in the predictors selected so far, we could observe an improvement at least as great as that obtained by including this new predictor. A very small p-value indicates that the predictor is effective.

By default, *Uncertainty reduction* is used for sorting the selected predictors and for the optional p-value computation. Also by default, the maximum number of predictors is fixed at eight, which is nearly always more than enough. Neither of these defaults can be changed when this test is executed from within a script file, although this should not be a problem because they are excellent choices. They can be changed by the user if this test is performed via the menu system, as discussed later.

Options for Nonredundant Predictor Screening

The options for nonredundant predictor screening are identical to those for the chi-square test described in a prior section. Nevertheless, they will be repeated here for the reader's convenience, and expanded upon later. Suppose we have the following basic test:

```
NONREDUNDANT PREDICTOR SCREENING
    [ X1 X2 X3 ] ( 3 ) WITH RET ( 3 ) ;
```

The test shown above splits each indicator candidate (X1, X2, X3) into three equal bins, and it does the same for the target, RET. We can replace the number of bins for the indicators with a fraction ranging from 0.0 to 0.5 (typically 0.05 or 0.1) and the word TAILS to specify that the test employ just two bins for the predictor, the most extreme values. The following command says that cases with indicator values in the highest 0.1 (ten percent) of all cases go into one bin, cases whose indicator is in the lowest 0.1 go into the other bin, and the 80 percent middle values of the indicator will be ignored.

```
NONREDUNDANT PREDICTOR SCREENING
    [ X1 X2 X3 ] 0.1 TAILS WITH RET ;
```

Another option is to specify that instead of using equal-count bins for the target, the sign of the target (win/loss) is used by splitting the bins at zero. This implies that there are only two target bins: win and lose. (Note that using three bins is often better than splitting at zero, because this splits the target into big win, big loss, and near zero.) We split at zero by replacing the bin count with the word SIGN. The following command does this:

```
NONREDUNDANT PREDICTOR SCREENING
    [ X1 X2 X3 ] ( 3 ) WITH RET SIGN ;
```

It was already mentioned that p-values can be computed. When you test many predictors and repeatedly look for the best to add, lucky predictors will be favored. This is called selection bias, and it can be severe, causing worthless predictors to be selected. A Monte-Carlo Permutation Test (MCPT) can be employed to approximately control for selection bias. This is done by appending MCPT = Nreps at the end of the command, as is shown in the following example:

```
NONREDUNDANT PREDICTOR SCREENING
    [ X1 X2 X3 ] ( 3 )  WITH RET ( 3 ) MCPT=100 ;
```

Finally, we can combine any or all of these options as shown below:

```
NONREDUNDANT PREDICTOR SCREENING
    [ X1 X2 X3 ] 0.1 TAILS  WITH RET SIGN MCPT=100 ;
```

Running Nonredundant Predictor Screening from the Menu

As an alternative to issuing commands in a script file, nonredundant predictor screening can be run from the menu by clicking Describe / Nonredundant predictor screening. The following dialog box will appear:
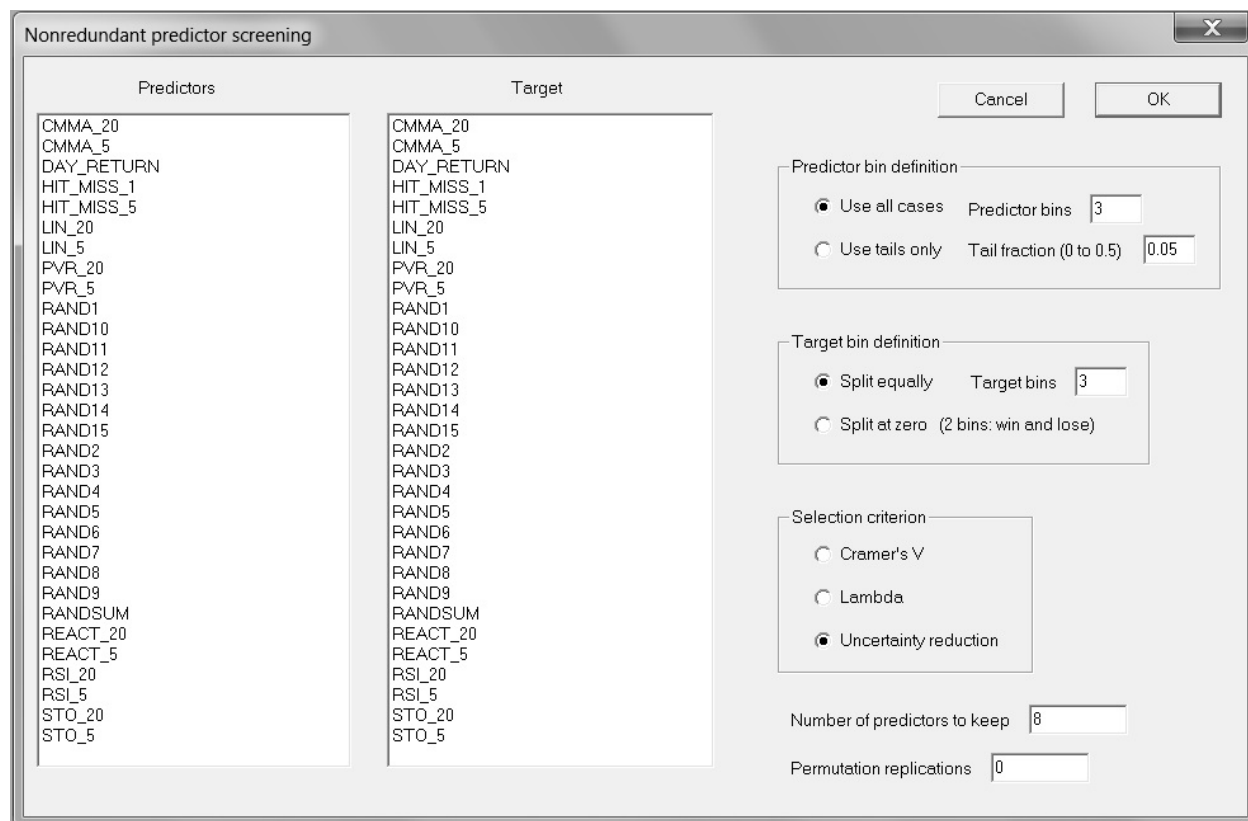
Figure 3: Dialog box for nonredundant predictor screening

The user selects one or more predictors (usually indicators, but target variabless can be selected as well) from the left-hand list. Selection can be done in any of three ways:

1) Drag across a range of predictors to select all of them.

2) Select a predictor, hold down the Shift key, and select another predictor to select all predictors that lie between them.

3) Hold the *Ctrl* key while clicking any predictor to toggle it between selected and not selected, without impacting existing selections.

A single target must be selected.

The user must specify how the predictor bins are defined. There are two choices:

1) Select 'Use all cases' and enter the number of equal-count bins to employ.

2) Select 'Use tails only' and enter the tail fraction on each side (greater than zero and less than 0.5) to employ two bins, one for each tail, with interior values ignored. Values of 0.05 or 0.1 are typical. Keeping too little of each tail usually results in such rapid reduction in the mean per-cell count that very few predictors can be safely selected.

The user must specify how the target bins are defined. There are two choices:

1) Select 'Split equally' and specify the number of equal-count bins to employ. In most cases this is the best choice, with three bins used. Three equal-count bins split the target into 'big win', 'big loss', and 'fairly inconsequential' trade outcomes. (This labeling assumes that the target distribution is fairly symmetric, the usual situation.)

2) Select 'Split at Zero', in which case there are two bins, with bin membership defined by the sign (win/lose) of the target. A target value of zero is considered to be a win.

Note that if the user selects 'Use tails only' for the predictor and 'Split equally' for the target, the target bin thresholds are defined by the entire target distribution, not the distribution in the predictor tails only. In most cases this results in a more sensitive test than determining the split points using predictor tails only.

The user may select which of the three available measures of predictive power is to be used for predictor selection and optional Monte-Carlo Permutation Tests. The default, Uncertainty reduction, is almost always far superior to the two alternatives, so it should be chosen unless the user wishes to experiment.

The number of predictors to keep defaults to 8. In nearly all cases this is more than enough, as cell count reduction will render the tests meaningless before even 8 are reached, unless there happens to be a huge number of history cases available. The user may set this to any desired quantity. For example, if extensive history of many markets is in the database, increasing this above 8 may be appropriate. The only advantage to specifying a smaller number to keep is that smaller numbers kept will speed a Monte-Carlo Permutation Test.

Finally, the user can specify 'Permutation replications' to be a number greater than one in order to perform a Monte-Carlo Permutation Test. If this is done, at least 100 replications should be used, and 1000 is not unreasonable. The smallest computed p-value possible is the reciprocal of the number of replications. The MCPT will produce a column labeled 'Inclusion p-value'

which contains the probability that, if the predictor truly has no predictive power beyond that already contained in the predictors selected so far, we could observe an improvement at least as great as that obtained by including this new predictor. A very small p-value indicates that the predictor is effective.

Examples of Nonredundant Predictor Screening

We end this discussion with a few progressive examples of nonredundant predictor screening. First, we'll explore how the algorithm behaves in a simple contrived application. We generate a set of random numbers by means of the following commands:

```
TRANSFORM RAND1 IS RANDOM ;
TRANSFORM RAND2 IS RANDOM ;
TRANSFORM RAND3 IS RANDOM ;
TRANSFORM RAND4 IS RANDOM ;
TRANSFORM RAND5 IS RANDOM ;
TRANSFORM RAND6 IS RANDOM ;
TRANSFORM RAND7 IS RANDOM ;
TRANSFORM RAND8 IS RANDOM ;

TRANSFORM RAND_SUM IS EXPRESSION (0) [
   RAND_SUM = RAND1 + RAND2 + RAND3 + RAND4 + RAND5
   ] ;

TRAIN ;
```

The commands just shown generate eight different random series, and then it creates a new series *RAND_SUM* by summing the first five of these eight random series. Note that the TRAIN command is required because transforms are not computed until they are needed, such as for training, walkforward testing, or cross validation.

Now suppose the following nonredundant predictor screening command appears:

```
NONREDUNDANT PREDICTOR SCREENING
    [ RAND1 RAND2 RAND3 RAND4 RAND5 RAND6 RAND7 RAND8 ](2)
    WITH RAND_SUM (2) MCPT=1000 ;
```

This command says that we will test all eight random series as predictors of a target which consists of the sum of the first five of the predictors. Two bins will be used for the predictors, and two for the target. A Monte-Carlo Permutation Test having 1000 replications will be performed. Because we have contrived this test, we know that we can expect RAND1 through RAND5 to be selected, and the others rejected.

The output produced by this command appears below:

Nonredundant predictive screening between predictors (2 bins) and RAND_SUM (2 bins) with n=5715
  Selection criterion is Uncertainty reduction

Variable	Mean count	100*Cramer's V	100*Lambda	100*UncertReduc	Inclusion p-value
RAND2	1428.8	32.74	32.73	7.88	0.0010
RAND4	714.4	45.72	33.11	16.36	0.0010
RAND3	357.2	55.38	50.72	25.33	0.0010
RAND5	178.6	63.86	51.17	35.30	0.0010
RAND1	89.3	72.74	68.95	47.75	0.0010
RAND7	44.6	72.97	68.95	48.23	0.3360
RAND6	22.3	73.31	68.95	48.97	0.6130
RAND8	11.2	73.90	68.95	50.09	0.9140

We see that as expected, RAND1 through RAND5 are selected first, ordered from most to least predictive. Each time one of them is added to the predictor set, the uncertainty about the target is reduced by a considerable amount. Moreover, the p-value for each of them is 0.001, the minimum possible with 1000 replications. Then, adding the remaining three candidates produces only a minor improvement in the predictability measures. Moreover, the p-values for these last three candidates are insignificant.

