Principal Component Transforms

The principal components of a set of variables are an equal number of new variables that are linear combinations of the original variables and that have a special property: the first principal component is the linear combination that has the maximum variance of all possible linear combinations. Subsequent principal components also have maximum variance, but subject to the restriction that they be uncorrelated with all prior principal components. In other words, the principal components of a set of indicators have two nice properties that the original indicators almost never have:

1) They are uncorrelated with one another.

2) Any subset consisting of the first principal component, the second, the third, et cetera, carries the maximum variation inherent in the original indicators that is possible to carry in a subset of that size. Roughly speaking, they carry the maximum variation in the minimum number of new indicators. This technique is an excellent way to reduce dimensionality and redundancy, two important goals in predictive modeling

As a simple example, suppose we have two indicators: trend over the most recent 10 bars, and trend over the most recent 20 bars. Obviously, these two indicators will be highly correlated. Now suppose we compute the principal components of these two indicators. Almost certainly, the first principal component will be something akin to the mean of the two indicators, a more or less generic measure of trend. The second principal component will be the suitably weighted difference between the two trend indicators, a measure of how much or little the trend has changed in the most recent 10 bars relative to what it has been over the last 20 bars.

That example is poor in that using just two indicators does not do justice to principal components. It would probably be as good to just supply the two indicators to the model, rather than bothering with principal components. But their real value comes to light when we have many members of a family. For example, we might measure volatility at four different lookbacks and three different types of volatility at each lookback. This gives us 12 volatility indicators. We may find that just the first three or four principal components capture the large majority of the variation observed in the original 12 indicators. Thus we end up with not only fewer indicators (which reduces the chance of overfitting), but as a nice bonus they are also uncorrelated!

It should be mentioned that capturing the majority of *variation* in the indicators is not necessarily equivalent to capturing the majority of the *predictive information* that they contain. It may be that the most useful information is contained in a small-

variance component that we reject. So by computing principal components and keeping the highest-variance subset, we might end up discarding vital information. On the other hand, much practical experience indicates that in most applications, the largest principal components do carry the most predictive information, while the smallest tend to be mostly or entirely noise.

Once the principal components have been computed and a maximum-variance subset of them has been retained, the user can optionally request Varimax rotation of the reduced space. This rotation scheme preserves exactly the same information as is contained in the subset before rotation. Hence, if the indicators will *all* be fed to a *linear model*, the choice of whether to rotate 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, rotation can have an impact on performance. This is because the ordering of most-to-least variation is sacrificed in order to often obtain composite indicators that are easier to interpret than principal components. These rotated components tend to have correlations with the original indicators that are either near +/- one, or near zero. In other words, rotation tends to produce new indicators that represent subsets of the original indicators. This may or may not be a good thing as far as model performance is concerned.

Invoking the Principal Components Transform

This transform is invoked with the following command:

```
Transform TransformName IS PRINCIPAL COMPONENTS [Specs] ;
```

The transform name may consist of letters, numerals, and the underscore character (_) and may be at most 15 characters. The following mandatory and optional specifications may be included:

INPUT = [VarName1 VarName2 ...] - This mandatory specification names two or more indicators that are the source for the transformation.

N KEPT = Integer
N KEPT = ALL - One or the other of these specifications is mandatory. This specifies how many principal components are kept. If more than 8 are specified, coefficients for only the first 8 are printed.

ROTATION = VARIMAX - This option decrees that Varimax rotation will be applied to the principal components in order to drive weights toward zero or +/-1. If this option is not specified, ordinary principal

components will be output.

The *N KEPT* principal components will be entered into the database. They will be given the name of the transform with the numerals 1, 2, 3, et cetera appended.

So, for example, we might declare a principal components transform as shown below. This declaration uses the indicators X1 through X10 as inputs, and it keeps the first (largest) three principal components. These will be named DemoTran1, DemoTran2, and DemoTran3. These new variables behave just like any other variables in the database. In particular, they can be used as model inputs and plotted with the various menu-accessed plotting functions.

```
TRANSFORM DemoTran IS PRINCIPAL COMPONENTS [
   INPUT = [ X1 - X10 ]
   N KEPT = 3
   ] ;
```

Tables Printed

The principal components transform will cause several sets of numbers to be printed in the audit log file. A specific example will appear in the next section. But first, here is a description of what will appear.

The first table printed is called the *factor loadings*. This is the correlation between each principal component and each original indicator. Thus, all entries in this table will be in the range -1 to 1. There is a row for each original indicator and a column for each principal component. This table is headed with two additional rows. The top row is the percent of the total variance which is accounted for by that principal component, and the second row is the cumulative percentage for that column and all prior columns.

At the far right of each row is a quantity called the *communality*. This is the percent of the corresponding indicator's variance that is accounted for by the principal components that are kept. If this quantity is near 100 percent, then the principal components kept capture almost all of the variation in this indicator. Conversely, if this quantity is small, then the retained principal components have kept almost none of the variation in this indicator. This missing variation resides in principal components that were discarded. (If you keep all principal components, the communalities will all be 100 percent.)

If the user specified the ROTATION=VARIMAX option in the principal components declaration, a second table of factor loadings (correlations with the original indicators) is printed. This is the loadings for the rotated factors.

The final table printed is called *Score Coefficients*. Again, there is a row for each original indicator and a column for each principal component kept. This table is of little interest to most people, but it is printed for completeness. These are the coefficients for the linear combination of standardized indicators which would be used to compute the (optionally rotated) corresponding principal component. In other words, if we were to standardize the original indicators by subtracting their means and dividing by their standard deviations, we could then multiply these standardized values by the printed score coefficients and sum them to get the principal component, rotated if so specified.

An Example

Nine volatility indicators were chosen for this example. PVARRAT_5, 10, and 20 are the PRICE VARIANCE RATIO at lookbacks of 5, 10, and 20 days. The CVARRAT indicators are the CHANGE VARIANCE RATIO. The ATRRAT indicators are the ATR RATIO. These are all described in the *Variables* chapter. Here is the principal components declaration, as well as the declaration of a model that uses the newly created variables, along with a single trend indicator:

```
TRANSFORM PCO_ROTATE IS PRINCIPAL COMPONENTS [
   INPUT = [ PVARRAT_5 PVARRAT_10 PVARRAT_20
             CVARRAT_5 CVARRAT_10 CVARRAT_20
             ATRRAT_5  ATRRAT_10  ATRRAT_20 ]
   N KEPT = 3
   ROTATION = VARIMAX
   ] ;

MODEL ROTATE IS LINREG [
   INPUT = [ LIN_ATR_7 PCO_ROTATE1 PCO_ROTATE2 PCO_ROTATE3
]
   OUTPUT = DAY_RETURN_1
   MAX STEPWISE = 2
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   ] ;
```

Observe that in addition to the trend indicator LIN_ATR_7, the model specifies as inputs the three principal components produced by the transform. These are the name of the transform, PCO_ROTATE, with the numerals 1, 2, and 3 appended.

The transform produces the following table of factor loadings (before rotation):

```
Factor loadings...

                      1       2       3     Comm

Percent            35.9    20.8    12.7
Cumulative         35.9    56.7    69.4

      PVARRAT_5   0.322   0.611  -0.437    66.9
     PVARRAT_10   0.452   0.080   0.297    29.9
     PVARRAT_20   0.248  -0.464   0.740    82.4
      CVARRAT_5   0.509   0.597   0.156    64.0
     CVARRAT_10   0.834   0.077  -0.066    70.6
     CVARRAT_20   0.676  -0.535  -0.278    82.1
       ATRRAT_5   0.604   0.539   0.286    73.6
      ATRRAT_10   0.835  -0.113   0.043    71.2
      ATRRAT_20   0.625  -0.568  -0.354    83.8
```

Notice that even though there are nine volatility indicators, the first principal component alone accounts for over one-third of the total variance of these nine indicators. Also note that this first principal component has positive correlation, generally large, with all nine indicators. Thus, we can consider the first principal component to be a broad measure of overall volatility.

The second principal component accounts for another 20.8 percent of the total variance, meaning that these two alone account for 56.7 percent of the total variance. This principal component has high positive correlation with lookbacks of 5 days, almost no correlation with lookbacks of 10 days, and high negative correlation with lookbacks of 20 days. In other words, this second principal component measures the contrast between 5-day and 20-day volatility lookbacks, the degree to which the total volatility is produced by the short-term versus the long-term indicator. It is fascinating to note that just these two new indicators alone account for well over half of the total variability of the nine indicators we began with.

The third principal component is more difficult to interpret. We'll leave that to the reader! Also, because it includes only 12.7 percent of the total variance, it is debatable whether it's even worth bothering with.

It's somewhat interesting that these three principal components capture the majority of the variance in all of the original indicators except for one: PVARRAT_10. The communality of this indicator is just 29.9, barely over one-quarter. This means that the three principal components that we kept are missing the majority of the variation in this indicator. This may or may not be consequential.

Because the declaration specified the ROTATION=VARIMAX option, the rotate loadings are also printed, This table is as follows:

	1	2	3	Comm
Percent	27.4	26.7	15.3	
Cumulative	27.4	54.1	69.4	
PVARRAT_5	0.062	0.417	-0.701	66.9
PVARRAT_10	0.163	0.479	0.207	29.9
PVARRAT_20	0.146	0.186	0.876	82.4
CVARRAT_5	-0.032	0.774	-0.200	64.0
CVARRAT_10	0.585	0.597	-0.085	70.6
CVARRAT_20	0.902	0.034	0.082	82.1
ATRRAT_5	0.018	0.856	-0.058	73.6
ATRRAT_10	0.650	0.526	0.112	71.2
ATRRAT_20	0.913	-0.051	0.037	83.8

It is crucial to understand the difference that rotation makes. Recall that the whole point of rotation is to keep exactly the same information as in the unrotated principal components, but drive the coefficients toward -1, 0, or +1 as much as possible so as to emphasize and de-emphasize the importance of individual indicators. This may or may not be a good thing, depending on the application and the goals of the developer.

With this in mind, it is not surprising to see that the percent of variance captured by the three principal components is the same, 69.4, before and after rotation. Also, the communalities remain the same. This makes sense, because no information has been gained or lost by rotation. But the distribution of variance has changed. The three components are now more equal in their contributions. Also, we no longer have the clear labeling of the first component as overall volatility and the second as the contrast between short-term and longer-term lookbacks. Instead, we see that the first rotated principal component is primarily made up of CVRRAT_20 and ATRRAT_20, with secondary contributions from CVRRAT_10 and ATRRAT_10 The other five indicators are practicallyignored. The second rotated component primarily picks up the 5-day lookbacks. The third rotated component features a contrast between a 5-day and a 20-day lookback, but only for PVARRAT. The other seven indicators make little or no contribution. So rotation has done exactly what it is supposed to do: give up the clear ranking of variance from greatest to least, and replace it with maximum conceptual clarity between indicators. Neither is universally superior to the other, and both convey the same information. The choice of which to use is often a shot in the dark, although engineers do tend to prefer avoiding rotation.

Linear and Quadratic Regression Transforms

The principal components transform described in the prior section is a common and effective means of reducing the number of indicators by distilling their information down into a much smaller set of derived indicators. However, it suffers from one worrisome drawback: there is no guarantee that the largest-variance principal components, the ones kept, carry the most useful predictive information. It may be that the information we would most like to have lies in one or more of the smaller-variance components that are discarded. The good news is that in practice, this unfortunate situation seems to be the exception rather than the rule. But the bad news is that it can and does happen.

There is a simple way to get around this potential weakness: train a multiple-input model to predict a target that is equal or similar to our ultimate target, and use the predictions of this model as our 'distilled' indicator. The problem with this approach is that we get only one indicator out of the transform, whereas with principal components we can get multiple indicators. But there is a bright side: the one indicator that we do get is guaranteed in some reasonable sense to optimally capture the predictive information in the multiple input indicators that will be most useful to us. This can be a lot more reassuring than using principal components and crossing our fingers, hoping that the information we want does not lie in the components we discarded.

TSSB provides two methods for accomplishing this optimally predictive transform. One is simple linear regression, and the other is quadratic regression, the principle of which is described here. These transforms are invoked with the following commands, respectively:

```
Transform TransformName IS LINEAR REGRESSION [Specs] ;
Transform TransformName IS QUADRATIC REGRESSION [Specs] ;
```

The transform name may consist of letters, numerals, and the underscore character (_) and may be at most 15 characters. The following two specifications are required:

INPUT = [VarName1 VarName2 ...] - Names two or more indicators that are the source for the transform.

OUTPUT = TargetVar - Names an indicator that will be the target for the transform. The generated variable will be an optimal fit to this target.

All input variables are used, and for the quadratic model, all possible terms up to second order are used. No stepwise selection or internal cross validation is

performed. This is because the point is to create an optimal mapping, not to engender optimal out-of-sample predictions. That is the task of the ultimate prediction model into which the transformed indicators will be fed. Thus, we want to capture all available information.

A Regression Transform Example

Here is an example of using regression transforms. We use two linear regressions and one quadratic regression. The two linear regressions use different indicator sets, and the quadratic regression uses a different target. It is legitimate to use the same target for a regression transform as for the ultimate model, but it is often better to use a related but not identical target so as to provide variety. Finally, the targets look ahead by 10 days, so all declarations include the OVERLAP=9 option See here for details on this option.

```
TRANSFORM LIN1 IS LINEAR REGRESSION [
   INPUT = [ CMMA_5 CMMA_10 CMMA_20 LIN_ATR_7 LIN_ATR_15 ]
   TARGETVAR = FUTSLP_10
   OVERLAP = 9
   ] ;

TRANSFORM LIN2 IS LINEAR REGRESSION [
   INPUT = [ PENT_1 - PMUTINF_3 ]
   TARGETVAR = FUTSLP_10
   OVERLAP = 9
   ] ;

TRANSFORM QUAD IS QUADRATIC REGRESSION [
   INPUT = [ CMMA_5 CMMA_10 CMMA_20 LIN_ATR_7 LIN_ATR_15 ]
   TARGETVAR = RSQFUTSLP_10
   OVERLAP = 9
   ] ;

MODEL REGDEMO IS LINREG [
   INPUT = [ LIN1 LIN2 QUAD ]
   OUTPUT = DAY_RETURN_10
   MAX STEPWISE = 2
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   OVERLAP = 9
   ] ;
```

All regression coefficients are printed in the audit log file. Here is the output for these three transforms:

```
Training LINEAR REGRESSION TRANSFORM LIN1
LIN1 read 11187 cases for training

Transform LIN1 regression coefficients:
         -0.061653  CMMA_5
          0.116553  CMMA_10
         -0.185353  CMMA_20
          0.024601  LIN_ATR_7
          0.112782  LIN_ATR_15
          3.027141  CONSTANT

Training LINEAR REGRESSION TRANSFORM LIN2
LIN2 read 11187 cases for training


Transform LIN2 regression coefficients:
          0.014179  PENT_1
         -0.016498  PENT_2
          0.006523  PENT_3
          0.046827  PENT_4
         -0.021096  PMUTINF_1
         -0.063403  PMUTINF_2
          0.029090  PMUTINF_3
          2.773316  CONSTANT

Training LINEAR REGRESSION TRANSFORM QUAD
QUAD read 11187 cases for training

Regression coefficients:
        -0.056716  CMMA_5
         0.053772  CMMA_10
        -0.091056  CMMA_20
         0.019502  LIN_ATR_7
         0.056980  LIN_ATR_15
         0.007442  CMMA_5 Squared
         0.009810  CMMA_10 Squared
         0.008363  CMMA_20 Squared
         0.005146  LIN_ATR_7 Squared
         0.010850  LIN_ATR_15 Squared
        -0.008801  CMMA_5 * CMMA_10
        -0.012168  CMMA_5 * CMMA_20
         0.001762  CMMA_5 * LIN_ATR_7
         0.013164  CMMA_5 * LIN_ATR_15
         0.001154  CMMA_10 * CMMA_20
        -0.014178  CMMA_10 * LIN_ATR_7
         0.000042  CMMA_10 * LIN_ATR_15
         0.005368  CMMA_20 * LIN_ATR_7
        -0.019181  CMMA_20 * LIN_ATR_15
        -0.003459  LIN_ATR_7 * LIN_ATR_15
         0.653100  CONSTANT
```

The Nominal Mapping Transform

The vast majority of predictive models, including all of those available in *TSSB*, take one or more numeric values (indicators) as inputs and produce a numeric predicted output. But occasionally we may have an indicator that is *nominal*, meaning that it names classes rather than having a numerical value. For example, the month of the year may be useful as a predictor, but it certainly does not have a numerical value. Even if we code the month as an integer from 1 through 12, that does not make it numeric. It may be a literal number, but it's still just a label.

In many cases there may be a discoverable relationship between a nominal indicator and future behavior of the market. *TSSB* contains a trainable *Nominal Mapping Transform* that computes an optimal (in a reasonable sense) mapping from two or more classes to a single indicator with multiple numeric values. This numeric value can then be used as an input to predictive models.

Inputs and the Target

There are two different methods for presenting a nominal value to the transform. The most straightforward way is to use the NOMINAL INPUT option (described i the option summary later) to name a single variable whose values define classes. Each unique value of this variable in the database defines a class. This must be a TEXT variable (Page ?).

The other method for presenting the nominal value to the transform is to use the FLAG INPUT option (described in the option summary later) to name two or mor variables. Each variable corresponds to a class, and the class of a case will be defined by whichever variable has the maximum value. This is a general method for handling a common technique: a class coding scheme may be binary, in which all variables have the value '0' except for one, which has the value '1' and thereby identifies the class membership of the case. However, *TSSB* takes the more versatile approach of defining the class by the variable having the largest value. This, of course, subsumes the common 0/1 coding scheme.

The user must also name a target variable via the TARGETVAR specificatio described in the option summary later. This is the variable that determines the ordering of the classes, and it will generally be a variable that looks into the future, although this is not required. For example, suppose we have three classes. Also suppose that the target variable tends to be unusually large for members of Class 2 and unusually small for members of Class 1. Cases in Class 3 are middle-of-the-road. Then the mapping will be computed in such a way that the new indicator has relatively small values for Class 1, large values for Class 2, and middle values for

Class 3. In this way, the nominal indicator that gives the class membership provides a new numeric variable that correlates with the target variable. If the target variable looks into the future, which will nearly always be the intelligent choice, then this new indicator will hopefully have predictive power and be usable as an input to a predictive model.

The output of the nominal mapping transform is computed in one of two ways. The temptation is to just use the mean of each class. However, the heavy tails of many 'future return' variables renders this idea useless. It would be more reasonable to use the median of the target in each class. However, much experimentation revealed that even this can be problematic. A more stable method is to rank the target variable across the entire training set and then use the mean target percentile for each class. This preserves the order relationship of the targets while being fairly immune to distribution problems. Percentiles are constrained to 0-100.

The mapping method just described is performed by specifying the MEAN PERCENTILE option described in the option summary later. However, in rar instances the user may wish to remove the mapping rule even further from the target distribution. This can be done by specifying the RANKED PERCENTILE option. In this case, the mean percentiles are computed as just described, and then they themselves are ranked. The map output is thus the percentile of the mean percentile.

Gates

The nominal transform includes an advanced option that allows subdivision into more categories based on one or two additional inputs called *gates*. Gates are premised on the idea that the assignment of a transform output value to each input class may depend on other conditions. For an analogy, consider the popular ADX indicator. It measures the strength of a trend, but it does not indicate the direction of the trend. So if ADX were used to predict price movement in the near future, the relationship between ADX and future price would be different according to whether the market were trending up or trending down. This is not a perfect analogy, because ADX is continuous while the nominal mapping transform applies to nominal variables, but the idea is the same. We may have one mapping of classes to new indicator values when the market is in one state (category), and a completely different mapping when the market is in a different state. Thus we may have two different categories of class mappings.

We may employ one or two gate variables. A positive value of a gate variable defines one mapping, and a nonpositive value defines another mapping. Thus, if we use one gate variable we will have two different mapping categories, and if we use two gate variables we will have four different mappings.

The idea of gating can be taken a step further. It may be that one or more values of a gate variable signify an 'undefined' condition, one which is believed to contribute no useful information. To handle this situation, we need three possible mappings, the choice of which depends on the value of a gate variable. Two of these are the mappings already described, while the third is a 'neutral' mapping in which the output value of the transform is not influenced by the presumably uninformative input class. Rather, the output is always set to a 'neutral' value, the median of the target, for all input classes. Thus, we can use a gate to tell the transform to ignore the input class for one or more values of the gate variable. This is done by means of the IGNORE option, described in the option summary later. If two gates are employed, the IGNORE option applies only to the first. The program contains no provision for ignoring values of the second gate, if used.

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.

Focusing on Extreme Targets

Mapping from classes to numeric values is a dicey operation when the target variable is extremely noisy, as is the case in market price prediction. If there are more than a very few classes, it is almost certain that noise will cause some errors in assigning numeric values to classes. These errors will probably be of little consequence, because faulty ranking will usually be confined to local areas of the ranking. Maybe the correct ranking should be ABCDEF but the computed ranking turns out to be BADCFE. All that happens is that a class gets switched with a near neighbor. But if local errors are likely, why bother resolving the mapping to such a fine level? We gain little or nothing, for we are only propagating input noise to the output.

It is also well known in market price modeling that the most predictive power usually lies in the tails, the extreme values of the indicators. Central values of the indicators often have little predictive power. This is the motivation for the KEEP option, which will appear in the option summary later.

With the KEEP option, the user specifies a minimum percent of the cases that must be mapped at each extreme output value. Then the program maps as few classes as possible at each extreme, subject to the user's restriction. All other classes (the classes corresponding to central values of the target) are mapped to the median target percentile.

For example, suppose the user specifies that we are to KEEP 30 PERCENT. Th program will compute the mapping as described here. Then it will see how many cases are in the class having the minimum output value. If this is at least 30 percent of the total number of cases, it will finish dealing with the low end of the outputs and move on to the high end. But if not, it will advance to the class having the second-smallest output value, and add its cases to the number in the lowest class. This process continues, moving upwards until 30 percent of the cases have been accounted for. At that point, it will repeat the process, but beginning with the class having the largest output value. All of the extreme classes, at both the low and high end, retain their original mapping. But the interior cases are mapped to the median percentile.

Declaring the Transform and its Options

The syntax for declaring a nominal mapping transform is as follows:

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

The transform name may consist of letters, numerals, and the underscore character (_) and may be at most 15 characters. The following specifications (in addition to the usual OVERLAP option) are available:

FLAG INPUT = [Var1 Var2 ...]- 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 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. Either this specification or the NOMINAL INPUT specification must appear.

NOMINAL INPUT = Variable- Names a single variable whose value defines the nominal class membership. This must be a TEXT variable (Page ?) Either this specification or the FLAG INPUT specification mus appear.

TARGETVAR = Variable - Names a variable, usually a forward-looking variable, that will be used to train the mapping. The nominal-to-ordinal mapping will be defined so as to maximize a nonparametric measure of correlation between the mapped values and the target variable. This specification is mandatory.

GATEVAR = Variable

GATEVARS = Variable1 Variable2 - 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. These are optional.

IGNORE = (Value1 Value2 ...) - It may be that one or more values of the first gate variable indicate an 'undefined' condition, such as a missing input measurement. For example, a popular market pundit may offer one of several prognostications most mornings. You may not believe his/her predictions of upcoming market direction, but you may nevertheless consider them potentially useful, even if they may be anti-predictive! Furthermore, you may believe that these pronouncements map to future market movement in one manner when the market is in a high-volatility state, and in another manner when the market is in a low-volatility state. So you would use a gate variable to specify volatility. But what if some mornings a substitute prognosticator appears, someone whom you believe has no sense of the market. When this happens, you would specify a value for the gate variable that signals to TSSB that the substitute pundit's prediction is to be ignored. This value (or more than one if needed for complex situations) can be listed in the IGNORE specification. 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. This option may be used only if at least one gate is employed.

MAPPING = MEAN PERCENTILE - 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. (Most standard statistics references define interval scaling.) This is usually a valid assumption. Either this or the MAPPING = RANKED PERCENTILE specification m appear.

MAPPING = RANKED PERCENTILE - The MEAN PERCENTILE mapping 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 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 values of the transform are considered useful. Those that produce only a small deviation of the target from its median should be considered noise. This number, whichmust be specified 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.

A Nominal Mapping Example

The following script file uses two nominal mapping transforms to demonstrate a useful approach as well as most options. The four VLM indicators used in this example are all members of the VOLUME MOMENTUM family described here. Both of these examples use the FLAG INPUT specification to determine which of the four lookbacks is dominant. This is not a classic binary example, with the flag variables explicitly naming class membership. Instead, this illustrates the more general ability of the transform to choose the largest value from among competing indicators, which is an indirect assessment of class membership.

Here are the transform and model declarations. The OVERLAP=4 option (here) is used because the target looks ahead five days.

```
TRANSFORM MapDemo1 IS NOMINAL MAPPING [
   FLAG INPUT = [ VLM_3_4 VLM_5_4 VLM_10_4 VLM_20_4 ]
   GATEVARS = LIN_ATR_7 LINDEV_10
   TARGETVAR = DAY_RETURN_5
   MAPPING = MEAN PERCENTILE
   OVERLAP = 4
   ] ;

TRANSFORM MapDemo2 IS NOMINAL MAPPING [
   FLAG INPUT = [ VLM_3_4 VLM_5_4 VLM_10_4 VLM_20_4 ]
   GATEVARS = LIN_ATR_7 LINDEV_10
   TARGETVAR = DAY_RETURN_5
   MAPPING = MEAN PERCENTILE
   KEEP 20 PERCENT
   OVERLAP = 4
   ] ;

MODEL Mod1b IS LINREG [
   INPUT = [ MapDemo1 MapDemo2 ]
   OUTPUT = DAY_RETURN_5
   MAX STEPWISE = 2
   CRITERION = PROFIT FACTOR
   RESTRAIN PREDICTED
   MIN CRITERION FRACTION = 0.1
   OVERLAP = 4
   ] ;
```

The class input for these transforms considers only volume, not trend direction. Thus, we would probably not expect for there to be a meaningful relationship between the class membership and the *direction* of the future price movement. With no supplementary information about *historical* trend, this transform would be useless for predicting *future* trend. The volume information provided by the class input may be useful for predicting the *strength* of the future trend, which in turn may

be useful to a predictive model. But if we could also coax direction information from the transform we would be providing something even more useful to the model. This is where the two gate variables come into play.

LIN_ATR_7 is a simple trend variable that looks back 7 days. LINDEV_10 look back 10 days and uses a linear projection to estimate the most recent closing price. This indicator is the deviation of the actual value from the predicted value, so it detects a sudden departure from recent trend. These two indicators provide a lot of information about the price behavior of the market.

There is a second aspect of gating that is relevant. It would be unreasonable to assume that the relationship between the volume-based input class and future price movement is the same for all trend conditions. It could well be that when the market is trending upward, one volume class corresponds to an upcoming reversal, while when the market is trending downward a different volume class corresponds to an upcoming reversal. We should not count on everything being perfectly symmetric. This is why the nominal mapping transform computes a separate, independent mapping for each gate category.

The output produced by the *MapDemo1* transform appears on the here. There are four input classes, two categories for the first gate, and two categories for the second gate, making a total of 4*2*2=16 bins altogether. The output values corresponding to each category are listed in ascending order, along with the number of cases that fall into each bin. The signs enclosed in parentheses after each input 'class' name depict the gate status for the class, with the first sign being the first gate and the second sign being the second gate.

For example, here is the third line in the table:

```
VLM_5_4(+-)     687     47.455
```

This line corresponds to the indicator VLM_5_4 having the greatest value of the four input class indicators, LIN_ATR_7 being positive, and LINDEV_10 bein negative or zero. There were 687 cases that satisfied this set of conditions. In the future, any case satisfying this set of conditions will be assigned a value of 47.455, which is the mean of the percentiles of targets within this category.

```
Training NOMINAL MAPPING MAPDEMO1
Gate 1 is LIN_ATR_7
Gate 2 is LINDEV_10
MAPDEMO1 read 11187 cases for training
```

```
Number of cases and mean target percentile for each class

          Class      N  Mean pctile
VLM_10_4(++)     561     46.060
VLM_20_4(++)    1037     46.874
 VLM_5_4(+-)     687     47.455
VLM_10_4(-+)     343     47.983
VLM_20_4(+-)     917     48.678
VLM_10_4(+-)     696     48.956
 VLM_3_4(+-)     529     49.315
 VLM_3_4(++)    1284     49.809
 VLM_5_4(-+)     336     49.884
 VLM_5_4(++)     537     50.469
VLM_20_4(-+)     802     50.823
VLM_10_4(--)     597     51.621
 VLM_3_4(--)     805     52.075
VLM_20_4(--)    1115     52.406
 VLM_3_4(-+)     513     54.415
 VLM_5_4(--)     428     54.744
```

Nothing notable pops out of this chart in regard to the volume-bases input classes, except perhaps the remarkable *lack* of consistency. For example, VLM_10_4 has the smallest output in the ++ and -+ gate conditions, but this same input 'class' is well above the median for the -- gate condition.

On the other hand, something quite remarkable is revealed for the first gate, LIN_ATR_7, a measure of short-term trend. With only one exception, the fourth line in the chart, the smallest outputs correspond to a positive value of this trend. Also with only one exception, the largest transform outputs correspond to negative (or zero) values of this trend. In other words, we clearly see a mean reversion going on here, as opposed to trend persistence.

The second nominal mapping transform, *MapDemo2*, is identical to the first, except that the KEEP 20 PERCENT option is used. This results in the chart of output shown below. The 20 percent outer class values are identical to those in the chart just seen, while the central vales are all changed to the mean percentile, 50.004. Twenty percent of 11187 (the total number of cases) is 2237. The lower 'unchanged' set contains 561+1037+687=2285 cases, and the upper set contains 428+513+1115+561=2861 cases. At both ends, this is the smallest number of unchanged bins such that the total number of cases at that extreme is at least 2237. With the KEEP option, information is retained and passed on to the transform output only for those bins which correspond to extreme values of the target. Information in the muddy middle is discarded.

Revised after keeping outer 20.0 percent...

Class	N	Mean pctile
VLM_10_4(++)	561	46.060
VLM_20_4(++)	1037	46.874
VLM_5_4(+-)	687	47.455
VLM_10_4(-+)	343	50.004
VLM_20_4(+-)	917	50.004
VLM_10_4(+-)	696	50.004
VLM_3_4(+-)	529	50.004
VLM_3_4(++)	1284	50.004
VLM_5_4(-+)	336	50.004
VLM_5_4(++)	537	50.004
VLM_20_4(-+)	802	50.004
VLM_10_4(--)	597	50.004
VLM_3_4(--)	805	52.075
VLM_20_4(--)	1115	52.406
VLM_3_4(-+)	513	54.415
VLM_5_4(--)	428	54.744

The ARMA Transform

An ARMA (autoregressive, moving average) model can be fit to a moving window in the database (an indicator) or a market, and any of several quantities can be computed to generate a new database variable. Readers unfamiliar with ARMA models may consult any of the widely available references.

ARMA models use historical values of a time series to estimate future values, often just the next value. The ARMA model does this by exploiting either or both of two tendencies that may be present in a time series: autoregressive (AR) and moving average (MA). The ARMA modeling method looks for these effects and builds an optimal model that incorporates them. The autoregressive tendency refers to the fact that the expected current value of a time series is significantly related to recent past values of the series. The moving-average tendency, as it is used in the ARMA context, refers to the fact that each actual value of a time series deviates from its expected value due to some random shock. The expected current value of the time series is significantly related to recent past shocks that impacted the series. ARMA models can use either or both of these tendencies to create a forecast: the current and recent past values of the series let us use the AR component of the model to predict the next expected value, and the current and past shocks experienced by the series let us use the MA component of the model to predict the next expected value. Both of these predictions (AR and MA) may be combined to make a pooled prediction (ARMA) of the next value of the series. *TSSB* uses the ARMA model to derive a number of different types of indicators that can serve as inputs to a prediction model. 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, an @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 for use as an indicator 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 (deviation from the predicted value) 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 specified lag*

MA PARAMETER Integer - *The value of the trained MA parameter at the specified 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. This is useful when the source series is market prices.

