# Table of Contents

## Foreword

### Part I Introducing Optimax

1 Why Optimax? .......................................................................................................................... 3

### Part II Introducing Genetic Algorithms ................................................................. 6

1 Genetics and Evolution ........................................................................................................ 6
2 Genetics and Mating .............................................................................................................. 8
3 Selecting A Mate .................................................................................................................... 11
4 Fitness ................................................................................................................................ 16
5 Generations .......................................................................................................................... 16
6 Convergence ........................................................................................................................ 17
7 Curve Fitting and Adaptation ............................................................................................... 18
8 Optimization Results ........................................................................................................... 20
9 Summary ............................................................................................................................... 22

### 2 Optimizing ......................................................................................................................... 22

#### Preparing A Strategy ......................................................................................................... 23

1 Adding Optimax Calls Manually ......................................................................................... 24
2 Preamble ............................................................................................................................... 26
3 Converted Inputs .................................................................................................................. 26
4 Prolog .................................................................................................................................. 26
5 Your Strategy ....................................................................................................................... 27
6 Epilog .................................................................................................................................. 28
7 Enhancing the Epilog .......................................................................................................... 29

1 Adding Optimax Calls Automatically .................................................................................. 32
2 Required Conventions ......................................................................................................... 32
3 Procedure ............................................................................................................................. 33

#### Setting Optimization Parameters .................................................................................... 34

1 Inputs ................................................................................................................................... 35
2 Search Types ....................................................................................................................... 38
3 Changeling Methods .......................................................................................................... 38
4 Genetic Options .................................................................................................................. 39
5 Price Modulation ............................................................................................................... 41
6 Modulating Multiple Streams ............................................................................................. 47

#### Running Optimizations ................................................................................................... 48

1 Evaluating Results ............................................................................................................... 51
2 The Equity Grid .................................................................................................................... 51
3 The Inputs Grid .................................................................................................................... 53
4 The Inputs Grid Detail Canvas .............................................................................................. 54
5 The Search Space Indicator ................................................................................................. 63
6 The Input Space Indicators ................................................................................................. 65
7 The Equity Detail Report .................................................................................................... 69
8 Displaying Trades in TradeStation ..................................................................................... 72
9 Covering the Terrain .......................................................................................................... 72
10 Evaluation In a Nutshell ...................................................................................................... 73
11 Managing Optimizations .................................................................................................. 75
12 Restrictions ........................................................................................................................ 79

©2004 Modus Novus Inc.
1 Introducing Optimax

Welcome to the Optimax User Guide and thank you taking the time to learn about our ground-breaking optimization product!

If you have any questions that this documentation doesn't answer, please visit our website at www.modus-novus.com/optimax or drop us a line. We're anxious to answer them for you. You can email us anytime at optimax@modus-novus.com.

What is Optimax?
Optimax is a stand-alone program that runs on your PC and works with TradeStation 7 during a strategy optimization to super-charge its' back-testing and optimization capabilities.

Optimax works with TradeStation to control it and display results as the optimization runs.

Optimax assembles a set of easy-to-read views of your strategy's behavior that quickly show you how it responds as the inputs and price series change. The biggest Optimax advantage comes from using our powerful genetic algorithms that slash your optimization times by factors of thousands, and our extremely popular built-in price modulator that modifies the price series dynamically during optimization to drastically reduce the probability of curve-fitting the results.

Optimax was developed by and is only available from Modus Novus, Inc.

1.1 Why Optimax?

Save time
Optimax helps you perform comprehensive optimizations quicker.
Optimizing 10, 20 or even 250 inputs is a matter of hours, not years.

**Save money**
The optimization of a strategy is one of the keys to success and cannot be over-valued. A professional high-performance tool dedicated to this purpose can drastically improve your results and decrease the chances of costly mistakes.

**Concentrate on the important thing - your strategy**
Quickly understand your strategy performance in new ways, not just with the ways TradeStation provides. Optimax is designed for speed and easy of use; it's' intuitive user interface is transparent, straight-forward and shields you from the nuts and bolts of optimization analysis and it's ground breaking genetic algorithms save you time for more strategy-writing, not optimization-waiting.

**All strategies supported**
Take all of your strategies and optimize them with Optimax. It's easy with the pre-tested EasyLanguage functions we provide. Leave nothing to chance - you can optimize them all with Optimax.

**Search options galore**
Whether you need died-in-the-wool exhaustive searches, or want to explore the leading edges of genetic optimization technology, Optimax is the right tool for you. It performs standard exhaustive searches displaying the results in several time-saving formats, and it performs genetic optimizations with the same simple ease-of-use interface and battery of intuitive indicators that professionals want and need.

**Dare to compare**
There aren't any other programs out there that do what Optimax does. Compare Optimax with Rina Systems software or the other add-ins available for TradeStation - what you conclude is up to you. Other tools make a lot of noise about delivering, but Optimax delivers and packs a punch. Forget the fuss about what more to buy with your hard-earned income, Optimax is the one essential add-in every TradeStation user needs.

**Optimizing strategies can be done with the computer you have**
You used to believe this, before you added 10 or more inputs to your
strategy and seriously started wondering how you could fit a Cray into your den or office. Well, give it a try now with Optimax. You'll see a world of difference, and you won't get cussed out by the missus for floor plan violations.

2 Getting Started

What is a Genetic Optimization?
A genetic optimization is like running a TradeStation back testing optimization on a thousand Cray supercomputer computers. Simply put, it's a smart search. TradeStation's current algorithm, the exhaustive search, evaluates every possible combination of strategy inputs and determines the best of them all. It is thorough, and time consuming. A genetic algorithm searches in a much more efficient manner; it samples many different combinations at random to find a set of promising inputs, and then modifies those combinations using methods that mimic biological evolution to determine optimum solutions. When used on problems that involve a great number of combinations, genetic searches are many orders of magnitude faster than exhaustive searches, with results that are less likely to be curve fitted.

How can Optimax be that fast?
Optimax employs genetic algorithms to speed up processing. These work by eliminating calculations that have the lowest probability of being useful. Using this method, the increase in optimization speed can be dramatic.

Genetic algorithms work smarter than exhaustive searches. Instead of checking every possible input combination, genetic algorithms quickly narrow down the desired combinations, finding and focusing on the areas that are most profitable and most stable, enabling you to optimize your strategies in a fraction of the time. With Optimax it is practical to optimize many inputs and combinations; 10, 25, or even 100 inputs resulting in millions of combinations can be optimized in a few hours.

For example, in a test case we optimized a strategy with 16 inputs having a possible 5.184E+21 combinations. On a 2.8GHz system our test strategy took approximately 4.3 seconds per iteration. Using a genetic search that converged in 50 generations and having 100 individuals in each generation, the optimization provided satisfactory results in 6 hours. With these same parameters, an exhaustive search would take 706,849,314,984,576 years, or approximately 706 trillion years.

If you could run TradeStation on a top-of-the-line Cray X1 supercomputer, since TradeStation doesn't support multiple CPUs, the best you could do is use is to use a single processor which would run at 11.55 gigaflops (billions of calculations per second.) The Cray would process the optimization 4,125 times faster than our 2.8GHz machine, and the optimization would still take about 171 billion
If you could afford a Cray, you could probably afford to pay someone to re-write every new version of your strategy in C. If you did this, it would run approximately 1,000 times faster (based on our own experiences,) and could be written to take advantage of the Cray's multiple CPUs. Running your C program on a Cray X1 maxed out at 60 processors the optimization would run about 247.5 million times faster (4125 * 1000 * 60.) Despite this astronomical gain, your optimization would still take 2.8 million years.