Two comments about these p-values must be made. First, it is largely coincidental that they increase in this example (0.3360, 0.6130, 0.9140). In the chi-square test discussed in the prior section, the p-values always increase because by definition the candidates are less predictive as we move down the table. But in this test, the p-values do not refer to the candidates in isolation. Rather, they refer to the *additional* predictability gained by including each successive candidate. Naturally, there is a tendency for additional power to decrease as the candidate set shrinks. We will be choosing from a less and less promising pool of candidates as the best are taken up. But it is always possible that after some only slightly advantageous predictor is added, its presence suddenly makes some previously worthless candidate a lot more powerful due to synergy. In this situation its p-value will probably be better (smaller) than the prior p-value.

The other thing to remember about p-values, not just in this test but in *any* hypothesis test, is that if the null hypothesis is true (here, the candidate predictor is worthless), the p-values will have a uniform distribution in the range zero to one. It is a common misconception among statistical neophytes that if a null hypothesis is true, this will always be signaled by the p-value being large. Not so. If the null hypothesis is true, one will still obtain a p-value less than 0.1 ten percent of the time, and a p-value less than 0.01 one percent of the time. This gets to the very

core of a hypothesis test: if you choose to reject the null hypothesis at a given p-value, and the null hypothesis is true, you will be wrong the p-value fraction of the time.

Now let's modify this test by increasing the resolution. The target will have three bins instead of two, and the predictors will be increased to ten bins. This is accomplished with the following command:

```
NONREDUNDANT PREDICTIVE SCREENING
    [ RAND1 RAND2 RAND3 RAND4 RAND5 RAND6 RAND7 RAND8 ](10)
    WITH RAND_SUM (3) MCPT=100 ;
```

The output, shown below, is rendered considerably different by this increase in resolution.

Nonredundant predictive screening between predictors (10 bins) and RAND_SUM (3 bins) with n=5715
  Selection criterion is Uncertainty reduction

Variable	Mean count	100*Cramer's V	100*Lambda	100*UncertReduc	Inclusion p-value
RAND5	190.5	29.71	22.34	8.31	0.0010
RAND3	19.1	44.53	34.57	19.40	0.0010
------------------> Results below this line are suspect due to small mean cell count <------------------					
RAND4	1.9	64.50	53.67	44.49	0.0010
RAND1	0.2	93.32	88.08	88.88	1.0000
RAND2	0.0	99.78	99.55	99.62	1.0000
RAND8	0.0	100.00	100.00	100.00	1.0000
RAND6	0.0	100.00	100.00	100.00	1.0000
RAND7	0.0	100.00	100.00	100.00	1.0000

We see here that the mean count per cell plummets by a factor of ten as each new candidate is added, because the predictors have ten bins. As a result, by the time we get to the third candidate, the count is below the rule-of-thumb threshold of five, so a warning is printed. The basic results are still reasonable: RAND1 through RAND5 are selected before the others, and the addition of each considerably reduces the uncertainty. Adding the final three does almost nothing. However, we do note that although the p-values for the first three predictors are nicely small, they jump to 1.0 for RAND1 and RAND2, when we would have liked to see them also be small. This is a direct result of the diminished mean count per cell, and it is a strong argument for using as few bins as possible.

We now advance to a more practical example involving actual market indicators and a forward-looking target. Here is the command:

```
NONREDUNDANT PREDICTIVE SCREENING
    [ CMMA_5 CMMA_20 LIN_5 LIN_20 RSI_5 RSI_20 STO_5
      STO_20 REACT_5 REACT_20 PVR_5 PVR_20 ] ( 2 )
    WITH DAY_RETURN ( 3 ) MCPT=1000 ;
```

This configuration is one of the three most commonly used. Splitting the target into three bins is generally a perfect choice because the three bins have practical meaning: the trade is a big winner, a big loser, or has an intermediate profit/loss that is relatively inconsequential. The predictor is a little more complex. If an important goal is to capture as many predictors as possible, then using two equal bins as is done here is the best choice. The mean count per cell decreases most slowly with this choice. Another option is to use three equal bins. This captures more information from the predictors, with the price being more rapid decrease in mean count per cell. Finally, the user may opt to use only the tails of the predictors. Since for most indicators, the majority of the predictive information is in the tails, this usually provides more predictive power and produces results that correlate relatively well with model-based trading systems. But if the tail fraction is small, the dropoff in mean count per cell is so rapid that few predictors will be retained.

The command just shown produces the following output:

Nonredundant predictive screening between predictors (2 bins) and DAY_RETURN (3 bins) with n=5715
  Selection criterion is Uncertainty reduction

Variable	Mean count	100*Cramer's V	100*Lambda	100*UncertReduc	Inclusion p-value
CMMA_20	952.5	10.14	6.17	0.47	0.0010
PVR_20	476.3	8.41	6.17	0.66	0.0020
CMMA_5	238.1	9.65	6.96	0.86	0.0140
LIN_20	119.1	10.43	7.03	1.02	0.4410
STO_20	59.5	11.98	7.74	1.38	0.2420
REACT_5	29.8	14.59	9.48	2.09	0.1500
RSI_20	14.9	17.56	11.21	3.06	0.4300
REACT_20	7.4	20.77	12.65	4.36	0.6950
LIN_5	---				
PVR_5	---				
RSI_5	---				
STO_5	---				

By default (changeable only from the menu interface), the best eight predictors are kept. Those not selected are still listed for the user's convenience, but the names are followed by dashes (---). The first two predictors are extremely significant, and the third is also quite significant. Then significance plummets with the fourth predictor.

There is a vital lesson in this example: later predictors improved the uncertainty much more than the early predictors, yet their significance is inferior. For example, adding REACT_20 improved the uncertainty reduction from 3.06 to 4.36, yet its p-value was 0.695. This is a common outcome; when numerous predictors are present, adding even a random, worthless predictor which is optimally selected from the remaining candidates can refine the predictive power of the kept predictors to a considerable degree. The p-value provides a valuable indication of whether the improvement is due to true predictive power or just due to selection bias.

Finally, we'll slightly modify the example above by examining only the tails of the predictors. Here is that command, followed by its output:

```
NONREDUNDANT PREDICTIVE SCREENING
    [ CMMA_5 CMMA_20 LIN_5 LIN_20 RSI_5 RSI_20 STO_5
      STO_20 REACT_5 REACT_20 PVR_5 PVR_20 ]   0.1 TAILS
    WITH DAY_RETURN ( 3 ) MCPT=1000 ;
```

Nonredundant predictive screening between predictors (2 bins) and DAY_RETURN (3 bins) with n=5715
  Selection criterion is Uncertainty reduction
  Tails only, each tail fraction = 0.100

Variable	Mean count	100*Cramer's V	100*Lambda	100*UncertReduc	Inclusion p-value
RSI_20	190.3	21.76	12.38	2.18	0.0010
PVR_5	26.2	19.85	10.26	3.82	0.3720
REACT_5	6.5	26.15	13.40	6.43	0.6540

------------------> Results below this line are suspect due to small mean cell count <------------------

Variable	Mean count	100*Cramer's V	100*Lambda	100*UncertReduc	Inclusion p-value
PVR_20	1.4	37.37	21.05	12.33	0.0790
RSI_5	0.6	37.96	26.67	13.21	0.3820
CMMA_20	0.3	37.96	26.67	13.21	0.4480
CMMA_5	0.1	37.22	26.92	12.65	0.5560
LIN_5	0.1	35.19	24.00	11.09	0.5540
LIN_20	---				
STO_5	---				
STO_20	---				
REACT_20	---				

There are several items to note concerning these results:

- Performance measures in this example, which employed only the tails (most extreme values) of the predictors showed predictability very much greater than in the prior example, which used the entire distribution.

- Restricting data to tails only caused the mean count per cell to drop much more rapidly than when the entire distribution was used. As a result we can trust this report for only the first three predictors.

- The order of selection is different, indicating that the information in just the tails can be different from the information in the entire distribution.

- Only one predictor had a significant p-value. Remember that this procedure is different from the chi-square procedure of the prior section. The chi-square procedure tested each indicator separately. On the other hand, the nonredundant predictor procedure here tests them sequentially, evaluating the successive inclusion of more predictors. So what we see here is that the information in the tails of RSI_20 contains all of the predictive power available in the set of candidates. No other candidate's tails can significantly add to the predictive information available in RSI_20's tails.

- The uncertainty reduction obtained using the tails of just RSI_20, which was 2.18 percent, was almost three times as great as that obtained using the entire distribution of all three significant predictors (0.86 percent) in the prior example! So although we pay a high price in mean per-cell count by using tails only, the increase in predictive power with fewer predictors is nearly always a worthy tradeoff.

- Note that it is possible for these predictive power measures to decrease as we add a predictor. This is extremely rare for Uncertainty reduction and Lambda, essentially a pathological situation. However, it is fairly common for Cramer's V and is a natural part of its definition.

Triggering Trades With States and Events

By default, every case in the dataset is presented to models during training and testing. In some situations, it can be useful to allow the possibility of trades for only a subset of the dataset. We may wish to allow trading only when the market is in a certain state, or when a certain event occurs. This is enabled with the following commands:

```
TRIGGER ALL VarName ;
TRIGGER FIRST VarName ;
REMOVE TRIGGER ;
```

VarName is the name of a variable that will define the trigger. The trigger variable should ideally be binary 0/1, although this is not required. The threshold for triggering is 0.5. Values greater than this are treated as *true* and values less than or equal to this are considered to be *false*.

The *TRIGGER ALL* command allows trades whenever the trigger variable is true. The audit log will note how many cases and the percent of the database have the value *true*.

The *TRIGGER FIRST* command allows trades for only the first day (or bar) in a string of *true* days (bars) in the same market. In other words, a trade is allowed for days in which the trigger variable is *true* AND the trigger variable is *false* for the prior day in that market. The day prior to the first day in the database is assumed to be *false*. The audit log will note how many cases are *true* and how many are *first true*.

The trigger variable may be part of the database, or it may be an expression transform.

The *REMOVE TRIGGER* command removes the trigger and makes the entire database eligible for trading.

You may use multiple triggers in a script file to perform multiple tests. For example, the following commands would test two different trigger scenarios, and then test the entire database:

```
TRIGGER ALL HiTrig ;
CROSS VALIDATE BY YEAR ;
TRIGGER ALL LoTrig ;
CROSS VALIDATE BY YEAR ;
REMOVE TRIGGER ;
CROSS VALIDATE BY YEAR ;
```

Cross-sectional and pooled indicators are computed without regard to trigger states. If the FRACTILE THRESHOLD option is used, only triggered cases take part in the rank ordering. In this situation, it is recommended that the trigger variable be based on an index.

CUDA Processing

If your computer contains a CUDA-enabled device, it can be used for accelerating some applications. (Currently, only GRNN training can use a CUDA device.) In order to implement CUDA acceleration, the following line must appear anywhere in a script control file:

```
CUDA ;
```

CUDA processing will occur *after* this command is encountered. Thus, you should insert this command before any TRAIN, CROSS VALIDATE, or WALK FORWARD command if you want it to employ CUDA acceleration.