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 o 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 change of 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 compute differences so that the ARMA model is operating on changes in the logs, not actual market prices. The resultant series is graphed at the bottom of this page in Figure 19.

```
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
   ] ;
```

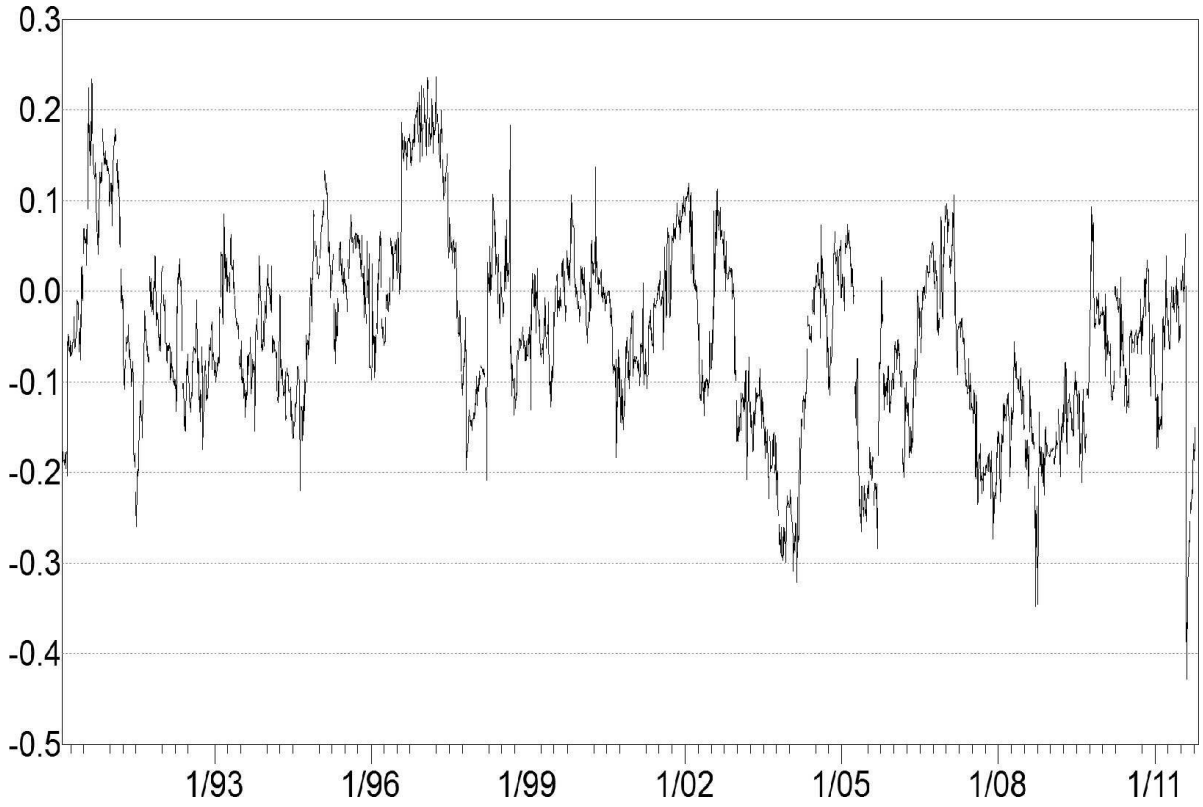


Figure 19: The AR parameter for changes in the close of OEX

It is interesting to examine the mean squared error (ARMA OUTPUT = MSE) as the window moves along. This is just the variance of the shocks within the window.

This series, shown in Figure 20 below, is obviously worthless as an indicator because it is so nonstationary. But we may find the corresponding R-square (ARMA OUTPUT = RSQUARE) to be of some value. It is shown in Figure 21.

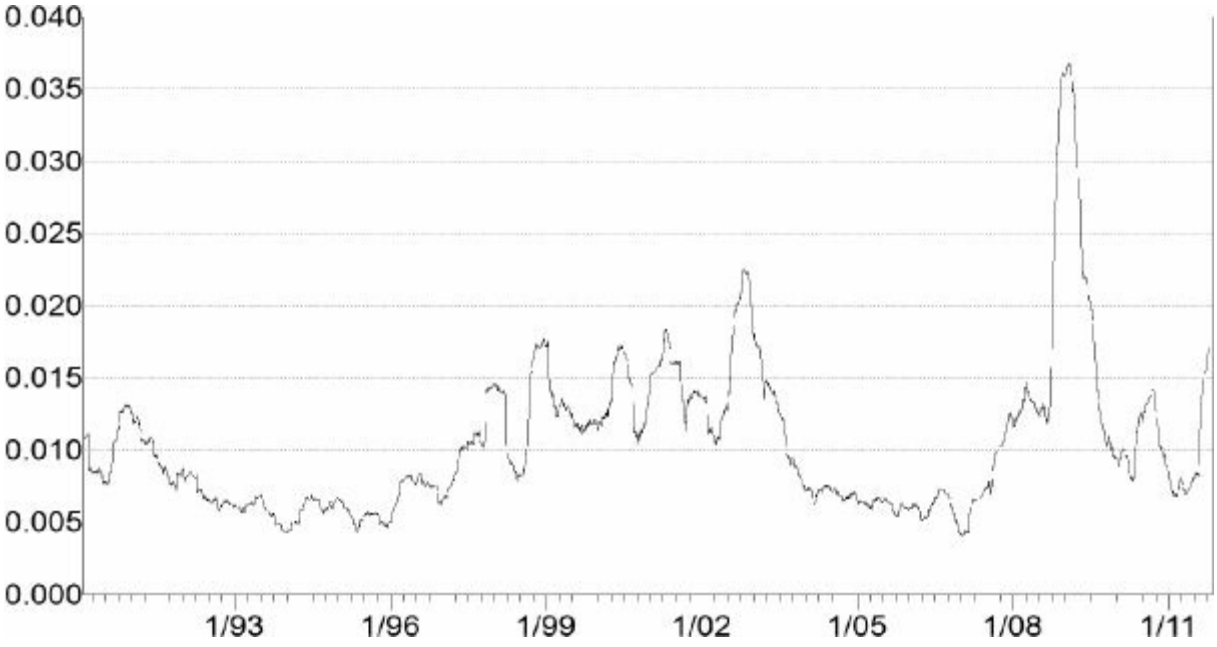


Figure 20: Variance of the shocks

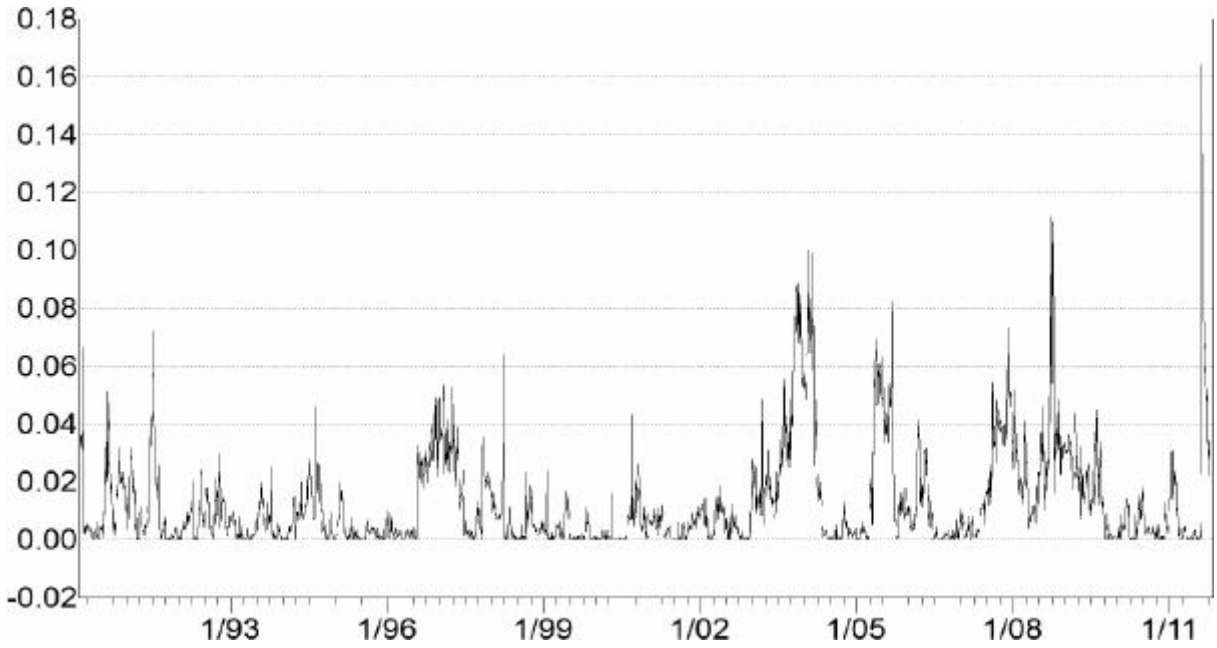


Figure 21 :R-square within the window

In many cases, the indicator we are most interested in is the shock series. For each day, this is the component of today's move that is not predictable by the ARMA model. The shocks for the OEX series (*ARMA OUTPUT = SHOCK*) are plotted in Figure 22 below. Clearly the indicator is too nonstationary in its variance for this series to be usable in a prediction model. However, by dividing each day's shock

by the standard deviation of the shocks within the window we can standardize them and obtain the much better behaved series *(ARMA OUTPUT = STANDARDIZE SHOCK*) shown in Figure 23.

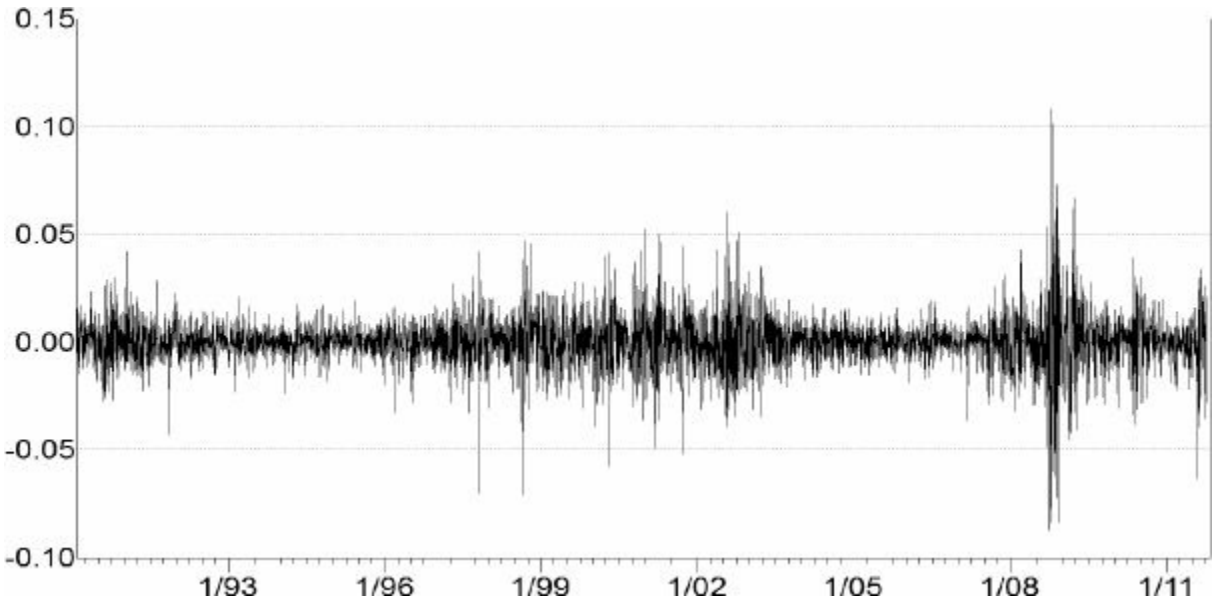


Figure 22: ARMA shocks as the window moves

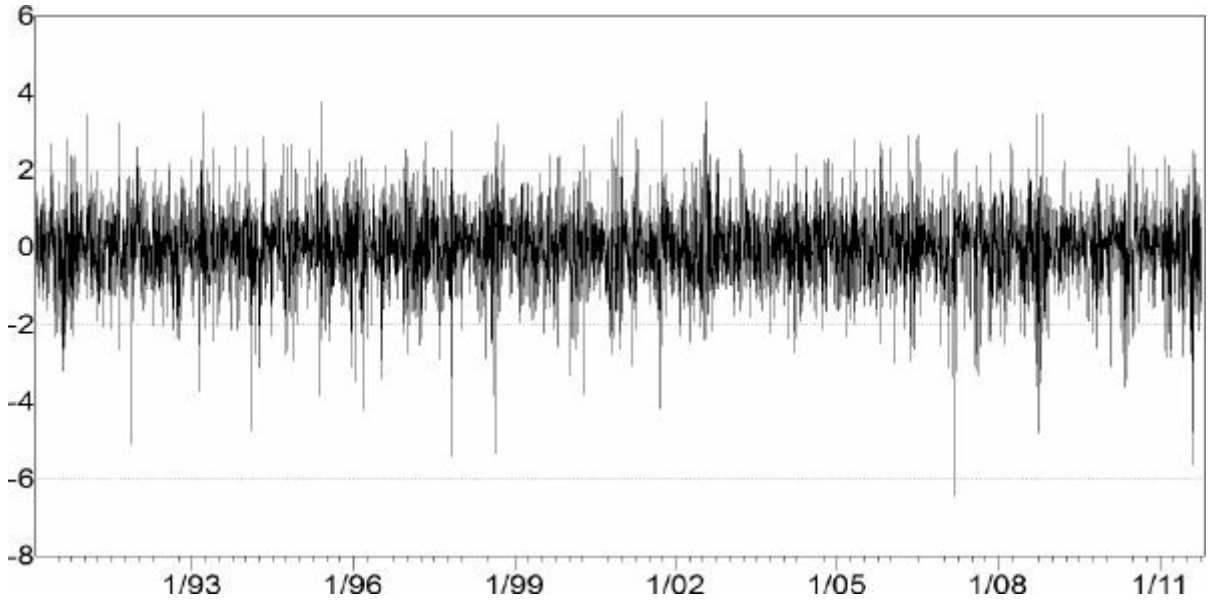


Figure 23: Standardized shocks (shocks divided by the window shock std dev)

The VIX series is extremely nonstationary, so one would be inclined to difference it to induce, or at least improve, stationarity. On the other hand, the use of a 100-day moving window produces decent stationarity within each window, meaning that an ARMA model may be effective. We'll try each. Figure 24 below shows the AR parameters of the raw series, while Figure 25 shows that for the differenced series. It's interesting to note the strong tendency in the latter toward negative serial correlation. See here for the script declarations that produced these series.

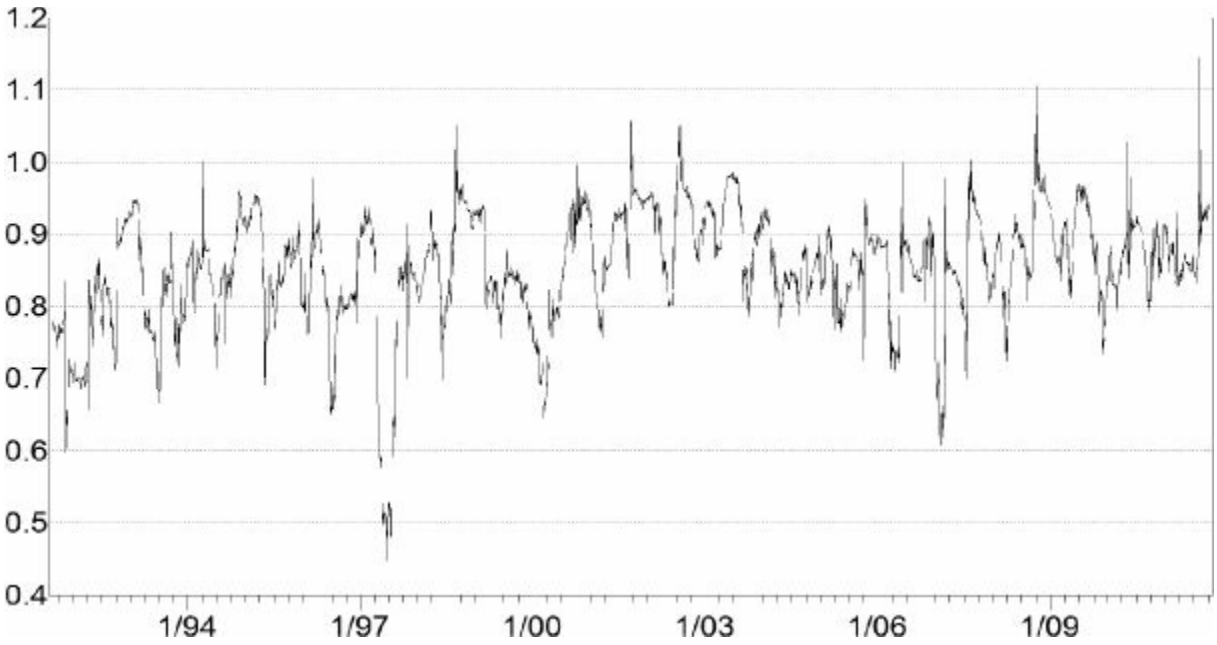


Figure 24: AR parameters of the original VIX series

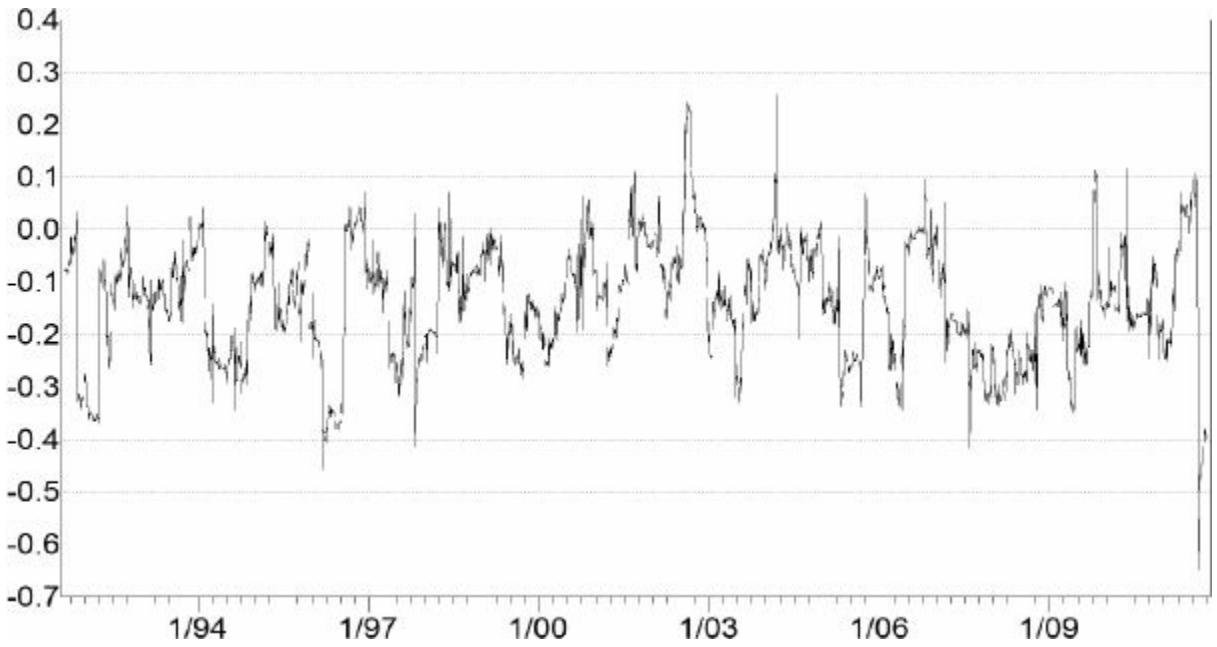


Figure 25: AR parameters of the differenced VIX series

As with OEX, we are probably most interested in the shock series as an indicator. Figure 26 below is the standardized shocks for the original VIX series, and Figure 27 is that for the differenced series. Notice how similar they are, even though the ARMA models are totally different. This makes sense, because the shock each day is the unpredictable component of the new value, and this should be about the same, whether you are predicting an actual value or a difference. However, if you look closely you will see that the shocks from the differenced series are slightly more stationary in the mean than those from the raw series.

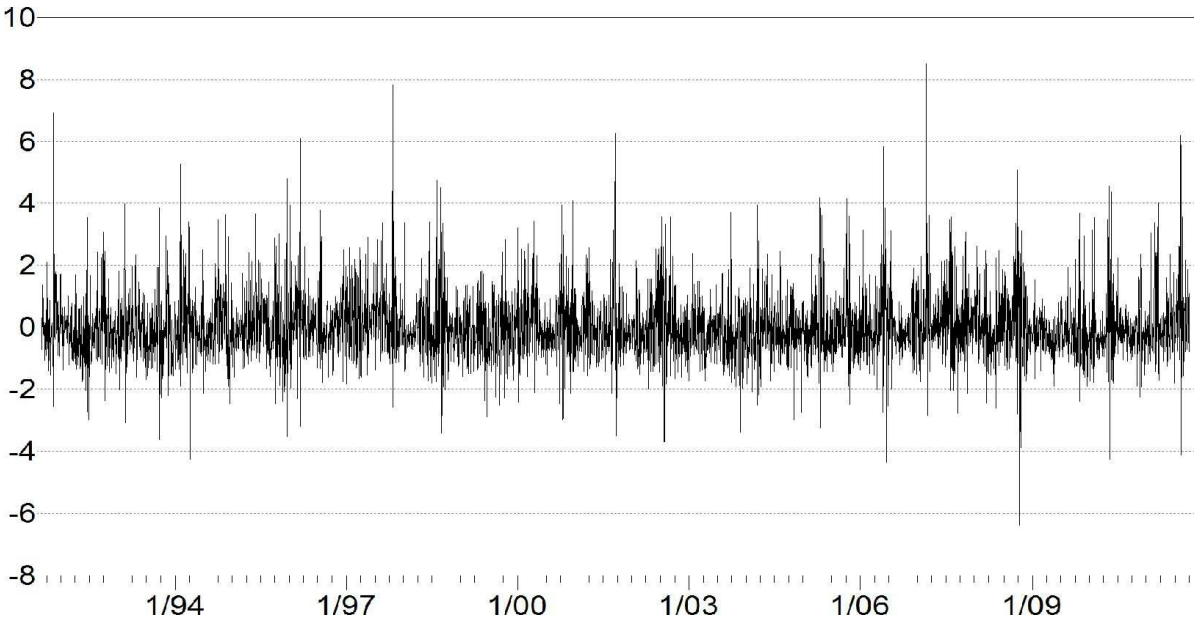


Figure 26: Standardized shocks of the original VIX series

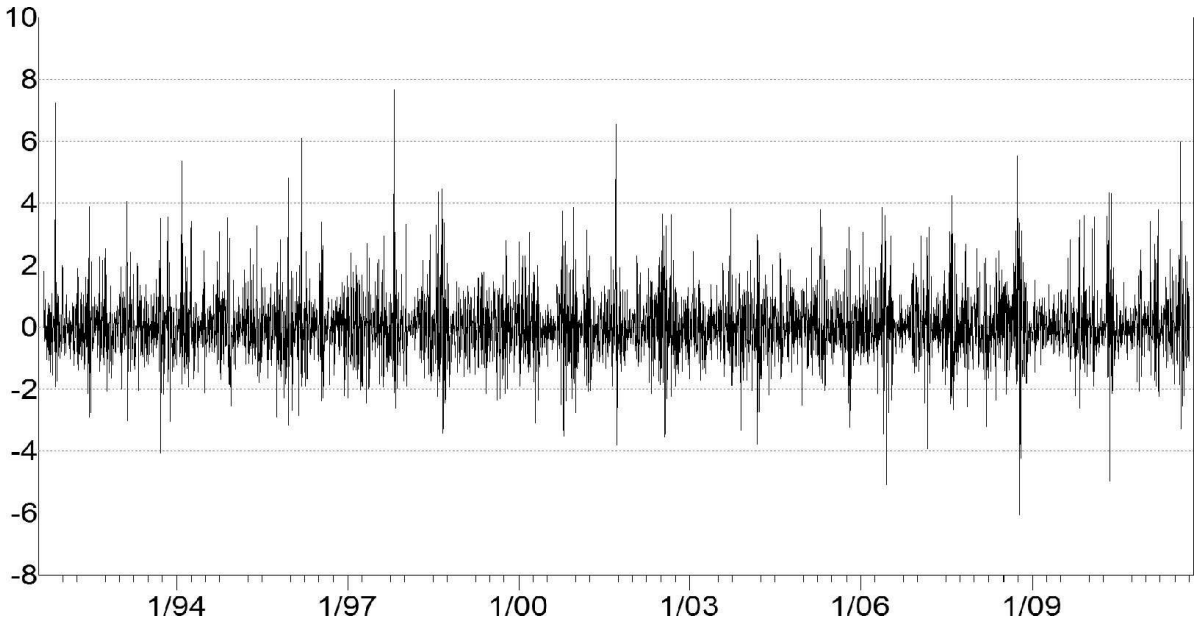


Figure 27: Standardized shocks of the differenced VIX series

Here are the script file declarations that produced Figures 24 through 27:

```
TRANSFORM VIX10_AR IS ARMA [
   SERIES = @CLOSE:VIX
   ARMA WINDOW = 100
   ARMA OUTPUT = AR PARAMETER 1
   ARMA AR = 1
   ARMA MA = 0
   ] ;
```

```
TRANSFORM VIX10_D_AR IS ARMA [
   SERIES = @CLOSE:VIX
   ARMA WINDOW = 100
   ARMA TRANSFORM = DIFFERENCE
   ARMA OUTPUT = AR PARAMETER 1
   ARMA AR = 1
   ARMA MA = 0
   ] ;

TRANSFORM VIX10_STD IS ARMA [
   SERIES = @CLOSE:VIX
   ARMA WINDOW = 100
   ARMA OUTPUT = STANDARDIZED SHOCK
   ARMA AR = 1
   ARMA MA = 0
   ] ;

TRANSFORM VIX10_D_STD IS ARMA [
   SERIES = @CLOSE:VIX
   ARMA WINDOW = 100
   ARMA TRANSFORM = DIFFERENCE
   ARMA OUTPUT = STANDARDIZED SHOCK
   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 an 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 LOOKBACK 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 th 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 th 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 b 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 PURIFIE 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 man 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 computi 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 wil 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 sever 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 purifie 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 th 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 indicato **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 f 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.

Complex Prediction Systems

Prior chapters have discussed Transforms, Models, Committees, and Oracles, and simple tutorial examples of each have been presented. Those examples are probably close to the sorts of prediction systems that most users would employ for a practical trading system. However, much more exotic systems are possible in *TSSB*. This chapter will demonstrate how to construct an extraordinarily complex prediction system using the basic building blocks available in *TSSB*.

First, we should briefly review some of the elementary principals involved. Modeling methodologies in *TSSB* are divided into two broad families: *Models* and *Committees*. (For the purposes of the discussion in this section, *Oracles* are considered to be *Committees*. Also, we will use the lower-case *model* and *committee* to refer to them in a generic sense only, and the upper-case *Model* and *Committee* to refer to them in the specific *TSSB* sense as well as the generic sense.)

Models and Committees are very similar, often identical in their usage in *TSSB*. Nonetheless, there are some important distinctions which will now be discussed. Their most basic similarity is that they both take one or more inputs and map values of the inputs to an output. In general, the inputs (often called *indicators* for Models) will be backward-looking in time (they depend only on past and current values of market history), while the output will be a prediction of a forward-looking variable often called the *target*. The overall goal of a model-based trading system is to map *indicators* to a *target* and base trading decisions on predicted values of the target. Models and Committees play a crucial role in this process.

TSSB executes Transforms, Models, and Committees in the order in which they appear in the script file. The user can feed the output of one of these items into the input of another item. For example, Model outputs can serve as Transform inputs. Committee outputs can serve as the inputs of other Committees. Many other sequences of data flow are possible. We will present an example of this in the next section. But first, we will explore the similarities and differences between Models and Committees.

On a philosophical level, the fundamental difference between a Model and a Committee is that a Model (usually) takes indicators as inputs, while a Committee (always) takes predictions made by Models or other Committees as inputs. As will be seen, *TSSB* allows Models to be used as committees, so this rule is not inviolate, but it is a good basis of understanding and motivation.

Here are the specific issues involving Models and Committees in *TSSB*:

- Many of the most common modeling methodologies are available for both Models and Committees. These are LINREG, GRNN, MLFN, TRE BOOSTED TREE, and FOREST. However, some methodologies are rare appropriate for committees and hence are available only for Models. These are QUADRATIC, OPSTRING, and SPLIT LINEAR. Similarly, so methodologies are inherently committees, not generally suitable for use as models. These are AVERAGE and CONSTRAINED.

- A Model can take anything as an input, including indicators, predictions made by Committees, and predictions made by other Models. Thus, a Model car always be used as a committee if the user wishes. However, a Committee can take as input only predictions made by Models or other Committees. *Indicators cannot be used as inputs to a Committee*. This is in keeping with the underlying philosophy that the purpose of a committee is to combine multiple predictions into a single pooled prediction.

- The most important distinction between a Committee and a Model that is being used as a committee arises if the inputs to the 'committee' are from Models in which the FRACTILE THRESHOLD (here) option is used. In this case, if a Model is being used to combine those FRACTILE THRESHOLD predictions, the inputs to this Model are the fractiles, the same quantities that are preserved in the database as the component Models' predictions. But if a Committee is being used to combine the FRACTILE THRESHOL predictions, then the inputs to the Committee are the raw outputs of the component Models, the outputs *before* they are converted to fractiles. There is some modest theoretical justification to preferring this latter approach, although the reasoning is not compelling. Using the fractiles as committee inputs, which happens when a Model is used as a committee, is certainly a viable option.

Stacking Models and Committees

We now present an extreme example of the sort of stacking that is available in *TSSB*. It must be pointed out that this example is far more complex than any that most users would devise. However, it does demonstrate how multiple levels of Models and Committees can be stacked for sophisticated operation.

The first section of this script file is shown below. Model MOD_LOOK_5 examines four indicators (linear trend, RSI, Stochastic, and Reactivity) that have a short lookback, just five bars. Model MOD_LOOK_20 is similar, but its indicators look back 20 bars. It makes sense to combine these two models with an Oracle which is gated by a measure of volatility. The idea is that one volatility regime may favor a short lookback distance, while another volatility regime may favor a longer lookback. But it is not obvious how far the volatility indicator should look back. One volatility lookback may be more effective than another, or they may actually complement each other. So we employ two Oracles. One uses PVR_5, a short lookback measure of volatility, as its gate. The other uses the same volatility family but with a longer lookback. Finally we combine the predictions of these two Oracles using a Committee.