It is easy to see from these numbers that the speed and power of Optimax are unmatchable, assuming of course you are happy with the results of the optimization compared to those of an exhaustive search. Considering the fantastic gain in speed, you might be tempted settle for a trade-off in optimization results, but with Optimax you don't have to. In fact, the results obtained are in our opinion, better than those obtained using an exhaustive search because Optimax is much less likely to focus on combinations of parameters in unstable regions; exhaustive searches have no knowledge of near-neighbour performance. In fact, you must spend additional time after an exhaustive search is complete to validate that the results set is stable and the combination is not curve-fitted. With Optimax, a curve-fitted result is much less likely because of the nature of the search algorithm.

If this is so much better, why hasn't TradeStation done it already?
I don't know.

How do I use Optimax?
Optimax runs alongside TradeStation. You run them both, and they communicate with each other via functions that you call from TradeStation.

Strategies must be adapted for use with Optimax. They must call DLL functions at appropriate times to get inputs from and send results to Optimax.

2.1 Introducing Genetic Algorithms

In order to get the most out of Optimax's myriad and powerful capabilities, you will need to understand some of the theory behind genetic algorithms. An introduction to this theory is explained in the following sections.

2.1.1 Genetics and Evolution

Genetic optimization algorithms are essentially smart search algorithms. Given a goal, they will try many combinations in order to reach that goal. The way they create these combinations is based on the science of genetics and evolution.

Evolution - a process in which something passes by degrees to a different stage (especially a more advanced or mature stage); "the evolution of his ideas took
The evolution of species is nature's way of solving problems. Traits are passed on through generations, and with each generation, individuals change - usually a little, sometimes a lot. Depending up on the traits inherited from the parents and a certain amount of randomness inherent in the reproduction mechanism, an individual will have certain capabilities, and the complex interplay of these capabilities will determine the ability of the individual to survive, to attract a mate, and the qualities of the mate they attract. All of these determine something we call the "fitness" of the individual.

**Fitness** - the overall ability of an individual to survive.

Fitness is synonymous with desirability, and not only determines one's ability to survive but also one's ability to attract a mate.

Evolution occurs as organisms that survive breed and pass their combined genetic material on to their offspring and thus pass on the traits that enable survival. In a given environment, organisms that are able to make the most of the environment and are able to adapt to new, unforeseen situations are more likely to survive. In nature, survival usually depends on more than mere strength; it depends upon the interplay of the millions of parameters that make up any complex organism. In the insect, plant and animal kingdoms survival can depend on an innumerable number of qualities such as stealthiness, intelligence, color, smell, eyesight, judgement, speed, height, weight, eating speed, and so on ad infinitum. So, in effect, nature is optimizing an enormous number of analog parameters and arriving at a single characteristic we commonly call "fitness." Thus, the complex interplay of the organism's parameters determines survival.

In Optimax, we employ evolutionary algorithms to mimic the complex mechanisms used in nature to find the optimum inputs for investment strategies. These algorithms are also known as genetic algorithms, hence this product is termed a genetic optimizer, meaning that it uses algorithms which mimic the genetic processes used in evolution to optimize a set of parameters, or inputs, for a investment strategy.

Evolutionary programming is useful when it is necessary to search through a very large number of combinations to arrive at an optimal solution from a high number of possible solutions, and when the number of combinations precludes a traditional exhaustive search. This approach is widely known, well accepted and used in many fields; it has been used for optimizing circuit board layouts, jet engines, analog filters, gas pipelines, and financial applications.

**Comparisons and Terms**

Comparing strategies to nature, investment strategies can have a number of inputs with many possible values. Any single combination of those values of
inputs we call an "individual."

**Individual** - a unique combination of values for the inputs of a strategy.

Each individual input variable for a strategy is called a "gene."

**Gene** - one of the input variables of a given strategy.

For example, in a moving average crossover strategy, you may have just two input variables: SlowLength and FastLength. In this case the strategy has two genes. An example of an individual would be the case where SlowLength = 50 and FastLength = 20. Another individual would be SlowLength = 55 and FastLength = 21. In fact, any combination would be considered an individual.

We combine genes to form chromosomes, as is done in nature.

**Chromosome** - a set of genes, usually related in function.

Chromosomes are the basic unit of reproduction. When you set up the inputs for your strategy, you will group the inputs into chromosomes. For example, we may add stops and targets to our moving average crossover strategy. Our input variables may then be:

<table>
<thead>
<tr>
<th>Chromosome</th>
<th>Inputs (Genes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SlowLength</td>
</tr>
<tr>
<td>1</td>
<td>FastLength</td>
</tr>
<tr>
<td>2</td>
<td>ProfitTargetAmt</td>
</tr>
<tr>
<td>2</td>
<td>StopLossAmt</td>
</tr>
<tr>
<td>2</td>
<td>BreakevenFloorAmt</td>
</tr>
<tr>
<td>2</td>
<td>DollarTrailingAmt</td>
</tr>
<tr>
<td>2</td>
<td>PctTrailingFloorAmt</td>
</tr>
<tr>
<td>2</td>
<td>PctTrailingPct</td>
</tr>
<tr>
<td>2</td>
<td>ExitOnClose</td>
</tr>
</tbody>
</table>

Combining genes which work together onto a common chromosome improves the speed and quality of the reproduction process. Here we have defined our entry logic inputs into chromosome 1, and the exit logic inputs into chromosome 2.

### 2.1.2 Genetics and Mating

Mating forms the basis for evolution. Two individuals exchange genetic material, thus passing on some of the capabilities of the mother and some from the father to the child. The child is thus a blend of the characteristics of both the mother
and father. If two very fit individuals mate, the child is also likely to be fit. However it will not be the same fitness as the parents - it cannot be - because the parents are not identical themselves. It will be a random blend of the parents and so sometimes be less fit and sometimes more fit.

The process of mating uses two primary operations:

- crossover
- mutation

**Crossover**
Crossover is the process of combining the genes of a chromosome from each parent to create the corresponding chromosome for the child. This is done by selecting a split point at random and using half of the genetic material from the mother and the other half from the father.

Using the chromosomes from the table in the previous topic, and the two individuals shown below, the first chromosome can have only one split point - at the 2nd gene. If we randomly select the split point for chromosome 2 at the 4th gene, we obtain a child as show below.

<table>
<thead>
<tr>
<th>Chromosome</th>
<th>Genes</th>
<th>Mother</th>
<th>Father</th>
<th>Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SlowLength</td>
<td>50</td>
<td>45</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>FastLength</td>
<td>60</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>2</td>
<td>ProfitTargetAmt</td>
<td>500</td>
<td>600</td>
<td>500</td>
</tr>
<tr>
<td>2</td>
<td>StopLossAmt</td>
<td>100</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>BreakevenFloorAmt</td>
<td>75</td>
<td>50</td>
<td>75</td>
</tr>
<tr>
<td>2</td>
<td>DollarTrailingAmt</td>
<td>50</td>
<td>60</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>PctTrailingFloorAmt</td>
<td>100</td>
<td>120</td>
<td>120</td>
</tr>
<tr>
<td>2</td>
<td>PctTrailingPct</td>
<td>2</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>ExitOnClose</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Split points are chosen at random, and whether the first half is copied from the mother's or the father's genes is also random. If the mother's genes are used for the first half, then the father's are used for the second, and vice versa.