Note that the CUDA option has temporarily been disabled. It will likely be restored in a future release.

Performance statistics

This section discusses the performance statistics that appear in the audit and report logs. Different applications will use different statistics, so it is likely that not all of the information presented here will appear in every application. Here is a section of the output from a walkforward run, with a running commentary:

First, the year and number of training and testing cases is listed.

```
-------------------------------------------------------
YEAR 2006 training 35497 cases, testing 4399
-------------------------------------------------------
```

The model about to be presented, as well as its target and optimization criterion (pf; profit factor), is named and described. The fact that the user specified the FRACTILE THRESHOLD and XVAL STEPWISE options is noted. The user demands that at least 0.1 of the training cases lie outside each threshold (upper and lower). The best model has a criterion of -0.2592, which is the log of its profit factor. This is not a good omen, as it shows that even the best in-sample model loses money in a cross-validated evaluation. The model is linear regression (LINREG) with one predictor chosen. That predictor's coefficient and the constant offset are printed for the user's edification.

```
LINREG Model VAL1 predicting RETURN1
FRACTILE THRESHOLD specified
XVAL Optimizing pf with min fraction=0.100  (Final best crit
= -0.2592)
Regression coefficients:
        0.019780  FILTER_VALUE
        2.362443  CONSTANT
```

In-sample performance results now appear. The upper threshold (outer, or primary) is 0.79832, which results in 3550 of the 35497 training cases equaling or exceeding it. This is ten percent of the training cases, which is to be expected because the user specified FRACTILE THRESHOLD and the optimized upper threshold is about 0.8. The lower threshold and inner thresholds are similarly displayed. Other performance criteria include the mean squared error, R-squared (which can be negative if the model is anti-predictive), and area under the ROC curve. The buy-and-hold and sell-short-and-hold profit factors (reciprocals, of course) are printed for reference. If we were to buy all cases above the upper threshold and sell all cases below the lower threshold, we would achieve a profit factor of 0.887. This losing strategy is expected when we note that this training period is strongly up-trending (buy-and-hold pf=2.4) but the separately optimized thresholds result in more than twice as many short trades as long trades. When the long and short sides are separated, we see that a long-only strategy results in a profit factor of 2.966, which is 1.236 times the buy-and-hold profit factor of 2.4. The short side is similarly shown. Finally, since the user specified the FRACTILE THRESHOLD option, profit factors from balanced positions with 0.01, 0.1, 0.25, and 0.5 trades outside the threshold are shown. Observe that these are all profitable, in sharp contrast to the unbalanced long/short strategy.

```
Outer hi threshold = 0.79832  with 3550 of 35497 cases at or
above (10.00 %)
Inner hi threshold = 0.65001  with 4551 of 35497 cases at or
above (12.80 %)
Inner lo threshold = -0.41133  with 9935 of 35497 cases at
or above (27.99 %)
Outer lo threshold  = -0.51498  with 8591 of 35497 cases at
or below (24.20 %)
MSE = 71.04773   R-squared = 0.00032    ROC area = 0.53523
Buy-and-hold profit factor = 2.400 Sell-and-hold = 0.417
Dual-thresholded outer PF = 0.887
Dual-thresholded inner PF = 0.803
   Outer long-only PF = 2.966   Ratio = 1.236
   Inner long-only PF = 2.505  Ratio = 1.044
   Inner short-only PF = 0.441   Ratio = 1.058
   Outer short-only PF = 0.493  Ratio = 1.183
Balanced (0.01 each side) PF = 1.352
Balanced (0.05 each side) PF = 1.295
Balanced (0.10 each side) PF = 1.251
Balanced (0.25 each side) PF = 1.220
Balanced (0.50 each side) PF = 1.133
```

The prior results were for in-sample data. These same statistics are now shown for the out-of-sample year.

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

These in-sample and out-of-sample results are printed for each walkforward or cross-validated year. When the run is complete, a summary appears:

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

Model VAL1...

Out-of-sample: 635 of 6343 cases at or above (10.01%), 1506
at or below (23.74%)
MSE = 69.43532   R-squared = -0.00547   ROC area = 0.50588
Buy-and-hold profit factor = 1.852   Sell-short-and-hold =
0.540
Dual-thresholded outer PF = 0.899
Dual-thresholded inner PF = 0.847
   Outer long-only PF = 1.993   Ratio = 1.076
   Inner long-only PF = 1.973   Ratio = 1.065
   Inner short-only PF = 0.560  Ratio = 1.037
   Outer short-only PF = 0.570  Ratio = 1.055
Balanced (0.01 each side) PF = 1.177
Balanced (0.05 each side) PF = 1.159
Balanced (0.10 each side) PF = 1.138
Balanced (0.25 each side) PF = 1.053
Balanced (0.50 each side) PF = 0.989
```

Since the user requested a Monte-Carlo permutation test, its results are now given. These are the approximate probabilities that results as good as or better than those obtained could have arisen from a random worthless model.

```
Monte-Carlo Permutation Test p-vals...
   R-square: 0.2830
   ROC area: 0.3460
   Profit factor: 0.1750
      Long only: 0.4220
      Short only: 0.2930
   Balanced 0.01: 0.3570
   Balanced 0.05: 0.2240
   Balanced 0.10: 0.1190
   Balanced 0.25: 0.2580
   Balanced 0.50: 0.8430
```

Since this run was testing a filter model (Haugen Alpha in this case), a table appears comparing the performance of the original figures being filtered versus the model just tested. The leftmost column shows the fraction of the cases being kept. The next column shows the profit factor obtained by buying the specified highest fraction of the data being filtered. The next column shows the corresponding profit factor obtained by buying that fraction of the model's highest predictions. The last two columns show the corresponding profit factors on the short side. In this case, the original filter data and the model have identical performance because the only variable selected for each trial year was alpha itself, and the coefficient was always positive! So these models are transparent, giving exactly the same ranked predictions as the original filter data. We see, for example, that if we were to buy the largest 25 percent of the alphas or model predictions, we would obtain a profit factor of 1.904. Similarly, if we sell the smallest one percent of the alphas or model predictions, we would obtain a profit factor of 0.911. When we train a practical model, we hope that most or all of the model profit factors exceed the filter-variable profit factors.