```
MODEL MOD_LOOK_5 IS LINREG [
   INPUT = [ LIN_5 RSI_5 STO_5 REACT_5 ]
   OUTPUT = HIT_MISS_1
   MAX STEPWISE = 2
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   ] ;

MODEL MOD_LOOK_20 IS LINREG [
   INPUT = [ LIN_20 RSI_20 STO_20 REACT_20 ]
   OUTPUT = HIT_MISS_1
   MAX STEPWISE = 2
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   ] ;

ORACLE ORAC_NO_PS_5 [
   INPUT = [ MOD_LOOK_5 MOD_LOOK_20 ]
   GATE = [ PVR_5 ]
   OUTPUT = HIT_MISS_1
   MIN CRITERION FRACTION = 0.1
   ] ;
```

```
ORACLE ORAC_NO_PS_20 [
   INPUT = [ MOD_LOOK_5 MOD_LOOK_20 ]
   GATE = [ PVR_20 ]
   OUTPUT = HIT_MISS_1
   MIN CRITERION FRACTION = 0.1
   ] ;

COMMITTEE COMM_NO_PS IS CONSTRAINED [
   INPUT = [ ORAC_NO_PS_5 ORAC_NO_PS_20 ]
   OUTPUT = HIT_MISS_1
   MAX STEPWISE = 0
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   ] ;
```

Now we work toward a similar goal, but using an entirely different approach, somewhat opposite the approach just taken. In the example shown above, the user explicitly specified lookbacks, and then volatility-gated Oracles combined the predictions that came from the different lookbacks. The approach shown below reverses the technique. It uses PRESCREEN model options to create specialis models based on volatility, and provides a diverse set of indicators from which the model may choose. An Oracle with the HONOR PRESCREEN option combines tl models. Because we don't know which lookback (5 or 20) will be best, we try both and then combine them with a Committee, much as we did above. Here are these commands:

```
MODEL MOD_PS_POS_5 IS LINREG [
   INPUT = [ CMMA_5 CMMA_20 LIN_5 RSI_5 STO_5 REACT_5
               LIN_20 RSI_20 STO_20 REACT_20 ]
   OUTPUT = HIT_MISS_1
   PRESCREEN PVR_5 > 0.0
   MAX STEPWISE = 2
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   ] ;

MODEL MOD_PS_NEG_5 IS LINREG [
   INPUT = [ CMMA_5 CMMA_20 LIN_5 RSI_5 STO_5 REACT_5
               LIN_20 RSI_20 STO_20 REACT_20 ]
   OUTPUT = HIT_MISS_1
   PRESCREEN PVR_5 <= 0.0
   MAX STEPWISE = 2
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   ] ;
```

```
ORACLE ORAC_PS_5 [
   INPUT = [ MOD_PS_POS_5 MOD_PS_NEG_5 ]
   GATE = [ PVR_5 ]
   HONOR PRESCREEN
   OUTPUT = HIT_MISS_1
   MIN CRITERION FRACTION = 0.1
   ] ;

MODEL MOD_PS_POS_20 IS LINREG [
   INPUT = [ CMMA_5 CMMA_20 LIN_5 RSI_5 STO_5 REACT_5
             LIN_20 RSI_20 STO_20 REACT_20 ]
   OUTPUT = HIT_MISS_1
   PRESCREEN PVR_20 > 0.0
   MAX STEPWISE = 2
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   ] ;

MODEL MOD_PS_NEG_20 IS LINREG [
   INPUT = [ CMMA_5 CMMA_20 LIN_5 RSI_5 STO_5 REACT_5
             LIN_20 RSI_20 STO_20 REACT_20 ]
   OUTPUT = HIT_MISS_1
   PRESCREEN PVR_20 <= 0.0
   MAX STEPWISE = 2
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   ] ;

ORACLE ORAC_PS_20 [
   INPUT = [ MOD_PS_POS_20 MOD_PS_NEG_20 ]
   GATE = [ PVR_20 ]
   HONOR PRESCREEN
   OUTPUT = HIT_MISS_1
   MIN CRITERION FRACTION = 0.1
   ] ;

COMMITTEE COMM_PS IS CONSTRAINED [
   INPUT = [ ORAC_PS_5 ORAC_PS_20 ]
   OUTPUT = HIT_MISS_1
   MAX STEPWISE = 0
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   ] ;
```

Now we take still a third approach, completely different from those just shown. We train two linear models using the EXCLUSION GROUP option, and then combir them using an MLFN Model. This nonlinear Model lets us pick up an nonlinearities in the relationships. Here are these commands:

```
MODEL MOD_EX1 IS LINREG [
   INPUT = [ CMMA_5 CMMA_20 LIN_5 RSI_5 STO_5 REACT_5
             LIN_20 RSI_20 STO_20 REACT_20 ]
   OUTPUT = HIT_MISS_1
   MAX STEPWISE = 2
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   EXCLUSION GROUP = 1
   ] ;

MODEL MOD_EX2 IS LINREG [
   INPUT = [ CMMA_5 CMMA_20 LIN_5 RSI_5 STO_5 REACT_5
             LIN_20 RSI_20 STO_20 REACT_20 ]
   OUTPUT = HIT_MISS_1
   MAX STEPWISE = 2
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   EXCLUSION GROUP = 1
   ] ;

MODEL MOD_EX IS MLFN [
   INPUT = [ MOD_EX1 MOD_EX2 ]
   OUTPUT = HIT_MISS_1
   OUTPUT LINEAR
   DOMAIN REAL
   FIRST HIDDEN = 2
   MAX STEPWISE = 0
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   ] ;
```

At this point we have three predictions available:

> **COMM_NO_PS** is based on the first example, in which the user explicitly specified lookbacks for the model inputs, and used volatility-gated Oracles to combine the models.
>
> **COMM_PS** is based on the second example, in which all indicators were supplied to the component models, which were then trained to be volatility-based specialists using the PRESCREEN command.
>
> **MOD_EX** is based on the third example, in which a Model is used as a committee to combine the predictions of EXCLUSION GROUP models.

We can wrap this all up into a single prediction by combining these three predictions with a final Committee:

```
COMMITTEE COMM_FINAL IS CONSTRAINED [
   INPUT = [ COMM_NO_PS COMM_PS MOD_EX ]
   OUTPUT = HIT_MISS_1
   MAX STEPWISE = 0
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   ] ;
```

Once again it must be stated that few users would want to create a trading system as complex as the multi-layered behemoth just shown. The complete script file is over 150 lines long! Still, this example illustrates the nearly unlimited ability of *TSSB* to stack prediction upon prediction using assorted Models, Committees, and Oracles.

Graphics

TSSB contains a variety of methods for plotting variables. These variables can be indicators and targets in the database, whether computed internally or read in from an external file. They can also be predicted values from walkforward or cross validation runs, as long as the predicted values were saved via the PRESERVE PREDICTIONS command (here).

Plots are displayed on the screen, but they can also be printed via the *File/Print* menu selection. In order to print a plot, the user must click on the plot so that Windows highlights it. If no plot is highlighted, the *File/Print* option will be grayed out so that it cannot be selected.

The plotting options available through the menu system under the *Plot* selection include the following:

- ***Series*** - The variable is plotted as a time series, with dates labeled at the bottom of the plot.
- ***Series + Market*** - This is identical to a *Series* plot except that the log of the market close is overlaid with the variable plot. Also, the user can plot a horizontal threshold line.
- ***Histogram*** - A histogram of a variable is plotted. In a multiple-market environment, the histogram can be based on a single market or all markets pooled.
- ***Thresholded Histogram*** - This is similar to the *Histogram* plot except that subsets of the complete dataset (all markets) based on indicators or predicted values can be plotted.
- ***Density Map*** - Various density estimators are used to produce a sophisticated version of a scatterplot for showing the relationship between two variables.
- ***Bivariate Plot*** - Shows the relationship between a target and two indicators.
- ***Trivariate Plot*** - Shows the relationship between a target and two indicators, conditional on a third indicator.
- ***Prediction Map*** - For a model having two inputs, this displays a two-dimensional map showing how values of the indicators affect the predicted target, thus revealing the inner workings of the model.

- ***Indicator-Target relationship*** - Shows a historical time series plot of the relationship between an indicator and a target.

NOTE... Many of the plots shown in this chapter are black-and-white renderings of color images, and hence they do not reproduce well. The authors will at some point include full color versions on the TSSB website. For now the reader must use his or her imagination.

Series Plot

The variable is plotted as a time series, with dates labeled along the bottom of the plot. The user specifies the following items:

Variable - The variable to be plotted. They are listed in alphabetical order.

Market - All markets are listed, even if there is only one. The user selects one.

Connect - If this box is checked, the values will be connected with straight lines. If it is not checked, each value will be isolated and represented by a vertical line.

After the plot is displayed, an individual section can be magnified. Place the mouse cursor at the left edge of the section to be magnified. Press and hold the left mouse button while dragging the cursor to the right. The selected area will be highlighted. Release the mouse button when the desired right boundary is attained.

If a section of the plot has been magnified, a horizontal scroll bar will appear. The user can move the magnified section along the entire time-range of the series in three ways:

1) Drag the elevator button to move a large distance

2) Click anywhere inside the elevator bar on either side of the button to move in medium jumps.

3) Click on the small buttons on the extreme left or right of the bar to move in small jumps.

The following page shows series plots with and without the *connect* option.

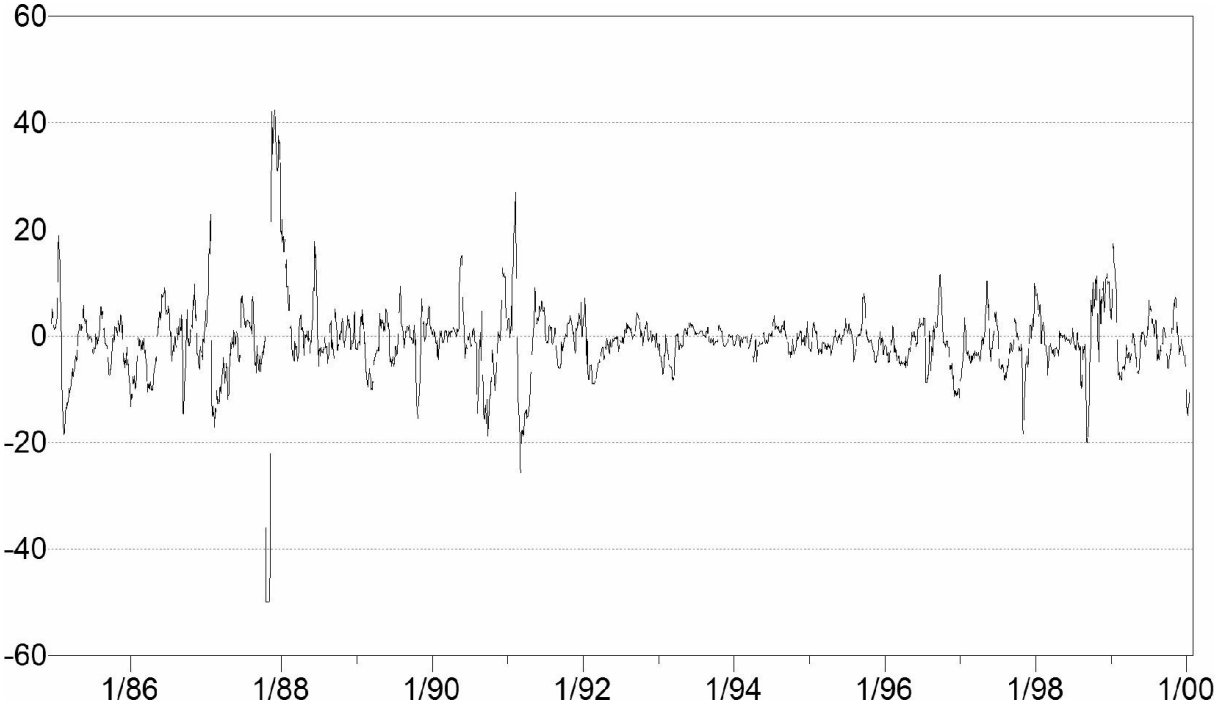


Figure 28: Series plot with *connect* option

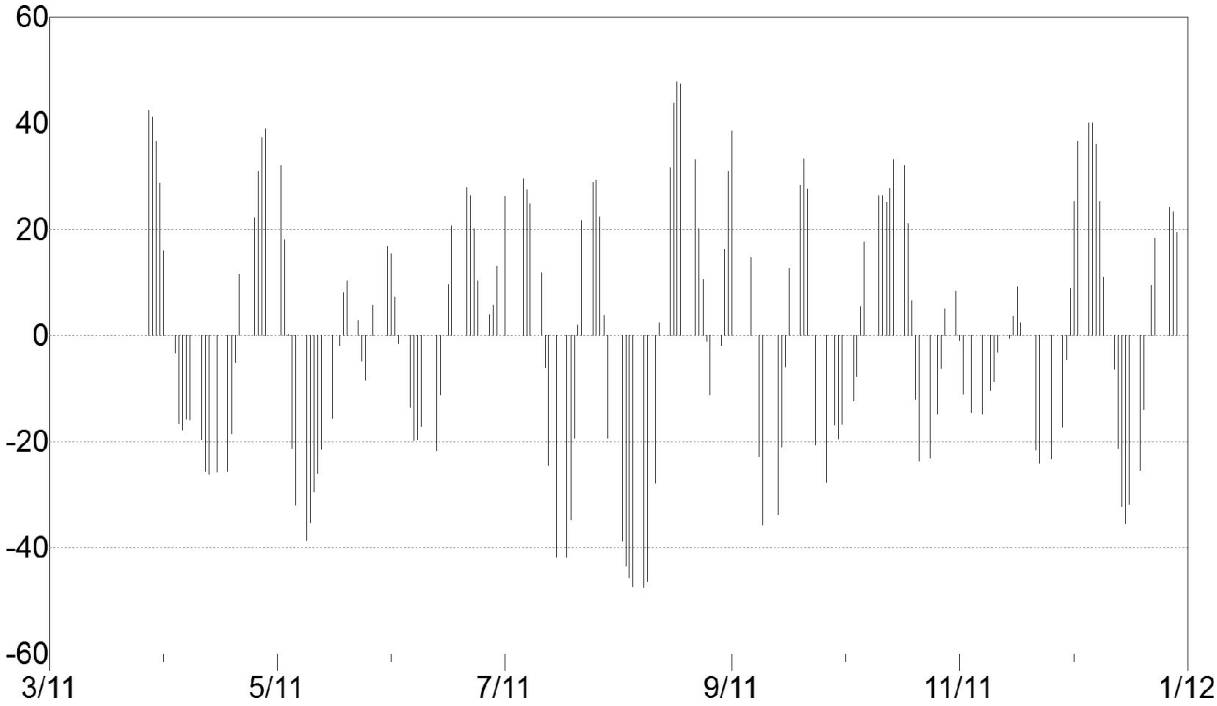


Figure 29: Series plot without *connect* option

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. An example of this plot is shown below:

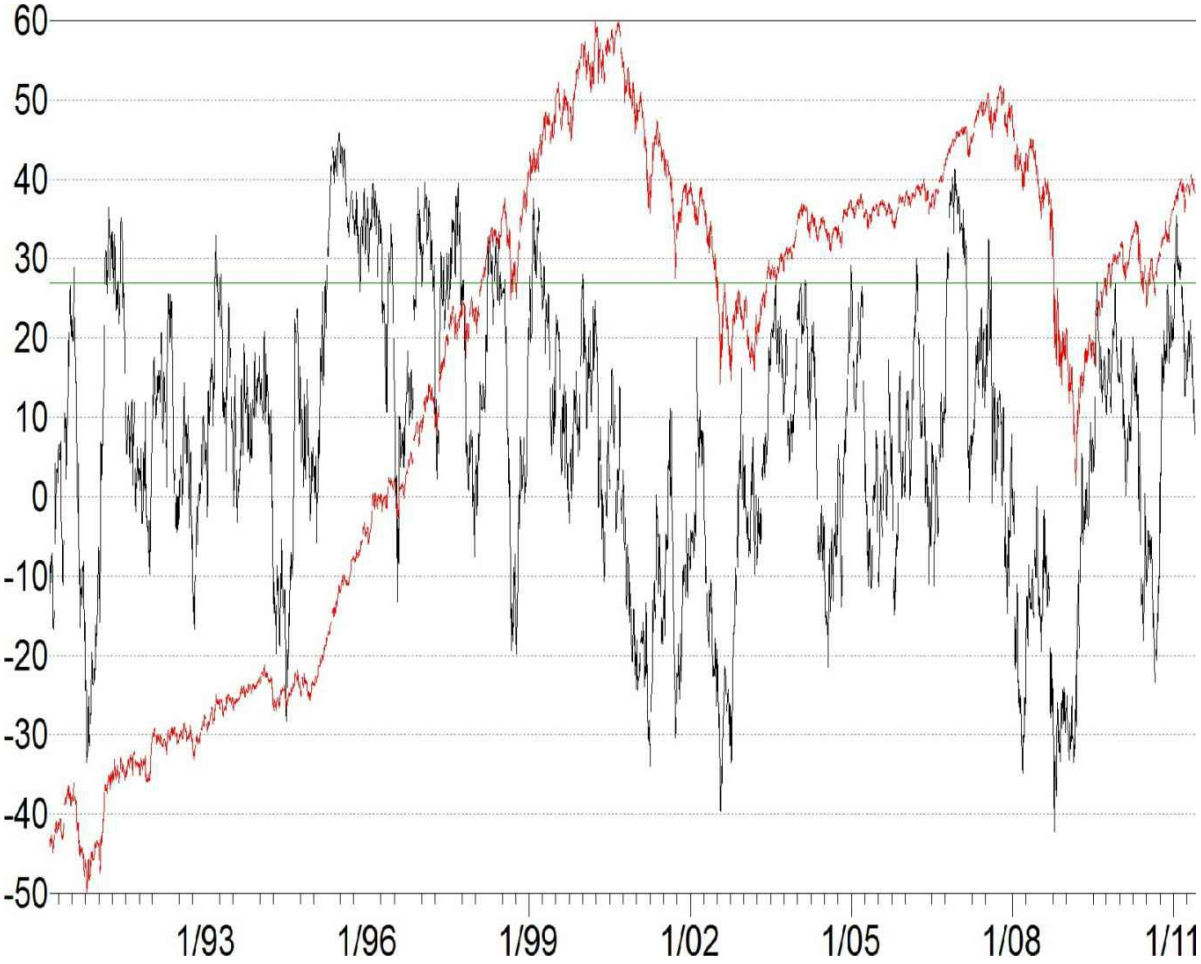


Figure 30: Series + Market plot, also using *threshold* option

Histogram

A histogram of a variable is plotted. The height of each bar represents the fraction of the total number of observations that fall into that bin. The user specifies the following items:

Number of Bins - This many bars will be displayed, each corresponding to a bin that counts how many cases fall into a range. Making this an odd number causes the graph to be centered at zero if the upper and lower limits are equal (of opposite sign), which is visually appealing.

Histogram Variable - The variable to be plotted. The candidates are listed alphabetically.

Primary Market - This lists all markets, even if there is only one. In addition, *Pooled* appears as the option at the top of the list. The user may select an individual market, in which case only data from that market will be used to compute the histogram. If instead the user selects *Pooled*, data from all markets will be pooled to compute the histogram.

Use Second Market - If this box is checked, the histogram will show the variable's distribution in two markets, using bars with different appearances. The primary market is displayed as solid black bars, and the secondary market is displayed as cross-hatched bars. This enables the user to easily compare the distributions in different markets, or in a single market versus pooled data.

Secondary Market - If the *Use Second Market* box is checked, this list of markets will appear so that the user can select a second market or pooled data for display.

Lower Limit - This comprises a check box and a field for entering a numeric value. If the box is checked, the user must enter a number in the field. This will be the lower (left) limit for the plot. All values less than or equal to this number will be cumulated in the leftmost histogram bin. This is handy for variables that have one or a few extreme values that cause the graph to become unnaturally compressed.

Upper Limit - This is similar to the *Lower Limit* except that it specifies the upper (right) limit for the plot.

Normalize Areas - This is relevant only if the user checks the *Use Second Market* box to display overlaid histograms. If the *Normalize Areas* box is not

checked, the height of each bar will represent the fraction of cases in each bin relative to the *total number of cases* and the vertical axis will be labeled as this fraction. If this box is checked, the height of each bar will represent the fraction of cases in each bin relative to the *number of cases in that market* and the vertical axis will be labeled zero to one. This latter option is usually preferable to the former.

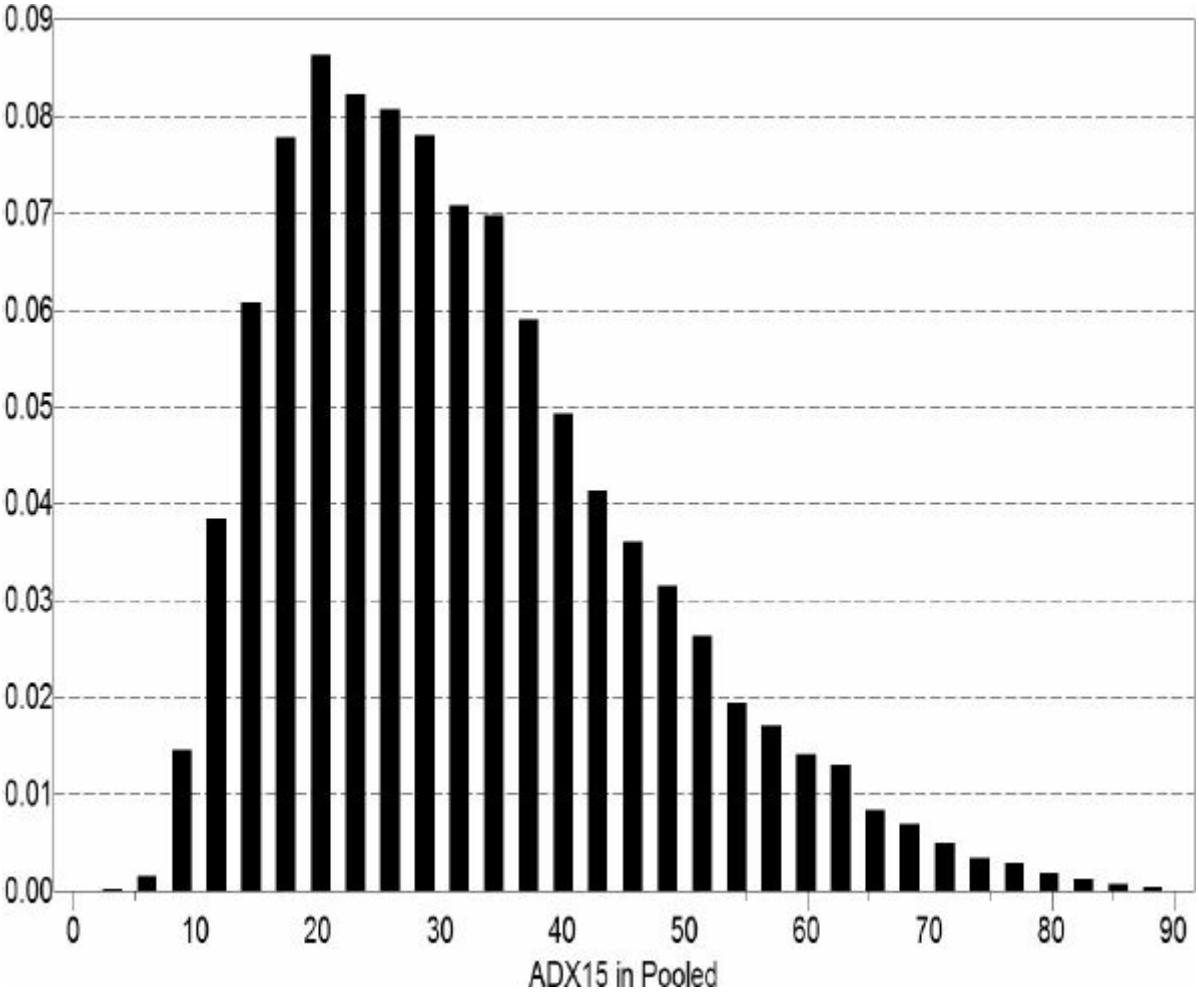


Figure 31: A typical histogram

Thresholded Histogram

This is similar to the ordinary histogram described on the prior page, but with a twist. 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 is based on only a subset of the cases, and the particular subset is determined by the value of the thresholded variable for each case.

The user specifies the following items:

Number of Bins - This many bars will be displayed, each corresponding to a bin that counts how many cases fall into a range. Making this an odd number causes the graph to be centered at zero if the upper and lower limits are equal, which is visually appealing.

Plotted Variable - The variable whose histogram is to be plotted. The candidates are listed in alphabetical order.

Thresholded Variable - The variable whose thresholded values determine which cases are plotted. The candidates are listed in alphabetical order.

Lower Limit - This comprises a check box and a field for entering a numeric value. If the box is checked, the user must enter a number in the field. This will be the lower (left) limit for the variable being plotted. All values less than or equal to this number will be cumulated in the leftmost histogram bin. This is handy for variables that have one or a few extreme values that cause the graph to become unnaturally compressed.

Upper Limit - This is similar to the *Lower Limit* except that it specifies the upper (right) limit for the plotted variable.

Lower Percent - This comprises a check box and a field for entering a numeric value that refers to the thresholded variable. If the box is checked, the user must enter in the field a number greater than zero and less than 100. The specified percent of cases having the smallest values of the thresholded variable are included in the plot and represented as solid bars, black by default and red if *overlayed colors* is selected..

Upper Percent - This is similar to the *Lower Percent* except that it specifies a high cutoff for the thresholded variable. The specified percent of cases having the largest values of the thresholded variable are included in

the plot and are represented as hatched (or solid blue if *overlayed colors* is selected) bars if the user also specified a lower percent.

Normalize Areas - This is relevant only if the user specifies both a lower percent and an upper percent for the thresholded variable. In this case, the lower percent will be represented by solid bars and the upper percent by hatched bars. If the *Normalize Areas* box is not checked, the height of each bar will represent the fraction of cases in each bin relative to the number of cases that meet *either* threshold criterion and the vertical axis will be labeled as this fraction. If this box is checked, the height of each bar will represent the fraction of cases in each bin relative to the number of cases that meet the *individual* (lower or upper) threshold criterion and the vertical axis will be labeled zero to one.

Overlayed colors - This is useful only if both *Lower Percent* and *Upper Percent* are employed. If this box is not checked, the lower percent bars are solid black and the upper are hatched black. If this box is checked, the lower bars are red and the upper blue.

This axis labeling when the *Normalize Areas* box is checked means that the numeric value attached to bar heights has a very different meaning from that when the areas are not normalized. As stated above, when the histogram is normalized, each bar's height represents the concentration of cases in that bar relative to the total number of cases that meet the associated threshold criterion. *The numeric value associated with that bar height is its concentration relative to the bar having the highest concentration*. So for example, suppose a normalized bar has a height of 0.25. This means that the concentration of cases in this bar is one-quarter of the concentration of the most concentrated (i.e., tallest) bar.

One particularly useful application for the thresholded histogram is when we are filtering trades with a single variable such as an indicator or the predicted profit from a model. 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. Of course, the utility of this display is not limited to filtering systems. The histogram can display any target.

Another approach is to display only the extremes of the of the thresholded variable, for example the highest and lowest five percent. The histogram shown on the here uses the output of a model (MOD_10) as the thresholded variable to plot values of

the target, *ScaledProfit*. The lower (solid bars) and upper (hatched bars) five percent of predictions are used. Observe that when the model has a very large prediction, the target also tends to be large. The converse is also true. This is good!

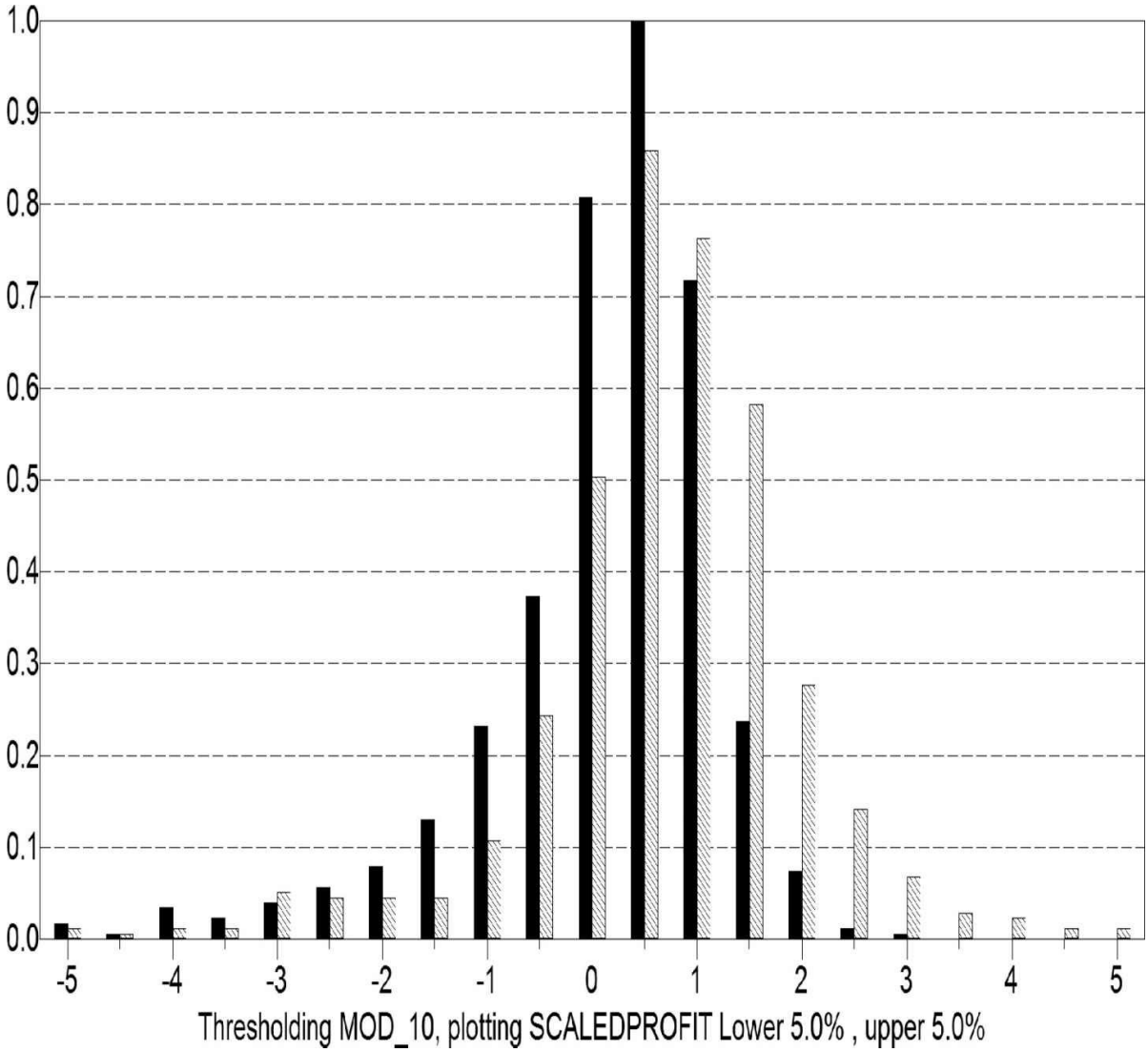


Figure 32: A thresholded histogram

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 a scatterplot 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.

A density plot overcomes this problem by using the gray level or color of the display to portray the *density* (or, optionally, measures related to the density) of the pair of variables at every point on the display. It does this by fitting a statistical smoothing window to the data in order to produce an estimated bivariate density. (Users interested in more details on this topic should search the internet for density estimation using Parzen windows.) Examples will appear later.

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.

Lower Limit - This comprises a check box and a field for entering a numeric value. If the box is checked, the user must enter a number in the field. This will be the lower (left or bottom) limit for the plot. Cases outside this range still take part in computations, but the plot does not extend to include them. Limits can be set separately for the horizontal and vertical variables.

Upper Limit - This is similar to the *Lower Limit* except that it specifies the upper (right or top) limit for the plot.

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 controls the width of the Parzen smoothing window, relative to the standard deviation of the data. 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 information 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 translate to grey levels or color on the display. Positive values will push the display toward yellow (or black for B&W display), and negative values toward blue (or white). Use this option to highlight desired features, experimenting as needed.

Tone spread - Like *Tone shift*, this has no effect on computation, but it controls how the density estimates translate to grey levels or colors on the display. Negative values are legal but rarely useful. Positive values, typically less than 10 or so, 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, experimenting as needed.

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

Histogram equalization - If this box is checked, the assignment of tone/color to each displayed value is done such that every possible shade is used in equal amounts. This makes details more visible. However, in many cases these details are mostly noise. Use of this option has the potential to seriously clutter the display with random noise.

Sharpen - Checking this box causes the display to highlight detail near extremes at the expense of detail in intermediate values.

Actual density - This, the default, plots the smoothed bivariate density of the two specified variables. A density plot is analogous to a scatterplot. The display defines a grid, with the value of one variable defining a position along the horizontal axis, and the value of the other variable defining a position along the vertical axis. Any pair of values for these two variables defines a point in the display, and the color or tone at that point depicts the concentration of data near that point. If the *Plot in color* box is checked, areas of high density, which are the result of

numerous cases clustering in that region, are colored yellow. Areas of low density (sparsely populated with cases) are colored blue. If the *Plot in color* box is not checked, low density is displayed as white and high density as black. The *Actual density* option is the easiest to understand and is recommended for most users. Note that financial data typically contains so much noise compared to predictive information that predictive relationships are difficult to see. This display is most useful for examining relationships between indicators or visualizing indicator/target relationships on a macro level.

Marginal density - This option is not generally useful, but it can be an interesting adjunct to the *Actual density* when they are compared. The display is identical that of the *Actual density* option, except that the densities plotted are not the actual bivariate densities. Rather, the tone/color at each point is determined by the product of the two marginal densities. (The marginal density of a variable is its probability distribution, considered alone, ignoring other variables.) Therefore, this plot shows what the actual density would look like if the horizontal and vertical variables had their observed univariate distributions, but there was no relationship between them. By comparing the *Actual density* to the *Marginal density* one can visually assess the degree of relationship between the horizontal and vertical variables. If the two plots are identical, chances are the two variables have little relationship. But if the *Actual density* shows a pattern not observed in the *Marginal density*, there probably is a significant relationship between the horizontal and vertical variables.

Inconsistency - A potential problem with scatterplots and their analogous *Actual density* plots is that these plots unavoidably include marginal densities in the display. For example, if one of the variables clusters at one extreme of its range, most of the cases will be plotted in a band there. Even if the two variables are unrelated, the map will show a possibly confusing gradient that may appear to show a relationship when none is actually present. What many people are most interested in is inconsistencies in the actual bivariate density *after taking marginal distributions into account*. We want to see where the actual density differs from the product of the marginals (the density that would be expected if the variables were not related). Visually comparing an *Actual density* plot to a *Marginal density* plot, as suggested earlier, reveals only large discrepancies. But by numerically comparing them, we can plot their inconsistency with great sensitivity. In this plot, areas which have a large value of the bivariate density compared to the product of the marginal densities are shown as black (for black-and-

white plots) or yellow (for color plots). Areas containing fewer cases than the product of the marginal distributions would predict are depicted as white (for B&W plots) or blue (for colored plots).

Mutual information - The mutual information between two variables is a sophisticated measure of how much their values are related. It is roughly analogous to a correlation coefficient, but it is totally nonlinear. It captures any sort of relationship, not just monotonic relationships. The mutual information between two variables is the sum of the mutual information in regions across their entire domain. The *Mutual information* plot shows by tone/color the contribution of each area to the total mutual information. Areas that contribute greatly to the mutual information are displayed as black (for B&W plots) or yellow (for color plots). Areas that contribute little mutual information are shown as white (for B&W plots) or blue (for color plots). This plot is useful only if there is significant mutual information between the variables. In a high noise environment, as is typically the case when one variable is an indicator and the other is a target, the mutual information is so small that the contributions of individual regions are mostly noise

In order to provide easily understood examples of these four types of density plot, a pair of variables having high negative correlation was used. These plots appear on the here.

Figure 33 is an *Actual density* plot of these two variables. The prominent yellow band shows that low values of the horizontal value are strongly associated with high values of the vertical variable, and vice versa.

Figure 34 is a *Marginal density* plot of the same data. Notice that the relationship between the two variables is gone, although they individually cover the same territory. If one compares this figure to the *Actual density* plot, the relationship between the vertical and horizontal variables is obvious.

Figure 35 is the corresponding *Inconsistency* plot. Naturally, the negative correlation is prominent. The brighter yellow at the corners indicates that the inconsistency is greater at the extremes than in the interior (mid-range), a common phenomenon.

Figure 36 is the *Mutual information* plot. It shows that cases along the band of high density are the primary contributors to the total mutual information.

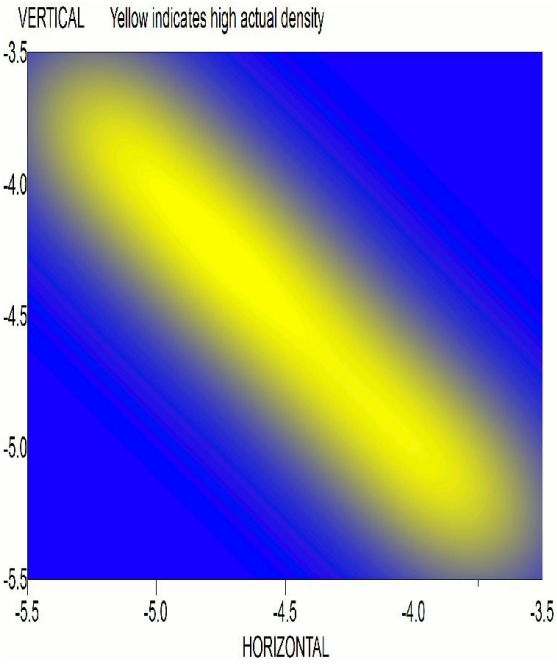


Figure 33: Actual density

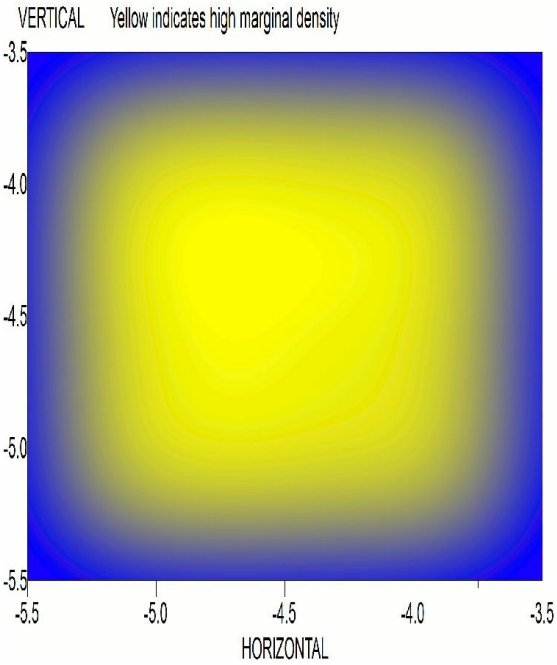


Figure 34: Marginal density

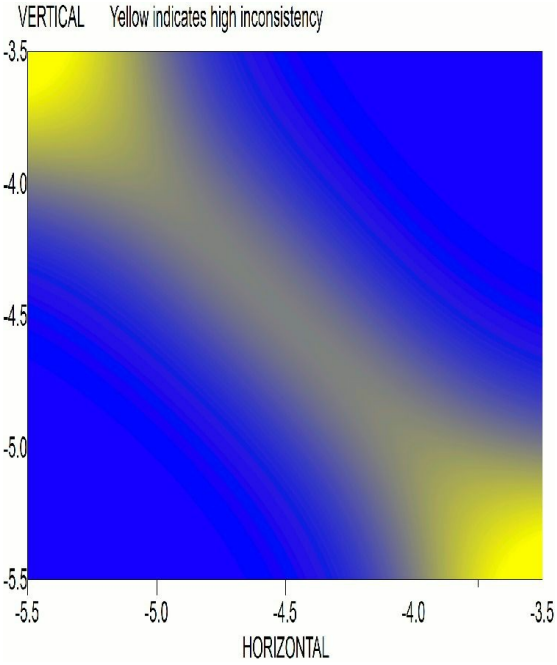


Figure 35: Inconsistency

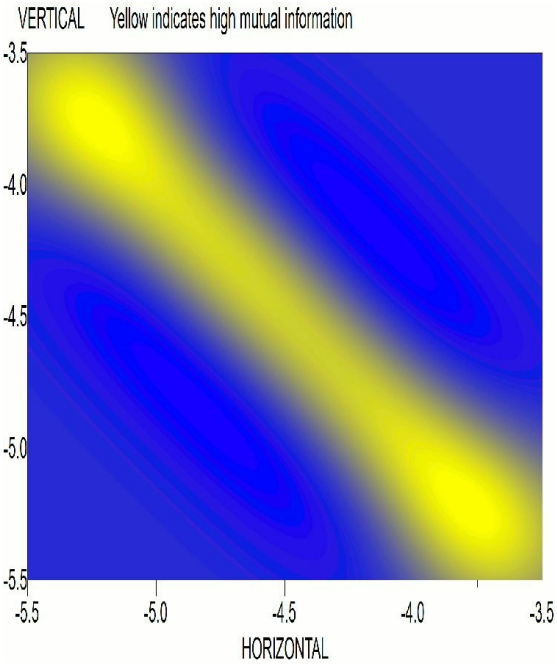


Figure 36: Mutual information

Figures 37 and 38 below employ actual data. Figure 37 is the *Actual density* of an indicator called PURIF_LOG and a forward-looking variable called RETURN. is a nearly flat ellipse, meaning that there is little relationship between the variables. If one looks closely, it is apparent that there is a slight upward tilt to it, so that larger values of PURIF_LOG tend to be slightly associated with large values of RETURN. However, the relationship is tiny. A*Marginal density* plot, not shown, looks similar, though without the slight tilt. A *Mutual Information* plot,

also not shown, is totally worthless. This is because the relationship between the two variables is so small that their mutual information across the display region is swamped out by random noise.

However, the *Inconsistency* plot, shown in Figure 38 below, reveals a fascinating bit of useful information. The bright yellow area in the upper-right corner shows that there is an unexpectedly large concentration of cases that simultaneously have very large values of PURIF_LOG and very large values of RETURN. These concentrations are all relative; this plot does not mean that such cases are common in an absolute sense, only that they are common *relative to what would be expected*. Such large values of PURIF_LOG are quite rare, as are such large values of RETURN. Because these values are rare, their simultaneous occurrence should be even more rare if they were unrelated. So what this plot is showing is that their simultaneous occurrence is not nearly as rare as would be expected if the two quantities were unrelated. This sort of event, in which the appearance of combinations of variables is relatively common *compared to what one would expect based on their individual distributions*, cannot be seen on an ordinary *Actual density* plot or its conventional analog, a scatterplot.

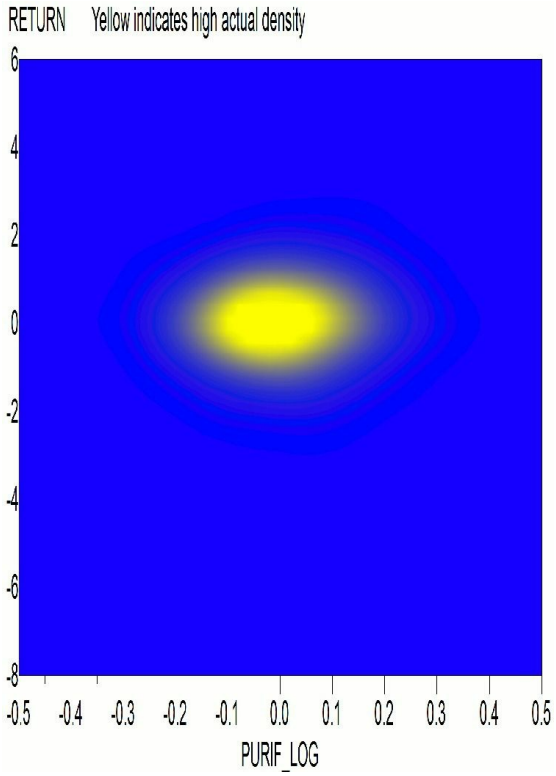


Figure 37: Actual density

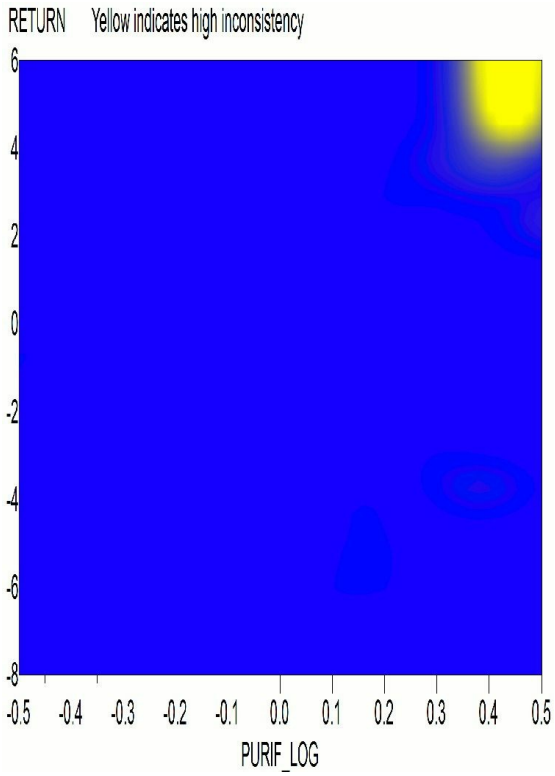


Figure 38: Inconsistency

Bivariate and Trivariate Plots

We saw in the prior section how the relationship between two variables (often an indicator and a target) could be displayed with a density map. It can also be useful to see how the value of one variable, usually a target or other measure of future market behavior, relates to *two* other variables, typically indicators. It may be that certain combinations of indicator values are associated with unusually large or small values of the target. Such patterns can be revealed with a bivariate plot.