If you had used only a single chromosome for all of these inputs there would have been only one split point and one crossover operation. With this you can see why combining related inputs into separate chromosomes improves the efficiency of the algorithm. You get multiple-point crossover with each mating, and each crossover mixes only the genes that relate to each other. This increases the speed with which the algorithm can search and the quality of the search.

**Mutation**
Sometimes, during mating, a gene will mutate and receive a value that is neither
in the mother nor the father. Mutation is nature's way of increasing diversity in the gene pool. Some mutations are beneficial, while others are detrimental to fitness.

The probability of mutation is low in nature; generally only a very small percentage of the total number of genes making up an individual mutate. If many mutations occur in a single individual the result is usually chaotic, and the individual is likely to be less fit because of it. However, a small amount of mutation can be good.

In Optimax, genes will mutate at random with a probability that you control. The value given to a gene during mutation will be within the optimization range for the input and a multiple of the incremental value.

For example, in the table below, one mutation has occurred and is shown in red.

<table>
<thead>
<tr>
<th>Chromosome</th>
<th>Genes</th>
<th>Min</th>
<th>Max</th>
<th>Inc</th>
<th>Mother</th>
<th>Father</th>
<th>Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SlowLength</td>
<td>45</td>
<td>80</td>
<td>5</td>
<td>50</td>
<td>45</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>FastLength</td>
<td>20</td>
<td>60</td>
<td>5</td>
<td>60</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>2</td>
<td>ProfitTargetAmt</td>
<td>100</td>
<td>1000</td>
<td>100</td>
<td>500</td>
<td>600</td>
<td>500</td>
</tr>
<tr>
<td>2</td>
<td>StopLossAmt</td>
<td>50</td>
<td>250</td>
<td>50</td>
<td>100</td>
<td>50</td>
<td>200</td>
</tr>
<tr>
<td>2</td>
<td>BreakevenFloorAmt</td>
<td>25</td>
<td>100</td>
<td>25</td>
<td>75</td>
<td>50</td>
<td>75</td>
</tr>
<tr>
<td>2</td>
<td>DollarTrailingAmt</td>
<td>20</td>
<td>100</td>
<td>10</td>
<td>50</td>
<td>60</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>PctTrailingFloorAmt</td>
<td>80</td>
<td>150</td>
<td>10</td>
<td>100</td>
<td>120</td>
<td>120</td>
</tr>
<tr>
<td>2</td>
<td>PctTrailingPct</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>ExitOnClose</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The StopLossAmt for the child does not exist in either the mother or father. It is within the range from 50 to 250, and is a multiple of 50.

**Genetic Complements**

Using genetic programming, we can do something that nature can't, we can create the opposite of an individual. This is call the *genetic complement* of an individual.

*Genetic Complement - the theoretical genetic opposite of an individual.*

Complements are useful for increasing the diversity of the gene pool in a way that is very different from mutation. By taking the complement of an unfit individual, there is the possibility of creating a fit individual. To determine the compliment, first imagine each input range wrapping around and creating a continuum on a circle. For example, if Min = 20, Max = 50 and Inc = 10, then the continuum would be the series 20, 30, 40, 50, 20, 30, etc, imagined as
For example, if an individual's gene has the value 30, the complement would be 50 - the value that is opposite to it on the continuum circle.

To create the complement of an entire individual, the complementary value of each gene would be taken to create a new individual, as shown in the following example.

<table>
<thead>
<tr>
<th>Chromosome</th>
<th>Genes</th>
<th>Min</th>
<th>Max</th>
<th>Inc</th>
<th>Individual</th>
<th>Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SlowLength</td>
<td>45</td>
<td>80</td>
<td>5</td>
<td>50</td>
<td>70</td>
</tr>
<tr>
<td>1</td>
<td>FastLength</td>
<td>20</td>
<td>60</td>
<td>5</td>
<td>35</td>
<td>55</td>
</tr>
<tr>
<td>2</td>
<td>ProfitTargetAmt</td>
<td>100</td>
<td>1000</td>
<td>100</td>
<td>500</td>
<td>1000</td>
</tr>
<tr>
<td>2</td>
<td>StopLossAmt</td>
<td>50</td>
<td>250</td>
<td>50</td>
<td>200</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>BreakevenFloorAmt</td>
<td>25</td>
<td>100</td>
<td>25</td>
<td>75</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>DollarTrailingAmt</td>
<td>20</td>
<td>100</td>
<td>10</td>
<td>50</td>
<td>90</td>
</tr>
<tr>
<td>2</td>
<td>PctTrailingFloorAmt</td>
<td>80</td>
<td>150</td>
<td>10</td>
<td>120</td>
<td>80</td>
</tr>
<tr>
<td>2</td>
<td>PctTrailingPct</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>ExitOnClose</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Because of their complexity, very few genetic algorithms implement complements. We make them available to you in Optimax, and you can control their behaviour from the optimization settings screen alongside all of the other options.

### 2.1.3 Selecting A Mate

For mating to occur, the individuals that will mate with each other must somehow be selected as a couple. In real life, mates chose each other, usually according to an attraction based upon a perceived fitness for mating. Quite a bit of variation occurs during the evaluation of fitness, which directly translates into the
desirability for mating. This variation depends upon many factors, such as personal judgement, geographical limitations, whim, fancy, availability and emotional makeup, which all contribute to the variability in fitness determination. In Optimax, we simulate the mate selection process using a variety of algorithms, known as Fitness Calculations. You will choose the fitness calculation that Optimax will use to evaluate the fitness of an individual (a combination of inputs.) We will discuss this in more detail in the next section.

After Optimax has sent all of the individuals of a generation to TradeStation and fitness values received for each, the next step is to execute the mating process in order to build the next generation. Optimax begins this process by creating a mating pool. A mating pool is an area in memory into which all individuals that will be eligible for mating will be placed. The mating pool can hold more individuals than are in the current generation. Exactly which individuals and how many individual Optimax places within it depends on the mating options that you select.

**Survival and Elitism**

Another advanced feature found within Optimax is the ability to mate individuals spanning several generations. In real life, a person from an older generation can mate with a person of a younger generation. This can happen because the fitter the individual, the longer they survive and the better they are able to compete with younger individuals. Th fitter the individual, the longer they survive, and the more chances they have of mating to pass on their extremely fit genes.

This feature is known as elitism, and is an advanced, extremely useful feature uncommon in genetic optimization programs because of it's complexity in programming. The individuals that survive from previous generations to mate with the current population are known as elite individuals.

You will control how many individuals survive from generation to generation using the Elite Percent Taper value on the Optimization Settings window. For example, setting this value to 50 would mean that 50% of the individuals from the previous generation survived to mate with the current generation. These are the fittest 50% of the previous population, of course. With each passing generation, 50% of the population dies, so in the 3rd generation, 50% of generation 2 is available for mating, and 25% of generation 1.

The Maximum Life Span field controls the maximum number of generations an individual can survive. For example, when set to 4, elite individuals are available for four generations, including the first generation in which they existed. So if you are in generation 15, Optimax will select individuals from the current generation, the top 50% of the individuals from generation 14, the top 25% of the individuals from generation 13, the top 12.5% of the individuals from generation 12, and stop there.

The absolution maximum life span is 10 generations. A value of 0 turns the
elitism feature off.

**Filling the Mating Pool**
The first thing Optimax does when filling the mating pool is to copy the top N percent of the current population into the pool, where N is the value in the Top Percent field. For example, a value of 80 causes Optimax to copy the top 80% fittest individuals to the mating pool. This feature weeds out the poorest performers.