```
Filter versus model profit factors (raw data) for various
extremes
```

Fraction	Filter long	Model long	Filter short	Model short
0.01	1.579	1.579	0.911	0.911
0.05	1.860	1.860	0.700	0.700
0.10	1.984	1.984	0.635	0.635
0.25	1.904	1.904	0.572	0.572
0.50	1.822	1.822	0.531	0.531
0.75	1.886	1.886	0.545	0.545
0.90	1.888	1.888	0.544	0.544
0.95	1.877	1.877	0.540	0.540
0.99	1.860	1.860	0.539	0.539

Permutation Training

An essential part of model-based trading system development is training a model (or set of models, committees, et cetera) to predict the near-future behavior of the market. Training involves finding a set of model parameters that maximizes a measure of performance within a historical dataset. Much of the time, the resulting performance will be excellent. However, this superb performance is at least partially due to the fact that the training process treated noise or other unique properties of the data as if they were legitimate patterns. Because such patterns are unlikely to repeat in the future, the performance obtained due to training is optimistic, often wildly so.

This undue optimism requires the responsible developer to answer two separate and equally important questions:

1) What is the probability that if the trading system (model(s) et cetera) were truly worthless, good luck could have produced results as good as those observed? We want this probability, called the *p-value*, to be small. If it turns out that there is an uncomfortably large probability that a truly worthless system could have given results as good as what we obtained just by being lucky, we should be skeptical of the trading system at hand.

2) What is the average performance that can be expected in the future (assuming that the market behavior does not change, which of course may be an unrealistic but unavoidable assumption)?

It is vital to understand that these are different, largely unrelated questions, and a responsible developer will require a satisfactory answer to *both* of them before signing off on real-life trading. It is possible, especially if extensive market history has been tested, for the answer to the first question to be a nicely small probability, even though the expected future performance is mediocre, hardly worth trading real money. It is also possible for the expected future performance to be enticing, but with a high probability of having been obtained by random good luck from a truly worthless system. Thus, we need both properties: it must be highly unlikely that a worthless system would have had sufficient good luck to do as well as we observed, and the expected future performance must be good enough to be worth putting down real money.

The standard method for answering the second question, and sometimes the first as well, is walkforward testing. The performance obtained in the pooled out-of-sample (OOS) period is an unbiased estimate of future performance. (Note that the term *unbiased* is used here in the loose sense of being without prejudice, as opposed to the much stronger statistical sense.) If the OOS cases are independent (which they would not be if the targets look ahead more than one bar), an ordinary bootstrap test can answer the first question. Even if the cases have serial dependence, special bootstraps can do a fairly good job of answering the first question.

But the problem with the walkforward approach is that it discards a lot of data, often the majority of historical data. Only the pooled OOS data can be used to answer the two questions. This is a high price to pay. There is an alternative approach to answering the first question, and which can also provide at least a decent hint of an answer to the second question, while using *all* available historical data.

Unfortunately, this alternative approach has a high price of its own: time. Training time is multiplied by a factor of at least 100, and as much as 1000 if a thorough job is to be done. This precludes some analyses. But it is a great addition to the developers toolbox.

This approach is invoked with the following command:

`TRAIN PERMUTED Nreps ;`

If this command appears, *no database can be read or appended.* This is because all indicators and targets must be able to be recomputed from the original and permuted market(s). This recomputation would obviously not be possible for precomputed indicators and targets in a database. Also, the REMOVE ZERO VOLUME command described on Page 3 must appear before the READ MARKET HISTORIES command.

TRAIN PERMUTED operates similarly to the ordinary TRAIN command, except that in addition to the usual computation of indicators and targets from the data, it also recomputes indicators and targets, and retrains all models, *Nreps*-1 times. Each of these repetitions is done on market histories whose changes have been shuffled. Thus, the permuted markets have exactly the same bar-to-bar changes as the original, unpermuted markets, but in a different order. This random shuffling obviously destroys any predictable patterns in the markets, giving us a large number of examples of truly worthless trading systems; no matter how 'good' a trading system may be, it will be worthless when presented with random market data! Some of these shuffled market histories will allow the trained trading systems to be lucky, and others will not. Thus, by counting how often we obtain a trading system from shuffled data whose performance is at least as good as that obtained from the original, unshuffled history, we can directly answer the first question.

Another way of looking at this algorithm is by realizing that if the trading system being developed is truly worthless, presenting it with real market data is no better than presenting it with random data. Its performance is no more likely to be outstanding than permuted performance. We would expect its performance to be somewhere in the middle of the heap, better than some and worse than others. On the other hand, suppose our trading system is truly able to detect authentic patterns in market data. Then we would expect its performance on the real data to exceed that of most or all of the systems developed on shuffled data.

In particular, suppose our trading system is truly worthless, and also suppose that we have *Nreps* trading results. One of these is from the original, unpermuted data, and the others are from shuffled data. Then there is a 1/*Nreps* probability that the original will be the best (none of the others exceeded it). There is a 2/*Nreps* probability that it will be at least second-best (at most one other exceeded it) and so forth. This can be visualized by imagining the performance measures laid out in a sorted line and knowing that if the system is worthless, it can land in any position with equal probability. In general, if *k* shuffled performance measures equal or exceed the unshuffled measure, the probability addressed by Question 1 above is (*k*+1)/*Nreps*.

This method of estimating probabilities of performance results can be seen in Figure 4 below. This is a histogram of the long-only profit factors obtained by a model called LIN_LONG when trained on 99 randomly shuffled market histories as well as the original unshuffled history. The profit factor from the original data is shown as a vertical red line. Note that this performance is one of the best, with a probability of 0.06 under the null hypothesis that the model is worthless.
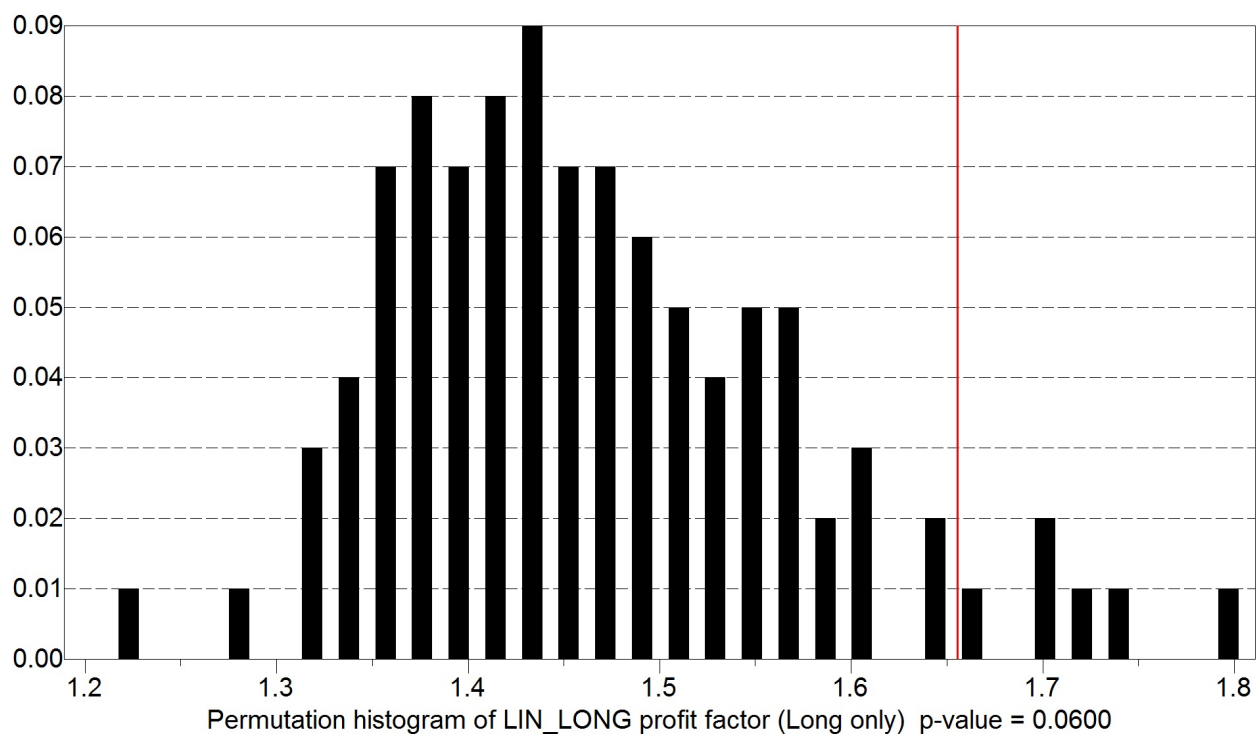
Permutation histogram of LIN_LONG profit factor (Long only) p-value = 0.0600
Figure 4: Histogram of TRAIN PERMUTED performance

The Components of Performance

The primary task of permutation training is estimating the probability that a worthless system could benefit from good luck sufficiently to attain the observed performance level. This provides an answer to Question 1 shown at the start of this chapter. However, Question 2 is equally important: What level of performance can we expect in the future? The gold standard for answering this question is walkforward testing. However, as a byproduct of probability estimation we can compute a rough but still useful estimate of one specific measure of future performance: Total return.

The fundamental basis of this estimation is the relationship among four loosely defined aspects of performance:

Total - The total return of the trading system after training

Skill - The true ability of the trading system to find and profit from authentic market patterns

Trend - If the market significantly trends up (or down) and the trading system is unbalanced in favor of long(or short) positions, this unbalance will generate profits on average even without intelligent individual trade decisions. If the market trend is large, and if the trading system development software has the ability to create unbalanced systems (the most common situation), the software will favor unbalanced systems that take advantage of the trend, even if these systems do not make many intelligent individual trades.

Bias - The process of training the system induces *training bias* by tuning its trade decisions to not only the authentic patterns represented in the dataset, but also to patterns that are unique to the historical dataset and may not be repeated in later use. If the system development software employs very powerful models, these models may go to great lengths to capture every pattern they see in the data, authentic and temporary, with the result that apparent performance will be greatly inflated.

Although Equation 7 below is far from a rigorous statement of how these four items are related, it is usually fairly close when these quantities are components of the total return of the trading system, and it is a useful mental tool for understanding the performance of a trading system after it has been trained:

$$Total \ = \ Skill \ + \ Trend \ + \ Bias \qquad\qquad (7)$$

The market data is permuted in such a way that any underlying trend is almost exactly preserved. As a result, the training algorithm will have the opportunity to generate unbalanced systems that take advantage of the trend, just as they do with the unpermuted market history. Therefore, we

can expect that the *Trend* component of the total return will be about the same for the permuted markets as for the unpermuted market.

Also, the permuted markets will, on average, present to the training algorithm the same opportunity to capitalize on inauthentic patterns as did the unpermuted market. Thus, we can expect that the *Bias* component of the permuted returns will, on average, be about the same as that for the unpermuted return.

The only difference between the returns from the unpermuted and the permuted markets is the *Skill* component, which cannot exist in permuted data. In other words, Equation 8 below represents the average total return obtained from training on permuted markets.

$$TotalForPermuted\ =\ Trend\ +\ Bias \qquad (8)$$

As a point of interest, look back at Figure 4 on Page 180. That figure is based on profit factors, while the equations just shown are more applicable to total return. Nonetheless, the principle is the same for either measure of performance. The bars in this histogram represent the values of Equation 8 above (plus the single observation for the unpermuted market, which is also used to compute the histogram). The red vertical bar is the value represented by Equation 7 above. We can thereby see the effect of *Skill* on performance relative to the performance of the trained trading systems when *Skill* is not involved. If the *Skill* component is significant, we would expect the vertical red bar to be on the far right side of the histogram, pushed there by the ability of the trading system to find authentic patterns in the market. Conversely, if the *Skill* component of performance is small, the red bar is more likely to be somewhere in the interior mass of the histogram.

Additional information can be gleaned from the permutation training by analyzing Equation 8 in more detail. When the trading system is trained by maximizing performance on a permuted dataset, the training algorithm will attempt to simultaneously maximize both components of Equation 8: it will try to produce an unequal number of long and short trades (assuming that the user allows unbalanced systems) in order to take advantage of any long-term trend in the market(s), and it will try to take individual trades in such a way as to capitalize on any patterns it finds in the data. Of course, since it is operating on randomly permuted data, any patterns it finds are inauthentic, which is why the *Skill* component plays no role.

Now suppose we have a system that makes only long trades, and it does so randomly; individual trades are made with no intelligence whatsoever. Equation 9 shows its expected return:

$$LongExpectedReturn\ =\ \frac{BarsLong}{TotalNumberOfBars}\ *\ TotalTargets \qquad (9)$$

In this equation, *BarsLong* is the number of bars in which a long position is signaled, and *TotalNumberOfBars* is the number of bars in the entire dataset whose performance is being evaluated. This ratio is the fraction of the number of bars in which a long position is signaled. Looked at another way, this ratio is the probability that any given bar will signal a long position. *TotalTargets* is the sum of the target variable across all bars. This will be positive for markets with a long-term upward trend, negative for those with downward trend, and zero for a net flat market. So if our system is without intelligence, with signals randomly given, we will on average obtain this fraction (the fraction of time a long signal is given) of the total return possible.

A similar argument can be made for a short-only system, although the sign would be flipped because for short systems a positive target implies a loss, and a negative target implies a win. Combining these two situations into a single quantity gives Equation 10, the expected net return (the *Trend* performance component) from a random system that contains both long and short signals.

$$Trend = NetExpectedReturn = \frac{BarsLong - BarsShort}{TotalNumberOfBars} * TotalTargets \qquad (10)$$

When we train a trading system using a permuted market, we can note how many long and short trades it signaled and then use Equation 10 to determine how much of its total return is due to position imbalance interacting with long-term trend. Anything above and beyond this quantity is training bias due to the system learning patterns that happen to exist in the randomly shuffled market. This training bias can be estimated with Equation 11.

$$Bias = TotalForPermuted - Trend \qquad (11)$$

It would be risky to base a *Bias* estimate on a single training session. However, as long as we are repeating the permutation and training many times to compute the probability of the observed return happening by just good luck, we might as well average Equation 11 across all of those replications. If at least 100 or so replications are used, we should get a fairly decent estimate of the degree to which the training process is able to exploit inauthentic patterns and thereby produce inflated performance results.

Two thoughts are worth consideration at this point:

- *TSSB* allows various BALANCED criteria which force an equal number of long and short positions in multiple markets. In this case, *Trend* will be zero.

- Powerful models will generally produce a large *Bias*, while weak models (not necessarily a bad thing) will produce small *Bias*.

Now that the *Bias* can be estimated, we can go one or two steps further. Equation 7 can be rearranged as shown in Equation 12 below.

$$\textit{UnbiasedReturn = Skill + Trend = Total - Bias} \qquad (12)$$

This equation shows that if we subtract the *Bias* estimated with Equation 11 from the total return of the system that was trained on the original, unpermuted data, we are left with the *Skill* plus *Trend* components of the total return. Many developers will wish to stop here. Their thought is that if a market has a long-term trend, any effective trading system should take advantage of this trend by favoring long or short positions accordingly. However, another school of thought says that since deliberately unbalancing positions to take advantage of trend does not involve actual trade-by-trade intelligent picking, the trend component should be removed from the total return. This can be effected by applying Equation 10 to the original trading system and subtracting the *Trend* from the *UnbiasedReturn*, as shown in Equation 13 below.

$$\textit{Skill = BenchmarkedReturn = UnbiasedReturn - Trend} \qquad (13)$$

This difference, the *Skill* component of Equation 7, is often called the *Benchmarked return* because the *NetExpectedReturn* (*Trend* component) defined by Equation 10 can be thought of as a benchmark against which to judge actual trading performance.

These figures are all reported in the audit log file, with p-values for the profit factor and total return printed first. Here is a sample:

```
Net profit factor p = 0.0600  return p = 0.0400
Training bias = 52.3346  (67.1255 permuted return minus 14.7909 permuted benchmark)
Unbiased return = 55.4320  (107.7665 original return minus 52.3346 training bias = skill + trend)
Benchmarked return = 39.8372  (55.4320 unbiased return minus 15.5947 original benchmark = skill)
```

The *Training bias* line is Equation 11, with the term *benchmark* referring to *Trend* in the equation, and these figures averaged over all replications. This figure (52.3346 here) is the approximate degree to which the training process unjustly elevates the system's total return due to learning of inauthentic patterns.

The *Unbiased return* line is Equation 12, the actual return that we could expect in the future after accounting for the training bias. This figure includes both the true skill of the trading system as well as the component due to unbalanced trades that take advantage of market trend.

The *Benchmarked return* line is Equation 13. This is the pure *Skill* component of the total return.

Permutation Training and Selection Bias

Up to this point we have discussed only training bias, which is usually dominated by models learning patterns of noise as if they were authentic patterns. There are other sources of training bias, such as incomplete representation of all possible patterns in the available history. However, these other sources are generally small and unavoidable. So what we have covered is sufficient to handle training bias in most practical applications.

However, there is another potential source of bias that inflates performance in the dataset above what can be expected in the future. This is *selection bias*, and it occurs when we choose one or more trained models from among a group of competitors. Bias occurs in the selection process because some of the models will have been lucky while others will have been unlucky. Those models that were lucky will be more likely to be selected than those that were unlucky, yet their luck, by definition, will not hold up in the future. Thus, the performance of the best models from among a set of competitors will likely deteriorate in the future as the luck that propelled them to the top vanishes.

If the user has employed any Portfolios of type *IS* (in-sample) in the script file, the TRAIN PERMUTED command will properly handle them, and the p-values and total return performance measures (unbiased and benchmarked) will be correct, or at least as correct as is possible under the loose assumptions of the technique. In other words, the selection bias inherent in the choice of an optimal subset will be accounted for in both the p-values and in the unbiased and benchmarked total returns.

There is one vital aspect of this to consider, though. Recall from the earlier discussion of the IS type of Portfolio that this is a potentially dangerous type. This is because if one or more of the competing models is extremely powerful, its in-sample performance will be unjustifiably excellent. As a result, it will likely be included in a Portfolio. This will, in turn, make this a seriously overfitted Portfolio. Of course, this will be detected in the permutation test, but that is little comfort if you are faced with a poorly performing Portfolio..

The proper way to handle this problem is to use the OOS type of Portfolio. Unfortunately, the TRAIN PERMUTED operation cannot handle OOS Portfolios. These can be evaluated only in a WALK FORWARD test because they need OOS model results for selecting the Portfolio components. A future release of *TSSB* may include a permutation version of this test.

An example of poor IS portfolio creation due to an overfit model can be seen in the example shown below. In this example, five weak models are defined: LIN1 through LIN5. They are all identical, using the EXCLUSION GROUP option to guarantee that their solo predictors are different. As stated earlier, this option is excellent for generating committee members, but probably not the best way to generate Portfolio candidates. Only LIN1 is shown here.

A strong model, LIN_POWER, is also defined. Unlike LIN1 through LIN5, which allow only one predictor, this one allows four predictors, which is almost certainly excessive. Finally, a Portfolio is defined which optimally selects two of these six candidate Models. It is LONG ONLY, so the six models all select their predictors by maximizing the long profit factor.

```
MODEL LIN1 IS LINREG [
   INPUT = [ CMMA_5 CMMA_20 LIN_5 LIN_20 RSI_5 RSI_20
     STO_5 STO_20 REACT_5 REACT_20 PVR_5 PVR_20 ]
   OUTPUT = DAY_RETURN
   MAX STEPWISE = 1
   CRITERION = LONG PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   EXCLUSION GROUP = 1
   ] ;

MODEL LIN_POWER IS LINREG [
   INPUT = [ CMMA_5 CMMA_20 LIN_5 LIN_20 RSI_5 RSI_20
     STO_5 STO_20 REACT_5 REACT_20 PVR_5 PVR_20 ]
   OUTPUT = DAY_RETURN
   MAX STEPWISE = 4
   CRITERION = LONG PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   ] ;

PORTFOLIO Port_Power [
   MODELS = [ LIN1 LIN2 LIN3 LIN4 LIN5 LIN_POWER ]
   TYPE = OPTIMIZE 2 IS
   NREPS = 1000
   LONG ONLY
   EQUALIZE PORTFOLIO STATS
   ] ;
```

The Portfolio optimizer selected LIN4 and LIN_POWER (of course) as the best components. Permutation training produced the distribution of profit factors shown on the next page.
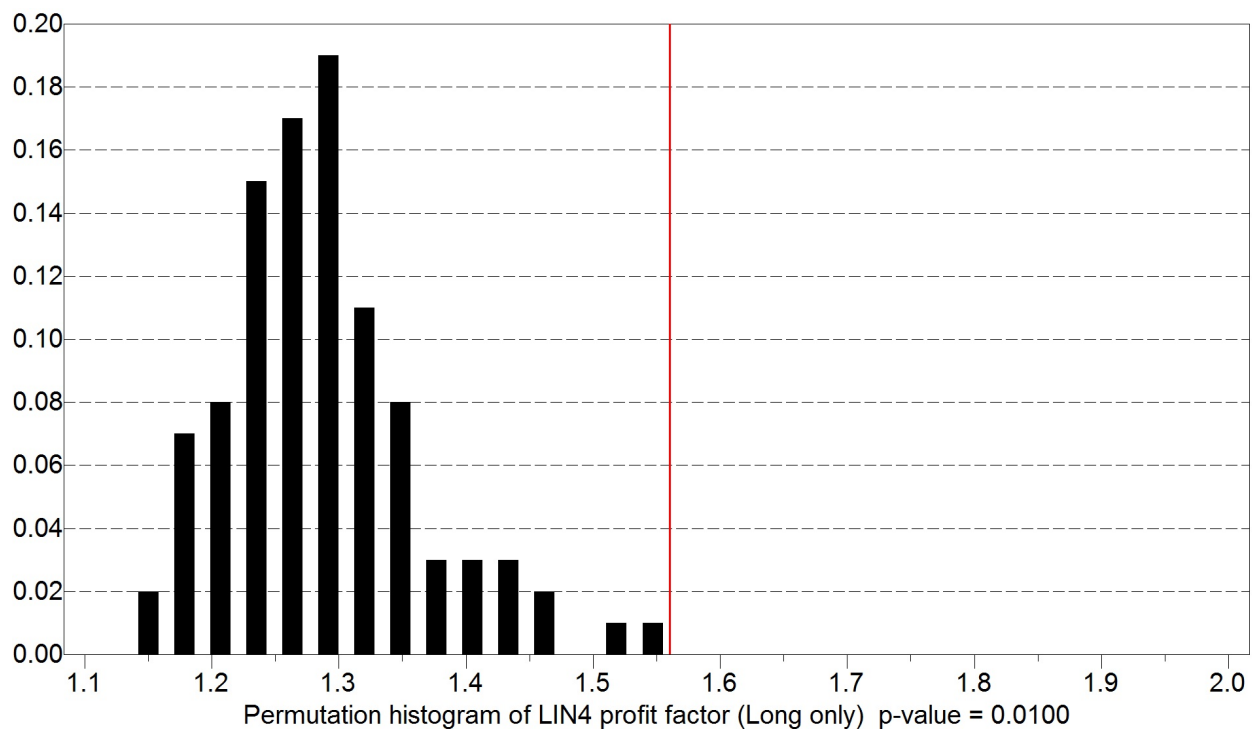
Figure 5: Permutation histogram of the weak model
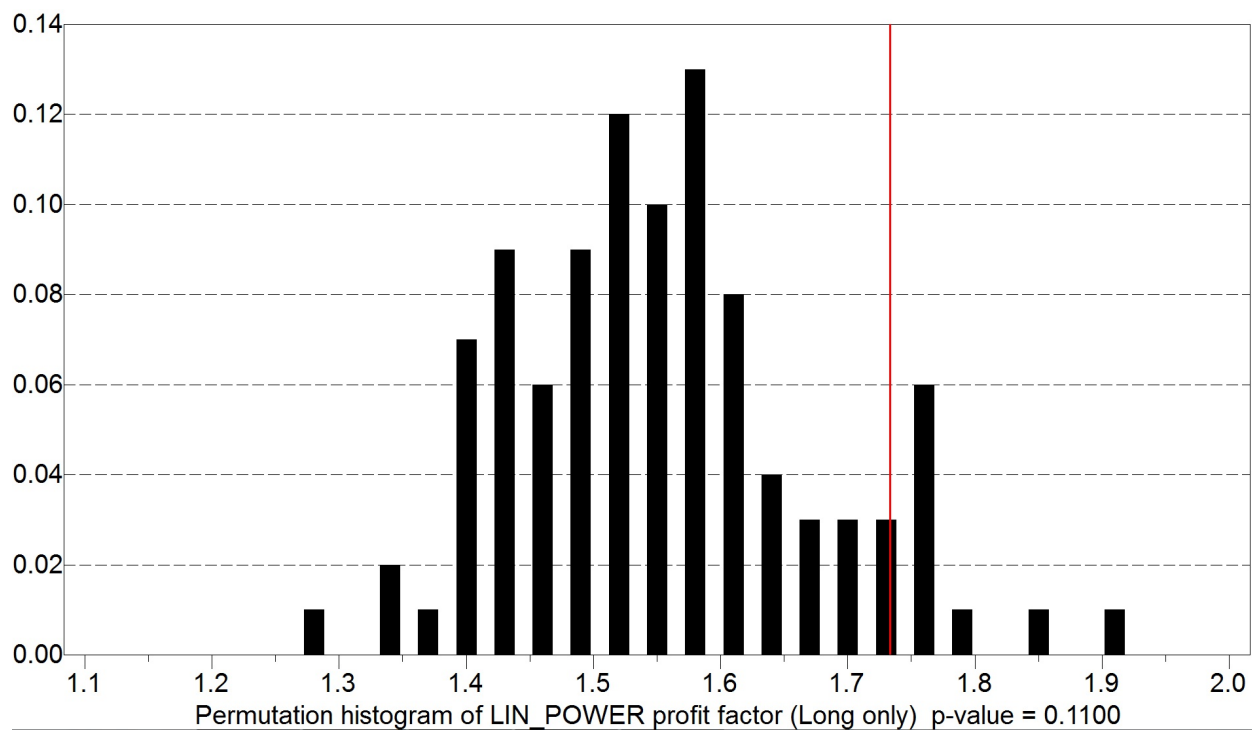
Figure 6: Permutation histogram of the overfitted model

Look at the histograms shown on the prior page. Observe that, as expected, LIN_POWER had a much higher profit factor than LIN4. This is because this is an in-sample computation, and LIN_POWER is a much more powerful model than LIN4. But along with this greater profit factor for the original data, the profit factors of shuffled data were also greater for LIN_POWER than for LIN4; LIN_POWER's histogram is considerably shifted to the right of that of LIN4. In fact, LIN_POWER's p-value of 0.11 is very inferior to that of LIN4, 0.01.

Finally, let's explore the performance partitioning for these models and the optimal Portfolio. These are shown on the next page. Note first that the p-values for the return have the opposite ordering as those for the profit factor (0.1 versus 0.01). This is not unusual; total return and profit factor have surprisingly little relationship. It may be that one model has few but excellent trades, resulting in a high profit factor and low total return, while another model has many trades that are only moderately good. This model may have a profit factor barely exceeding one, yet have a high total return by virtue of its large number of trades. Experience shows that profit factor is a vastly superior measure of performance than total return.

The most salient figure is the training bias: the weak model LIN4 has a training bias of 37.4672, while the strong model LIN_POWER has a training bias of 58.8356. This great disparity should not be surprising. Interestingly enough, when the bias is subtracted from the performance of each model, the unbiased returns of 59.6410 for LIN4 and 57.8350 for LIN_POWER are almost equal. The benchmarked returns are also almost equal. So we see that, at least in terms of total return, the weak and the strong model have about the same skill at detecting valid patterns in the market data. It's just that the more powerful model's returns are inflated by its greater ability to learn inauthentic patterns in the data.

Finally, let's look at the Portfolio results. Because its definition includes the EQUALIZE PORTFOLIO STATS option, all of these figures are based on the *average* of its two components, not their *sum*. We see that the Portfolio's training bias is nearly as great as that of LIN_POWER, probably reflecting the fact that this bias figure includes selection bias as well as the training bias of the two component models. Without the selection bias (if this were a fixed Portfolio), the training bias would have equaled the mean of the training biases of the two component models, or (37.4672 + 58.8356) / 2 = 48.1514. However, this model training bias is inflated further by the selection bias inherent in creating the optimal portfolio. This raises the Portfolio's training bias considerably.

After subtracting the training bias from the raw total return of the Portfolio, we see that the unbiased return is a little less than that of the two component models. This is probably just random variation. The important point is that the Portfolio results are in line with the components because the algorithm has compensated not only for the models' training bias, but the Portfolio's selection bias as well.

MODEL LIN4
   Long profit factor p = 0.0100 return p = 0.1000
      Training bias = 37.4672 (57.6183 permuted return minus 20.1511 permuted benchmark)
      Unbiased return = 59.6410 (97.1082 original return minus 37.4672 training bias = skill + trend)
      Benchmarked return = 43.3659 (59.6410 unbiased return minus 16.2751 original benchmark = skill)

MODEL LIN_POWER
   Long profit factor p = 0.1100 return p = 0.0100
      Training bias = 58.8356 (73.1141 permuted return minus 14.2786 permuted benchmark)
      Unbiased return = 57.8350 (116.6705 original return minus 58.8356 training bias = skill + trend)
      Benchmarked return = 42.2402 (57.8350 unbiased return minus 15.5947 original benchmark = skill)

PORTFOLIO PORT_POWER
   profit factor p = 0.0400   return p = 0.0500
      Training bias = 53.7958 (72.4237 permuted return minus 18.6279 permuted benchmark)
      Unbiased return = 53.0935 (106.8894 original return minus 53.7958 training bias = skill + trend)
      Benchmarked return = 37.1586 (53.0935 unbiased return minus 15.9349 original benchmark = skill)

Multiple-Market Considerations

When the TRAIN PERMUTED command is executed in a multiple-market situation, operation becomes slightly more complicated. The issue is that inter-market correlation must br preserved. A fundamental assumption of permutation tests is that all permutations must be equally likely to have occurred in real life. For example, it is well known that in times of large moves, markets tend to move together. When some world crisis arises, most markets go down. When good economic news is broadcast, most markets rise. If the markets were permuted separately, this coherence would be lost and the permutation test would be invalidated.

This implies that every market must have data available for every date. Otherwise it would be impossible to swap dates with data in some markets and not in others. In order to accomplish this, the first step in permutation training is to pass through the market history and eliminate all dates whose market set is incomplete. This results in a message similar to the following being printed in the audit log file:

```
NOTE... Pruning multiple markets for simultaneous
    permutation reduced the number of market-dates from
    503809 to 414427.
    The model(s) about to be printed and all permutation
    results reflect this reduced dataset and will differ
    from prior and subsequent results.
```

This message informs the user as to how many market-dates had to be removed from the history files. It also reminds the user that the results about to be shown reflect the market histories *after* this reduction, and hence they will generally be different from results of an ordinary TRAIN operation performed before or after the TRAIN PERMUTED operation.

If any dates had to be eliminated, then after the TRAIN PERMUTED operation is complete and all of its results are printed, the dates will be restored and as a convenience for the user the trading system will be trained with the complete dataset. A message similar to the following will be printed to make it clear to the user what is happening:

```
NOTE... Permutation is complete.  Prior to permutation we
    had to reduce  the number of market-dates so that
    every date had all markets.
    The original markets have now been restored.
    All models will now be re-trained with the original
    market data.  The results you are about to see reflect
    this original data, before pruning.
```

Trade Simulation and Portfolios

You may follow a TRAIN, WALK FORWARD, or CROSS VALIDATE command with the PRESERVE PREDICTIONS command, and follow this with a TRADE SIMULATOR command. This provides more extensive performance statistics than are provided by default, and it also allows you to execute slightly more sophisticated rules than those inherent in default operation. The syntax of this command is as follows:

```
TRADE SIMULATOR Model Detail  TradeRule  PerformanceMeasure
```

Model names a model that is in the script file. Trades for this model will be simulated.

The *Detail* may be GLOBAL or BY MARKET. This option, although always required, is relevant only if multiple markets are present. If you choose GLOBAL, only a summary with all markets pooled will be printed. If you choose BY MARKET, not only will the global summary be printed, but results for each individual market will also be printed. Note that this will produce a voluminous printout if a large number of markets are present. This option must be GLOBAL if an equity curve will be written or displayed.

The *TradeRule* must be one of the following:

```
LONG ( Enter Maintain )
LONG ( TRAINED Maintain )
SHORT ( Enter Maintain )
SHORT ( TRAINED Maintain )
DUAL ( LongEnter LongMaintain ShortEnter ShortMaintain )
DUAL ( TRAINED LongMaintain TRAINED ShortMaintain )
```

The rules that begin with LONG will take only long positions. The position will be opened when the model's prediction equals or exceeds the *Enter* value, and it will be closed when the prediction drops below the *Maintain* value. Obviously, the *Maintain* value must not exceed the *Enter* value. Negative values are permitted.

If instead of specifying a numeric *Enter* value, you specify the keyword TRAINED, the position will be opened when the predicted value equals or exceeds the optimal in-sample value determined during training. This is legal for cross validation and walkforward as well as ordinary training, because the program keeps track of the optimal value for each test fold.

If this TRAINED option is used, the *Maintain* specification is not an actual prediction threshold. Rather, it is a multiplier for the optimal open value. If it is specified as its legal maximum, 1.0, the maintain threshold will equal the open threshold. Specifying it as, say, 0.5 means that the

long position will be held as long as the prediction is at least half of the open threshold. Negative values are legal.

Note that most of the time, the optimal trained prediction threshold for a long position will be positive. In the unusual but occasional instance that the threshold is negative, the specified *Maintain* value will be ignored, and the maintain threshold will be set equal to the open threshold.

The rules that begin with SHORT will take only short positions. The parameters associated with these two commands are similar to those for LONG commands, with rules appropriately flipped.

The rules that begin with DUAL take both long and short positions. *Open* and *Maintain* thresholds must be separately specified for the long and short trades.

The *PerformanceMeasure* defines the statistic that will be used to measure performance. In all cases, we assume that Day 0 has closed and a decision has been rendered. The position is taken at the open of Day 1 and closed at the open of a subsequent day, at which point the return for the transaction is logged. This option may be the following:

PERCENT - This is the percent change in the market.

ATR(Dist) - This is the change in the market expressed as a multiple of ATR measured back in time over the specified distance, which must be at least 2. I recommend this if multiple markets are present, because it provides better cross-market conformity.

POINTS - This is the actual point change in the market.

Note that the TRADE SIMULATOR option obviously requires that market histories be present! If you are using a READ DATABASE command to read precomputed variables, you must *precede* the READ DATABASE command with READ MARKET LIST and READ MARKET HISTORIES commands.

Writing Equity Curves

After the trade simulator has been run, you may write the equity curve to a comma-delimited text file readable by Excel and most standard statistics packages. The units written are the units selected in the trade simulator: *Percent, ATR,* or *Points.*

To write the equity to a comma-delimited text file, use the following command:

WRITE EQUITY "FileName" ;

As with all file names in *TSSB*, the name must contain only letters, numbers, and the underscore character (_). The full path name may be specified. If no path is supplied, the file will be written to the current directory.

The fille will begin with the following header:

Date, Long, Short, Change, Cumulative

These quantities are defined as follows:

Date - The date as YYYYMMDD.

Long - The number of long positions open on the specified date (for multiple markets).

Short - The number of short positions open on the specified date (for multiple markets).

Change - The change in equity as of the open of the specified date relative to the open of the prior day's date.

Cumulative - The cumulative change in equity as of the open of the specified date.

If the data is intraday, the time as HHMM or HHMMSS will follow the date.

If there is only one market, it will obviously be impossible (except for extremely rare pathological situations involving crossed thresholds) to simultaneously have a long and short position. However, if several markets are present, some markets may be long on a given day while others are short.

It is vital to understand how the date, position, and change are related. The idea is that the market closes on what we will call *Day 0.* At this time we have all information needed to make a decision as to the position to take at the next trading opportunity (bar or day). This is the open of the next day (or bar for intraday data). So a position will be taken on *Day 1* and this position

will appear in the equity file for the date of *Day 1*. Equity will not change on that day, even though one might mark equity to the market throughout the day. At some point, after the close of the market on some day, the decision will be made to close the position at the next opportunity. This will be the open of the market on the day following the decision. Here is an actual example of an equity file:

```
Date, Long, Short, Change, Cumulative
20000104, 0, 1, 0.00000, 0.00000
20000105, 1, 0, 3.45964, 3.45964
20000106, 0, 1, 0.02766, 3.48730
20000107, 0, 1, -0.45047, 3.03683
```

As of the close of the day *prior* to 20000104 the system decided to take a short position. Thus, it takes this position at the open of 20000104. This short position is recorded in the file. The equity does not change because we need a full 24 hours to track the market change. Thus, the equity for 20000104 is still zero.

At the close of 20000104 the system decides it wants to turn long. At the open of 20000105 it closes the short position and opens a long position. This new position is recorded in the file. It made a 24-hour profit of 3.45964 as a result of closing the short position. The new long position does not enter into this figure at all yet.

At the close of 20000105 the system decides to go short again. At the open of 20000106 it closes the long position and opens a short position. This change is recorded for this date. The profit for the 24-hour long position is 0.02766, making a cumulative profit for the original short position and then the long position of 3.48730.

At the close of 20000106 it decides to continue holding the short position. This is reflected in the position field of 20000107. The short position that it held from the open of 20000106 to the open of 20000107 resulted in a loss of -0.45047, which drops the cumulative equity as of the open of 20000107 to 3.03683.

In summary, the position shown for each record is the position held on the specified date, including the overnight session that follows the specified date. The equity and cumulative equity for a specified date is the value as of the open of that date.

Performance Measures

For individual markets (printed if the BY MARKET option is specified), the first line for each market shows the total number of bars in the history, the number of those in which a long or short position is taken, and the percent of bars spent long or short. For the global summary, these numbers are combined across all markets. In addition, the global summary prints the unpooled but combined profit factor. In other words, this profit factor considers every trade in every market as a separate event rather than pooling them at the end of every time slice (bar).

The next line shows the total return across the entire historic time period, measured in whatever unit (PERCENT, ATR units, or points) was requested.

The next line shows the profit factor. For the global summary in a multiple-market situation, the trades are pooled across all markets at the end of each time slice (bar). This will generally produce a different profit factor than would be obtained by treating each market's trades as individuals. This latter quantity is printed on the first line, as mentioned above.

The last line shows the maximum drawdown over the historical time period, and the number of bars it covered. This drawdown is measured in the same units (PERCENT, ATR or points) as the other figures.

Note that a seeming anomaly can occur in some conditions. Intuitively, it would seem as if the long and short counts in a single DUAL run should equal the corresponding long and short counts in individual LONG and SHORT runs. This usually happens. However, suppose the following two conditions are true:
1) The enter threshold for a short position is positive. (This may be explicitly specified, or happen through internal automation if the TRAINED option is used.)
2) The long maintain threshold is less than the short enter threshold

Then the following sequence of actions may occur:
1) A prediction exceeds the long entry threshold, so a long position is taken.
2) The subsequent prediction exceeds the long maintain threshold but is less than the short enter threshold. Thus, under the LONG scenario, the long position would be maintained. But under the DUAL scenario, the short entry will take precedence over the long maintain status, resulting in a reversal of position. This is an unavoidable philosophical problem. Several work-arounds are possible, but my opinion is that they would cause more problems than they solve. I am willing to discuss the issue if you wish, but actually this is fairly rare, and happens only under conditions that could be considered pathological.

Portfolios (File-Based Version)

A portfolio is a set of trading systems that are combined into a single trading system that often has better net performance (better risk/reward ratio) than any individual trading system in the portfolio. *TSSB* contains two versions of portfolios: *File Type* and *Integrated*. They do almost the same thing (find and test optimal combinations of trading systems), but they do so in very different ways. The most important difference is that the *File Portfolio* reads equity files in order to obtain its component trading systems. As a result, *File Portfolios* can be used to process trading systems produced by other programs, such as *TradeStation*™. On the other hand, in order to build Portfolios based on *TSSB* models, the user must execute the *Trade Simulator* and write equity files. This is not only a nuisance, but it limits some capabilities. Therefore, *TSSB* also implements *Integrated Portfolios*. These are much more convenient as they can directly process trading results produced by *Models, Committees*, and *Oracles*. This section describes *File Portfolios*. *Integrated Portfolios*, which will be the Portfolio of choice for the vast majority of users, are presented in a separate chapter of their own (Page 201).

After the trade simulator has been run for two or more models, and corresponding equity files have been written, the user may be interested in computing net performance figures for a combination of equity curves. This can be done with the FILE PORTFOLIO command, which merges two or more files of equity curves to produce a single combined equity curve. The resulting curve can then be written to a file exactly like the curve produced by a TRADE SIMULATOR command.

Note that the FILE PORTFOLIO command does not need to be in the same script file as the TRADE SIMULATOR commands that produced the equity curves. This is because all necessary information is contained in the equity curve files that the PORTFOLIO command reads. The user can create a short, simple script file that contains nothing but one or more PORTFOLIO commands, thus separating the model training/testing operations from the study of possible portfolios. In fact, the equity curve files read by this command need not have been created from *TSSB*, as long as they are in the file format described in the section on writing equity curves, Page 193.

The syntax for the FILE PORTFOLIO command is as follows:

```
FILE PORTFOLIO  PortfolioName  [  PortfolioSpecs  ]  ;
```

The *PortfolioName* may be up to 15 characters long, and it may not contain spaces or any special characters other than the underscore (_). At this time the name has no use, but it has been included as a mandatory part of the syntax due to the fact that tentative enhancements to the program will require that portfolios be named.

There is only one *PortfolioSpec* that is required:

```
EQUITY FILE = [ File1  File2  ... ] ;
```

This specification names two or more equity files that will go into the portfolio. These files must be in the file format described in the section on writing equity curves, Page 193. Each file name must be enclosed in double quotes ("...") and may not contain spaces or special characters other than the underscore (_).

Here is an example of a simple PORTFOLIO command:

```
PORTFOLIO DEMO_PORT [
   EQUITY FILE = [ "EQTY1.TXT" "EQTY2.TXT" "EQTY3.TXT" ]
   ] ;
```

The example command just shown defines a portfolio called *DEMO_PORT* that merges three equity curve files.

There are several optional specifications available. These are:

EQUALIZE - By default, all returns, drawdowns, et cetera, are computed by summing the equity curves. Thus, for example, the net return of a portfolio is by default the sum of the returns of all of its components. Naturally, this inflates returns and drawdowns relative to those of individual components, making comparisons difficult. By specifying the *EQUALIZE* option, the portfolio performance figures are based on the mean of the components instead of their sum, which can make visual inspection of performance tables easier. Of course, this is only a scaling issue. Returns and drawdowns are scaled equally if this option is employed. Performance figures like the profit factor and Sharpe ratio are not affected at all by this option.

OPTIMIZE = NumberInPort Trials - By default, all of the named equity files are included in the portfolio. As an alternative, the *OPTIMIZE* command automatically selects an optimal subset of the named files. Currently, optimization is performed by finding the subset that maximizes the Sharpe ratio. Alternative optimization criteria may be included in a future release. The user specifies the number of files to be included in the portfolio, as well as the number of random trials that will be tested in order to find the best. This should be very large if the number of equity files is large. If n is the number of equity files and m is the portfolio size, the number of combinations is $n! / ((n-m)!\ m!)$, which blows up quickly as n increases. Ideally, *Trials* should be at least somewhat larger than this quantity, although this may not always be possible. However, it is not usually a serious problem if the number of trials is small relative to the number of possible combinations. This is because although the optimal portfolio found will probably not

be the true optimum, it will likely be close to the best. A future release of the program may include an advanced genetic algorithm for optimization.

WALK FORWARD FROM Year - The *OPTIMIZE* command produces a large selection bias. In other words, random luck plays a significant role in selecting the best portfolio over a given time period. It is likely that in a different time period this optimal portfolio will not perform as well as it did in the time period in which it was selected. The *WALK FORWARD FROM Year* option lets the user evaluate the degree to which this effect occurs. The time period prior to the specified year is used to find the optimal portfolio, and then this portfolio's performance is evaluated in the specified year. Next, the specified year is appended to the time period used for finding the optimal portfolio, and optimization is performed again. This new optimal portfolio is tested on the following year. This operation of walking forward one year at a time is repeated until the end of the supplied historical data. This command may be used only when the *OPTIMIZE* command is also used.

Here is an example of a PORTFOLIO command that finds an optimal portfolio and walks it forward to evaluate the degree to which optimality holds up. Also, the *EQUALIZE* option is used to make visual interpretation of result easier.

```
PORTFOLIO DEMO_PORT [
   EQUITY FILE = [ "EQTY1.TXT" "EQTY2.TXT" "EQTY3.TXT" ]
   EQUALIZE
   OPTIMIZE = 2 100
   WALK FORWARD FROM 2005
   ] ;
```

Numerous statistics are printed in the AUDIT.LOG file. These quantities are based on the daily values present in the equity file. See Page 192 for a discussion of the options available in creating an equity curve.

Minimum - The minimum daily profit, negative for a loss.

Maximum - The maximum daily profit.

Mean - The mean daily profit.

StdDev - The standard deviation of daily profits.

Rng/Std - The range of daily profits (maximum minus minimum) divided by their standard deviation.

Sharpe - The Sharpe ratio of the equity curve. This is the mean daily profit, divided by the standard deviation, and multiplied by the square root of 252, which approximately annualizes the value.

PF - The profit factor. This is the sum of positive profits divided by the sum of negative profits (i.e. losses).

Drawdown - The maximum net loss (peak cumulative equity dropping to subsequent trough).

Recovery - The number of days required for the worst drawdown to recover to the peak from which the drawdown began.

These statistics will be printed for each individual equity file, as well as for the portfolio obtained by combining all of the equity files. If the *OPTIMIZE* option is used, these statistics will also be printed for the optimal portfolio. The files selected for the optimal portfolio, as well as those not selected, will be listed. These are identified by the numeric order in which they were specified in the *PORTFOLIO* command. The first file named in the *EQUITY FILE* list is number 1, the second is number 2, and so forth.

If the *WALK FORWARD FROM Year* option appears, results for each out-of-sample year will be printed. This includes three lines:

Trn - The best portfolio's in-sample (portfolio selection) performance. This is the performance of the optimal portfolio in the time period in which it was selected, the data prior to the current walkforward year. This quantity will almost certainly be optimistically biased.

OOS - The performance of this portfolio for the current walkforward year, the year immediately following the training period. This out-of-sample quantity is an unbiased estimate of the true performance of the optimal portfolio. On average this will be less than that in the *Trn* training period, although natural market variation will often cause this to exceed the training performance.

All - The performance of trading all systems in the current walkforward year. Comparing this to the *OOS* performance shows the difference between simply trading all equity files versus trading just the previously selected optimal portfolio.

After walkforward is complete, all out-of-sample data is pooled into a single set of trades, and two more lines of performance statistics are printed. These are *OPT*, which is the performance of the optimal portfolio (unbiased, since the portfolio was selected before this performance is computed), and *All*, which is the net performance of all equity files. Comparing these two items lets the user evaluate the effect of finding an optimal portfolio versus just trading all systems.

Important note on WALK FORWARD folds: The folds in the FILE PORTFOLIO command are based on the dates in the equity file, which are the dates of actual change in equity. However, the folds in the ordinary WALK FORWARD command are based on the date on which the trade decision is made, which most users would consider more sensible. Thus, the folds do not exactly correspond. In general they are shifted with respect to each other by two days, and hence results may not exactly agree with each other. This is not incorrect, it is just an unavoidable difference of fold definition.

Integrated Portfolios

The prior section discussed file-based Portfolios. Those are useful in situations in which the developer wishes to import equity curves from other programs such as TradeStation™ and make use of *TSSB*'s Portfolio building and testing algorithms. However, in most situations the developer wants to build Portfolios from models, committees, and oracles already implemented in a *TSSB* script. In such cases, using the *Trade Simulator* to produce equity files, and then reading them with the FILE PORTFOLIO command, is a nuisance. For this reason, *TSSB* also includes a Portfolio building and testing algorithm fully integrated into the train/test cycle. This *Integrated Portfolio* is the subject of this chapter.

There are two subtle differences between the *File Portfolio* and the *Integrated Portfolio* that should be mentioned. First, the *File Portfolio* deletes from the beginning and end of the equity files all records that have no equity change. This reduces the number of cases included in the equity history. If a user employs the *Trade Simulator* to produce equity files, processes these files with the *File Portfolio*, and compares the results to those obtained strictly internally with the *Integrated Portfolio*, means and standard deviations may be slightly different. This is because the different number of cases results in dividing sums by different numbers of bars. This is of no practical consequence.

Second, the *File Portfolio* bases walkforward folds on the dates in the equity files. The *Trade Simulator* assigns the dates according to when equity changes are recorded. But the *Integrated Portfolio* bases walkforward folds on the dates on which trade decisions are made, which is a more sensible and practical approach. Since trade decisions obviously precede resulting changes in equity, walkforward folds will be slightly misaligned between the two versions, producing slightly different fold performance statistics.

A Portfolio is a collection of two or more Models, Committees, or Oracles. (Actually, it is legal for a Portfolio to contain only one component, but that would be pointless.) A Portfolio may contain a mixture of Models, Committees, and Oracles. The Portfolio must not be defined until all of its components have been defined. Portfolios are ignored for CROSS VALIDATE commands. They are evaluated only during TRAIN and WALK FORWARD commands.

An *Integrated Portfolio* is defined by means of the following command:

PORTFOLIO PortName [Specifications] ;

The following specifications are available:

MODELS = [Model_1 Model_2 Model_3 ...]

> *This mandatory specification lists the component Models (or Committees or Oracles) of the portfolio. Depending on other specifications, all of these Models may make up the Portfolio, or only a subset of the list.*

LONG ONLY
SHORT ONLY

> *By default, the Portfolio includes all trades, long and short, of the component models. Including one of these optional specifications causes the Portfolio to consider only long or only short trades of the components.*

TYPE = FIXED

> *The TYPE of the Portfolio must be specified. The FIXED Portfolio uses all of the components listed in the MODELS specification.*

TYPE = OPTIMIZE Nused IS

> *The TYPE of the Portfolio must be specified. The IS (In-Sample) type uses 'Nused' components of those listed in the MODELS specification. The subset is chosen by maximizing the Sharpe Ratio of the Portfolio. The NREPS specification, described below, must appear if the IS type of Portfolio is defined. When a TRAIN command is executed, the optimal subset of components is found by examining the entire dataset, just as is done to train the components. When a WALK FORWARD command is executed, the optimal subset is found by examining the in-sample fold data, the same data that is used to train the components. The performance of the Portfolio will be based on the out-of-sample data in each fold, the same data that is used to assess the components.*

TYPE = OPTIMIZE Nused OOS Nfolds FOLDS

> *The TYPE of the Portfolio must be specified. The OOS (Out-Of-Sample) type uses 'Nused' components of those listed in the MODELS specification. The subset is chosen by maximizing the Sharpe Ratio of the Portfolio. The NREPS specification, described below, must appear if the OOS type of Portfolio is defined. This type of Portfolio is evaluated only when a WALK FORWARD command is executed. Double folds are implemented: the components are trained on the training fold and evaluated on the out-of-sample fold. Then the Portfolio's optimal subset is chosen by examining the 'Nfolds' most recent folds of out-of-sample data, and the Portfolio's performance is evaluated based on the data in the next out-of-sample fold. A detailed example appears later in this chapter.*

EQUALIZE PORTFOLIO STATS

This option has no effect on the selection of the optimal subset of components, or on any other aspect of computation. It affects only printing of performance results. If this option does not appear in the Portfolio definition, printed results are based on the sum of the components' trades. If this option is used, printed results are based on the mean of the components' profits and losses.

OVERLAP = Integer

This option has the same use as in training the components. If the target has a lookahead greater than one bar, it is best (though not critical) to set the OVERLAP to one less than the target's lookahead.

NREPS = Integer

This option is required if the IS or OOS type is employed. It specifies the number of randomly selected subsets to try in order to find the subset that maximizes the Portfolio optimization criterion, currently the Sharpe Ratio.

A FIXED Portfolio Example

One useful application of a FIXED Portfolio is to facilitate training separate Models (or entire trading systems) to handle long and short trades independently. It is well known that it is difficult to find a single trading system (a Model or set of Models combined with a Committee or Oracle) that works well for both long and short trades. Specialization in one or the other almost always provides the best performance. By making use of a Portfolio we can train long and short systems separately, and then obtain net performance estimates. Consider the following simple example:

```
MODEL LIN_LONG IS LINREG [
   INPUT = [ CMMA_5 CMMA_20 LIN_5 LIN_20 RSI_5 RSI_20
             STO_5 STO_20 REACT_5 REACT_20 PVR_5 PVR_20 ]
   OUTPUT = DAY_RETURN
   MAX STEPWISE = 2
   LONG ONLY
   CRITERION = LONG PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   ] ;

MODEL LIN_SHORT IS LINREG [
   INPUT = [ CMMA_5 CMMA_20 LIN_5 LIN_20 RSI_5 RSI_20
             STO_5 STO_20 REACT_5 REACT_20 PVR_5 PVR_20 ]
   OUTPUT = DAY_RETURN
   MAX STEPWISE = 2
   SHORT ONLY
   CRITERION = SHORT PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   ] ;

PORTFOLIO Port [
   MODELS = [ LIN_LONG LIN_SHORT ]
   TYPE = FIXED
   ] ;
```

This example employs two models that are almost identical. Their only difference is that the first chooses its predictors to maximize the long profit factor, and it executes only long trades. The second maximizes the short profit factor and executes only short trades. The simple Portfolio combines these two sets of trades and prints net performance results.

An OOS Portfolio Example

The prior example showed how a fixed Portfolio could be used to pool long and short specialists in order to obtain net results from trading both sides of the market. Another use of Portfolios is to examine the performance of several (perhaps a great many) Models or complete trading systems that also contain Committees and Oracles. The Portfolio will then automatically select an optimal subset of the candidates.

At this time the only optimization criterion available is the Sharpe Ratio. The popular wisdom is that the Sharpe Ratio is a poor measure of financial performance, the leading argument being that it penalizes large upward moves in equity as much as it penalizes large downward moves. This is not the forum to examine the pros and cons of the Sharpe ratio, but for now please understand that this argument contains significant technical flaws. In fact, the Sharpe Ratio is an excellent way of comparing portfolios, primarily because it is sensitive to the desirable effect of losses in one component being offset by gains in another component. A laudable goal of finding a good portfolio is making its equity curve as smooth as possible. The Sharpe Ratio does an admirable job of achieving this goal.

Another minor weakness of the current Portfolio optimization algorithm is that it simply tries a large number (specified with the NREPS parameter) of randomly selected subsets and chooses the best. A future release of *TSSB* may employ genetic optimization, although unless the number of candidates is huge, the difference in execution speed between naive search and intelligent search is insignificant.

It should be obvious that choosing the best subset from among competing candidates introduces a large optimistic bias. The important question is not how well the portfolio performs in the time period in which it was selected. Rather, we need to know whether a portfolio's performance holds up in the future. This inspires us to use walkforward testing in evaluating a Portfolio.

The choice to use walkforward testing immediately leads to another question: what data do we use for finding the optimal subset? We could use the same time period as was used to train the candidate models. This is often reasonable, and it is the method employed with the TYPE=...IS option. However, there is a significant risk with this method. Suppose some component is based on an excessively powerful model, meaning that the model overfits the data and hence has superb in-sample performance. This component will likely be selected for the optimal portfolio, even though its out-of-sample performance may be dismal due to the overfitting. Thus, we are inspired to select the optimal portfolio based on the components' out-of-sample performance, which is much more honest than in-sample performance. This calls for double walkforward, in which multiple out-of-sample folds are needed. The example script file shown on the next page, and the explanation that follows, make this operation clear.

```
MODEL LIN_1 IS LINREG [
   INPUT = [ CMMA_5 CMMA_20 LIN_5 LIN_20 RSI_5 RSI_20
     STO_5 STO_20 REACT_5 REACT_20 PVR_5 PVR_20 ]
   OUTPUT = DAY_RETURN
   MAX STEPWISE = 2
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   EXCLUSION GROUP = 1
   ] ;

MODEL LIN_2 IS LINREG [
   INPUT = [ CMMA_5 CMMA_20 LIN_5 LIN_20 RSI_5 RSI_20
     STO_5 STO_20 REACT_5 REACT_20 PVR_5 PVR_20 ]
   OUTPUT = DAY_RETURN
   MAX STEPWISE = 2
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   EXCLUSION GROUP = 1
   ] ;

MODEL LIN_3 IS LINREG [
   INPUT = [ CMMA_5 CMMA_20 LIN_5 LIN_20 RSI_5 RSI_20
     STO_5 STO_20 REACT_5 REACT_20 PVR_5 PVR_20 ]
   OUTPUT = DAY_RETURN
   MAX STEPWISE = 2
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   EXCLUSION GROUP = 1
   ] ;

MODEL LIN_4 IS LINREG [
   INPUT = [ CMMA_5 CMMA_20 LIN_5 LIN_20 RSI_5 RSI_20
     STO_5 STO_20 REACT_5 REACT_20 PVR_5 PVR_20 ]
   OUTPUT = DAY_RETURN
   MAX STEPWISE = 2
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   EXCLUSION GROUP = 1
   ] ;
```

```
PORTFOLIO OptPort [
   MODELS = [ LIN_1 LIN_2 LIN_3 LIN_4 ]
   TYPE = OPTIMIZE 2 OOS 3 FOLDS
   NREPS = 1000
   ] ;

WALK FORWARD BY YEAR 5 2006 ;
```

This example defines four candidate Models using the EXCLUSION GROUP option. This is an excellent method for creating Committee components, but it is not necessarily the best method for building a Portfolio. Most users would want to be more explicit in defining the components. But it's quick and easy, so this is the method used for this example.

The TYPE option in this Portfolio specifies that two of the four candidates will be chosen for the Portfolio, and this selection will be based on three out-of-sample training folds. Let us look in detail at the sequence of operations that will be performed.

Train the Models using 2001-2005
Test the Models using 2006
Train the Models using 2002-2006
Test the Models using 2007
Train the Models using 2003-2007
Test the Models using 2008
Train the Models using 2004-2008
Test the Models using 2009
Train the Portfolio using 2006-2008 Model OOS results
Test the Portfolio using 2009 Model OOS results
Train the Models using 2005-2009
Test the Models using 2010
Train the Portfolio using2007-2009 Model OOS results
Test the Portfolio using 2010 Model OOS results
Et cetera

The Portfolio's performance will be reported in two ways in the AUDIT.LOG file. It will be reported for each fold in which the Portfolio is evaluated, and then reported one last time in the walkforward summary section. The following statistics are printed:

Minimum - The minimum daily profit, negative for a loss.

Maximum - The maximum daily profit.

Mean - The mean daily (bar) profit across all days.

StdDev - The standard deviation of daily (bar) profits.

Rng / Std - The range of daily profits (maximum minus minimum) divided by their standard deviation.

Sharpe - The Sharpe ratio of the equity curve. This is the mean daily profit, divided by the standard deviation, and multiplied by the square root of 252, which approximately annualizes the value.

PF - The profit factor. This is the sum of positive profits divided by the sum of negative profits (i.e. losses).

Drawdown - The maximum net loss (peak cumulative equity dropping to subsequent trough).

Recovery - The number of days required for the worst drawdown to recover to the peak from which the drawdown began.

Note that when these results are reported for the complete Portfolio, by default they will be based on the sum of component results. If the EQUALIZE PORTFOLIO STATS option is used in the Portfolio definition, these results reflect the mean of the components rather than their sum.

Here is an example of fold results for the example just show:

Portfolio OPTPORT statistics for 755 training dates
  (Mean and StdDev are across all dates, not just all trades.)

System	Minimum	Maximum	Mean	StdDev	Rng/Std	Sharpe	PF	Drawdown	Recovery
LIN_1	-3.423	3.588	0.0404	0.407	17.234	1.575	1.861	6.035	225
LIN_2	-3.423	3.588	0.0298	0.499	14.048	0.947	1.423	8.389	245
LIN_3	-3.423	3.588	0.0032	0.483	14.525	0.104	1.033	14.031	Never
LIN_4	-3.423	3.588	0.0364	0.479	14.644	1.208	1.575	6.959	145
All pooled	-13.694	14.352	0.1097	1.591	17.630	1.095	1.396	29.642	234
Optimal subset	-6.847	7.176	0.0768	0.802	17.488	1.521	1.697	11.447	213
Optimal in OOS	-1.805	1.725	-0.0079	0.354	9.968	-0.356	0.904	5.302	Never

Optimization kept the following candidates:
  LIN_1
  LIN_4

Optimization rejected the following candidates:
  LIN_2
  LIN_3

The table above first shows the individual performance of the four candidates in the Portfolio's training set (the recent out-of-sample Model data). Then it pools all candidates. Next it shows the performance for the optimal subset. Finally, it shows the performance of this optimal subset in the Portfolio's out-of-sample period.

The walkforward summary provides the same information, but in a different format:

Portfolio OPTPORT pooled OOS performance with 746 dates...
  Min = -4.02348 Max = 5.94531
  Mean = 0.00652 StdDev = 0.62720
  Profit factor = 1.05098 Annualized Sharpe ratio= 0.16498
  Max drawdown = 12.80320 320 days to recovery

Index