Of course, paper and computer screens are two-dimensional, and with three variables we have a third dimension. One solution is to plot a 3-D projection (a visual representation) of the curved surface defined by the target as a function of the two indicators. But this can often obscure important details, depending on the viewpoint from which the smoothed function is observed. A more reliable method is to use black-and-white tone or color as the 'third dimension' of the plot.

Figure 39 below shows the dialog box by which the user enters the parameters for the plot. Some sample plots are shown on the here, and a discussion follows.

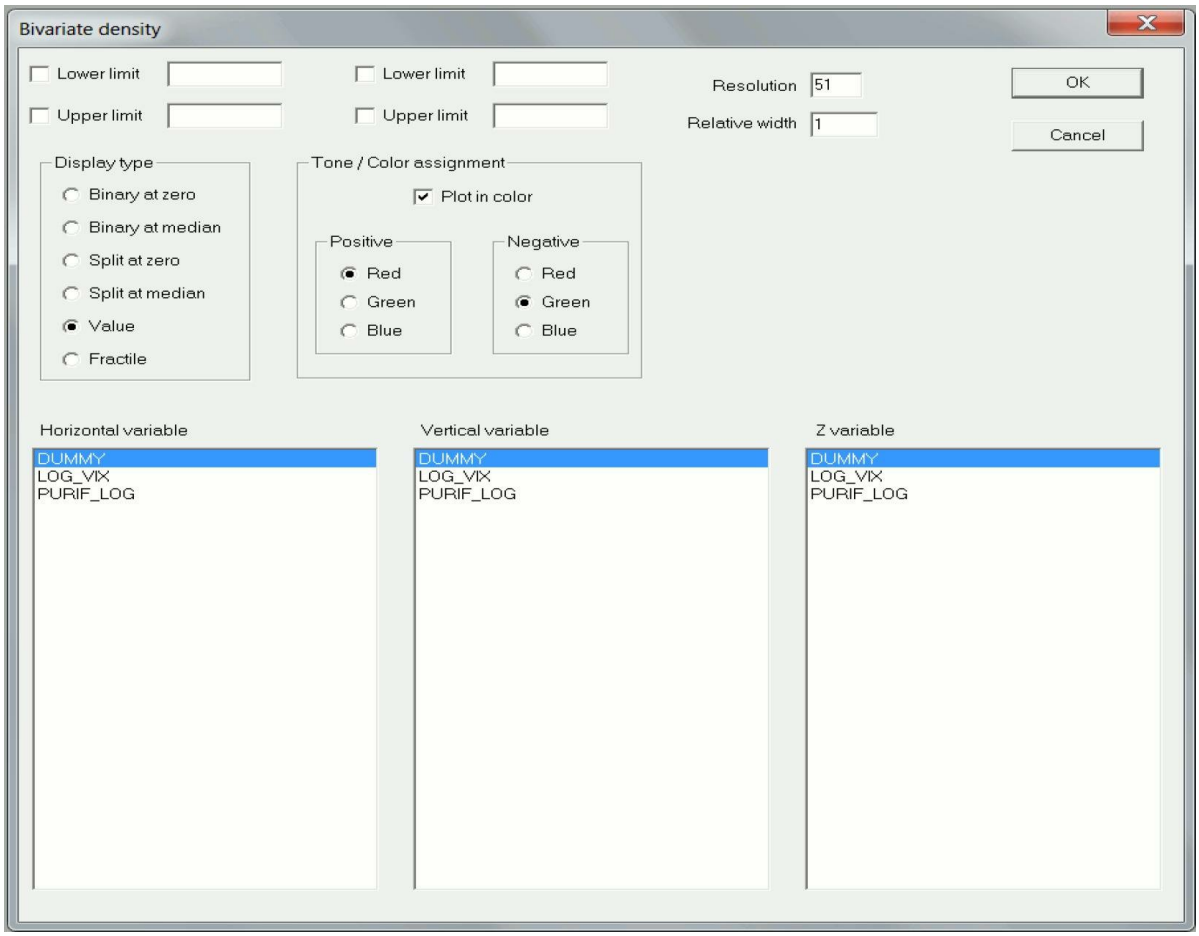


Figure 39: Entering parameters for a bivariate plot

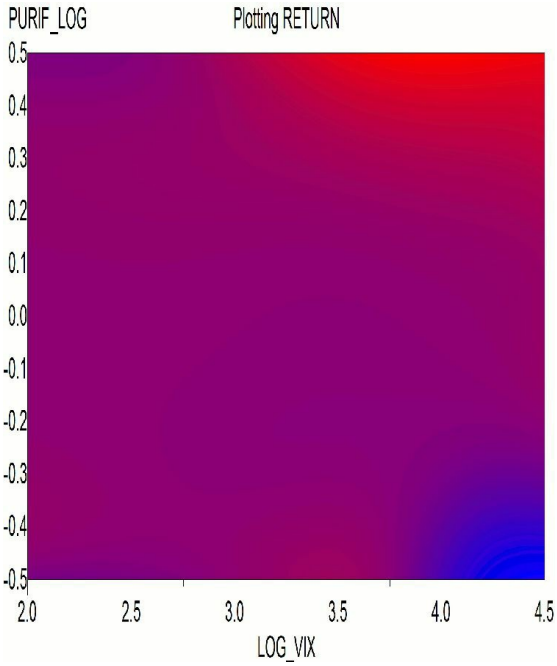


Figure 40: Value plot with width=1.0

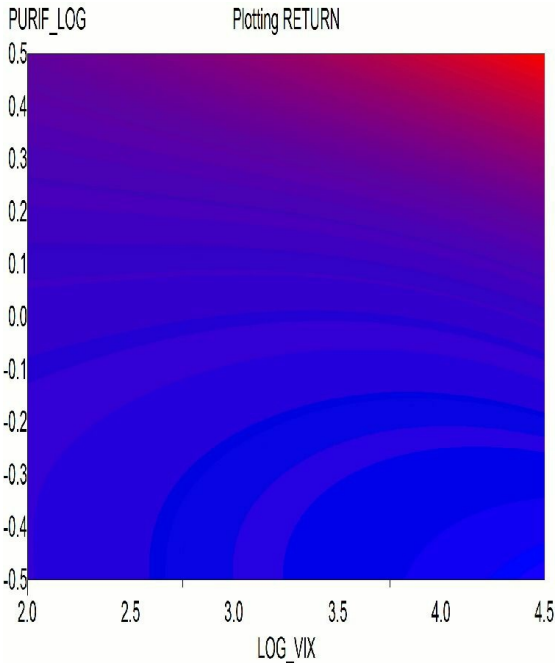


Figure 41: Value plot with width=2.0

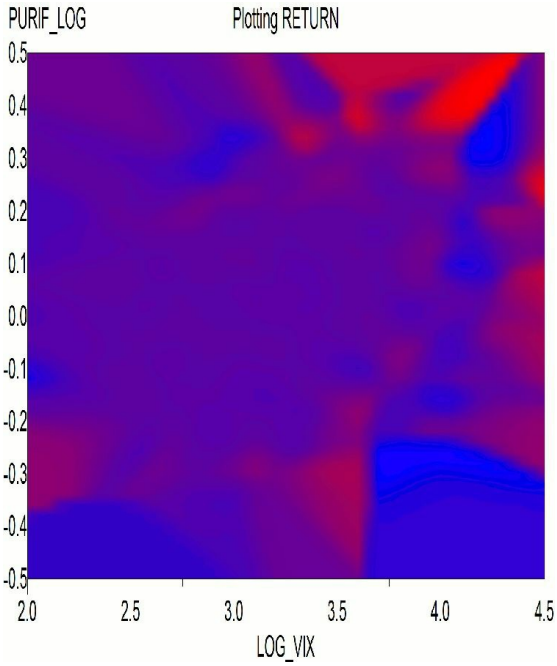


Figure 42: Value plot with width=0.2

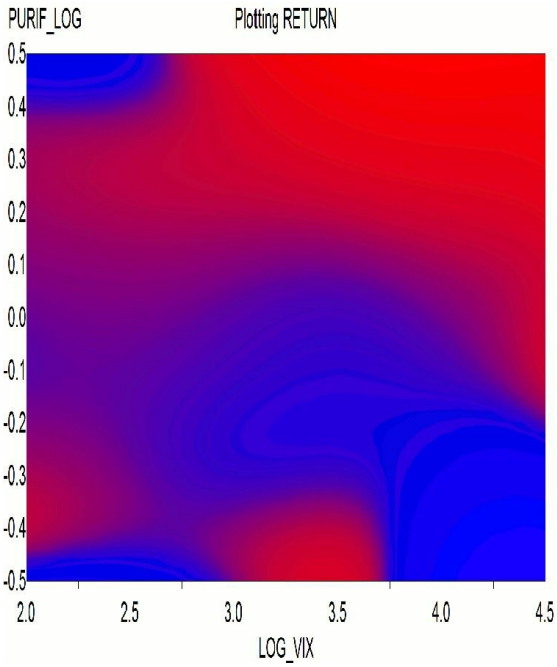


Figure 43: Fractile plot with width=1.0

Horizontal variable is the variable (usually an indicator) whose value lies along the horizontal axis. ***Vertical variable*** is the variable (also usually an indicator) whose value lies along the vertical axis. ***Z variable*** is the variable (usually a target or other measure of future market move) whose value is displayed by gray level or color.

If the ***Plot in color*** box is not checked, the minimum value of the Z variable will be

plotted as brightest white, and the maximum as darkest black. If this box is checked, the minimum value of the Z variable will be the brightest selected color for ***negative*** and the maximum will be the brightest ***positive*** color, and intermediate values will be dimmer shades. Central values of the Z variable will be shades of gray, regardless of whether the color box is checked. In keeping with engineering conventions and to avoid confusion for color-blind readers, all plots shown in this section are colored with red as positive (hot, or electrical positive) and blue (cold) as negative. Users who are more comfortable with the common financial convention of red for negative and green for positive may do so; in fact, this is the default, so no change is required. However, the user should be aware that many people are color-blind to red-green, and so this color combination, though the default, should be used with caution.

Resolution is the number of horizontal and vertical grid points at which the smoothed function is computed. Large values provide a better appearance but require more computer time.

Variations in the value of the Z variable are smoothed using a Gaussian window. (Users interested in more details on this topic should search the internet for *Parzen window*.) The width of this window relative to the standard deviation of the horizontal and vertical variables is specified with the ***Relative width*** parameter. The default value is 1.0, which sets the width equal to the standard deviation in each direction (indicator axis). Making the width a multiple of the standard deviation of each indicator prevents indicators with a smaller standard deviation from being overly smoothed. Larger values of the relative width result in more blurring (smoothing), which is good for diminishing the impact of random noise. Smaller values reveal more detail, which can be good in a low-noise situation but confusing if a lot of noise is present. Larger relative width values are would be analogous to a longer term moving average - more smoothing but greater loss of detail.

Display Type controls the basic nature of the display. The following types are available:

- ***Value*** is the simplest and, in a sense, the most 'honest' display type. This is because the full range of the Z variable is mapped to the full range of the display tone/color, and this mapping is done in a linear fashion. Figures here, 41, and 42 on the prior page demonstrate the *Value* option for three different widths. Notice that, as is commonly the case, the default relative width of 1 seems to be the best compromise between revealing detail while simultaneously blurring noise. Raising the width to 2.0 results in severe blurring, while lowering it to 0.2 fills the image with apparently random noise, a common situation in financial applications which have low

predictivity relative to noise.

- ***Fractile*** is a variation on the *Value* display method. As with *Value*, the full range of the Z variable is mapped to the full range of the display tone/color. But the *Fractile* display function is nonlinear, computed in such a way as to reduce the impact of extreme values of the Z variable on the display. Readers familiar with image processing and photo-editing programs will recognize this as *histogram equalization*: every tone/color appears on the display in equal amounts. This makes full use of the range of display tones/colors, which usually reveals much more detail than the *Value* option. Suppose, for example, that most data cases have Z values that cluster around a small range, but one or a few values are unusually large. The *Value* option would display these outliers at one extreme (such as green, if green were the selected positive color), while clumping the mass of 'usual' values in a narrow, unrevealing range of tones/colors (poorly differentiated shades of red if red were the negative color). The *Fractile* option, which spreads out such clumps across a wide range for better display, is shown in Figure 43 on the prior page. Note how this figure makes equal use of reds and blues, thus revealing more information, while the other figures on that page are dominated by one color.

- ***Split at zero*** is a variation on the *Fractile* option that uses the sign of the Z variable to choose the color. (This option is legal but pointless if the *Color* box is not checked.) The *Split at zero* option computes the fractiles of positive and negative values of the Z variable separately. The fractiles of negative Zs are displayed in shades of the chosen color, with the most negative values being depicted with brightest intensity. Similarly, positive values are displayed in the other selected color. For both signs, values of Z near zero are assigned such a dim tone of the appropriate color that they appear gray. In some cases it can be handy to know the sign of the Z variable in various regions of the indicator space. Neither the *Value* nor the *Fractile* display options allow this. An example of this display option is shown in Figure 44 on the next page.

- ***Split at median*** is similar to the *Split at zero* display option, except that instead of the split point being zero, it is the median of the Z variable's distribution. This is useful when one wants to split the fractile computation so that each color is processed separately and roughly equally, but the median of Z is not close to zero. An example of this display option is shown in Figure 45 on the next page. For example suppose blue is specified for Z values less than the median and red for Z values greater than the median. Cases near the median will appear nearly black. The further from the median the brighter will be the red or blue color.

- ***Binary at zero*** and ***Binary at median*** are similar to the split versions above, except that no gradations of color are done. Values of Z below the median (or zero) are one selected color, and values above are the other, both at uniformly bright intensity. If the *Color* box is unchecked, values below are white and those above are black.

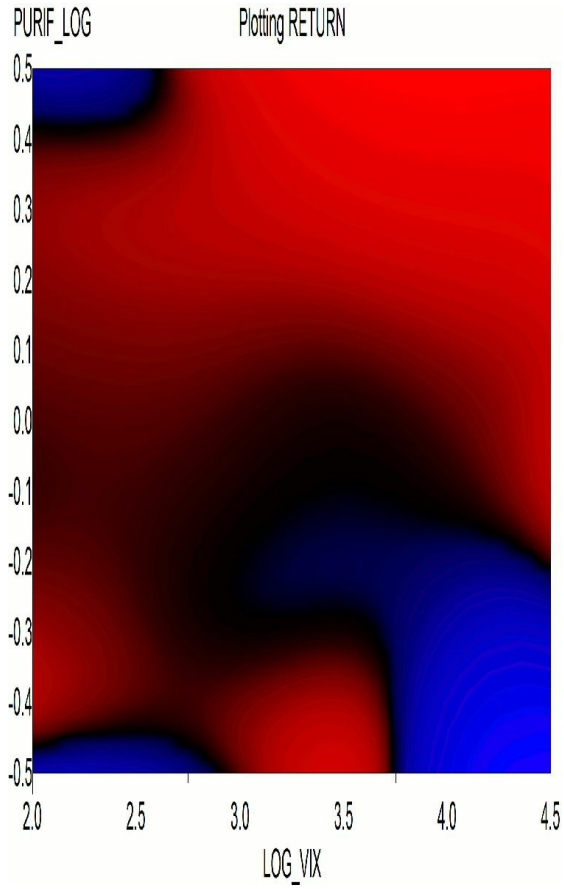


Figure 44: Split at 0

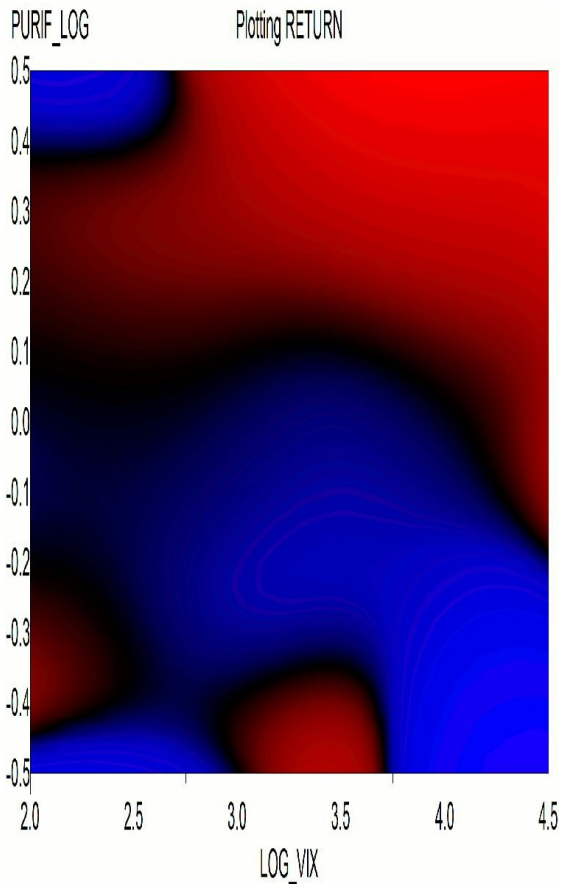


Figure 45: Split at median

Considerable experimentation with the various display types and smoothing widths may be required in order to discover a set of parameters that portrays the information in a clear manner. Indeed, sometimes certain features will be obvious with some display parameters and invisible with others. This is especially true with the *Value* display option compared to the others. The *Value* option is 'honest' in the sense that the mapping from Z to tone/color is linear. Thus, an extreme value in Z will be portrayed as a similarly extreme value of tone/color. This is good if you want such extreme values to be prominent. However, the price paid for this honesty is high: the remainder of the Z variable's distribution will have only a narrow range of tones/colors with which to be displayed. Detail in the bulk of the distribution will be obscured.

Finally, as with most plot functions in *TSSB*, the user has the ability to specify display limits for the variables. By default, the limits are computed automatically according to the actual range of the data. If the user checks any of the four *limit* boxes, the corresponding specified limit(s) will override the default choice. The left pair of limit blocks applies to the horizontal variable, and the right pair applies to the vertical variable.

Trivariate Plots

A trivariate plot expands a bivariate plot by bringing in a third indicator. This third variable is used to slice a cross section in the relationship defined by three indicators and the target. One can think of a trivariate plot as a bivariate plot that is conditional on a third indicator. The two bivariate indicators may relate to the target in one manner when the third variable is large, and in another manner when the third variable is small. A trivariate plot lets us visualize such relationships.

An example of this cross-sectioning process is shown in Figure 46 below. In this figure, each layer is a bivariate plot showing how PureVIX and ADX relate to a forward-looking target variable. However, the nature of this relationship depends on the current trend. So we see that the layers corresponding to each trend are different. A trivariate plot lets us examine a single such layer.

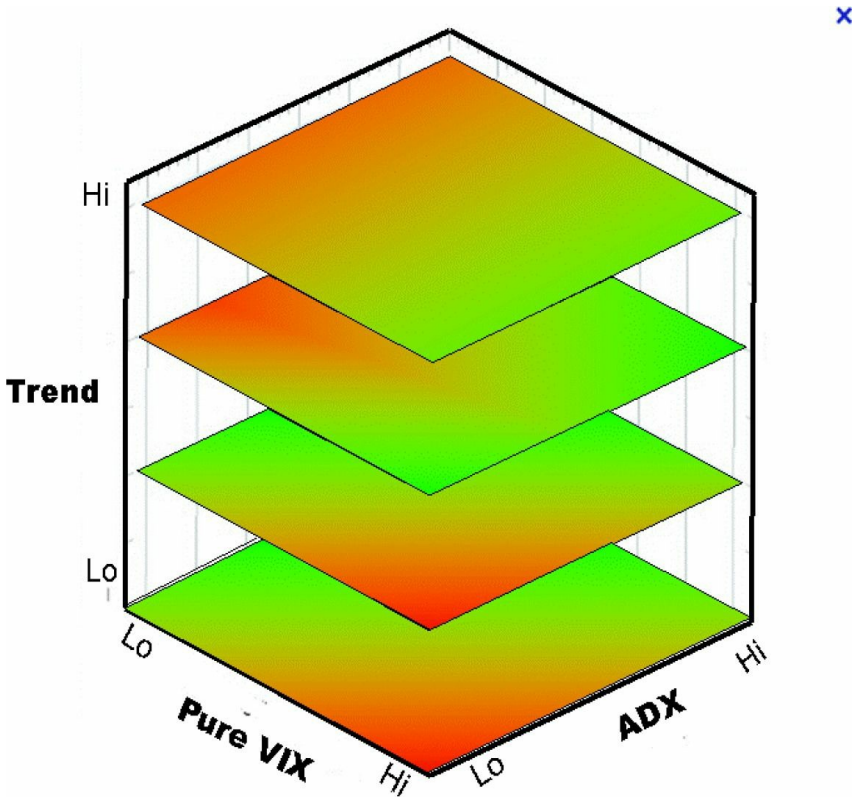


Figure 46: A trivariate plot in which PureVIX and ADX are cross-sectioned by Trend

The parameters for the trivariate plot are those for a bivariate plot plus three more. The user must specify the variable that is used for cross-sectioning, the value of this cross-sectioning variable, and the relative width of the cross-sectional slice. If this cross-sectional width is very narrow (much less than 1.0) and the dataset contains few cases in the vicinity of the specified value for the cross-sectional variable, the plot may be noisy and distorted. On the other hand, if this width is very large (much greater than 1.0), a good cross-sectional representation may not be obtained,

because too much of the volume will be blurred together. Some experimentation may be required to obtain the best display.

Figures 47 and 48 below depict the same relationship, with the same parameters, as the bivariate plot in Figure 43 here. However, Figure 47 shows this relationship when the indicator called DUMMY is near the value 30, while Figure 48 shows the relationship when DUMMY is near -30. Notice the significant difference especially at the top-right!

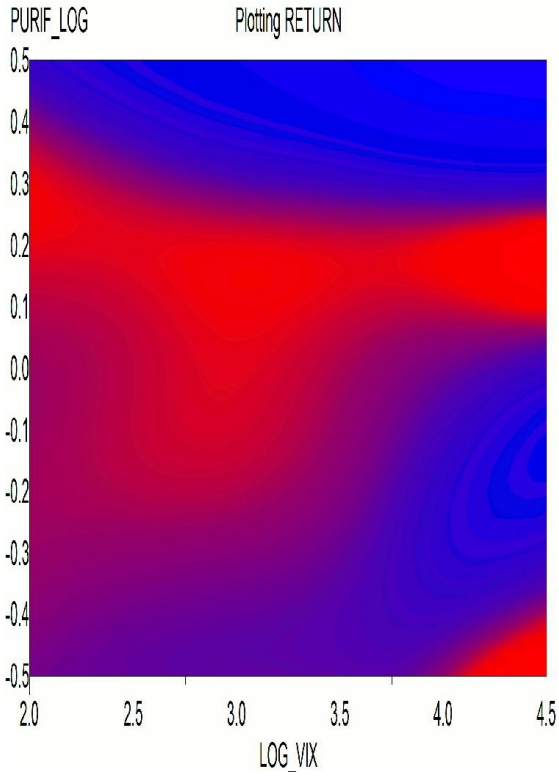


Figure 47: Plot with DUMMY near 30

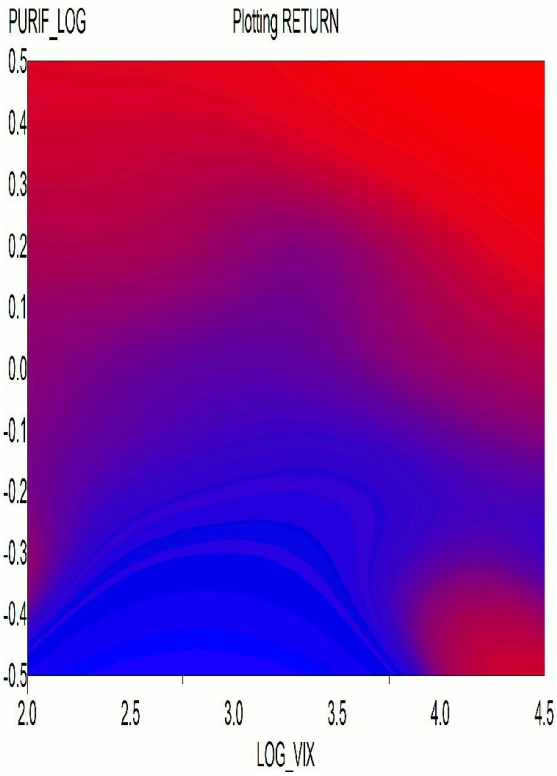


Figure 48: Plot with DUMMY near -30

Equity

If the rulebase contains one or more Models, Committees, Oracles, or Portfolios and a TRAIN, WALKFORWARD, or CROSS VALIDATE command has be executed, the equity can be graphed. If a WALK FORWARD or CROS 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. This is done by selecting one of the three choices shown in the upper-right corner of the Plot Equity dialog shown in Figure 49 below. 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.

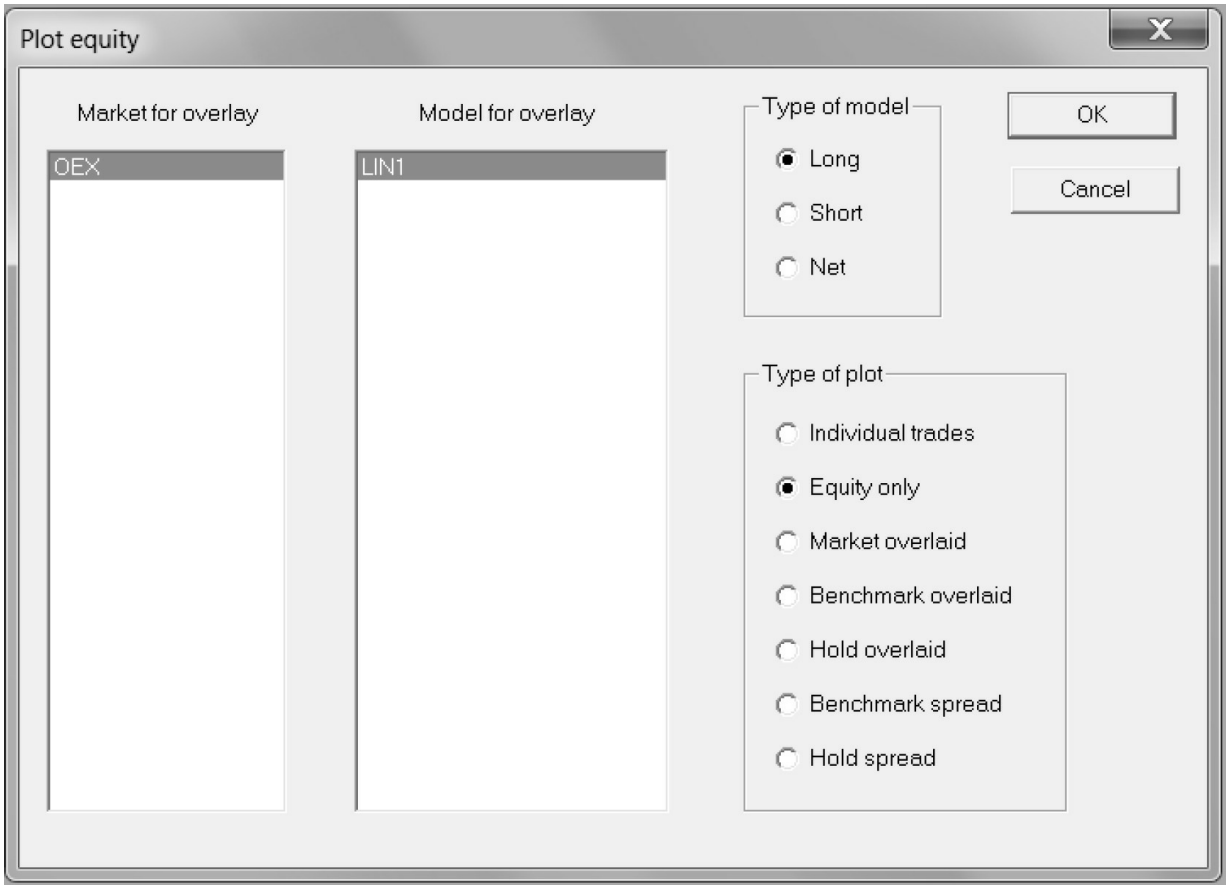


Figure 49: Dialog box for plotting equity

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. An example of a *Benchmark Overlaid* plot is shown in Figure 50 here.

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 a 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.

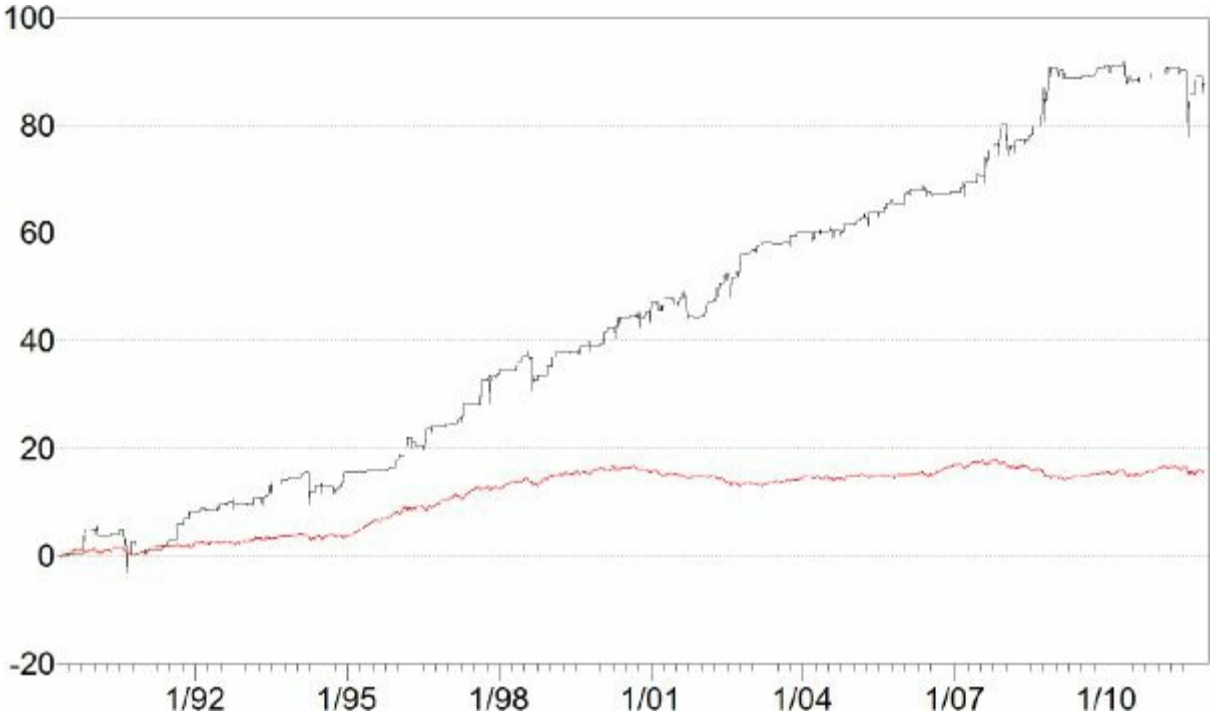


Figure 50: A *Benchmark overlaid* plot

Prediction Map

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

Beyond this, it can help the user distinguish between nonlinear 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. (Of course, this applies only to nonlinear models such as neural networks that are capable of such clumping.)

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 if you plan on displaying a prediction map. Otherwise, the map will be based on only the most recent fold, which is unlikely to be what the user wishes.

The user can set the following options:

Model - This lists all models that have two inputs. The user selects the desired model.

Horizontal Variable - The user selects which of the two inputs will be mapped along the horizontal axis.

Vertical Variable - The user selects which of the two inputs will be mapped along the vertical axis.

Lower Limit - This comprises a check box and a field for entering a numeric value. If the box is checked, the user must enter a number in the field. This will be the lower (left or bottom) limit for the plot. Limits can be set separately for the horizontal and vertical variables.

Upper Limit - This is similar to the *Lower Limit* except that it specifies the upper (right or top) limit for the plot.

Resolution - 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.

Show Cases - If this box is checked, training cases are 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.

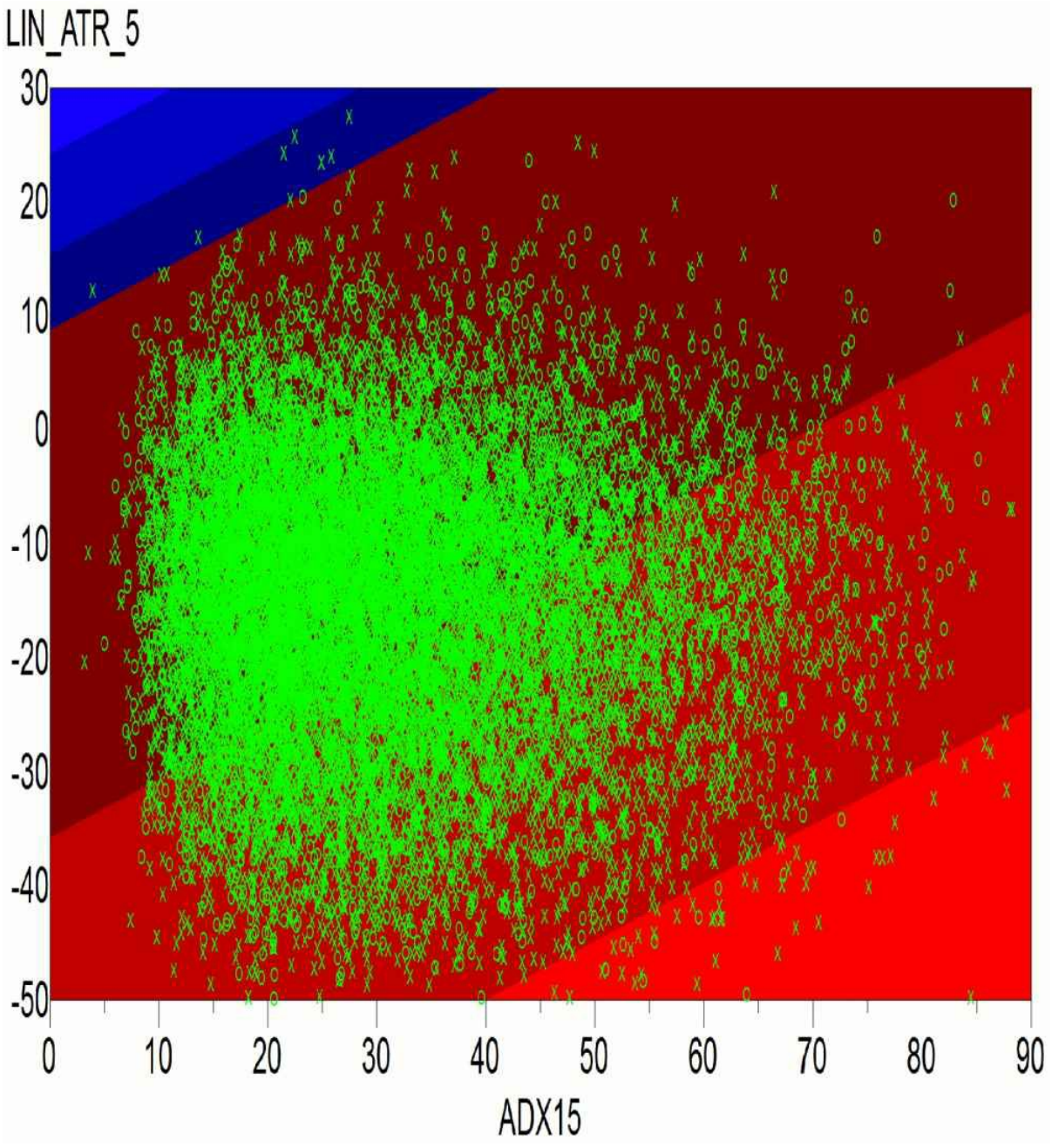
Multiple Tones - If this box is not checked, two colors will be used: red for positive predictions and blue for zero or negative predictions. By checking the *Multiple Tones* box, three levels of shading for each color will be used. Red will still be used for positive predictions, and blue for negative and zero. But the brightest red and blue will be reserved for the most extreme ten percent of predictions of that sign. The dimmest version of each color will be used for predictions below the median (in absolute value). Intermediate brightness will be used for the remainder (values between the median and the 90'th percentile). But see the next option, which impacts color assignment.

Split at Median - 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 profitable trades that most or all predictions are positive. When this box is checked, the split for color assignment will occur at the median of the target rather than at 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.

A typical prediction map is shown on the here. This is for a model that filters a decent long-only trading system. Because the trading system has many more wins than losses, the model is biased toward predicting positive trades (red), so the user may wish to check the *Split at Median* box to equalize colors. (It was not checked for this display.)

This map reveals some interesting things about operation of the model. Larger values of ADX15, a 15-bar ADX measure, result in predictions of more profit. At the same time, larger values of LIN_ATR_5, a 5-bar trend indicator, result in

predictions of less profit. Thus, we see that the filtering model is heavily based on short-term counter-trend; if the market has a strong short-term up-trend, the filtering model is likely to abort a trade recommended by the external trading system. This is especially true if ADX15 is small. Interesting.



Indicator-Target Relationship

TSSB offers several algorithms for relatively quickly screening batches of indicators to assess their degree of relationship to a target. The chi-square test ranks indicators based on their individual relationships to the target, and the nonredundant predictor screening test 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:

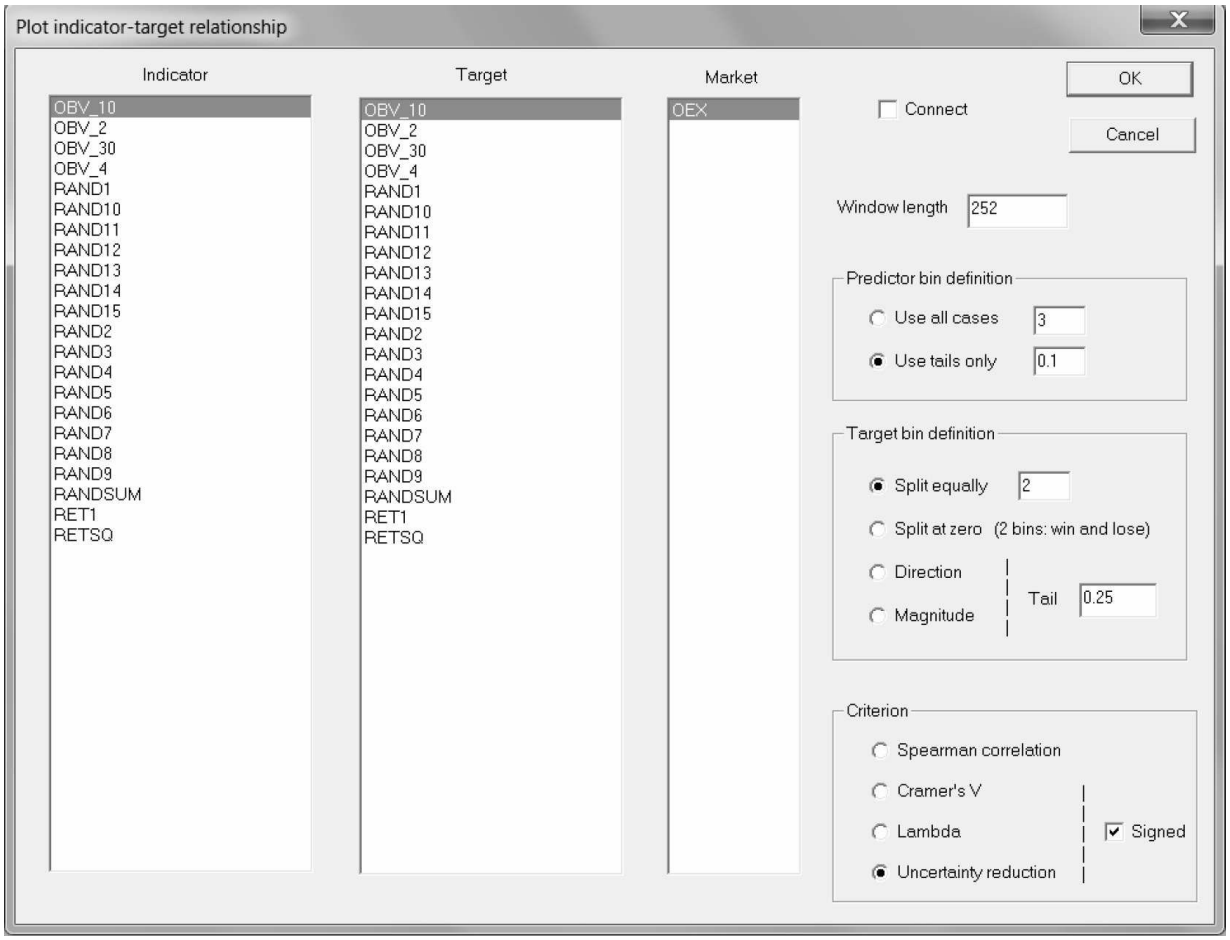


Figure 52: Dialog box for Indicator-Target relationship

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 approximately 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. However, this can lead to display anomalies as discussed in the examples to follow. The last two options (Direction and Magnitude) will be discussed in the context of anomalies later in this section.

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).

Lambda - A contingency-table-based measure of nominal (category) correlation which ranges form 0 (no correlation) to 1 (perfect correlation).

Uncertainty reduction - A contingency-table-based measure of nominal (category) correlation which ranges form 0 (no correlation) to 1 (perfect correlation). 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 by examining the relationship between the most extreme bins of the predictor and target. 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.

Figures here and here on the next page show examples of these plots, and they are discussed on the following page.

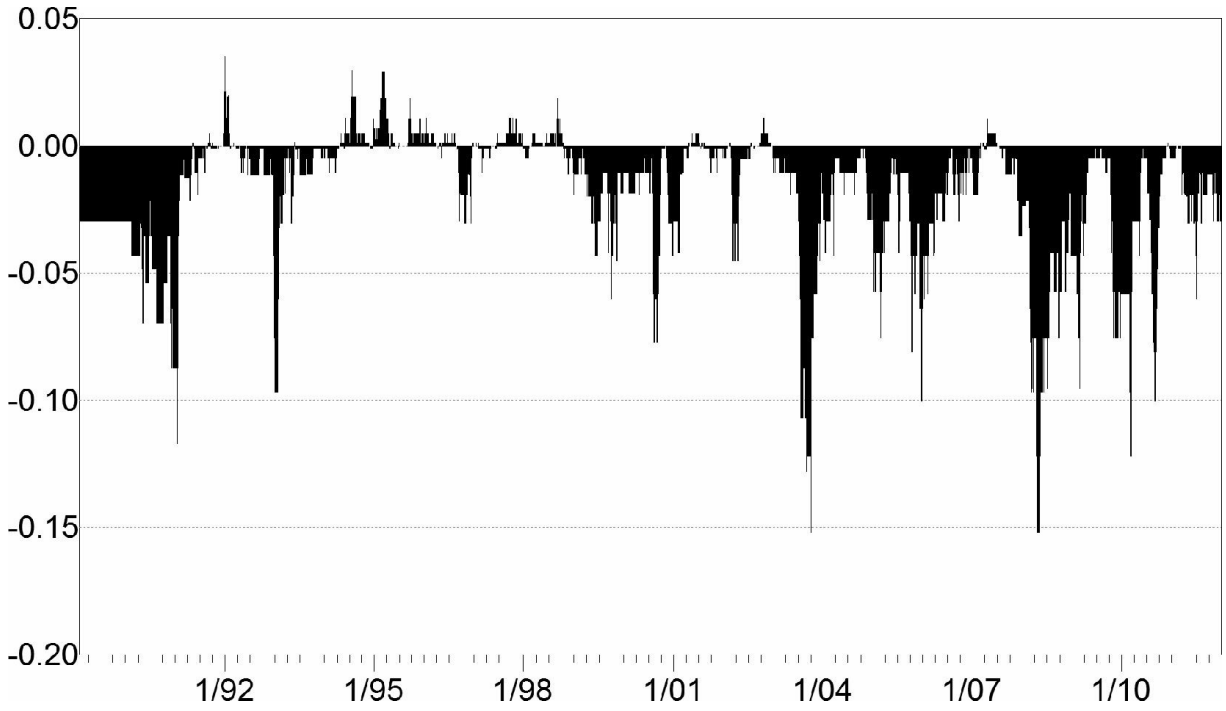


Figure 53: Signed uncertainty reduction with 2 target bins

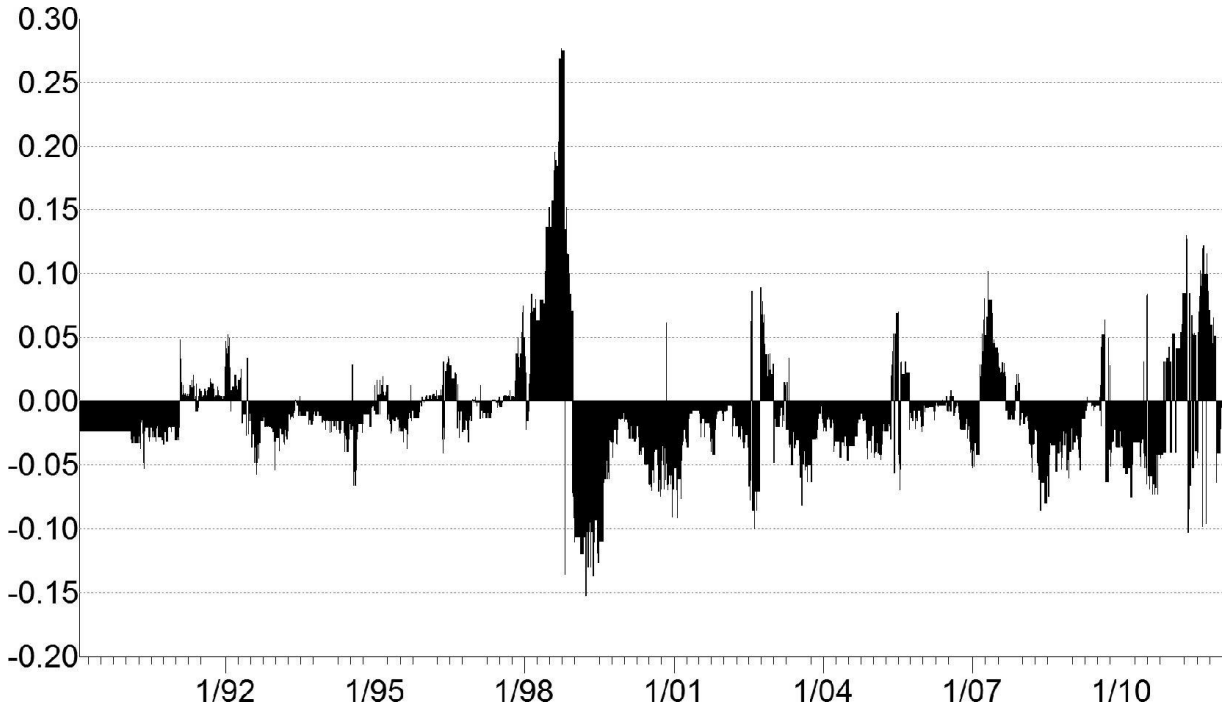


Figure 54: Signed uncertainty reduction with 3 target bins

Figure 53 on the prior page shows the signed uncertainty reduction based on two target bins. The degree of predictability is not very large, but probably usable because it does have a valuable trait: it almost always has the same sign. This means that the relationship between this indicator and target, though fairly small, is nicely consistent. This sort of consistency would not be seen if there were no legitimate relationship between them, and a well-designed trading system should be

able to take advantage of this relationship.

Figure 54 shows the same plot with one difference: there are three target bins instead of two. The plot does not look nearly as good, because in late 1998 the indicator-target relationship spikes and then shockingly switches sign! What's going on?

The answer is that for much of 1998 (and several other epochs) the indicator loses its ability to discriminate between large wins and large losses. The indicator-target relationship becomes small and temporarily switches to positive. Look at Figure 53 to see this. So why does the uncertainty reduction hit nearly 0.3 during this time, a huge value? It's because during this time the indicator becomes extremely powerful for discriminating between an upcoming flat market versus a large upcoming price change. It cannot say whether this large price change will be up or down, but it can say that it's coming. Thus, during most of 1998 the plot shows the huge predictive power of the indicator, but because its ability to determine the direction of the future move is positively correlated (though very small!), the plot's sign reverses right in the middle of this highly predictive period.

This sort of anomaly is the motivation for the two options discussed in the next section.

Isolating Predictability of Direction Versus Magnitude

We just saw that using three target bins can produce an anomalous display when the indicator's ability to predict the magnitude (absolute value) is large but its ability to predict the direction of the move is small. One can obtain a plot that shows high predictability, but the sign of the plot flips in ugly and confusing ways, even showing narrow reversed spikes in the midst of a mountain of predictability. The cause of this phenomenon can be visualized by using the *Direction* and *Magnitude* target bin options.

Both of these options require that the user specify a tail fraction, greater than zero and less than or equal to 0.5. For the *Direction* option, the graph displayed is the indicator's ability to predict whether the target is strongly positive or strongly negative. This is computed by examining the specified upper and lower tails, ignoring the central values. This provides two target bins, and the algorithm measures how well the bin of the indicator provides predictive information about the bin (upper tail versus lower tail) of the target. If the tail fraction is specified to be 0.5, this option is equivalent to selecting 'use all cases' and setting the number of bins to two.

The *Magnitude* option is slightly more complex. The entire distribution is used. The two tail bins (upper and lower) are pooled into a single bin, and this is contrasted with the center bin (all cases that do not lie in a tail). Thus, we have two target bins: center and (combined) tail. The algorithm measures how well the indicator bin provides information about whether the target lies in the center bin versus the tail bin (which encompasses both the left and right tails).

The utility of this can be seen by reviewing Figure 54 and then examining Figures here and 56 on the next page. Note from Figure 54 that the anomalous area occurs late in 1998. In Figure 55, which shows the ability of the indicator to predict the direction (unusually up versus unusually down) of upcoming market movement, we see that during this time period the indicator has very little power to predict the direction. But Figure 56 shows that during this same time period, the indicator has strong ability to predict the magnitude of the target. This is the power that appears in Figure 54 as a result of using three target bins, but the poor power to predict the direction of the market causes the sudden reversal in the plot.

Note that this indicator has strong and quite consistent power to predict the magnitude of the target. The fact that it is negative almost everywhere means that the relationship between the indicator and the magnitude of target is inverse: large values of the target imply small upcoming market movement.

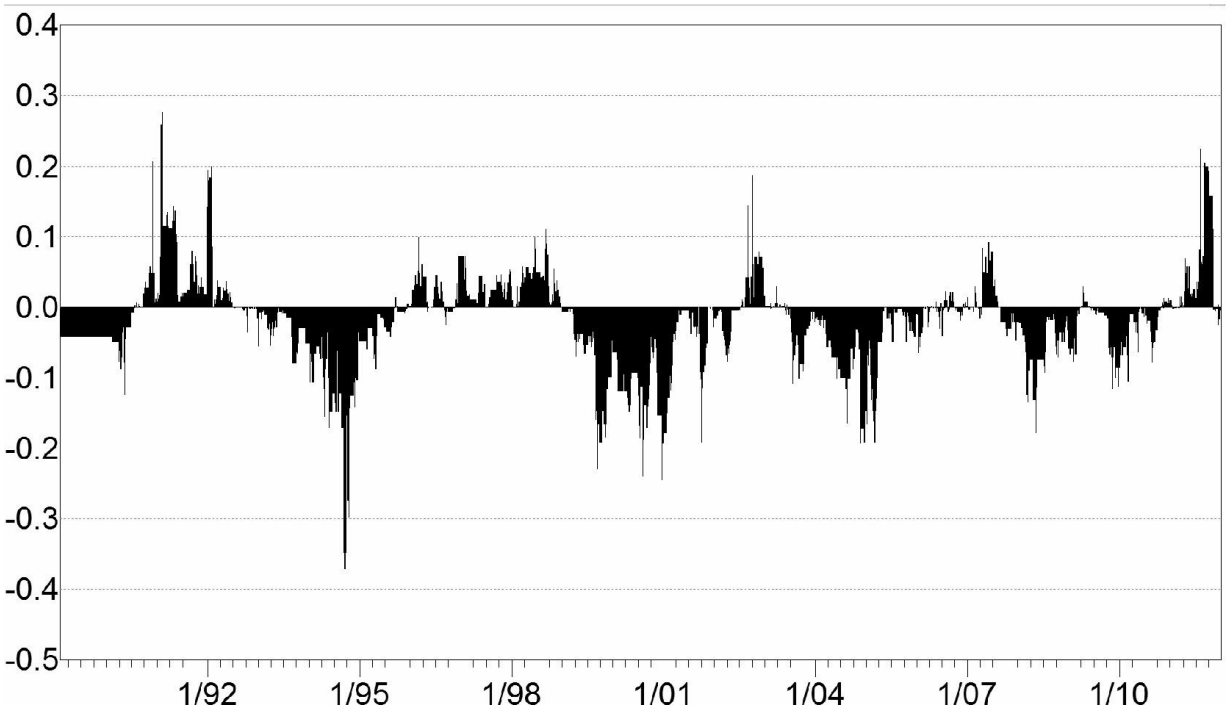


Figure 55: Signed uncertainty reduction showing power to predict direction

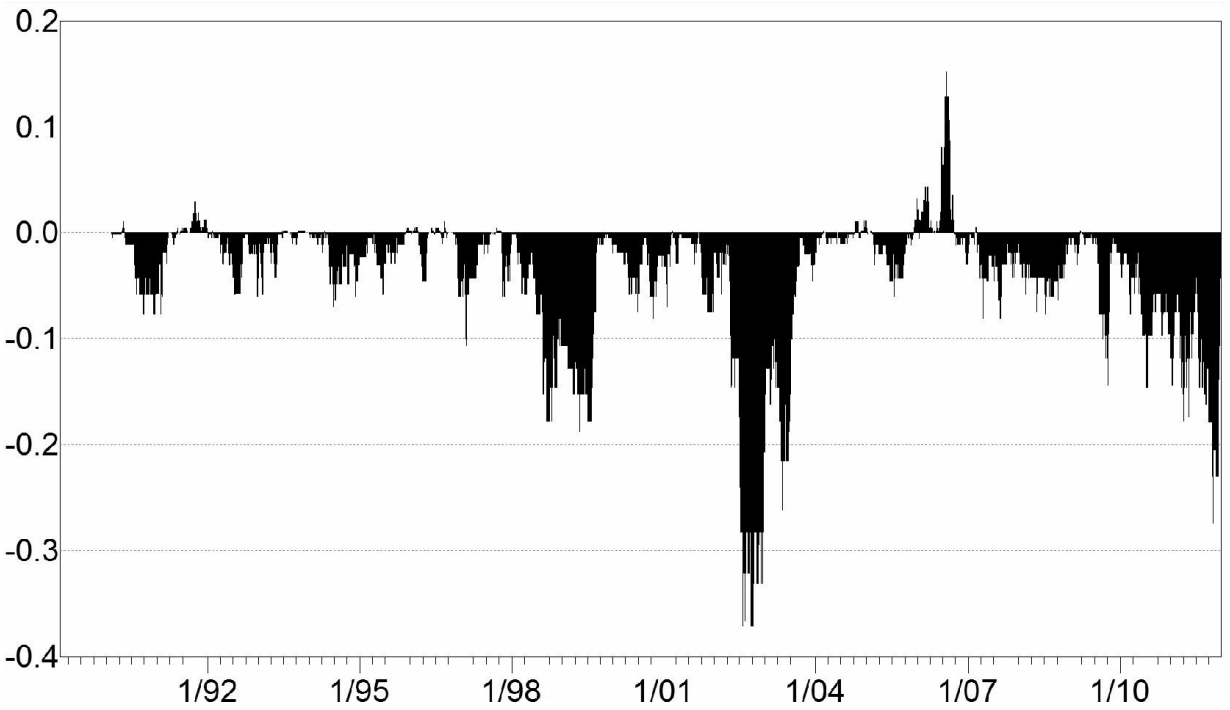


Figure 56: Signed uncertainty reduction showing power to predict magnitude

Finding Independent Predictors

here of the *Committees* chapter we saw many methods for creating models whose predictions will be combined using a committee. It is usually best to create these models in an intelligent, manual way. However, this may not always be possible, or even required. Sometimes we are well served by an automated approach. The FIND GROUPS command is a useful way of automating the discovery of model that can be effective committee members.

The idea is that we find small sets of predictors (typically 2 or 3) which, taken as a group, have power to predict a target variable. Moreover, the members of each predictor group are to some degree independent of the members of the other predictor groups, thus avoiding serious redundancy.

The algorithm to accomplish this is simple. The list of predictor candidates is examined and a linear model is found that optimally predicts the target variable based on a few indicators. 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 to the set just found and eliminated from future contention. An optimal linear model is found by testing the remaining candidates. 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  ] ;
```