Individuals with negative fitness values are always considered unfit for mating and are never copied into the mating pool.

Next, Optimax copies elite individuals from previous generations into the mating pool.

Next, using the least fit members of the current population, Optimax generates the number of genetic complements that you specified in the Genetic Complement Number field, and places them into the mating pool. If individuals with negative fitness values exist in the current population, they can and will be used to generate complements.

The mating pool is now full, and mate selection begins.

**Basic Mate Selection**
In basic mating selection mode, Optimax now chooses a pair of individuals for mating from the mating pool completely at random, and performs crossover and mutation operations to create a brand new individual. This process repeats until all individuals of the next generation have been created.

Basic mode is enabled when all other modes (described below) are disabled.

**Tournament Selection**
When you enter a Tournament Group Size field value greater than 0, you enable tournament selection mode. In this mode, when Optimax wants to choose an individual from the population for mating, it first selects a group of individuals at random. Imagine this group as if they were competing in a tiny sparring match, where the individual with highest fitness value wins. That individual is chosen for mating. This method mimics scenarios in nature where small groups of individuals arrive at the same location and compete with each other for a mate. In those scenarios, the fittest individual is the one who wins and mates with the individual they compete for. The same thing happens in Optimax when you enable tournament selection.

For example, if you specify Tournament Group Size 3, Optimax selects 3 individuals at random. Suppose the three individuals had fitnesses of 1.2, 6.4 and 5.39. Optimax would deem the individual with the 6.4 fitness as the winner of the tournament and would select it for mating. Optimax would repeat this for a
second individual, perhaps selecting 3 individuals with fit, and those two individuals would then mate.

This selection mode puts pressure on the selection process to select fitter individuals more frequently than the less fit individuals. The result helps to ensure that the fittest individuals mate more frequently than less fit individuals. The larger the group size, the more frequently the fittest individuals will be selected for mating. However, you need to maintain diversity in the gene pool, so Optimax will search many parts of the fitness landscape, so putting too much pressure on the fittest individuals will reduce your chances of discovering all of the landscape maximums, and thus reduce your chances of discovering the optimal solution.

Disable this selection mode by specifying 0 in the Tournament Group Size field.

**Roulette Selection**

In roulette selection mode, individuals are selected for mating at random but the probability of an individual being selected is proportional to its fitness (relative to the others in the mating pool.) In this mode, the fittest individuals are selected more frequently than the less fit individuals, according to the frequency distribution of the fitnesses in the current generation.

For example, if you had a generation with only 3 individuals with fitness values of 10, 20 and 30, then the individual with the fitness value of 30 would be selected from the pool for mating three times more often than the one with the fitness value of 10. The individual with the fitness value of 20 would be selected for mating twice as often as the individual with fitness value 10.

The value you specify in the Roulette Scaling Factor field controls the strength of the weighting. The example in the previous paragraph is an example of a scaling factor of 1. If you were to specify a scaling factor of 2, then the individual with fitness 30 would be selected six times more often than the fitness 10 individual, and the fitness 20 individual four times as often. If you specify a scaling factor of 0.5, the selection pressure is reduced and the fitness 30 individual would only be selected 1.5 times as often as fitness 10, and fitness 20 would be selected 1.25 times as often.

Specifying a Roulette Scaling Factor of 0 turns roulette selection off.

**Tournament and Roulette Selection**

If both tournament and roulette selection modes are enabled, the individuals for each tournament are selected using the roulette selection method instead of at random. This can place additional pressure on the fitter members of the generation by selecting them even more frequently than either method alone, or with a low scaling factor and a small number of members in the tournament, you can end up with approximately the same about of pressure on fit members but with a more complex selection method that mimics natural methods better than
either alone.

**Incest Filtering**
For all mate selection methods, before Optimax performs each mating operation it checks the Hamming distance between the two individuals to determine how closely related they are. If the Hamming distance is less than the value you specified in the Min nHamming Distance field, the individuals are considered too closely related, the match is rejected, and two new individuals are selected for mating instead. This is an advanced feature known as *incest filtering*. Every set of individuals is incest-checked before mating occurs.

Near the end of a successful optimization, individuals become more and more alike as they hone in on optimal solutions. Under those circumstances it is possible that all couples could be rejected because they fail the incest test, and the optimization would not continue. Optimax has a built-in mechanism to prevent this situation from occurring. If 10 couples are rejected because they are too closely related, after the 10th rejection, the Min nHamming Distance is divided in half and the process continues with the reduced distance value. This halving occurs each time 10 consecutive couples are rejected, and will always result in a Hamming value that will allow the most distant individuals to mate before the most closely related couples, no matter what the diversity may be in the mating pool.

When halved, the new Hamming distance is used for all future tests within the current optimization.

**Summary of Advanced Features**
Mate selection is an extremely important part of the evolutionary mechanism. Optimax gives you many more options than are available in any other genetic search programs:

- dynamic mating pool size
- elitism
- incest filtering
- multiple mate selection modes, including simultaneous modes (unavailable anywhere else)
- scalable roulette weighting (unavailable anywhere else)

These are only some of the advanced genetic features available in Optimax. No other genetic algorithms implement this much flexibility in mating, or provide as many options for your to control the mating process. Since mating is at the core of the evolutionary process, these advanced options are exceptionally valuable and having them available for your use can make a world of difference during your optimizations.
2.1.4 Fitness

Once an individual has been born, there must be some way to evaluate that individual's fitness, also known as the individual's desirability. Determining the fitness or desirability of a set of inputs to a strategy is as simple as running the strategy using those inputs and performing a calculation on the last bar of the strategy to compute fitness. You create the fitness computation in EasyLanguage and pass the fitness value to Optimax using a function call. When computing fitness, what you must do is assess your strategy's overall performance for this set of inputs and express that as a single floating-point number.

Fitness is the objective of your optimization expressed numerically. It is a floating-point value between -1E300 and 1E300. Positive fitness values are fit and negative values are unfit. Optimax considers higher values as fitter, and lower values as less fit. That doesn't mean you should attempt to have fitness values span the entire permissible range up to 1E300; values between 0.0 and 1.0 would work as well, given enough decimal places; it's up to you. Unfit values are excluded from mating and are not used when creating new individuals.

Sample fitness calculations you could use are:

- Sharpe Ratio
- Sortino Ratio
- Calmar Ratio
- Sterling Ratio
- Treynor Ratio
- % Gain Ratio

Each of these computations reduce the strategy's performance to a single numerical value. Optimax includes several fitness functions for you to use right out of the box. You can also find standard formulae for all of the above ratios on the Internet.

You can change the fitness calculation to suit your needs at any time and it can be different for each optimization you perform if you like. And it can be as complex or as simple as you need it to be.

Related Topics:

- Enhancing the Epilog
- Evaluation in a Nutshell
- OMX_F_MSRatio
- OMX_F_GPF

2.1.5 Generations

In Optimax, a generation is a collection of individuals all created at the same time and analysed as a set.
**Generation** - A group of individuals all born at about the same time. Also known as a population.

Each optimization is composed of several generations, usually 15 to 100, with each generation usually containing 50 to 200 individuals. Optimax begins the optimization using a randomly-generated set of individuals for the first generation, analyses them, and then creates a subsequent generation from them. It repeats this process throughout the optimization. You will specify how many individuals for Optimax to create for the first generation, and each subsequent generation will contain the same number of individuals as the first.