Most of the specifications for this command are identical to the similarly named specifications for models. Rather than repeat detailed descriptions that appear elsewhere in this document, wewill just provide a brief summary of each command and then give a page reference for users who want more details.

INPUT = [Var1 Var2 ...] - Lists the input variables that are candidates for inclusion. Range and family options are also available. See here.

OUTPUT = VarName-Names the variable that is to be predicted, the target. See here.

PROFIT = VarName- If PROFIT is not specified here, the OUTPUT variable which is the target being predicted, is also used for computing profit factor performance measures. However, if the target variable is not interpretable as a profit, a profit variable can be named with this optional specification. See here.

MAX STEPWISE = Number - The maximum number of predictor variables that may be included in each group. This must be specified. See here.

STEPWISE RETENTION = Number Optionally specifies the number of 'best' candidate predictor sets kept at each stage of stepwise selection. See here.

FRACTILE THRESHOLD - Optionallydecrees that, in a multiple-market situation, predicted values are normalized fractiles across markets instead of actual predicted values. See here.

CRITERION = Criterion - Names the criterion that is used to select optimal predictors. This may be any of the model optimization criteria, such as RSQUARE, PROFIT FACTOR, and so forth. See here.

MIN CRITERION CASES = Integer Specifies the minimum number of cases that must meet the optimal threshold. See here.

MIN CRITERION FRACTION = RealNumber Specifies the minimum fraction of cases that must meet the optimal threshold. See here.

RESTRAIN PREDICTED - Causes the values of the target variable to be compressed at the extreme values before training. See here.

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. Specifying 1.0 will effectivelyr emove all redundancy checks, which will maximize the number of groups that can be found, at the price of possibly serious redundancy. Specifying a value near zero will quickly eliminate candidates with even tiny redundancy. The result is that few groups will be possible, perhaps only one, though the groups that are able to be found will be highly independent.

MAX GROUPS = Number - The maximum number of groups that can be found. This maximum may not always be reached. Setting this to a huge value will result in all possible groups being found, though run time may be excessive.

Both AUDIT.LOG and REPORT.LOG contain detailed descriptions of each grou as it is defined. In addition, AUDIT.LOG lists the R-square of each candidate witl 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 ratio 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.

A FIND GROUPS Demonstration

We now consider a typical use of the FIND GROUPS command. This example wil use the command for two purposes: to judge the relative importance of predictors in a very large list of candidates (almost 300!), and to create reasonably independent models that will be combined via a committee. Here is the command that finds the groups:

```
FIND GROUPS [
  INPUT = [ CMMA_5 - VMUTINF_3 ]
  OUTPUT = DAY_RETURN_1
  MAX STEPWISE = 3
  STEPWISE RETENTION = 4
  MAX GROUPS = 5
  GROUP RSQUARE CUTOFF = 0.9
  CRITERION = PROFIT FACTOR
  MIN CRITERION FRACTION = 0.1
  RESTRAIN PREDICTED
  ] ;
```

Since one of our goals is to rank the utility of predictor candidates, we set STEPWISE RETENTION to 4, a small fraction of the total number of candidate There is nothing special about 4; any value in that neighborhood will do. In fact, it might be useful to try several nearby values and compare results. Also, we set MAX GROUPS to 5, the actual number of models we will later use in th committee. This guarantees that the candidate rankings will reflect this exact number of models and will ignore candidates that would only appear in later groups if MAX GROUPS were larger.

The audit log file for the FIND GROUPS command is extensive, so we will her show only select portions. The first model found is as shown on the here.

```
----------> Group 1 <----------

LINREG Model GroupFinder predicting DAY_RETURN_1
Regression coefficients:
          0.001267  CMMA_5N
          0.001775  QUODDEV_5
          0.000842  DAU_STD_32_2
          0.040350  CONSTANT

MSE = 0.28999   R-squared = 0.00439   ROC area = 0.54209
Buy-and-hold profit factor = 1.160   Sell-short = 0.862
Dual-thresholded outer PF = 1.500
   Outer long-only PF = 1.449   Improvement Ratio = 1.249
   Outer short-only PF = 1.567  Improvement Ratio = 1.818
```

After the model is shown, the log file lists the remaining candidates in the order in which they were defined or they appeared in the database, and the R-square for each of them relative to the model just discovered. Any whose R-square exceeds the user-specified elimination threshold are flagged. Here is a small subset of this output:

```
R-square of remaining candidates with group just created...

          CMMA_5    0.898
         CMMA_10    0.686
         CMMA_20    0.404
        CMMA_10N    0.769
        CMMA_20N    0.482
    DAU_NRG_32_1    0.672
    DAU_NRG_32_2    0.919
 Removing DAU_NRG_32_2 with R-square = 0.919
    DAU_NRG_32_3    0.391
    DAU_NRG_32_4    0.108
```

The remaining four models are similarly presented. Then, a selection summary is printed. If the candidates were imported from a different program via a database file, then only the variables themselves can be summarized. However, if the variables were computed internally by the *TSSB* program, more extensive summaries will be presented. The first such summary is the family, and a subset of this output is shown on the here. It is sorted in descending order of the ratio of the percent of times the variable was selected to the number of times that would be expected if all candidates were equally predictive.

```
--------------------------------
FIND GROUPS selection summary...
--------------------------------

Total best variables = 120

Families selected...

                Name   Percent  Expected    Ratio     Prob

  QUADRATIC DEVIATION  15.00      1.08   13.9500    0.0000
     LINEAR DEVIATION  12.50      1.08   11.6250    0.0000
     IMAG DIFF MORLET   9.17      1.08    8.5250    0.0000
      CUBIC DEVIATION   5.83      1.08    5.4250    0.0000
  IMAG PRODUCT MORLET   5.83      1.08    5.4250    0.0000
            ACCEL ADX   3.33      0.72    4.6500    0.0003
             DAUB STD  14.17      3.23    4.3917    0.0000
```

In the table shown above, we see for example that a member of the QUADRATIC DEVIATION family (seehere) was selected in a ‘best’ intermediate or final group

15.00 percent of the time during stepwise selection for the five groups. If all candidates had equal predictive value, a QUADRATIC DEVIATION predicto would have been selected on average just 1.08 percent of the time. So it was selected 13.95 times more often than expected. If all candidates were equally predictive, obtaining such over-representation by random chance has a probability of zero to at least four decimal places.

The next summary printed is by lookback length. Here is the beginning of this table:

```
Lookbacks...

   Value   Percent  Expected    Ratio     Prob

       5   17.50       7.17    2.4413   0.0000
      32   21.67       9.68    2.2389   0.0000
      16   10.83       7.17    1.5113   0.0598
      20   25.83      19.00    1.3599   0.0281
      10   14.17      11.11    1.2750   0.1434
      15    5.00       6.81    0.7342   0.2156
```

We see that candidates having a lookback distance of five days were part of a 'best' trial group17.5 percent of the time, when we would expected a rate of 7.17 percent. However, it is important to avoid reading too much into this table unless the candidates were carefully chosen to balance families and lookbacks. For example, it may be that many of the best families featured lookbacks of five days. If this happened to be the situation, the results are really determined by the families, and the lookbacks are just a byproduct of how the user defined the candidate variables.

Tables based on index usage (here) and historical normalization (here) are printed next. Finally, we see a table showing the individual candidate predictors. Here are the first few lines from this table. Note that even though the expected usage is the same for all variables, it is still printed for clarity and conformity with the other tables.

```
Variables selected...

          Name    Percent  Expected  Prob

    QUODDEV_10     11.67     0.36   0.0000
      LINDEV_5     11.67     0.36   0.0000
    IDMORLET_5      9.17     0.36   0.0000
  DAU_STD_32_3      7.50     0.36   0.0000
    CUBEDEV_20      5.83     0.36   0.0000
    IPMORLET_5      5.83     0.36   0.0000
  DAU_STD_32_2      4.17     0.36   0.0000
  DAU_NRG_32_3      4.17     0.36   0.0000
```

The user may then create a script file that employs the models discovered with the FIND GROUPS command and that combines the model outputs via a committee. Here is such a script file:

```
MODEL MOD_1 IS LINREG [
   INPUT = [ CMMA_5N QUODDEV_5 DAU_STD_32_2 ]
   OUTPUT = DAY_RETURN_1
   MAX STEPWISE = 0
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   RESTRAIN PREDICTED
   ] ;

MODEL MOD_2 IS LINREG [
   INPUT = [ CUBEDEV_20 DVLM_5_4_B DAU_STD_32_3 ]
   OUTPUT = DAY_RETURN_1
   MAX STEPWISE = 0
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   RESTRAIN PREDICTED
   ] ;

MODEL MOD_3 IS LINREG [
   INPUT = [ LINDEV_5 DPPV_10 IPMORLET_5 ]
   OUTPUT = DAY_RETURN_1
   MAX STEPWISE = 0
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   RESTRAIN PREDICTED
   ] ;

MODEL MOD_4 IS LINREG [
   INPUT = [ QUODDEV_10 DAU_MAX_32_3 VMUTINF_2 ]
   OUTPUT = DAY_RETURN_1
   MAX STEPWISE = 0
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   RESTRAIN PREDICTED
   ] ;

MODEL MOD_5 IS LINREG [
   INPUT = [ VLM_5_4 IDMORLET_5 DAU_NRG_32_3 ]
   OUTPUT = DAY_RETURN_1
   MAX STEPWISE = 0
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   RESTRAIN PREDICTED
   ] ;
```

```
COMMITTEE COMM IS CONSTRAINED [
   INPUT = [ MOD_1 MOD_2 MOD_3 MOD_4 MOD_5 ]
   OUTPUT = DAY_RETURN_1
   MAX STEPWISE = 99999
   STEPWISE RETENTION = 99999
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   RESTRAIN PREDICTED
   ] ;

WALK FORWARD BY YEAR 15 1999 ;
```

Here is an edited version of the audit log file output for this demonstration. Notice that even though this committee was generated by purely automatic means (the FIND GROUPS command), it greatly outperforms every component model in term of out-of-sample performance.

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

Model MOD_1...

Results for the restrained values...
MSE = 0.37591   R-squared = -0.00219   ROC area = 0.52003
Buy-and-hold profit factor = 1.099   Sell-short = 0.910
Dual-thresholded outer PF = 1.164
   Outer long-only PF = 1.293   Improvement Ratio = 1.177
   Outer short-only PF = 0.972  Improvement Ratio = 1.067


Model MOD_2...

Results for the restrained values...
MSE = 0.37687   R-squared = -0.00474   ROC area = 0.51942
Buy-and-hold profit factor = 1.099   Sell-short-and-hold =
0.910
Dual-thresholded outer PF = 1.113
   Outer long-only PF = 1.090   Improvement Ratio = 0.992
   Outer short-only PF = 1.143  Improvement Ratio = 1.256


Model MOD_3...

Results for the restrained values...
MSE = 0.37637   R-squared = -0.00342   ROC area = 0.50997
Buy-and-hold profit factor = 1.099   Sell-short-and-hold =
0.910
Dual-thresholded outer PF = 1.060
   Outer long-only PF = 1.428   Improvement Ratio = 1.300
   Outer short-only PF = 0.978  Improvement Ratio = 1.074
```

```
Model MOD_4...

Results for the restrained values...
MSE = 0.37710   R-squared = -0.00535   ROC area = 0.49917
Buy-and-hold profit factor = 1.099   Sell-short-and-hold =
0.910
Dual-thresholded outer PF = 1.029
   Outer long-only PF = 1.087   Improvement Ratio = 0.990
   Outer short-only PF = 0.958  Improvement Ratio = 1.052


Model MOD_5...

Results for the restrained values...
MSE = 0.37653   R-squared = -0.00383   ROC area = 0.51577
Buy-and-hold profit factor = 1.099   Sell-short-and-hold =
0.910
Dual-thresholded outer PF = 0.988
   Outer long-only PF = 1.130   Improvement Ratio = 1.028
   Outer short-only PF = 0.904  Improvement Ratio = 0.994


---> Committee results <---

Results for the restrained values...
MSE = 0.37520   R-squared = -0.00029   ROC area = 0.52996
Buy-and-hold profit factor = 1.099   Sell-short-and-hold =
0.910
Dual-thresholded outer PF = 1.302
   Outer long-only PF = 1.452   Improvement Ratio = 1.322
   Outer short-only PF = 1.124  Improvement Ratio = 1.234
```

Market Regression Classes

When working with a large collection of markets, such as the components of the S&P500, it is often too much to expect that a single prediction model will be useful across all markets in our universe. It may be that one model performs well for interest-rate issues, while another does well for consumer product companies. If we try to find one 'universal' model that performs well for all markets, we will probably end up with a model that performs excellently for none. For this reason, *TSSB* has the ability to categorize a universe of markets into subgroups that have similar relationships between predictors and the target. This is done with the following command:

```
REGRESSION CLASS  [ Specifications ] ;
```

Most of the specifications for this command are identical to the similarly named specifications for models. Rather than repeat detailed descriptions that appear elsewhere in this document, we will just provide a brief summary of each command and then give a page reference for users who want more details.

METHOD = HIERARCHICAL / SEQUENTIAL / LEUNG - Specifies the class separation algorithm to use. The HIERARCHICAL method use straightforward hierarchical clustering to produce classes that maximize the minimum ROC area (Receiver Operating Characteristic curve, here) 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 metho that is sub-optimal but runs at a reasonable rate. The LEUNG metho uses the algorithm of Leung, Ma, and Zhang (2001), slightly modified, to find classes. Each class is guaranteed to have a ROC area greate than 0.5 and a positive Spearman Rho between the predicted and the target.

INPUT = [Var1 Var2 ...] - Lists the input variables that are candidates for inclusion. Range and family options are also available. See here.

OUTPUT = VarName-Names the variable that is to be predicted, the target. See here.

MAX STEPWISE = Number - The maximum number of predictor variables that may be included in each group's prediction model. This must be specified. See here.

RESTRAIN PREDICTED - Causes the values of the target variable to be compressed at the extreme values before training. See here.

MIN CASES = Integer - 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. Larger values encourage greater stability by preventing minor markets from exerting undue influence on results.

MAX GROUPS = Number - This does not affect computation, only the printing of results in the log file. The hierarchical and sequential algorithms begin by considering each market to be its own class. As time passes, the number of classes is gradually reduced by merging classes. When the number of classes drops to or below the MAX GROUPS threshold detailed group membership information will be printed to the log files. The Leung method operates in the opposite direction: classes are generated one at a time. Class membership will be printed until more than MAX GROUPS classes are obtained. If you set MAX GROUPS to a very large number, the audit log file may be huge because it contains vast detail. In general, you should set this to the number of regression classes you are truly interested in, or a somewhat larger number, for economy in the log file.

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 (described in detail later). Execution time is a roughly linear function of this value, and larger values result in more accurate performance. Values around 1000 are probably a reasonable compromise between time and quality in most cases, although it should be set to as large a value as computer time allows.

REGRESSION CLASS Demonstrations

We now consider typical uses of the REGRESSION CLASS command. Thes examples will demonstrate each of the three algorithms, and more information about the nature of each algorithm will be provided in the context of discussing audit log output. These examples use the markets in the S&P 100, demanding that a market contain data for at least 500 days in order to be considered. They employ nine predictor candidates: linear, quadratic, and cubic trends at lookbacks of 5, 15, and 50 days. In most practical applications, considerably more than nine predictor candidates will be used. However, execution time increases rapidly as the number of indicators increases, so they are limited for now. At most two predictors will be used in each model, and details are to be printed for only the ten dominant groups. These specifications appear in the declarations of all three examples.

The Hierarchical Method

The hierarchical method for creating regression classes is the most versatile of the three methods, but it is also the slowest in most cases. If time allows and the user wants maximum control, this is the method that should be used. The Leung method is also excellent, but it is more rigid in terms of user control. This will be discussed later.

The hierarchical method examines all possible pairs of markets. For each pair (such as IBM and BOL) it pools the cases in the two markets and then fits an optimal linear model using stepwise predictor selection. Three ROC areas for this model (see here) are computed: that for each of the two markets individually, and that for the pooled pair of markets. The merging criterion for this pair of markets is defined the smallest of these three ROC areas. The motivation for this definition is that if we try to use a single model to handle two markets, then this model should be effective in each market individually as well as in the pooled data. To the degree that *any* of these three interests are not met, the pooling should be discouraged.

Once this ROC-based merging criterion is computed for every possible pair of markets (a *slow* process if there are a large number of markets!), some of the best market pairs are merged. After this initial merging is complete, the process is repeated. However, in this second round of merging, some pairs of markets will have been merged in the first round. Such merged pairs are treated as if they are a single market. Then this process is repeated over and over, until just two market sets remain. It should be apparent that in general, the ROC area at each round of merging will steadily decrease as we force each single model to handle more markets.

Here is the REGRESSION CLASS script file declaration for this exampl followed by the audit log output for the run:

```
REGRESSION CLASS [
   INPUT = [ LIN_ATR_5 - CUB_ATR_50 ]
   OUTPUT = RETURN
   MIN CASES = 500
   MAX STEPWISE = 2
   MAX GROUPS = 10
   RESTRAIN PREDICTED
   METHOD = HIERARCHICAL
   ] ;
```

```
-----------------------------------------------------
REGRESSION CLASS by hierarchical merging beginning...
-----------------------------------------------------

Starting with 74 valid markets
Iteration 1 leaves 65 groups  Crit=0.5546
Iteration 2 leaves 57 groups  Crit=0.5529
Iteration 3 leaves 50 groups  Crit=0.5471
Iteration 4 leaves 44 groups  Crit=0.5463
Iteration 5 leaves 39 groups  Crit=0.5439
Iteration 6 leaves 35 groups  Crit=0.5424
Iteration 7 leaves 31 groups  Crit=0.5381
Iteration 8 leaves 28 groups  Crit=0.5362
Iteration 9 leaves 25 groups  Crit=0.5351
Iteration 10 leaves 22 groups  Crit=0.5344
Iteration 11 leaves 20 groups  Crit=0.5339
Iteration 12 leaves 18 groups  Crit=0.5333
Iteration 13 leaves 16 groups  Crit=0.5316
Iteration 14 leaves 14 groups  Crit=0.5320
Iteration 15 leaves 13 groups  Crit=0.5308
Iteration 16 leaves 12 groups  Crit=0.5298
Iteration 17 leaves 11 groups  Crit=0.5296
Iteration 18 leaves 10 groups  Crit=0.5284
```

```
Markets in each group...

Class 1
   AA
   AVP
   AXP
   BAC

... {Remainder of Group1, followed by Groups 2 through 8}

Class 9
   MMM

Class 10
   SLB

... {Iterations 19 through 25}

Iteration 26 leaves 2 groups  Crit=0.5154

Markets in each group...

Class 1
   AA
   ABT
   AMGN
   AVP
   AXP
   BA
   BAC
... {Remainder of Group 1, followed by Group 2}
```

This audit log output begins by stating that there are 74 valid markets, each of which defines its own group. Actually, there were 100 markets, but the script imposed the restriction MIN CASES = 500 which removes from consideration any markets that do not contain at least 500 days. There were 26 such markets due to the changing membership of the S&P 100.

After the first pass, we are down to 65 groups because 9 pairs of markets were merged. At the end of this first pass, the ROC criterion was 0.5546. Successive passes continue merging pairs, steadily reducing the number of groups, with the price of also steadily reducing the ROC criterion. (There is one pass in which the criterion actually increases a little. This is fairly rare and due to nothing more than random behavior.)

The script specified MAX GROUPS = 10. Thus, when we reach the point of having 10 groups, the market membership of each group is printed. This output is extensive, so most of it is omitted here.

Notice that Groups 9 and 10 are single-market groups. Price behavior in these two markets (MMM and SLB) was so unique that they did not comfortably fit in with any other markets.

Iterations continue until at Iteration 26 we are down to just 2 groups. By this time the ROC criterion has dropped to 0.5154.

There is no definitive way to know the ideal stopping point. In rare cases it may be that a particular merge causes a sudden sharp dropoff in the ROC criterion. When this happens, one would be inclined to stop just before this awkward merge. Unfortunately, it is usually the case that the criterion slowly and steadily drops, with no sharp change. Thus, the user has to trade off the benefit of having a relatively large criterion with the alternative benefit of having few groups.

The Sequential Method

The hierarchical method just described has great theoretical and practical advantages. However, the computer time needed to find the optimal model for *every possible pair* of markets can be prohibitive. The sequential method does a respectable job of imitating the hierarchical method but with considerably more speed.

The sequential method begins by computing the ROC criterion for a subset of the possible pairs. In this example, we specify INITIALIZE = 1000 to tell it to compute the criterion for 1000 randomly selected pairs of markets. The pair with the maximum ROC criterion is merged. Bookkeeping is done to account for the fact that these two markets are no longer separate entities, and then the vector of *INITIALIZE* pairs (1000 in this example) is topped off with more randomly selected pairs. Once again the pair having the best ROC criterion is merged. This process is repeated until only two classes remain, just as was the case for the hierarchical method.

In essence, the only difference between the hierarchical method and the sequential method is that the former tests all possible pairs for merging at each step, while the latter tests only *INITIALIZE* pairs. If *INITIALIZE* is much less than the number of possible pairs, the usual situation, then the time savings will be substantial.

One might worry that there is a high probability that the 'best' merging pair will not be selected for the randomly chosen initialization set. This is true, but it is not usually a serious problem. The reason is that even though the initialization set will almost certainly fail to contain numerous pairs whose merging criteria are superior to all pairs in the complete set, nonetheless the best pair in the initialization set will

almost certainly be very good, a pair which should be merged. It may not be the best, which the hierarchical method would immediately merge, but it is still a pair that the hierarchical method would merge early in the process. And as soon as this best pair is merged in the sequential method, the set will be topped off with more randomly selected pairs. The end result is that the sequential method merges almost the same markets as the hierarchical method, but it does so in a slightly different order. This order may not be optimal, but it still ends up with about the same classes as optimal merging would.

Of course, it is possible that a severe streak of bad luck can subvert the sequential algorithm, while the hierarchical method is deterministic and hence immune to bad luck. Still, as long as *INITIALIZE* is set reasonably large, luck should play only a minimal role in results.

Here is the script file entry for sequential-method merging, followed by the audit log output for this run:

```
REGRESSION CLASS [
   INPUT = [ LIN_ATR_5 - CUB_ATR_50 ]
   OUTPUT = RETURN
   MIN CASES = 500
   MAX STEPWISE = 2
   MAX GROUPS = 10
   RESTRAIN PREDICTED
   INITIALIZE = 1000
   METHOD = SEQUENTIAL
   ] ;

-------------------------------------------------
REGRESSION CLASS by sequential merging beginning...
-------------------------------------------------

Starting with 74 valid markets
  74: Merging 26 and 54   Crit=0.5635
  73: Merging 5 and 69   Crit=0.5632
  72: Merging 39 and 55   Crit=0.5624
  71: Merging 26 and 65   Crit=0.5612
  70: Merging 17 and 68   Crit=0.5611
  { ... Iterations through 16}
  15: Merging 1 and 3   Crit=0.5298
  14: Merging 6 and 7   Crit=0.5297
  13: Merging 2 and 6   Crit=0.5304
  12: Merging 7 and 10   Crit=0.5284
  11: Merging 4 and 6   Crit=0.5278
```

```
--------------------------------
Markets in each of 10 groups...
--------------------------------

Class 1 has 12 markets and ROC area = 0.5423
   AA
   CAT
   CMCSA
   DD
   DELL

... {Remainder of Group1, followed by Groups 2 through 8}

Class 9
   MMM

Class 10
   SLB

... {More iterations}


   3: Merging 1 and 2   Crit=0.5189

--------------------------------
Markets in each of 2 groups...
--------------------------------

Class 1 has 69 markets and ROC area = 0.5222
   AA
   ABT
   AMGN
   AVP
   AXP
... {Remainder of Group 1, followed by Group 2}
```

As with the hierarchical method, this audit log output begins by stating that there were 74 valid markets, each of which defines its own group. Actually, there were 100 markets, but the script imposed the restriction MIN CASES = 500 whic removes from consideration any markets that do not contain at least 500 days. There were 26 such markets due to the changing membership of the S&P 100.

The hierarchical method merges groups in small chunks to speed operation, a trivial sacrifice of optimality that produces great speedup relative to merging one pair at a time. But the sequential method, which is already much, much faster than the hierarchical method, can afford to merge one pair at a time. The audit log identifies the group numbers of the merged pairs, which is of little interest to most users, but can provide useful information to advanced users by identifying patterns of merging. More importantly, the audit log lists the usually decreasing ROC

criterion. If it suddenly plunges, this is an indication of the point at which merging might best be terminated.

The script specified MAX GROUPS = 10. Thus, when we reach the point of having 10 groups, the market membership of each group is printed. This output is extensive, so most of it is omitted here. Notice that Groups 9 and 10 are single-market groups, as was the case with hierarchical merging. Price behavior in these two markets (MMM and SLB) was so unique that they did not comfortably fit in with any other markets. Note that even though the group memberships will usually be similar with hierarchical and sequential merging, the associated numbering may be different. In this case, Groups 9 and 10 happened to have the same group numbers in both methods, though Group 1 was different.

Iterations continue until we are down to just 2 groups. By this time the ROC criterion has dropped to 0.5189.

As with hierarchical merging, there is no definitive way to know the ideal stopping point. In rare cases it may be that a particular merge causes a sudden sharp dropoff in the ROC criterion. When this happens, one would be inclined to stop just before this awkward merge. Unfortunately, it is usually the case that the criterion slowly and steadily drops, with no sudden change. Thus, the user has to trade off the benefit of having a relatively large criterion with the alternative benefit of having few groups.

The Leung Method

The paper "A New Method for Mining Regression Classes in Large Data Sets" b Leung, Ma, and Zhang (IEEE Transactions on Pattern Analysis and Machin Intelligence, January 2001) presents a mathematically rigorous algorithm for discovering regression classes in large and noisy datasets. The algorithm is extremely complex, so only an overview will be presented here. The Leung method differs from the hierarchical and sequential methods in several key ways:

- The hierarchical and sequential methods begin with each market defining its own class, and it steadily merges markets and existing classes to *reduce* the number of classes as it proceeds. The Leung method begins with no classes, and it discovers new classes one at a time, thus *increasing* the number of classes as it proceeds.

- Once a class is defined by the Leung method, it remains unchanged throughout the remainder of processing. In the hierarchical and sequential methods, existing classes can be merged to create larger classes.

- The Leung method can quickly identify markets that are so unlike other markets that they can never be pooled with other markets in a single class. As such markets are encountered during processing, they will be identified and removed from further consideration. The hierarchical and sequential methods identify such markets only at the end of processing when they remain isolated, having been unable to merge with any other markets.

- Perhaps most important, the Leung method contains inherent statistical tests to determine if a group of markets is so statistically consistent that these markets can indeed be considered to be members of the same regression class. Such classes will be labeled 'Good' in the audit log. Classes that fail an internal consistency test will be labeled 'Bad' in the audit log.

It is important to understand that several different and extremely sophisticated criteria are involved in the Leung algorithm. Some criteria determine whether two or more markets are cast into the same class. Others rate the statistical uniformity of a class. Naturally, there is a high correlation between these criteria. Thus, if a class consists of two or more markets, it is likely that it receives a 'Good' rating. Conversely, a single isolated market is likely to be given a 'Bad' rating. Nonetheless, it is possible, in rare circumstances, for two or more markets to be assigned to the same 'Bad' group. Also, an isolated market may receive a 'Good' rating if it has excellent internal consistency and predictive power. The bottom line is that the user should be wary of any group, regardless of its size, which receives a 'Bad' rating.

Here is the script declaration for the Leung method, followed by the corresponding audit log output. Note that we set MAX GROUPS=50 because the Leung metho has a tendency to find small and poor groups first. We want to make sure that all interesting groups are printed.

```
REGRESSION CLASS [
   INPUT = [ LIN_ATR_5 - CUB_ATR_50 ]
   OUTPUT = RETURN
   MIN CASES = 500
   MAX STEPWISE = 2
   MAX GROUPS = 50
   RESTRAIN PREDICTED
   METHOD = LEUNG
   ] ;

-------------------------------------------
REGRESSION CLASS by Leung method beginning...
-------------------------------------------

One recalcitrant market; making it a class

1 classes; 73 markets remain unclassified
 Class  Markets
    1       1   Bad

Markets in each group...

Class 1  (Bad)
   MSFT


2 classes; 71 markets remain unclassified
 Class  Markets
    1       1   Bad
    2       2   Good

Markets in each group...

Class 1  (Bad)
   MSFT

Class 2  (Good)
   AMGN
   MRK

Making one or more poor markets their own class
```

```
3 classes; 70 markets remain unclassified
 Class  Markets
    1        1   Bad
    2        2   Good
    3        1   Bad

Markets in each group...