When you start the optimization, Optimax generates a set of completely random individuals which form the first generation. That means that the values for the genes of each individual in the first generation are completely random. Each of these individuals is passed to TradeStation one by one and evaluated, and a fitness value is sent back from TradeStation and stored with each individual. At the end of the first generation, Optimax generates a second generation by selecting individuals from the first generation for mating. The individuals are mated, a second generation produced, and then they are all sent one by one for processing to TradeStation and fitness values returned and stored. The process repeats until convergence is reached.

During optimization, every individual within a generation is guaranteed by Optimax to be unique. When each individual is created, no matter which method is used, it is compared with all other individuals of the generation in a "clone check" operation. If it is identical to another individual, Optimax uses one or more of three changeling methods: mutation, rival, or random, to create a unique individual.

When Mutation is chosen, a single gene is mutated at random and this process is repeated until the individual is unique within the population.

When Rival is chosen, roughly one-half of the genes of the individual are shifted up or down by one step to create a near likeness, or a rival to the existing individual. Each gene has a 50% chance of being shifted, and there is a 50% chance of an upward or downward shift. If, after shifting all genes and creating a rival, it is found that there is a clone in the population for this new individual, then the process is repeated until a clone is not found.

When Random is chosen, an entirely new, entirely random individual is generated.

### 2.1.6 Convergence

After a while, if the optimization is set up correctly, the individuals of several generations should all become very similar as the optimization hone in on the optimal solution. This increase in similarity is called *convergence.*
**Convergence** - a point in time reached during a genetic optimization when several generations have produced individuals without significant differences between them.

Once the individuals of a generation no longer show much difference between each other, the diversity of the population stabilizes and continuing that particular optimization will not yield much more useful information and so the run can be considered complete. At this point Optimax will stop generating subsequent generations.

The standard quantitative measure of the difference between two individuals is called the Hamming distance. The Hamming distance is computed as the sum of all of the distances between the genes of two individuals, where the distance between two corresponding genes is computed as follows:

\[
\text{distance} = \text{absvalue} \left( \frac{\text{GeneN\_IndividualA} - \text{GeneN\_IndividualB}}{\text{increment}} \right)
\]

where \(\text{GeneN\_IndividualA}\) is the value of gene N of an individual A, and \(\text{GeneN\_IndividualB}\) is the value of the corresponding GeneN of IndividualB, and \(\text{increment}\) is the increment value used during optimization for that gene.

Another way of determining if there is much difference between individuals is to compare their fitness values. If two fitness values are close, you could consider the individuals as being similar. The closer the fitness values, the more you would consider the individuals as equivalent. It is possible that two very different sets of inputs could produce similar fitness values. In that case, it is probable that the individuals are almost equivalent, provided that the fitness function is properly designed.

When determining convergence, instead of using Hamming distances to compare individuals, Optimax compares fitness values. When the standard deviation of the fitness values over a given number of generations falls below a preset value, convergence is reached and the optimization stops. The number of generations and the value are both set by you in the Optimization Settings window.

### 2.1.7 Curve Fitting and Adaptation

Curve fitting, also known as *over-optimizing*, is the pariah of strategy optimization. Curve fitting is the process of optimizing a strategy to such a degree that you get good back testing results only because you have tailored them for a specific set of price points, and not because you have unearthed some truism about the price series that will remain true in the future. You will know that you have curve-fitted when, as you test future data, your strategy yields drastically different results, usually for the worse.

Many so-called experts will tell you than you should first have a sound basis for
your strategy and then use parameters that work. But if you are not an expert and want to create a strategy, how can you do it and become an expert? Naturally you will try various things to see what works, and learn from that. And you will do it in the hopes that you find something that the experts have not, and in the process learn what works and what doesn't work. The major problems that people encounter in these endeavors are:

- not enough processor power to try all of the combinations of things they want to try
- an inability to determine if the results obtained are curve-fitted

Optimax addresses both of these points. We already know that Optimax's genetic search algorithms reduce optimization time to a minimum. To address the second point, Optimax uses a new technique: price modulation. Most people use a technique called walk-forward testing to address this issue. Walk-forward testing has both pros and cons, and we also plan to add this capability to Optimax in the future, so it is worthwhile to discuss here and compare it to our solution of price modulation.

**Pros and Cons of Walk-Forward Testing**

To overcome the problem of curve fitting, many people use walk-forward testing. This is a process by which you optimize over a historical chart period, holding in reserve a segment of the chart data, usually the most recent data. After you have optimized, you test again using the reserved data. If the strategy performs well on the data that wasn't used during optimization, then the strategy is probably not curve fitted. That's great, as long as your test goes well.

But what if your walk-forward test fails on the unseen data?

The biggest problem now is not what you have done so far, but what you will do next. If your test fails, do you throw all your work away and start from scratch? Or do you work with your strategy some more and see if you can come up with another approach that yields satisfactory results on the unused segment of data? If you are even only half-way human, you will probably modify your strategy and try again. Unfortunately, at that point, you are no longer able to say that your walk-forward test is reliable, because you are now trying to change your strategy so that the unused segment of data yields acceptable results as well. Once your strategy has seen the unused data, it is no longer unused. If you continue testing you are now curve-fitting, no matter how much thou dost protest.

So, the only reliable way to use a walk-forward test is as a final go/no-go test. If it fails, you should reject the basis of the strategy and try again from scratch. Else, you have fallen into the curve-fitting trap - albeit via a circuitous route.

As if that weren't enough, there is yet another problem with walk-forward testing - it requires you to optimize over less data than you currently have. An inefficiency arises because you've performed your optimization with some of the
data missing - you haven't used it all and thus you couldn't have optimized your strategy to take full advantage of all of the price action.

So what's a poor strategy-designer to do?

That's where price modulation, available only in Optimax, comes in. Using the price modulation options on the Optimization Settings window, you can cause the price series to change during the optimization process, causing each individual to see a slightly different price series. You do this by applying a modulation wave and a random noise function at the same time, and this will cause the price series to be different for every test during the optimization. The strength of the modulation you apply determines the amount of difference, thus reducing the possibility of curve fitting by the amount of variability you inject into the price series. Injecting the maximum amount of variability will reduce the likelihood of curve fitting nearly to zero.

The reason we chose to implement this technique before walk testing is because there are more advantages in using modulation than walk testing:

- you can use all of your data for testing
- you can adjust the amount of variability, and thus "stress-test" your strategy
- you can add a random walk to the price series, thus performing "what-if" tests with your strategy

There is always a danger in optimizing when you are optimizing over a static data stream. With random walk price modulation, that series is no longer static, and you now have a way of testing the adaptability of your strategy as well as the profitability. Since the stock market isn't static, but is dynamic, optimizing over a static datastream is an unrealistic way to back test. Testing over a dynamic datastream increases the reality of the tests and adds a never-before-available level of integrity of your simulations.

It is extremely important to modulate the price series during the genetic optimization.

2.1.8 Optimization Results

Imagine a 3-dimensional plane with hills, valleys, troughs, crests and peaks. Imagine further that this is a map of all of the fitness values obtained using all of the possible input combinations for your strategy. That map is known as a fitness landscape.

**Fitness landscape** - the theoretical set of fitness values for all possible input combinations.

This map in reality is not actually 3-dimensional, it is \( n \)-dimensional, where \( n \)
equals the number of inputs you are optimizing, but it is easier to imagine in 3 dimensions.