Class 1  (Bad)
   MSFT

Class 2  (Good)
   AMGN
   MRK

Class 3  (Bad)
   DELL

... {More iterations}
```

```
34 classes; 0 markets remain unclassified
 Class  Markets
    1        1   Bad
    2        2   Good
    3        1   Bad
    4        2   Good
    5        1   Good
    6        1   Good
    7        1   Bad
    8        2   Good
    9        2   Good
   10        1   Bad
   11        1   Bad
   12        4   Good
   13        2   Bad
   14        1   Good
   15        2   Bad
   16        3   Good
   17        2   Good
   18        1   Good
   19        2   Bad
   20        2   Bad
   21        1   Bad
   22        3   Good
   23        1   Good
   24        1   Bad
   25        4   Good
   26        1   Bad
   27        6   Good
   28        2   Good
   29        1   Bad
   30        1   Bad
   31        5   Good
   32        2   Good
   33        1   Bad
   34       11   Good
```

The first thing the algorithm does here is to identify what the program calls a recalcitrant market. This is a market that is so unusual and internally inconsistent that it cannot merge with any other market. We soon see that this market is MSFT.

It then identifies a regression class consisting of two markets: AMGN and MRK This class is labeled ‘Good’ because these two markets can be pooled into a single dataset for which a single prediction model is effective.

A third class consisting of the single market DELL is identified. This market is labeled ‘Bad’ because it fails one or more of the internal consistency tests.

This process is repeated until a total of 34 classes are found. As is typical of the algorithm, it tends to find small and single-market classes first. It isn’t until the 12'th class that it finds a ‘Good’ class consisting of more than 2 markets. Class 27

is ‘Good’ and contains 6 markets, and it is not until its final discovery, Class 34, that we find a class that contains 11 markets. The markets that make up each of these classes are printed in the audit log but not reproduced here because the listing is extensive.

One disadvantage of the Leung method compared to the hierarchical and sequential methods is that the user has no control over the number and size of classes. In those other two methods, merging can be terminated at any point according to the user’s choice of class properties. But the Leung method uses sophisticated statistical tests to determine the optimal classes, and the user has no alternative but to accept those decisions. On the other hand, the user does have the comfort of knowing that the choices made by the Leung algorithm have a decent degree of statistical rigor, while the choice of stopping point made by the user in the hierarchical and sequential methods is more or less arbitrary.

Developing a Stand-Alone System

In this chapter much of the material in prior chapters comes together. We will now walk through every step in the development of a stand-alone trading system. It must be emphasized that the approach shown here is not by any means the only correct approach. An infinite number of variations are possible. And due to time and space constraints, some aspects of this development will be abbreviated. Still, the procedures described here cover the most common and important aspects of system development.

Choosing Predictor Candidates and the Target

If there is one rule that trading system developers should remember, it is this: *predictive power should come primarily from the predictor variables, not from the model*. In fact, powerful models (those able to fit complex patterns) are more likely to harm performance by overfitting than they are to help performance by harnessing subtle information in the predictors. Thus, it is almost always in the best interest of the developer to expend massive effort in finding effective predictor variables and then present them to a relatively weak model such as linear regression.

Choosing the Target

Before delving into the complexities of predictor selection, we'll mention target selection. In most cases, selecting a target does not involve a difficult decision because the target is intimately connected to the application, such as the desired trade duration, and hence not open to much discussion. Thus, we will not devote much attention to the target. Nonetheless, here are a few common situations:

- An excellent choice is a *one-day-ahead* trading system in which a position is taken at the next open of the market after a signal is given, and then the trade is closed at the following open. This method has the valuable advantage that reasonably valid statistical tests can be performed on out-of-sample trades to estimate the probability that results as good as those obtained could have been obtained by pure random good luck. Such tests are difficult to impossible for trading systems that look ahead more than one day. One-day-ahead systems also have at most one position open at a time, which simplifies margin and risk calculations. In this case, a target such as NEXT DAY LOG RATIO or NEXT DAY ATR RETURN is appropriate.

- Another popular choice is to open a position at the next open after a signal is given, and then hold the position until either a specified profit is made or a specified loss is suffered. This use of the HIT OR MISS target has the advantage of corresponding to closing trades with *limit* and *stop* orders, a common trading habit. It also has the enormous advantage that the distribution of returns has exceptionally light tails; extreme wins and losses are impossible because they are limited by the fixed exit points. This is a great benefit to model building, as the training algorithm will not focus undue energy on eliminating severe losses or capturing huge wins. On the other hand, as with all model-based systems other than *one-day-ahead*, multiple positions are likely. As long as buy/sell signals are given (the prediction is more extreme than the threshold), positions are piled on, even if one or more

positions are already open. This can complicate real-life trading due to margin requirements and risk wariness. Unfortunately, with any model-based development system (not just *TSSB*), this problem is impossible to avoid without introducing other problems that are usually even more serious. The moral of the story is that one should always set the stops and limits in the HIT OR MISS target to be as tight as possible so that positions are kept open for only short stretches of time. Also, one can set the cutoff bar count of the HIT OR MISS target to a low value to preclude extended time in a position.

- Perhaps the least desirable choice is to specify in advance that a trade will be kept open for a fixed length of time, such as a number of days, using a target such as SUBSEQUENT DAY ATR RETURN. This shares with HIT OR MISS targets the disadvantage that multiple positions will often be open. It also shares with one-day ahead systems the disadvantage that extreme wins and losses are possible, making effective training difficult. Finally, although the *tapered block bootstrap* and the *stationary bootstrap* can theoretically be used to provide statistical tests of the validity of the trading system, these tests are notoriously unreliable in practice, and they should not be overly trusted to give valid results. In summary, a fixed multiple-day-ahead target has all of the disadvantages of the other choices, and none of the advantages. This should be a last resort.

One final word is needed concerning the trading systems that allow multiple positions to be open. This includes all systems except one-day-ahead. In addition to the obvious practical problems of large margin requirements and high risk of ruin, there is a subtle but enormous statistical danger: the variance of net returns of such a system is huge. This is because there is high serial correlation between trades. Suppose our strategy is to hold positions for ten days. Suppose also that a trade we open today is destined to score a large win. Then if we open a new position tomorrow, it, too, will likely have a large win, because nine of their ten days overlap. The net effect is that wins and losses are exaggerated, resulting in a huge error variance (uncertainty of the estimate) of total returns. Thus, a developer may discover a system that has an enormous profit factor and conclude that wealth is right around the corner, when in fact it's just a random lucky string of highly correlated wins.

Here's another way of looking at it. Suppose you play a game in which you toss a coin 20 times. Every time it comes up heads you win a dollar, and every time it comes up tails you lose a dollar. Most of the time you will probably come out about even because wins and losses will roughly cancel each other. Now suppose you modify the game. You still win a dollar for heads and lose a dollar for tails. But if it comes up heads, for the next toss you must use a coin that is strongly biased toward heads. If, as is very likely, that toss comes up heads, your next toss again is

with a coin strongly biased towards heads. The same is true for tails: if a toss comes up tails, your next toss is with a coin strongly biased towards tails. You can see how wins and losses would not cancel well. Instead, you will experience long strings of wins or losses. In the original, fair game your final total would be the sum of a large number of small wins and losses. But in the modified game, your final total would be the sum of a few strings of extraordinarily large wins and losses. This will result in huge variance for your total gain, making it nearly impossible to decide, based on your gains, whether the coin is fair on average..

The bottom line is this:

- Your best choice will often be a *one-day-ahead* system because it is most amenable to statistical confirmation of results, although it does have the disadvantage of occasional large wins and losses which can distort performance figures and possibly training.

- Another excellent alternative is a *hit-or-miss* system with very tight limit and stop exits. This system allows multiple positions, which complicate statistical analysis and margin requirements. However, if the exit points are tight, special bootstrap tests like the tapered-block and stationary are generally applicable, and large positions are unlikely. The huge advantage of a *hit-or-miss* system is that extreme wins and losses are impossible (at least in the theoretical analysis, in which markets cannot blow through stops!). This facilitates effective training and interpretation of results.

- A *hit-or-miss* system with a distant target and/or stop still has the advantage of a light-tailed distribution, but the possibility of large positions accumulating makes statistical analysis and margin calculations difficult. This is generally not recommended.

- A system with a fixed holding period of many days is the worst possible choice. It has every disadvantage discussed above, and no real advantage. It should be used only if circumstances make it mandatory.

Quality Does Not Equal Quantity for Predictors

Developers of model-based trading systems face a fundamental quandary. One does not want to inadvertently ignore an indicator that has powerful predictive ability. At the same time, the degree of noise inherent in market dynamics makes it likely that if you examine a sufficient number of indicators, you will stumble upon some whose pattern of noise over the historical sample happens, by pure random chance, to appear to be highly predictive of the target. Naturally, if such an indicator is included in the trading model, that deceptive pattern will soon vanish in real life, leaving the trader with a worthless indicator.

It would be unrealistic to expect the system developer to specify in advance the exact predictors (perhaps two or three) that his or her model will employ. Virtually all developers rely on intelligent automated algorithms to select effective predictors from among a set of candidates. The task of the developer is to limit as much as possible the size of the candidate list. Herein lies what many or most experts agree is the most difficult yet single most important aspect of model development. If the list of candidates is extensive, it is likely that one or more truly useful indicators will be among them. This is good. On the other hand, a large candidate list increases the likelihood that one or more of them will have a pattern of random noise that impersonates predictive power. This is bad. Thus, the developer is faced with the unenviable task of coming up with a list that is likely to contain useful predictors but unlikely to contain more than the inevitable few useless indicators. He or she will then trust an intelligent automated selection algorithm to separate the sheep from the goats, trusting that careful initial selection of candidates will minimize the possibility that the selection algorithm will make an accidental poor choice.

How does one come up with a candidate list? Here are a few thoughts in no particular order:

- As in much of life, there is no substitute for experience. A person who has designed model-based trading systems for many years in many markets will have a solid intuitive feel for which indicators work in a given environment and which do not. The new, inexperienced developer should make use of whatever experienced colleagues are available.

- Market professionals have developed many theory-based hypotheses about market behavior. For example, momentum-driven systems are based on the notion of investor under-reaction to new information; only slowly do investors become unanchored from prior beliefs. As a result, prices do not move instantaneously to a new level implied by breaking news. That delayed reaction is a gradual trend that momentum indicators can exploit. Conversely,

overbought and over-sold indicators are based on over-reaction to news, with investors taking prices beyond a level implied by the news.

- There is almost always a huge correlation between the target distance and the historical lookback period for computing useful indicators. If you are designing a one-day-ahead system, it would be unusual to find useful predictive information in an indicator that is based on more than 20 or so days of history. There are exceptions to this rule, but not many. In fact, looking back as few as five days is often ideal for a one-day-ahead system. Similarly, if you are designing a hit-or-miss system that will typically be in the market for 30-50 days, it is unlikely that an indicator based on the most recent five days will be of much value.

- When in doubt, look to price momentum indicators. Recent trend, and sudden deviation from recent trend are often the best choices. They tend to have monotonic relationships with targets, so they are particularly useful indicators for weak models such as linear regression.

- More exotic indicators based on volume, information content, and other esoteric quantities can contain valuable predictive information, but it almost always is in a highly nonlinear relationship with the target, necessitating powerful nonlinear models.

- Stationarity, at least at a reasonable visual level, is crucial for predictors. It is impossible to overemphasize the importance of this. For this reason, visual examination of a time-series plot of every predictor candidate is mandatory. If an indicator hovers around, say, 20-40 for a year or two, then drops to 5-15 for a few years, this predictor is worthless. The indicators built into *TSSB* have been designed to have decent stationarity in most markets and time periods, but it is still important to look at them before putting them to use.

- In a multiple-market situation, it is crucial that the distribution of each indicator be similar in all markets. If this is not the case, some markets will dominate training and trading while others will be ignored. This will almost always be a serious problem. As with stationarity, the indicators built into *TSSB* have been designed to have distributions that are fairly independent of the market in which they operate. Still, one should take advantage of the cross-market conformity tests built into the program in order to confirm this property. This will be discussed more later.

Predictor and Target Selection for this Study

This section discusses the particular variables selected for this development of a stand-alone trading system.

The first choice made was to develop a one-day-ahead trading system for the longest-lived markets in the Standard and Poors 100. We will deal with the problem of extreme wins and losses by means of the RESTRAIN PREDICTE command (here), which while not perfect, is a decent way of handing the problem. The statistical tractability of one-day-ahead systems is a powerful incentive to make this choice. Hence, our target is NEXT DAY ATR RETURN 250. This targe assumes that we will take a position at the open of the day following the signal and close it at the open of the following day. The size of the position will be inversely related to the *average true range* over the prior year (250 days).

The type of system developed here is different from what has been seen in any other example in this tutorial document. Here we present what is often called a *balanced* or *market-neutral* system. In order to remain relatively immune to unexpected large movement of the total market, a balanced system holds an equal number of long and short positions each day. In theory at least, if the market suddenly moves en mass, losses on one side will be compensated by gains on the other side. If the trading system can do a good job of finding a few markets that are expected to move upward (at least relative to the market as a whole), and also find a few that are expected to fall, then the system can take long positions in the former and short positions in the latter. Over the long term, a balanced system can expect to make money in any sort of market. (As an interesting side note, the supposed market neutrality of a balanced system has tempted some traders to take on absurd positions, and subsequent market unwinding in unanticipated directions has caused some monumental collapses of major equity firms. Beware.)

The variables used in this study are saved to the database and tested using the following script file. The commands shown are all basic and discussed in detail elsewhere in this tutorial, so detailed discussions are not provided here.

```
READ MARKET LIST "D:\BOOSTER\TEST\SP100_MAJOR.TXT" ;
INDEX IS OEX ;
READ MARKET HISTORIES "E:\SP100\C.TXT" ;
CLEAN RAW DATA 0.4 ;
READ VARIABLE LIST "StandAloneVars.TXT" ;
OUTLIER SCAN ;
CROSS MARKET IQ ;
WRITE DATABASE "StandAlone.dat" "StandAlone.fam" ;
```

Here is a listing of the predictor candidates chosen for this study. Note that they

follow the precepts mentioned in the prior section. They are various ways of measuring trend or deviation from trend, indicators that have a long history of effective predictive power. No measures of volatility or more complex variables are included in this study because they generally require nonlinear models, which in turn require very large training sets and extended training time.

```
LIN_ATR_5: LINEAR PER ATR 5 100
LIN_ATR_15: LINEAR PER ATR 15 100

MOM_5_100: PRICE MOMENTUM 5 100
MOM_10_100: PRICE MOMENTUM 10 100

ADX15: ADX 15
ADX30: ADX 30

LINDEV_10: LINEAR DEVIATION 10
LINDEV_40: LINEAR DEVIATION 40
QUADDEV_10: QUADRATIC DEVIATION 10
QUADDEV_40: QUADRATIC DEVIATION 40

INDDEV_10: DEVIATION FROM INDEX FIT 10 0
INDDEV_40: DEVIATION FROM INDEX FIT 40 0

INT_10: INTRADAY INTENSITY 10
INT_20: INTRADAY INTENSITY 20

DINT_10: DELTA INTRADAY INTENSITY 10 10
DINT_20: DELTA INTRADAY INTENSITY 20 20
```

Details concerning these predictor candidates can be found in the *Variables* chapter. However, here is a basic summary:

LIN_ATR_5 and *LIN_ATR_15* are price velocity (linear trend, or slope of price change) indicators at lookbacks of 5 and 15 days.

MOM_5_100 and *MOM_10_100* are similar to the preceding two except that they are based on a simple change in price rather than a linear fit.

ADX15 and *ADX30* are the traditional ADX trend indicators. Because they are non-directional they are likely more suitable for nonlinear models than linear. For this reason, a quadratic model will be included in the development.

LINDEV_10 through *QUADDEV_40* measure the deviation of the current price from what would be predicted by a linear or quadratic fit of recent prices. On first glance it might appear that using a lookback of 40 days violates the earlier precept that one-day-ahead systems should focus on very recent history, rarely looking back more than 20 days. However, these four indicators measure *today's* deviation,

which is as recent as it gets! Here, the lookback just determines the length of time which is used to compute the prediction. This, of course, is relevant, but it is swamped out by the immediacy of using today's deviation. Also, especially for a quadratic fit, using 40 days is reasonable in order to get stable coefficients.

INDDEV_10 and *INDDEV_40* compute a least-squares linear regression function for estimating the market price from the index (OEX in this example). They then subtract today's estimated value from today's actual value. This measures how much a particular market has suddenly deviated from what would have been expected based on its recent relationship to the index. As with the prior deviation indicators, even though this is a one-day-ahead system it is reasonable to try going back 40 days for the fit, because the primary information content of this indicator comes from the current price.

INT_10 and *INT_20* are the intraday intensities at lookbacks of 10 and 20 days. These measure daily changes relative to daily true ranges.

DINT_10 and *DINT_20* are the differences in intraday intensity over short lookback periods.

Stationarity

If an indicator slowly wanders up and down, staying in one range for an extended period before moving to another range for another extended period, this indicator will usually be useless for model-based trading. This is because the predictions of a model in which this indicator is used will similarly wander, and it will be impossible for the training algorithm to find a signal threshold that produces at least reasonably uniformly distributed trades. The model would trade frequently for a long period of time, and then shut down for a long period of time. The result would be a model whose training is dependent on market behavior over particular time periods. Sometimes this is what the user wants, but usually this lack of universality is problematic. In fact, if the user want models that specialize in specific market conditions, there are better ways of doing this, such as prescreening, oracles, and triggering. These topics are all discussed elsewhere.

For an example of the sort of stationarity that is almost always ideal for an indicator, look at Figure 57 below, which shows one of the indicators used in this demonstration. Imagine that you were to draw a horizontal line anywhere in that figure, but especially near the extremes of the series (around plus or minus 40). The series would cross outside that line regularly across its entire history. In nearly all applications this is exactly the sort of behavior that you want. Note that *TSSB* contains a variety of stationarity tests built into its tool set. However, most of these tests are far more strict than is needed for most applications. Simple visual confirmation of the sort shown in this plot is almost always sufficient.

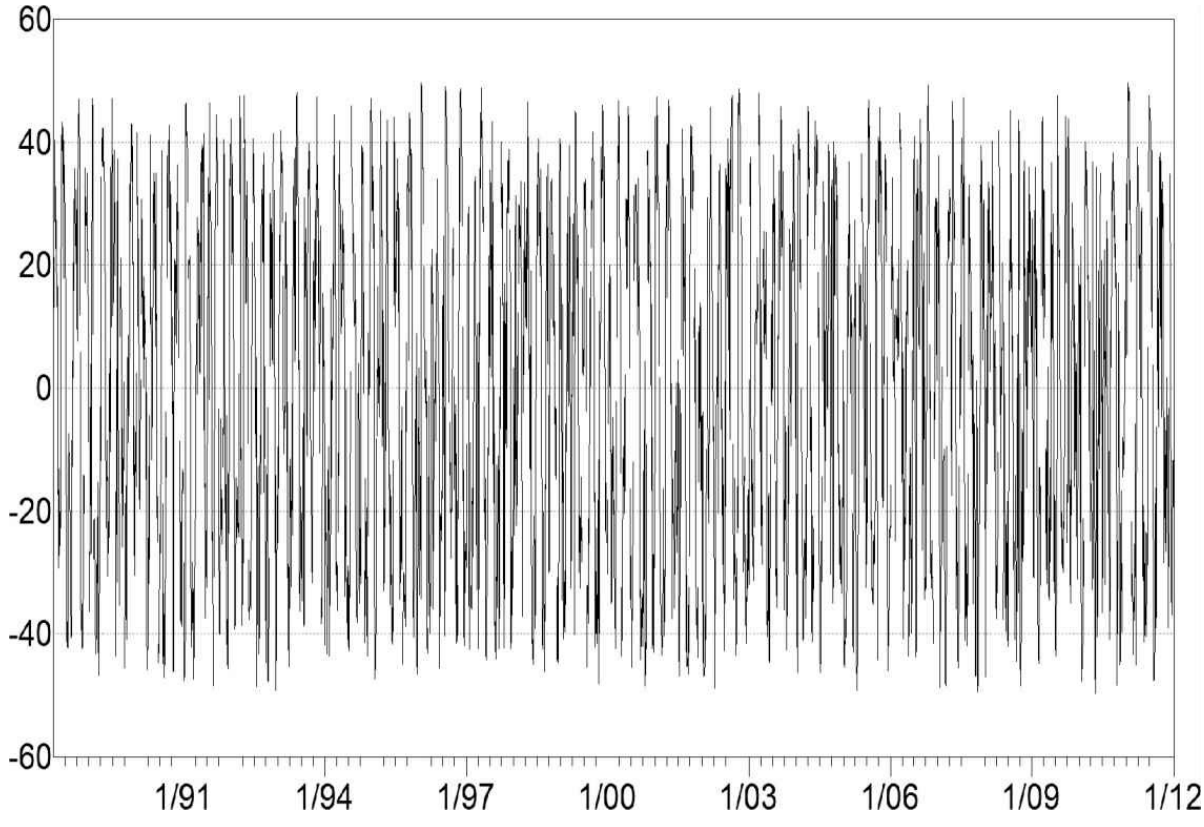


Figure 57: An ideally stationary indicator

The Problem of Outliers

If one or more cases have a predictor or target value that is extreme relative to the majority of cases in the training set, most models will be compromised in their training. This is because the training algorithm will go to great lengths to reduce the impact of prediction error for outlier cases, at the expense of performance for the majority of 'normal' cases. *TSSB* contains a model training option called *RESTRAIN PREDICTED* (here) which compresses the target as needed to reduce the effect of extreme target values. This is necessary because some targets, such as market moves over a fixed time period, are inherently at risk of extreme values; markets occasionally experience wild gyrations.

Unlike the situation with targets, the developer has no real excuse for presenting a model with training cases that have extreme values of predictors. The user has full control over the definition of indicators, and suitable compression can (and should!) be built into the definition. All of the indicators in *TSSB*'s library have been designed to avoid extreme values. Nonetheless, before embarking on a development task, the user should confirm that no outliers are present. This is done with the *OUTLIER SCAN* command described here. This command was run on the indicator set used for the development task of this chapter, obtaining:

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

  Variable     Min     Mkt     Max     Mkt      IQ     Rn  Ratio
Entropy
 LIN_ATR_5  -50.00    UNH   49.99    AAPL   25.48    3.9   0.90
LIN_ATR_15  -50.00    MRK   49.95     AVP   25.78    3.9   0.90
 MOM_5_100  -49.94    AEP   49.67     AVP   23.35    4.3   0.88
MOM_10_100  -49.77     BA   48.93     AVP   23.63    4.2   0.88
     ADX15    2.99    AEP  100.00    DELL   19.79    4.9   0.81
     ADX30    3.40    BMY  100.00    DELL   14.34    6.7   0.71
 LINDEV_10  -43.76    UNH   43.76    DELL   43.83    2.0   0.98
 LINDEV_40  -49.97    MRK   49.94    MSFT   45.28    2.2   0.98
QUADDEV_10  -37.94    UNH   37.94    DELL   35.95    2.1   0.98
QUADDEV_40  -49.95    MRK   49.90     AVP   42.14    2.4   0.97
 INDDEV_10  -47.92    CVS   47.93    DELL   49.10    2.0   0.99
 INDDEV_40  -49.94    MRK   49.87     AVP   41.67    2.4   0.97
    INT_10  -48.08    DVN   48.32     DVN   22.23    4.3   0.85
    INT_20  -49.10   CMCSA   47.11     AEP   22.80    4.2   0.86
   DINT_10  -49.97    DVN   49.60     DVN   29.79    3.3   0.92
   DINT_20  -49.94    DVN   49.89   CMCSA   30.10    3.3   0.92
    RETURN  -14.07    MRK   10.99     BAX    0.84   29.5   0.33
```

```
Sorted by ratio, worst to best

       Variable     Ratio  Entropy
         RETURN     29.5    0.338
          ADX30      6.7    0.714
          ADX15      4.9    0.815
         INT_10      4.3    0.858
       MOM_5_100     4.3    0.881
         INT_20      4.2    0.867
      MOM_10_100     4.2    0.882
       LIN_ATR_5     3.9    0.907
      LIN_ATR_15     3.9    0.901
         DINT_10     3.3    0.926
         DINT_20     3.3    0.928
       INDDEV_40     2.4    0.973
      QUADDEV_40     2.4    0.978
       LINDEV_40     2.2    0.984
      QUADDEV_10     2.1    0.985
       LINDEV_10     2.0    0.986
       INDDEV_10     2.0    0.993
```

The most important figure in this table is the ratio of the range (max minus min) divided by the interquartile range. The range (numerator) encompasses extreme values, while the interquartile range (denominator) exemplifies 'normal' behavior of the variable. There is no magic threshold for declaring that an indicator is good or bad, but it would not hurt to examine a histogram of any indicators that have an unusually large ratio. If the value of an indicator for one or a very few cases are far from the mass of cases, that indicator should be considered suspicious. Here we see that ADX30 is a little suspicious, but not seriously so, and all others are fine.

Cross-Market Compatibility

The trading system being developed in this chapter is designed to be used with the component markets in the Standard and Poors 100. Since multiple markets are involved, it is crucial that the markets behave similarly for every indicator and the target. Suppose one market has huge variance of an indicator (or indicators, or the target) used in the prediction model, much greater than the variance in other markets. Then this market will dominate training and execution. The model will be trained so as to optimize performance on this dominant market, to the near exclusion of performance in other markets. Thus, it is crucial that we always test that all indicator candidates, as well as the target, behave similarly in all markets. The indicators and targets in the built-in *TSSB* library have been designed to have good cross-market compatibility. Still, it does not hurt to verify this. The *CROSS-MARKET IQ* test was run on the dataset used in this development effort, and satisfactory results were obtained. The following is an abbreviated listing of the

results:

```
Cross market IQ Range overlap test for variable LIN_ATR_5
These results are sorted worst to best
    OEX has insufficient data or statistic is undefined
    DVN   0.83250
     SO   0.86832
    CAT   0.89873
    PEP   0.90642
    XOM   0.91244
     AA   0.92112
    CPB   0.92295
    FDX   0.93014
```

The exact nature of this test is described here. What we see in the partial table just shown is that for the indicator *LIN_ATR_5* the worst market is DVN, with an IQ range overlap of 0.83250. It is very good to see the worst being this good. Each indicator should be checked this way.

This test also produces two useful summary tables. Here is one:

```
Median IQ range overlap across markets, worst to best...
      LIN_ATR_15  0.95841
          INT_20  0.95956
           ADX30  0.96073
         DINT_20  0.96324
       LIN_ATR_5  0.96607
          INT_10  0.97242
         DINT_10  0.97477
      MOM_10_100  0.97493
          RETURN  0.97560
       MOM_5_100  0.97611
           ADX15  0.97978
       INDDEV_40  0.98312
       LINDEV_40  0.98597
      QUADDEV_40  0.98663
      QUADDEV_10  0.98914
       LINDEV_10  0.98930
       INDDEV_10  0.98947
```

The table just shown says that the worst indicator in terms of IQ range overlap is *LIN_ATR_15*, and even it has a median (across all markets) IQ range of 0.95841, which is fabulous cross-market compatibility.

The second summary table shows the median (across all indicators and the target) IQ range overlap for each market. The first few markets in this table are shown below. Note that OEX is excluded from the study because it is an index market, and index markets never appear in the database. This is of no consequence for this test.

In this table we see that DVN is the worst market for cross-market compatibility, and even it has a median IQ range overlap of 0.95117, which is a wonderful figure.

```
Median IQ range overlap across variables, worst to best...
    OEX has insufficient data or statistic is undefined
    DVN   0.95117
     AA   0.95153
    FDX   0.96056
    CPB   0.96070
    CAT   0.96290
    XOM   0.96357
    WFC   0.96424
```

Data Snooping: Friend or Foe?

The bane of every trading system developer is the paucity of training and testing data. In most engineering and social science fields, the availability of data is limited only by resources such as time and money. While these can be difficult hindrances, they are not impossible to overcome in most cases. But markets only go back so far in time, and their behavior changes. Behavior in a market that may have been predictable fifty years ago, if the market even existed then, is unlikely to be predictable in the same way today. And we cannot just say that the thousands of equity and commodity markets currently available provide nearly unlimited data, because these markets have (or must be assumed to have) significant correlation. Sure, you can develop a trading system for IBM, test it in T, and rejoice if it does well in that market. But your joy must be tempered by the fact that IBM and T are impacted by numerous common events and fundamental factors, so one certainly cannot consider data in a different market to be out-of-sample, a source of independent confirmation of performance.

This dilemma leads to the necessity of a difficult decision: do we use all available history, right up to the current date, give the development effort one good shot, and hope for the best if we put the system into practice? This choice makes maximum use of available history, which is good, but it requires a lot of faith in the ability of the developer and his or her development tools. Properly done, as with the data-pooling walkforward capabilities in *TSSB*, this will provide an unbiased estimate of future performance *as long as the user does not attempt multiple methodologies to develop the final trading system, and then choose the methodology having best walkforward performance*. If the developer picks the best of these, the chosen model will probably be good, but its performance figure will be optimistically biased. In other words, if you *use all available data and then experiment* to find the best system, you will not have a reliable estimate of future performance. You will just have to hope for the best when you begin trading the system with real money.

The other option is to hold out a recent chunk of history, such as the most recent year, and devote enormous research power to producing an excellent system with the data prior to the segment excluded from study. This has the advantage of giving the developer complete freedom to study, experiment, and tweak performance. In particular, if the developer is inexperienced or is using new, unstudied indicators or targets, this method may be required. After a highly effective trading system is developed with the older, excluded data, it can then be tested on the most recent, virgin data. This will provide an unbiased estimate of future performance. Of course, the price paid for this delightful accomplishment is that the experimentation phase, in which the developer laboriously constructed a beautiful trading system, did not have access to the most recent market data. This is a high price to pay, but

this is the route taken in this tutorial, because it allows demonstration of some basic experimental techniques.

Checking Stability with Subsampling

While not strictly necessary, it's always interesting to use subsampling (or resampling for models other than a GRNN) to check on the stability of the relationship between the predictor candidates and the target. There are three essential core components of the development of a model-based trading system:

- Chose one or several predictors
- Train a model to predict a target using these predictors
- Find a threshold for making trade decisions based on the model's predictions

Ideally, these components should be stable. In other words, changing some aspect of the training and test data should cause little change in the choice of the predictors, the nature of the models, and the location of the threshold.

The obvious way to examine stability is with walkforward testing, and certainly every fold of a walkforward run should be studied. In particular, one should pay close attention to the out-of-sample performance in each fold. If it varies widely from fold to fold, especially if the folds are large (a year) and the change is massive (a bad loss amidst many wins), one should be suspicious.

However, some variation in walkforward folds is unavoidable. We may be testing a long-only system, and if a huge bear market comes along it may be that the only reasonable thing we can expect from our system is that it avoid being in the market during the worst downward moves. We should be happy if our system loses a lot less money than a buy-and-hold strategy would lose. Thus, time-slicing for stability checks has limited utility.

Examination of stability can be approached from a different direction which avoids interference from market trends. Instead of slicing by time, we take random samples from the entire extent of the dataset. This can be done with the SUBSAMPLE o RESAMPLE model option. The SUBSAMPLE option is usually the better choi because the RESAMPLE option is incompatible with GRNN models due t duplication, and also because subsampling accomplishes essentially the same thing as resampling, but with a smaller training set.

One can subsample the entire dataset and use the TRAIN command, but a key measure of the quality of a model is how well its predictions hold up out of sample. This lets us distinguish a model that is so powerful (overfitted) that it learns noise, from a model that has just enough power to learn the true patterns in the data. Hence, the ideal testing method is to embed subsampling into a walkforward run and examine the summary. With this in mind, here is a script file that trains and tests

five subsampled models:

```
RETAIN YEARS 1900 THROUGH 2010 ;
READ DATABASE "StandAlone.dat" "StandAlone.fam" ;

MODEL LINMOD1 IS LINREG [
   INPUT = [ LIN_ATR_5 - DINT_20 ]
   OUTPUT = RETURN
   RESTRAIN PREDICTED
   SUBSAMPLE 60 PERCENT
   MAX STEPWISE = 2
   STEPWISE RETENTION = 10
   FRACTILE THRESHOLD
   CRITERION = BALANCED_10
   MIN CRITERION FRACTION = 0.1
   SHOW SELECTION COUNT
   ] ;
```

... Models 2, 3, and 4

```
MODEL LINMOD5 IS LINREG [
   INPUT = [ LIN_ATR_5 - DINT_20 ]
   OUTPUT = RETURN
   RESTRAIN PREDICTED
   SUBSAMPLE 60 PERCENT
   MAX STEPWISE = 2
   STEPWISE RETENTION = 10
   FRACTILE THRESHOLD
   CRITERION = BALANCED_10
   MIN CRITERION FRACTION = 0.1
   SHOW SELECTION COUNT
   ] ;

WALK FORWARD BY YEAR 5 1999 ;
```

For economy, only two of the five models are shown above. They are all identical. Each uses the full set of predictor candidates and predicts the restrained target. Other model parameters mimic the parameters that will be used in the final prediction models:

- A maximum of two predictors are used.
- STEPWISE RETENTION = 10 is reasonable, large enough to guarant thorough testing of various indicator combinations, yet not so large as to require excessive computation time.
- Since this is to be a balanced system, we must use the FRACTILI THRESHOLD option and a BALANCED criterion (ten percent here).

- The MIN CRITERION FRACTION should be set to match the BALANC fraction (0.1 is 10 percent), even though this parameter plays no practical role in training a balanced system at this time. It may play a role in a future release of *TSSB*, so it's a good habit to set it to match.

- The SHOW SELECTION COUNT option lets us see which parameters we most and least popular for each model.

The key here is that all five models include the SUBSAMPLE 60 PERCEN option. This decrees that a randomly selected 60 percent of the dataset will be used to train the model. If the model is able to find little or no *true* relationship between the predictor candidates and the target, the model will just be learning random noise, and each of the five models will learn a different set of noise points. In this case, the predictors selected will be more or less random, and out-of-sample performance will vary widely, since the out-of-sample predictions will be in a random relationship with the target. But if the combination of the predictor candidates and the linear model is able to find a true, repeatable pattern, this pattern will appear in the randomly selected training sets of all five models. This will result in at least approximately the same predictors being selected for all five models, and at least similar out-of-sample performance. Here is a table showing the out-of-sample profit factor (top row) of each of the five models, and the predictors selected, ordered from most to least popular:

1.081	1.081	1.079	1.082	1.072
INT_10	INT_10	INT_10	INT_10	INT_10
LIN_ATR_5	LIN_ATR_5	LIN_ATR_5	LIN_ATR_5	LIN_ATR_5
MOM_5_100	MOM_5_100	MOM_5_100	MOM_5_100	MOM_5_100
MOM_10_100	MOM_10_100	MOM_10_100	MOM_10_100	MOM_10_100
DINT_10	DINT_10	DINT_10	DINT_10	DINT_10
INT_20	LIN_ATR_15	LIN_ATR_15	LIN_ATR_15	LIN_ATR_15
INDDEV_10	LINDEV_40	INT_20	INT_20	INT_20
LIN_ATR_15	INDDEV_10	INDDEV_10	QUADDEV_40	QUADDEV_40
LINDEV_40	INT_20	INDDEV_40	INDDEV_10	INDDEV_10
QUADDEV_40	INDDEV_40	LINDEV_10	INDDEV_40	INDDEV_40
QUADDEV_10	QUADDEV_10	QUADDEV_40	LINDEV_40	LINDEV_40
LINDEV_10	QUADDEV_40	LINDEV_40	LINDEV_10	LINDEV_10
INDDEV_40	LINDEV_10	DINT_20	QUADDEV_10	QUADDEV_10
DINT_20	DINT_20	QUADDEV_10	DINT_20	DINT_20
ADX30	ADX30	ADX30	ADX30	ADX30
ADX15	ADX15	ADX15	ADX15	ADX15

Note the remarkable uniformity among the five models, especially for the most and least popular indicators. Of course, with a 60 percent overlap, the models do have many training cases in common. Still, this degree of agreement among the models is encouraging. We don't yet know how much practical performance we will be able to get out of this predictive system, but the news is good so far.

Another interesting observation is that the two ADX indicators are at the bottom of the list for all five models. Because ADXis nondirectional (it indicates trend strength, not direction), one would expect ADX15 and ADX30 to have no predictive power whatsoever with a linear model, and this is exactly what we are seeing. But if the models were learning only random noise instead of true patterns, ADX would have just as much 'power' as directional variables. Hence ADX15 and ADX30 could appear anywhere in the lists. Yet all five models are ranking all of the directional variables above ADX. Nice!

As a final note, this test was re-run with a subsample rate of just 20 percent. Obviously, this produces tremendous diversity in the training sets. Nonetheless, for the five models, out-of-sample profit factor ranged from 1.069 to 1.088, a fairly narrow range. Moreover, the most, second-most, and third-most popular variables were INT10, LIN_ATR_5, and MOM_5_100 respectively for all five models! This is remarkable conformity and is very encouraging.

How Long Does the Model Hold Up?

Before one gets too far along in the development process, it is useful to see how far into the future a trained model holds onto its predictive power. Whenever computer time allows, this should also be the final step in development, although resource limitations often prevent this final step. But it should nearly always be feasible to test at least some of the predictive components early in the process. Information about the usable life of a model before it needs retraining is obviously important to deployment. But it also plays a key role in the development process. If one finds that a model needs frequent retraining, one should account for that in any experimentation, or else revise the plan.