A maximum on this plane would look like a hill, a mountain or a even a sharp peak, a minimum would look like a trough or a valley. Because strategy fitness landscapes can be very complex, there are usually many hills and valleys of varying heights and depths. There can even be rugged terrains composed of sharp cliffs, crevasses, spikes, holes, and cracks. A set of hills grouped together are called a set of local maximums. The highest point in the entire fitness landscape is called the global maximum. Naturally, this is the point you are seeking to discover, provided that the approach to the global maximum is smooth from all sides. The smoother your fitness landscape, the more predictably your strategy will perform in changing situations.

If the global maximum is a needle sharp peak, this is probably not the most desirable solution for the optimization, because a small change in the stock's action would probably cause a large change in the strategy's performance. Instead, the more desirable results are maximums that have smooth sides so that changes in the price action should have a better chance of resulting in a smooth drop in strategy performance.

Some elementary search algorithms are of the "hill climbing" variety; they modify the inputs in several directions to see which one detects a rise in fitness, and then they explore that direction, always seeking the higher ground until it starts to decline, and then they stop. If the terrain contains several hills, they usually explore only one of them and then stop, and may miss many better solutions to the problem. They can also discover and return a needle-like maximum as the solution if it exists, but the likelihood of finding it is very small because the feature itself covers only a small area of the landscape.

Genetic algorithms, like the ones used by Optimax, are much more advanced. They explore many areas simultaneously and can hone in on multiple maximums at the same time, arriving at multiple solutions to the same problem at once. They don't get stuck on local maximums like hill-climbing algorithms do, and are also unlikely to find needle-like maximums. Unlike hill-climbing algorithms, the terrain approaching a maximum is important too - in the mixing of genetic material Optimax explores many facets of the approach, and must arrive at many unique nearby solutions for there to be enough individuals in the population to survive, mate and pass on their genetic material to subsequent generations. If there are only a few individuals with sufficient fitness at a local maximum, they must mate with others, and in doing so they become weeded out as their genetic material is diluted with the material from other local maximums. In other words, it's not only survival of the fittest, but the smoothest as well. For this reason, solutions for a needle-like point on the landscape will not survive given at least one other maximum with a more gradual and smoother approach.

Theoretically, after enough generations have passed, the population will be
entirely composed of individuals whose fitness values cluster around the global maximums on the fitness landscape. Whether all maximums are found depends upon the number of maximums to be found, the number of dimensions in the landscape (also known as the number of genes in the individual), the area of the landscape (also known as the number of possible permutations in the optimization) and the number of individuals in the first generation. The more complex the landscape, the more individuals you should have in the initial generation. One way of knowing if you have enough individuals in the initial generation is to run the optimization more than once. If each run produces different results, then the landscape is more complex than the diversity that is possible given the size of the initial population.

2.1.9 Summary

Summary of the Process
1. Begin with a randomly generated population of input combinations (a.k.a. individuals)
2. Calculate the fitness of each individual, where fitness is a measure of how well the strategy performs using a set of inputs
3. Retain the fittest members, discarding the least fit members
4. Generate a new population of individuals from the remaining members of the old population by applying the crossover, mutation and complement operations
5. Calculate the fitness of these new individuals retaining the fittest, discarding the least fit.
6. Repeat until convergence is reached

Further Reading
This is only a brief introduction to genetic algorithms. We encourage you to read more about them to fully understand their functions, capabilities, and limitations. Many excellent guides can be found on the Internet:

- The Hitch-Hiker's Guide to Evolutionary Computation
- A Brief Primer on Genetic Algorithms
- Local Search Algorithms
- Genetic Algorithms as a Computational Tool for Design
- Using Multiple Chromosomes To Solve a Simple Mixed Integer Problem

2.2 Optimizing

Optimizing a strategy with Optimax is far less time consuming than using the standard methods available via TradeStation. Optimax presents a wealth of information in a compact format including several exciting new indicators not available in any other packages. The optimization information is presented in
these forms:

- equity grid
- inputs grid
- input space map
- search space map
- input space canvas
- detailed report

To run an Optimax optimization, you must:

1. Prepare the strategy
2. Set the optimization parameters
3. Run the optimization

### 2.2.1 Preparing A Strategy

To prepare a strategy you must insert some standard Optimax code into your strategy. If you use multiple strategies at once, for example if you use yours plus the Stops & Targets strategy, then you must combine them into a single strategy. You will also need to add an additional datastream, identical to Data1.

Once your strategy meets these requirements, you can now add the Optimax calls. Adding the calls can be done via either of two methods (you do not need to do both):

- manually, or
- automatically using our code converter utility

Both of these methods are described in the following sections.

---

**Important**

Although the code converter can perform most of the work for you, we recommend you read both sections so that you understand the changes that the code converter will make to your strategy and so that you can verify that the changes have been done correctly. You may also need to modify the code when you make future changes to your strategy.
2.2.1.1 Adding Optimax Calls Manually

The strategy code must first be prepared for use with Optimax by inserting OMX function calls. In this section we describe how to do it manually.

The following table gives you an overview of the code of an Optimax-converted strategy:

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>OMX Preamble</td>
<td></td>
</tr>
<tr>
<td>Converted Inputs</td>
<td>OMX Prolog Code</td>
</tr>
<tr>
<td>Your Strategy Code</td>
<td>Your Strategy Code</td>
</tr>
<tr>
<td>OMX Epilog Code</td>
<td></td>
</tr>
</tbody>
</table>

Here is a sample strategy we will use in this chapter:

```plaintext
inputs:
    Len1( 15 ),
    Len2( 20 );

if AverageFC( c, Len1 ) crosses over AverageFC( c, Len2 ) then
    buy next bar at market ;
if AverageFC( c, Len1 ) crosses under AverageFC( c, Len2 ) then
    sell all shares next bar at market ;
```

Here is the same strategy modified for use with Optimax. The added code is highlighted in pale green.
The following topics will cover each code segment in detail.
2.2.1.1 Preamble

Your strategy must begin with an OMX Preamble, as follows:

```plaintext
{_____OMX PREAMBLE______}
vars:
   OMX_Init( 0 ),
   OMX_Temp( 0 ),
   OMX_Fitness( 0 ) ;
inputs:
   OMX_IterationNum( 0 ),
   OMX_Generation( 0 ),
   OMX_Individual( 0 ),
   OMX_StreamNum( 2 ),
   OMX_EquityResolution( 100 ) ;
```

This code is stored in TradeStation in the OMX_CutAndPaste strategy. From there you can simply cut and paste these lines into your code. (Cut and paste from here will not give you proper indentation.) Or, you may find it easier to copy the entire CutAndPaste strategy and then copy your code into it in the appropriate places.

2.2.1.2 Converted Inputs

For each input you want to optimize you must create a variable of the same name, and then prefix the input name with `I_`, as seen here:

**Before:**

```plaintext
inputs:
   Len1( 15 ),
   Len2( 20 );
```

**After:**

```plaintext
{_____CONVERTED INPUTS__}
inputs:
   I_Len1( 15 ),
   I_Len2( 20 );
Vars:
   Len1( 15 ),
   Len2( 20 );
```

This is only necessary for the inputs you want to optimize. The default values of the new variables don’t matter.

2.2.1.3 Prolog

Next, prolog code must follow. A sample is shown here:
This code sets the values of the converted input variables used in your strategy. It will do this on the first bar of each strategy evaluation using either the values from the \_inputs or by calling Optimax to obtain the values.

The lines highlighted in bright blue are the lines that change for each strategy. For each input, you must have two lines, one in the first section and one in the second.

**Explanation**

- **Line 3** will be true when you are not optimizing causing the values for your strategy to come directly from the input variables (lines 4 and 5.)