TSSB contains a facility to get a good idea about how long a model retains its predictive power. This is done by running two or three different walkforward fold (out-of-sample) sizes. At a minimum, fold sizes of a year and a month should be tested. If at all possible, a fold size of a day should also be used to provide an optimal-performance baseline. Here is a script file to test the linear model and predictor candidate set which form the main foundation of the predictive system:

```
RETAIN YEARS 1900 THROUGH 2010 ;
READ DATABASE "StandAlone.dat" "StandAlone.fam" ;

MODEL LINMOD IS LINREG [
   INPUT = [ LIN_ATR_5 - DINT_20 ]
   OUTPUT = RETURN
   RESTRAIN PREDICTED
   MAX STEPWISE = 2
   STEPWISE RETENTION = 10
   FRACTILE THRESHOLD
   CRITERION = BALANCED_10
   MIN CRITERION FRACTION = 0.1
   ] ;

WALK FORWARD BY YEAR 5 1999 ;
WALK FORWARD BY MONTH 60 199901 ;
WALK FORWARD BY DAY 1250 19990101 ;
```

The testing walks forward always using about five years of training data. This is 60 months, or 1250 days. Out-of-sample profit factors were as follows:

By year: 1.070
By month: 1.085
By day: 1.085

The first thing many people will note is that these profit factors are not terribly impressive. However, one must remember that this is a very special trading system:

it's balanced. Every day it *must* take a long position in ten percent of its markets, and it must also take a short position in another ten percent of its markets. This is a double-edged sword. It has the disadvantage of often requiring the trader to open long positions when the prediction is for a strong downward move, and similarly taking short positions when the prediction is for an upward move. Systems that only need to take a position in the direction of the predicted move have an obvious advantage! But the payoff is substantial, because a balanced system is far more immune to mass market moves than unbalanced systems. Thus, one can safely (at least most of the time!) use leverage to take on far larger positions than one can with an unbalanced system. This multiplies the potential profits without similarly multiplying the potential losses (except in exceptional circumstances).

The important information in these three numbers is that the model appears to hold up well for at least a month, but it loses considerable power when asked to survive for a year without retraining. It would be beneficial to conduct all further experiments using a monthly walkforward. Unfortunately, computer resources for this demonstration (as is often the case in real life) preclude monthly retraining.

Finding Models for a Committee

It almost goes without saying that modern model-based trading systems use a committee of some sort for making the final trading decisions. The performance of a committee of models is almost always superior to that of a single model, often extremely so. We already discussed here some general guidelines for committee formation. This example will use a few simple linear models, always a good choice, along with a single nonlinear model, to be committee members.

If one is using the one-shot approach discussed in the prior section, then the EXCLUSION GROUP option is a good way to find committee members. Howeve in an experimentation environment, the FIND GROUPS command used on the entir training period may be preferable. The disadvantage of this approach is the walkforward results in the experimentation phase will be optimistically biased due to using all of the data to find the best component models. But this is not a disaster, because the final test involving the excluded data will still be unbiased, and it's nice to have the stability of knowing in advance exactly which indicators will be used in all component models. With the EXCLUSION GROUP option, differe indicators will generally be used in different folds, and none of these indicator sets will benefit from having access to all training data simultaneously. Of course, this is a judgement call with no hard answer; the FIND GROUPS method seemed bette for this demonstration. Thus, we use the following script file:

```
RETAIN YEARS 1900 THROUGH 2010 ;
READ DATABASE "StandAlone.dat" "StandAlone.fam" ;

FIND GROUPS [
   INPUT = [ LIN_ATR_5 - DINT_20 ]
   OUTPUT = RETURN
   MAX STEPWISE = 2
   STEPWISE RETENTION = 10
   MAX GROUPS = 3
   GROUP RSQUARE CUTOFF = 0.8
   FRACTILE THRESHOLD
   CRITERION = BALANCED_10
   MIN CRITERION FRACTION = 0.1
   RESTRAIN PREDICTED
   ] ;
```

Notice that this script file begins by retaining only years through 2010. The complete dataset runs through the end of 2011, which will be the test (out-of-sample) year. The data begins in 1988, but it's also safe to begin the retained period at any year prior to this, such as 1900.

The FIND GROUPS command uses all candidate indicators, setting a limit of tw

predictors in each model. A STEPWISE RETENTION of 10 is reasonable, lar enough to guarantee thorough testing of various indicator combinations, yet not so large as to require excessive computation time. The GROUP RSQUARE CUTOF of 0.8 is also reasonable.

Here are the three indicator groups found. We will use these three linear models, along with a quadratic model for token nonlinearity, in a committee.

```
----------> Group 1 <----------
Regression coefficients:
         -0.000747  LIN_ATR_5
         -0.000872  INT_10
          0.023183  CONSTANT

----------> Group 2 <----------
Regression coefficients:
         -0.000732  INT_20
         -0.000826  DINT_10
          0.022511  CONSTANT

----------> Group 3 <----------
Regression coefficients:
         -0.000981  MOM_10_100
         -0.000212  LINDEV_10
          0.023257  CONSTANT
```

The Trading System

Development of the actual trading system is where the majority of experimentation takes place. Typically, one would try several model types, several sets of predictors, and several committee types. Because in this demonstration we have held out the entire year 2011 for testing, we are free to experiment and tweak performance as much as we like. For reasons of economy this tutorial will limit experimentation to varying only the committee type, but in an actual application other variations should be attempted.

As has been mentioned earlier, it is nearly always best to use powerful predictors in a weak model, as opposed to weak predictors in a powerful model. For this reason, three linear models will form the foundation of this trading system. However, one should usually consider the possibility that some nonlinearity is present in the predictor set, and hence include at least one nonlinear model in the design. A quadratic model generally has an excellent combination of training speed and nonlinear predictive power, so that type is chosen here. Here is the entire trading system script file:

```
RETAIN YEARS 1900 THROUGH 2010 ;
READ DATABASE "StandAlone.dat" "StandAlone.fam" ;

MODEL LIN1 IS LINREG [
   INPUT = [ LIN_ATR_5 INT_10 ]
   OUTPUT = RETURN
   RESTRAIN PREDICTED
   MAX STEPWISE = 0
   STEPWISE RETENTION = 10
   FRACTILE THRESHOLD
   CRITERION = BALANCED_10
   MIN CRITERION FRACTION = 0.1
   ] ;

MODEL LIN2 IS LINREG [
   INPUT = [ INT_20 DINT_10 ]
   OUTPUT = RETURN
   RESTRAIN PREDICTED
   MAX STEPWISE = 0
   STEPWISE RETENTION = 10
   FRACTILE THRESHOLD
   CRITERION = BALANCED_10
   MIN CRITERION FRACTION = 0.1
   ] ;
```

```
MODEL LIN3 IS LINREG [
   INPUT = [ MOM_10_100 LINDEV_10 ]
   OUTPUT = RETURN
   RESTRAIN PREDICTED
   MAX STEPWISE = 0
   STEPWISE RETENTION = 10
   FRACTILE THRESHOLD
   CRITERION = BALANCED_10
   MIN CRITERION FRACTION = 0.1
   ] ;

MODEL QUAD1 IS QUADRATIC [
   INPUT = [ LIN_ATR_5 - DINT_20 ]
   OUTPUT = RETURN
   RESTRAIN PREDICTED
   MAX STEPWISE = 2
   STEPWISE RETENTION = 10
   FRACTILE THRESHOLD
   CRITERION = BALANCED_10
   MIN CRITERION FRACTION = 0.1
   ] ;

COMMITTEE COMM1 IS CONSTRAINED [
   INPUT = [ LIN1 LIN2 LIN3 QUAD1 ]
   OUTPUT = RETURN
   RESTRAIN PREDICTED
   MAX STEPWISE = 4
   FRACTILE THRESHOLD
   CRITERION = BALANCED_10
   MIN CRITERION FRACTION = 0.1
   MCP TEST = 1000
   ] ;

COMMITTEE COMM2 IS CONSTRAINED [
   INPUT = [ LIN1 LIN2 LIN3 QUAD1 ]
   OUTPUT = RETURN
   RESTRAIN PREDICTED
   MAX STEPWISE = 0
   FRACTILE THRESHOLD
   CRITERION = BALANCED_10
   MIN CRITERION FRACTION = 0.1
   MCP TEST = 1000
   ] ;
```

```
COMMITTEE COMM3 IS AVERAGE [
   INPUT = [ LIN1 LIN2 LIN3 QUAD1 ]
   OUTPUT = RETURN
   RESTRAIN PREDICTED
   MAX STEPWISE = 0
   FRACTILE THRESHOLD
   CRITERION = BALANCED_10
   MIN CRITERION FRACTION = 0.1
   MCP TEST = 1000
   ] ;

WALK FORWARD BY YEAR 5 1999 ;
```

The first three models are the three linear models discovered with the FIND GROUPS command discussed earlier. In order to accommodate nonlinearity, a QUADRATIC model is also included. Finally, three committees are employed. The first is a CONSTRAINED committee which uses stepwise selection to choose the models that it will use. The second is also CONSTRAINED, but all four models are forced to be used. Finally, an AVERAGE committee is used to simply average the predictions of the four models. These are all tested with yearly walkforward and a five-year training period. Pooled out-of-sample profit factors are as follows:

```
LIN1        1.093
LIN2        1.070
LIN3        1.082
QUAD1       1.072
COMM1       1.077
COMM2       1.091
COMM3       1.092
```

We now have a bit of a quandary: the first linear model is the best performer, though only by a trivial amount. If the difference were larger, it would be tempting to base the trading system entirely on this one model. However, in this nearly tied situation, it is scary to base trading on just two predictors. Thus, we choose to base trade decisions on COMM3, the average of the four models' predictions. Keep in mind that these performance figures are subject to random variation, so there is no guarantee that the observed rank ordering will exactly correspond to the rank ordering of true quality. For this reason, it seems good to favor the known general superiority of committees over single models.

The ordering of committee performance is no great surprise. COMM1 is the only one of the three that uses stepwise selection to choose its component models. This additional power encourages overfitting. COMM2 has adjustable coefficients, whichmakes it more powerful than COMM3 which is a simple average. So we see that out-of-sample performance is inversely related to committee power.

The Final Test

After the developer has performed numerous experiments to learn as much as possible about the predictors and target, and has tweaked performance to perfection, it is time to take a deep breath and test the complete trading system on the data that was withheld during the experimentation phase. The script file for this is identical to that used in the development and shown in the prior section, with one exception. It does a single walkforward year, 2011:

```
WALK FORWARD BY YEAR 5 2011 ;
```

Results for this test year are as follows:

```
LIN1        1.043
LIN2        1.045
LIN3        1.015
QUAD1       1.032
COMM1       1.041
COMM2       1.019
COMM3       1.040
```

This is a considerable disappointment. At least all models and committees made money in the test year, but it was not much. Also, it turns out that the decision to use COMM3 rather than LIN1, even though LIN1 was slightly superior, may have bee a bad choice, because LIN1 is also better than COMM3 in this test year.

This brings up a subtle but important issue. We already know that LIN1 was superior to COMM3 in the development phase, and now we see that it is alsc superior in the test year. So when we begin to trade the system for real money (assuming that we choose to do so), should we trade LIN1 instead of COMM3? I fact, LIN2 was the best of all in the test year. Maybe we should choose to trade it.

There is no simple answer to this dilemma, but there are some vital issues to consider. Perhaps the most important issue is that we chose *in advance* to trade COMM3, and we just tested it on virgin data. The implication is that its performance of 1.040 is an unbiased estimate of its expected future performance. If we change our mind after seeing that LIN1 (or LIN2) is superior, we can no longer say that the observed performance of LIN1, 1.043, or that of LIN2, 1.045, is an unbiased estimate of future performance. These superior figures have been inflated by selection bias, the result of the real possibility that random good luck intervened to subvert the natural ordering of performances. By definition, good luck will not continue for long, if at all.

There is yet another aspect of this issue. Suppose we had not done any

experimentation. Instead, suppose we had taken the (nearly) one-shot approach described early in this chapter. But also suppose that instead of just one shot, we had tried several models and walked them forward on all available data, and then chosen the best performer for real-money trading. Again, the performance of the best model would be optimistically inflated by selection bias. Nonetheless, under normal conditions it would be in our best interest to trade the best performer, because it is most likely to be the truly best. We should just understand that the performance we will obtain in the future will probably not be as good as what we observed, due to selection bias.

Does this mean that in this case we should trade LIN1 (or perhaps LIN2) instead o COMM3? The answer is no, because we are in a different situation here. In ou current situation, we are not looking at just the relative performances in the test year. We also have experimentation-phase walkforward results pooled from 1999 through 2010, and those results showed COMM3 to be a good performer, practically tied with LIN1 and much superior to LIN2 (which was the grand*loser* then). We would be foolish to discount those results, especially since they are based on 12 years of pooled trade results, and the figures obtained with this final test are for a single year.

Does this leave you craving more rigorous decision-making rules? Indeed it should, because there are few hard-and-fast rules. Sometimes you have to fly by the seat of your pants and weigh alternatives without the benefit of rigorous theory. But we'll leave you with two definitive rules that should always be given strong consideration:

1) The more years that go into a study, the more reliable are the results. Rank orderings produced by 12 years of pooled data are more stable than those produced by a single test year.

2) The moment you examine competing models and choose the best you have introduced selection bias. You have favored not only the truly best model, but also the luckiest model. Thus, the performance of the chosen best will, on average, be optimistic.

If you keep these two rules in mind you should be able to navigate the development waters with reasonable safety.

Trade Simulation and Portfolios

You may follow a TRAIN, WALK FORWARD, or CROSS VALIDATE comma with the PRESERVE PREDICTIONS command, and follow this with a TRAI SIMULATOR command. This provides more extensive performance statistics tha are provided by default, and it also allows you to execute slightly more sophisticated trading rules than those inherent in default operation. (Recall that the default trading rule in TSSB is based on a simple threshold: if the model's prediction exceeds an upper (long) or lower (short) threshold a new position is taken and the trade's realized return is either the value of the target variable or a designated profit variable if the target can not interpreted as a gain or loss.)

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 alway required, is relevant only if multiple markets are present. If you choose GLOBAL only a summary with all markets pooled (aggregate performance) 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* threshold value, and it will be closed when the prediction drops below the *Maintain* threshold value. In other words, a position is entered if the prediction is equal to or more extreme than the *Enter* value and it is held so long as the forecast value remains at or beyond the *Maintain* value. Obviously, the *Maintain* value must not exceed the *Enter* value. Negative values are permitted, in which case the long position will be

maintained even though the prediction is negative, as long as the prediction equals or exceeds the *Maintain* value.

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 found during training. 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(Distance) - 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. This option is recommended 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 historie

be present! If you are using a READ DATABASE command to read precompute variables, you must *precede* the READ DATABASE command with REA 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.

We conclude with a two-part example of computing equity and writing it to a text file. It is certainly legal to perform all operations in a single script file. However, it is more efficient to split the operations into a preliminary file that computes and saves all predictors and targets, and a subsequent script file that reads this dataset, fits the model, and writes the equity. Here is the initial file, with the variable definition file shown after:

```
READ MARKET LIST "D:\BOOSTER\TEST\OEX_VIX.TXT" ;
INDEX IS VIX ;
READ MARKET HISTORIES "E:\SP100\VIX.TXT" ;
READ VARIABLE LIST "D:\BOOSTER\TEST\VIX_STUDY_VARS.TXT" ;


TRANSFORM VIX10_NODIFF IS ARMA [
   SERIES = @CLOSE:VIX
   ARMA WINDOW = 50
   ARMA OUTPUT = STANDARDIZED SHOCK
   ARMA AR = 1
   ARMA MA = 0
   ] ;

WRITE DATABASE "ARMA_EQUITY.DAT" "ARMA_EQUITY.FAM" ;
```

Here is the variable definition file, VIX_STUDY_VARS.TXT:

```
PURIF_DAVE2: PURIFIED INDEX 2 300 2 3 6 11 22 44 65 130 260
RETURN: NEXT DAY ATR RETURN 250
HITMISS16: HIT OR MISS 16.0 16.0 1 0
```

This variable definition file computes a purified index, which we may choose to use in a subsequent study. It also computes two types of return.

There is an extremely important and subtle aspect to this variable definition file. Whenever possible, *TSSB* assumes that all markets in the market list are to be traded. However, in some situations it is clear that an index is not to be traded. This is the case when the MINUS INDEX modifier is used for an indicator, c when a PURIFIED INDEX is computed. Thus, in this example the VIX market w be excluded from trading, which is good. (In the unusual situation that you want to trade VIX along with OEX you could copy the VIX market to another market fil and include this copied file in the market list.) In this example, if you knew in advance that you would have no use for a purified index indicator and did not wish to compute it, you should include only OEX in the market list. If you also included VIX in the market list, the program would assume that VIX is just another tradeable market and do so!

Once the database of indicator and target candidates is written by this preliminary script file, this database can be used in a model, and the model's equity curve can we written. The script file to do this is as follows:

```
READ MARKET LIST "D:\BOOSTER\TEST\SYM_OEX.TXT" ;
READ MARKET HISTORIES "E:\SP100\OEX.TXT" ;
READ DATABASE "ARMA_EQUITY.DAT" "ARMA_EQUITY.FAM" ;
```

```
MODEL NODIFF IS LINREG [
   INPUT = [ VIX10_NODIFF ]
   OUTPUT = HITMISS16
   MAX STEPWISE = 1
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   ] ;

WALK FORWARD BY YEAR 5 2000 ;
PRESERVE PREDICTIONS ;
TRADE SIMULATOR NODIFF GLOBAL DUAL (TRAINED 1.0 TRAINED
     1.0) POINTS ;
WRITE EQUITY "EQTY.TXT" ;
```

The first thing to notice is that in addition to reading the database created by the prior script file, we also need to read the OEX market. If we were just training and testing models this would not be necessary. But since we are invoking the TRADE SIMULATOR, it needs market history. The SYM_OEX.TXT file names just on market, OEX. We do not want to get VIX into the mix at this point. It served its purpose in computing indicators, and now we are done with it. We do not want to trade it!

The model is walked forward, and the PRESERVE PREDICTIONS comma preserves its out-of-sample predictions in the database as just another variable. The TRADE SIMULATOR is invoked for this model. The GLOBAL option is us even though there is just one market, because this is required for equity computations. The DUAL option specifies that we will count both long and shor positions. The TRAINED option for both long and short positions says that we wil use the trained threshold for each fold, and 1.0 is the factor for holding a position. This implies that we will open and close positions according to the prediction relative to the trained threshold. All equities will be in terms of raw point values.

Finally, we write the equity file.

Performance Measures

For individual markets (printed if the BY MARKET option is specified), the firs 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 or ATR) 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 or ATR) 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. Thi 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 i 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 positio 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 they would cause more problems than they solve. 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 (here).

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 t produce a single combined equity curve. The resulting curve can then be displayed or written to a file exactly like the curve produced by a TRADE SIMULATOI command.

Note that the FILE PORTFOLIO command does not need to be in the same scri file as the TRADE SIMULATOR commands that produced the equity curves. Thi is because all necessary information is contained in the equity curve files that the FILE PORTFOLIO command reads. The user can create a short, simple script fi that contains nothing but one or more FILE 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, here.

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

planned 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, here. 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, 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. Ratio-based performance figures such as 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 entire available 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 uses 100 trials to find a optimal portfolio of two systems (out of three candidates) and walks it forward from 2005 to evaluate the degree to which optimality holds up. Also, the *EQUALIZE* option is used to make visual interpretation of results 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 here 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 dailyprofits (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 equity 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 be optimistically biased on average.

OOS - The performance of this portfolio for the current walkforward year, the year immediately following the training (portfolio optimization) period. This out-of-sample quantity is an unbiased estimate of the true performance of the optimal portfolio (assuming that the individual portfolio componentss themselves are unbiased). 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 pooled performance of trading all systems in the current walkforward year. Comparing this to the *OOS* performance shows the difference

between simply trading all systems versus trading just the previously selected optimal portfolio.

After walkforward is complete, all out-of-sample data is pooled into a single collection 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.

A Portfolio Example

We conclude the discussion of file portfolios with a complete example. This example employs five predictive models:

MOD1LONG - Linear model with indicators selected based on long profit factor

MOD1SHORT - Linear model with indicators selected based on short profit factor

MOD1DUAL - Linear model with indicators selected based on combined profit factor

MOD2DUAL - Linear model with indicators selected based on combined profit factor, looking back a short distance

MOD3DUAL - Linear model with indicators selected based on combined profit factor, looking back a longer distance

Here is the script file through the model definitions. We will not bother reproducing the variable definition file EQUITY.TXT here, as files of this type have been treated extensively already.

```
READ MARKET LIST "D:\BOOSTER\TEST\SYM_OEX.TXT" ;
READ MARKET HISTORIES "E:\SP100\OEX.TXT" ;
CLEAN RAW DATA 0.6 ;
READ VARIABLE LIST "D:\BOOSTER\TEST\EQUITY.TXT" ;
```

```
MODEL MOD1LONG IS LINREG [
   INPUT = [ LIN_ATR_5 QUA_ATR_5 CUB_ATR_5 LIN_ATR_15
             QUA_ATR_15 CUB_ATR_15 ]
   OUTPUT = RETURN
   MAX STEPWISE = 2
   CRITERION = LONG PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   ] ;

MODEL MOD1SHORT IS LINREG [
   INPUT = [ LIN_ATR_5 QUA_ATR_5 CUB_ATR_5 LIN_ATR_15
             QUA_ATR_15 CUB_ATR_15 ]
   OUTPUT = RETURN
   MAX STEPWISE = 2
   CRITERION = SHORT PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   ] ;

MODEL MOD1DUAL IS LINREG [
   INPUT = [ LIN_ATR_5 QUA_ATR_5 CUB_ATR_5 LIN_ATR_15
             QUA_ATR_15 CUB_ATR_15 ]
   OUTPUT = RETURN
   MAX STEPWISE = 2
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   ] ;

MODEL MOD2 IS LINREG [
   INPUT = [ LIN_ATR_5 QUA_ATR_5 CUB_ATR_5 ]
   OUTPUT = RETURN
   MAX STEPWISE = 2
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   ] ;

MODEL MOD3 IS LINREG [
   INPUT = [ LIN_ATR_15 QUA_ATR_15 CUB_ATR_15 ]
   OUTPUT = RETURN
   MAX STEPWISE = 2
   CRITERION = PROFIT FACTOR
   MIN CRITERION FRACTION = 0.1
   ] ;
```

Next we have the commands that walk the models forward, preserve the predictions, and execute the trade simulator for each model. The data which will later be used for portfolio building and testing will begin at the year 2000, the first model walkforward year. We could, of course, TRAIN the models instead of walking them forward, which would let portfolio building examine the entire dataset. However, building and testing portfolios on in-sample data makes little

sense, because the returns are optimistically biased. By building portfolios based on the models' out-of-sample trades we are using honest, unbiased trade results to find and test optimal portfolios. Here are these commands, followed by the AUDIT.LOG output produced by them:

```
WALK FORWARD BY YEAR 5 2000 ;
PRESERVE PREDICTIONS ;

TRADE SIMULATOR MOD1LONG GLOBAL LONG (TRAINED 1.0) ATR 250
;
WRITE EQUITY "LONG1.TXT" ;

TRADE SIMULATOR MOD1SHORT GLOBAL SHORT (TRAINED 1.0)ATR
250;
WRITE EQUITY "SHORT1.TXT" ;

TRADE SIMULATOR MOD1DUAL GLOBAL DUAL (TRAINED 1.0 TRAINED
1.0) ATR 250 ;
WRITE EQUITY "DUAL1.TXT" ;

TRADE SIMULATOR MOD2 GLOBAL DUAL (TRAINED 1.0 TRAINED 1.0)
ATR 250 ;
WRITE EQUITY "DUAL2.TXT" ;

TRADE SIMULATOR MOD3 GLOBAL DUAL (TRAINED 1.0 TRAINED 1.0)
ATR 250 ;
WRITE EQUITY "DUAL3.TXT" ;


MODEL MOD1LONG
Bars=6025  Long=347 (5.76%) Short=0 (0.00%)
Total return = 20.026 ATR units
Profit factor = 1.1766
Maximum drawdown = 11.649 on 20080804
   289 bars to bottom and 736 bars to recovery


MODEL MOD1SHORT
Bars=6025  Long=0 (0.00%) Short=396 (6.57%)
Total return = -5.276 ATR units
Profit factor = 0.9605
Maximum drawdown = 23.633 on 20070802
   1903 bars to bottom and never recovered


MODEL MOD1DUAL
Bars=6025  Long=382 (6.34%) Short=362 (6.01%)
Total return = 16.444 ATR units
Profit factor = 1.0673
Maximum drawdown = 23.485 on 20070816
   1907 bars to bottom and 2203 bars to recovery
```

```
MODEL MOD2
Bars=6025  Long=457 (7.59%) Short=506 (8.40%)
Total return = -7.200 ATR units
Profit factor = 0.9769
Maximum drawdown = 28.953 on 20070801
   1898 bars to bottom and 2236 bars to recovery

MODEL MOD3
Bars=6025  Long=462 (7.67%) Short=465 (7.72%)
Total return = 28.977 ATR units
Profit factor = 1.0969
Maximum drawdown = 15.197 on 20060921
   1023 bars to bottom and 1223 bars to recovery
```

We now consider two different portfolios. The first is a simple combination of the *MOD1LONG* and *MOD1SHORT* models. This would be a common operation for many users. It is solidly established that trying to find a single model that performs well for both long and short trades is not wise. Performance is almost always improved by using separate models for long and short trades. We can then use the *PORTFOLIO* command to compute the net performance obtained by trading both models simultaneously. Here is the command to do so, followed by the AUDIT.LOG file output produced by this command:

```
PORTFOLIO MOD1_NET [
   EQUITY FILE = [ "LONG1.TXT" "SHORT1.TXT" ]
   ] ;

-------------------------------------------------------------------------
Processing Portfolio MOD1_NET
-------------------------------------------------------------------------

Basic profit statistics for 3004 records
System  Minimum  Maximum   Mean   StdDev   Rng/Std  Sharpe   PF   Drawdown Recovery
  1    -2.861   3.588   0.0067   0.319    20.248   0.332  1.177   11.649  736
  2    -2.842   3.531  -0.0018   0.329    19.382  -0.085  0.961   23.633  Never
 All   -2.861   3.588   0.0049   0.418    15.412   0.186  1.069   16.683  433
```

In this example we see that the long system (System 1 because LONG1.TXT i named first in the file list) is a decent winner, while the short system actually loses money. Thus, it's not surprising that the net performance of the two systems is inferior to that of the long-only system. We should not judge this example too harshly, because the indicators provided to the models are limited, just a few simple trends, to keep things simple. More exotic indicators improve performance a lot.

Note, by the way, that the trade simulator reported 6025 bars, while this portfolio reports 3004 records (one equity record per bar) in the equity files. This is because the market file being traded, OEX, contains prices for 6025 days. But the portfolic study is the out-of-sample walkforward data, which begins about halfway through the market history. When the equity files are created, no records are written until

the first trade is signaled. This eliminates having an equity file inflated by a large number of leading records whose equity is zero.

The second PORTFOLIO example is more complex because it demonstrate optimization and walkforward testing of the optimal portfolios. Here are the script file commands that read the equity files that were previously written (see here) and do the optimization and testing:

```
PORTFOLIO PORT_OF_TWO [
  EQUITY FILE = [ "DUAL1.TXT" "DUAL2.TXT" "DUAL3.TXT" ]
  EQUALIZE
  OPTIMIZE = 2 100
  WALK FORWARD FROM 2005
  ] ;
```

This defines a portfolio called *PORT_OF_TWO*. It reads the equity curve files for the three long/short models. The *EQUALIZE* option is invoked to make reading the table of results easier. The *OPTIMIZE* option specifies that our portfolio will consist of two of these three models. It also says that 100 trials will be used to find the best portfolio, a value that is ridiculously excessive. There are only three ways that one can choose two models out of three! Still, the selection process is very fast, so making the number of trials huge is cheap insurance against randomness playing a cruel trick and causing the program to miss the optimum. Finally, the walkforward process begins with 2005 as the first out-of-sample year.

It's worth exploring in a little more detail exactly what happens when we combine the walkforward model training/testing with the walkforward portfolio training/testing. This is the best, most honest approach, but it is a little complicated. Here is a summary of the steps that are happening behind the scenes:

Model training ends with 1999; the models are evaluated in 2000.
Model training ends with 2000; the models are evaluated in 2001.
Model training ends with 2001; the models are evaluated in 2002.
Model training ends with 2002; the models are evaluated in 2003.
Model training ends with 2003; the models are evaluated in 2004.
Portfolio optimization ends with 2004; the optimal portfolio is evaluated in 2005.
Model training ends with 2004; the models are evaluated in 2005.
Portfolio optimization ends with 2005; the optimal portfolio is evaluated in 2006.
Model training ends with 2005; the models are evaluated in 2006.
Portfolio optimization ends with 2006; the optimal portfolio is evaluated in 2007.
Model training ends with 2006; the models are evaluated in 2007.
Portfolio optimization ends with 2007; the optimal portfolio is evaluated in 2008.
Et cetera, until the available market history is exhausted.

Notice that portfolio out-of-sample testing always stays one year ahead of the

model out-of-sample testing. This ensures that the optimal portfolio is always being based on out-of-sample performance of the models.

The following output is generated by this example. We will break it up into sections for easier presentation.

```
---------------------------------------------------------------------
Processing Portfolio PORT_OF_TWO
---------------------------------------------------------------------
   Keeping 2 systems in portfolio
   Using 100 replications
   Equalizing portfolio to one contract total
   First walkforward year is 2005

Basic profit statistics for 3011 records
System  Minimum  Maximum   Mean   StdDev   Rng/Std  Sharpe   PF   Drawdown Recovery
   1    -5.301    3.588   0.0055   0.468   19.001   0.185  1.067   23.485   2203
   2    -5.301    3.588  -0.0024   0.502   17.720  -0.076  0.977   28.953   2236
   3    -5.301    4.891   0.0096   0.526   19.376   0.290  1.097   15.197   1223
 All    -5.301    3.588   0.0042   0.354   25.099   0.190  1.054   16.312   2202
 Opt    -5.301    3.588   0.0075   0.393   22.610   0.305  1.097   14.732   2026

Selected:
    1
    3

Not selected:
    2
```

The first three lines report the performance statistics for the three component systems. System 2 is a money-loser. If we trade all three systems we obtain poor results due to the unfavorable presence of System 2. By trading just the best pair of systems, we obtain a portfolio Sharpe ratio of 0.305, which is better than any individual component and also better than trading all three. Note that the *days to recovery* reported here for the individual systems can sometimes be a day or so different from that reported by the trade simulator. This is due to alignment issues related to matching trade dates among multiple systems. It should never be anything more than a trivial difference.

The table just shown concerns the optimal portfolio derived from the entire available model walkforward period, which begins in the year 2000. The next step is to walk the optimization process forward. Each year is printed in the AUDIT.LOG file, but for economy we will show only the first year here.

```
For walkforward OOS year 2005, trained best portfolio is: 2 3
 Data  Minimum  Maximum    Mean    StdDev  Rng/Std  Sharpe   PF   Drawdown Recovery
 Trn   -2.431    2.416   -0.0044   0.317  15.292   -0.221  0.941   12.670   Never
 OOS   -0.891    0.934   -0.0050   0.229   7.951   -0.346  0.919    4.106   Never
 All   -0.989    1.245   -0.0015   0.223   9.999   -0.107  0.974    4.020   Never
```

In the table above we see that based on model out-of-sample data ending in 2004, the best portfolio, consisting of Systems 2 and 3, had a net loss. It's Sharpe ratio is -0.221. When this two-system portfolio is tested in 2005 it obtains an even worse Sharpe ratio of -0.346. In this year (2005) we would actually have been best off trading all three systems, because this portfolio has a Sharpe ratio of -0.107.

After all walkforward years have been tested, a summary table is computed by pooling all of the OOS data. Here is this table:

Data	Minimum	Maximum	Mean	StdDev	Rng/Std	Sharpe	PF	Drawdown	Recovery
Opt	-5.301	3.588	0.0138	0.431	20.621	0.509	1.155	10.086	36
All	-5.301	3.588	0.0124	0.388	22.931	0.509	1.148	9.451	Never

In this case, the Sharpe ratio happens (purely by coincidence) to be the same whether we traded the optimal portfolio or all systems in each test year. We do see that the optimal portfolio had a slightly higher mean return per day, as well as a slightly higher profit factor.

It must be emphasized that this optimization example has been kept deliberately simple for the purposes of this tutorial. In a real application, the component models would use far more sophisticated indicators and thereby have improved performance. Also, we might have a dozen candidate models, or perhaps even hundreds, and choose a substantial fraction of them for the optimal portfolio. In such a situation, performance will likely be much improved over that obtained in this example.

Important note on WALK FORWARD folds: The folds in the 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 ir 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, inmost 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 beer defined. Portfolios are ignored for CROSS VALIDATE commands. They ar 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 Portfoli 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 here, 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 GROU 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 using 2007-2009 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 PORTFOLI 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
Pooled -13.694    14.352   0.1097   1.591   17.630   1.095  1.396  29.642     234
Opt sub -6.847     7.176   0.0768   0.802   17.488   1.521  1.697  11.447     213
Opt 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