- **Line 3** will be false when you are optimizing, causing the OMX_GetParm functions (lines 11, 12) to be called and the values retrieved from Optimax.

**Note**

Optimax accomplishes price modulation via the call to OMX_PriceModX shown in bright blue near the end of the above code snippet.

2.2.1.1.4 Your Strategy

Place your unchanged strategy code between the prolog and epilog.
If OMX_IterationNum = 0 Or OMX_DateOk Then begin
{______YOUR STRATEGY______}
  if AverageFC( c, Len1 ) crosses over AverageFC( c, Len2 ) then
    buy next bar at market ;
  if AverageFC( c, Len1 ) crosses under AverageFC( c, Len2 ) then
    sell all shares next bar at market ;
{______OMX EPILOG_______}
end ;

You can change this code as you normally do when working with your strategy. If you want to add another input, you will need to change the prolog and the converted inputs.

Notice that your strategy is contained within a higher-level IF statement that begins in the prolog and end in the epilog.

2.2.1.1.5 Epilog

Place the following epilog code at the end of your strategy:

{______OMX EPILOG_______}
end ;
If OMX_Init = 1 And OMX_IterationNum > 0 Then begin
  OMX_Temp = OMX_Barinfo( OMX_EquityResolution, OMX_IterationNum ) ;
  If OMX_LastBarInWalk Then begin
    OMX_Fitness = Div( NetProfit, absvalue(MaxIDrawDown) ) ;
    OMX_Temp = OMX_EndPass( OMX_Fitness, OMX_IterationNum,
                            OMX_FirstCalcDate, OMX_FirstCalcTime ) ;
  end ;
end ;

There are two important parts of the epilog for you to consider, BarInfo and Fitness.

BarInfo

OMX_Temp = OMX_Barinfo( OMX_EquityResolution, OMX_IterationNum ) ;

In this line, you are sending information about each bar back to Optimax. The first parameter is a granularity value. If you are optimizing an intra day strategy, you can modify this parameter to control the resolution of the equity curve displayed in Optimax.

Common granularity values are:
5  report equity at 5 minute intervals
10  report equity at 10 minute intervals
100 report equity at 10:00, 11:00, 12:00, etc.
200 report equity at 10:00, 12:00, 14:00, etc.
1500 report equity daily at 3:00pm

Equity is reported to Optimax when mod(time,EquityResolution) = 0. The lowest equity between two granularity points will be the equity reported to Optimax.

Changes to this parameter have no effect on daily, weekly or monthly charts.

Performance Tip
Lower resolution values increase detail within Optimax, and also decrease performance during optimization and display.

Fitness

OMX_Fitness = Div( NetProfit, absvalue(MaxIDDrawDown) ) ;

Fitness is the objective of your optimization expressed numerically. It is a floating-point value between -1E300 and 1E300. Positive fitness values are fit and negative values are unfit. Optimax considers higher values as fitter, and lower values as less fit. That doesn't mean you should attempt to have fitness values span the entire permissible range up to 1E300; values between 0.0 and 1.0 would work as well, given enough decimal places; it's up to you. Unfit values are excluded from mating and are not used when creating a subsequent generation.

2.2.1.5.1 Enhancing the Epilog

Calculating Fitness
The fitness calculation is the heart of your optimization and the primary input into the mating process. The default fitness calculation Optimax uses is shown below in blue.
if OMX_Init = 1 and OMX_IterationNum > 0 then begin
    OMX_temp = OMX_BarInfo( OMX_EquityResolution, OMX_IterationNum ) ;
    If OMX_LastBarInWalk then begin
        Fitness = Div( NetProfit, absvalue(MaxIDDrawDown) ) ;
        OMX_temp = OMX_EndPass( OMX_Fitness, OMX_IterationNum, OMX_FirstCalcDate, OMX_FirstCalcTime ) ;
    end ;
end ;

This is the default fitness calculation provided with Optimax.

Optimax also provides other fitness calculations for you to try. Here is a list of all fitness calculations provided:

1. Div( NetProfit, absvalue(MaxIDDrawDown) )
2. OMX_F_MSRatio
3. OMX_F_VanTharp

Insert one of these functions into your strategy, replacing the Fitness computation line shown above in blue.

The first computation - NetProfit / absvalue(MaxIDDrawDown) - is a profit-to-drawdown ratio. It maximizes profit while minimizing drawdown. For example if you had two equity curves both with $10,000 of net profit, the first having $1,000 of drawdown and the second $2,000, the resulting fitnesses would be 10 and 5, identifying the first result as being twice as desirable as the second. This is a good fitness function to use when your strategy has a possibility of large drawdowns - for example a swing trading strategy with loose stops. It would not be as useful in a strategy with tight stops, since the drawdown is limited within the strategy itself and the function would not provide a wide range of values to choose from.

The second computation is a modified Sharpe Ratio function. The Sharpe Ratio is a time-based measure of the consistency of returns. It will assess the consistency of your gains over a period of days, weeks or months. 

The third computation is a modified Van Tharp function. The Van Tharp is profit-to-risk ratio standardized to the number of days per year. The more often your strategy trades and the more it makes on average per trade (measured as dollars made per each dollar risked,) the higher the score.

Click here for a more detailed description of these fitness functions.

In addition to the above, you can create your own fitness functions to use. We encourage you to experiment according to your needs.

Sending Additional Results
Using the OMX_SendVar function, you can send additional equity results to
if OMX_Init = 1 and OMX_IterationNum > 0 then begin
  OMX_temp = OMX_BarInfo( OMX_EquityResolution, OMX_IterationNum ) ;
  If OMX_LastBarInWalk then begin
    Fitness = Div( NetProfit, absvalue(MaxIDDrawDown) ) ;
    OMX_temp = OMX_SendVar( "MyVar1", value1 ) ;
    OMX_temp = OMX_SendVar( "MyVar2", value2 ) ;
    OMX_temp = OMX_EndPass( OMX_Fitness, OMX_IterationNum,
          OMX_FirstCalcDate, OMX_FirstCalcTime ) ;
  end ;
end ;

You make up the name for the first parameter, and pass any value you wish for the second. You can call this function as many times as you wish, passing a different variable each time. These values will be stored and later displayed as part of the equity detail report for an individual within Optimax. Shown below are the two variables sent from the code above, with sample values.
2.2.1.2 Adding Optimax Calls Automatically

Optimax contains an EasyLanguage code converter than can prepare most strategies automatically. We recommend that after conversion you check the code to ensure that conversion was carried out properly.

**Required Conventions**

**Procedure**

2.2.1.2.1 Required Conventions

For the code converter to work successfully, your strategy must adhere to the following formatting conventions. Before using the code converter, modify your strategy first according to these formatting rules:
Inputs must be the first executable statement in the strategy. Comments may precede the inputs statement.

- The word Inputs must be the only statement on that line.
- Place each input variable on a separate line.
- Comments on inputs must not span multiple lines.
- Comments on input variable lines must not contain a semicolon (;)

Right:

```
Inputs:
  var1( 15 ), { input is the 1st stmt and is on a line by itself }
  var2( 20 ); { each variable is on a separate line }
```

Wrong:

```
Inputs: var1( 15 ), var2( 20 ); { variables should be on separate lines }
```

Wrong:

```
Inputs: var1( 15 ), { this variable should be on the 2nd line }
  var2( 20 );
```

Wrong:

```
Inputs: var1( 15 ), { this comment spans multiple lines }
  var2( 20 ); { this one contains a semicolon }
```

Restrictions

String variables can't be optimized - this includes High/Low/Open/Close and True/False. Change True/False variables to use 1/0 values instead.

Your EL must be syntactically correct before it can be successfully converted. Incorrect EL syntax will give undefined results, and may result in an infinite Optimax loop or program termination.

2.2.1.2.2 Procedure

To convert your code for use with Optimax:

1. In the Optimax Optimization Settings window, click Convert EL.
2. Paste your code into the left side of the window, and click Convert.
3. Copy your code into a new EL strategy.

Suggestions:
- Keep your original code in a separate strategy for reference.
- Use a consistent naming convention between related strategies. For example: Strat1, Strat1_OMX

**Note**
if the default input value is either 0 or 1, the converter will assume it is a binary 0/1 value and on the Optimization Settings screen will automatically assign Min=0, Max=1 and Inc=1 for that input.

### 2.2.2 Setting Optimization Parameters

The quality of the results you obtain from an optimization depend a great deal on the settings you use. The settings must be appropriate for your fitness landscape, the number of optimization parameters you use, the number of permutations, and the number of generations you will let it run.

To set the optimization parameters, click the Optimization Settings button.

*Clicking the Optimization Settings button will display the Optimization Settings window.*

When running a strategy, Optimax uses the parameters you specify on the Optimization Parameters screen to determine how to generate individuals.

While the optimization is running you can view the parameters, but you cannot change them.

**Current and Historical Settings**
After you have run several optimizations, you will see a list of them on the left of the Main Window in the Run History pane. By clicking a historical row in the Run History pane, you can display the results from previous optimization runs.
The Run History pane is on the left in the main window.

When you do this, Optimax copies those optimization settings to the current settings. If you then click the Optimization Settings button, you can modify the current settings. This does not modify the settings for that historical run. Instead, it modifies the settings that Optimax will use for the next run you perform. Each time you click a historical run, the current settings will be overwritten with a copy of the historical settings.

2.2.2.1 Inputs

Before running your optimization, you need to set up your inputs. If you used the code converter, the inputs will be automatically populated for you. You will probably need to adjust them to suite your requirements.
For each input in your strategy, the inputs frame contains the Input Number, Chromosome Number, Gene Number, Input Name, and Min/Max/Inc values.

If the inputs that Optimax automatically added during code conversion are not what you require or if you manually inserted the OMX calls into your strategy, you can cut and paste the inputs into this screen. Click Select All, then Delete, then place your cursor in the Input Name field, then click Paste.

There must be an input listed here for each OMX_GetParm function call in your strategy. The names listed here must match the names listed on the OMX_GetParm calls. (The comparison however, is case-insensitive.) For example, if you have two GetParm calls, there must two inputs listed here, as shown above. If the names on the GetParm calls are Len1 and Len2, then the figure above displays how they should appear. The order doesn't matter.

Adjust the minimum, maximum and increment values as you desire and set the chromosome numbers on each gene. Group inputs together that have something in common. For example, if you use two stochastics in your strategy, you might use two chromosomes, as shown below.
The Estimates calculator can help you determine how long your optimization is likely to run.

If you do not wish to use one of the inputs for an optimization, set the maximum equal to the minimum and set the increment to 0. You could also delete the input, but you would then need to go to the strategy and remove the GetParm call for that input as well.

When you have finished setting up your inputs, click the Validate button. This will validate your inputs and it will update the Estimate fields at the bottom of the window.

**Estimates**
This section is a small calculator for estimating durations, permutations and generations. You can type values into any of the fields, and it will calculate the rest. For example, to determine how long a search will take, you can type the number of permutations, or let Optimax fill it in. Then, enter the number of seconds (approximately) each iteration of your strategy takes in the Seconds box. The duration of both exhaustive and genetic searches will be calculated and displayed.

Or, if you want to determine the number of permutations that will fit within a given duration, change the duration field and the permutations or generations will automatically update. For example, if you want to run a genetic optimization overnight, change the duration to 8 hours, and the number of generations that can be completed in that time will be computed from the number of seconds per iteration, and the Initial Population Size.

Clicking Validate updates the Permutations and Duration fields from the inputs.
above it.

2.2.2.2 Search Types

When you run an optimization, you can choose between the Exhaustive or Genetic search methods.

Choosing between exhaustive and genetic search algorithms affects the optimization characteristics.

The exhaustive search will search all possible combinations of the inputs. This option is like the exhaustive search used within TradeStation. In addition, it enables you to use the extensive graphical and history bank capabilities of Optimax.

The genetic search enables the powerful genetic algorithms within TradeStation, and makes additional fields modifiable on this screen.

2.2.2.3 Changeling Methods

Whenever Optimax is about to add a new an individual to the population, it first checks to see if that individual already exists in the population. This is call a clone check. The clone check ensures that trials are not duplicated within the same generation, increases diversity and forces exploration of the nearby fitness terrain.

If the clone check determines that a clone exists, Optimax will use one of the changeling methods you select to modify the current individual so that it is no longer a clone.
These are the changeling methods available in the optimization settings.

**Mutation**
When using Random, Optimax creates a new individual by mutation. A single, randomly chosen gene is mutated to a new, valid value for that gene.

**Rival**
When using Rival, roughly one-half of the genes of the individual are shifted up or down by one step to create a near likeness, or a rival to the existing individual. Each gene has a 50% chance of being shifted, and there is a 50% chance of an upward or downward shift. If, after shifting all genes and creating a rival, it is found that there is a clone in the population for this new individual, then the process is repeated until a clone is not found.

**Random**
When using Random, Optimax generates and entirely new, entirely random individual.

**Clone Checking**
After applying any changeling method, Optimax performs the clone check again. If the changeling is itself determined to be a clone, the current method is repeated until the changeling is found not to be a clone.

**Multiple Selections**
If you select more than one changeling method, Optimax chooses between all of the selected methods at random each time it needs to create a changeling.

**Related topic:** Generations

### Related topic: Generations

#### Max Generations
This controls the maximum number of generations that Optimax will record. Once this number is reached, if the number of permutations specified in TradeStation has not been reached, the strategy will "spin," meaning that the remaining permutations in TradeStation will execute over and over using the same parameters as the final individual in the final generation. During this phase the OMX_InitPass call will return 0 and so no further results will be send to Optimax.
**Tournament Group Size**
Tournament Group Size is described [here](#).

**Roulette Scaling Factor**
Roulette Scaling Factor is described [here](#).

**Initial Population Size**
The number of individuals in the first generation, and in all subsequent generations. The number of individuals in a generation determines the maximum possible population diversity, and thus the probability of searching all features within the entire fitness landscape.

The greater number of permutations and inputs, the larger your initial population needs to be to cover the entire fitness landscape successfully without getting stuck on local maxima.

**Maximum Life Span**
Maximum Life Span is described [here](#).

**Elite Percent Taper**
Elite Percent Taper is described [here](#).

**Min nHamming Distance**
Min nHamming Distance is described [here](#).

**Top Percent**
Top Percent is described [here](#).

**Crossover Rate**
A fractional value between 0 and 1 controlling the frequency with which chromosomal crossover occurs during mating. When set to 1, crossover occurs for every chromosome in the mating operation. When set to 0, crossover never occurs. At 0.5, crossover occurs half the time. In nature, crossover almost always occurs, so a value very close to 1 is normal, for example, between .90 and .99. For those chromosome operations where crossover doesn't occur, Optimax simply copies the chromosome from either the mother or father; the donating parent is chosen at random.

**Mutation Rate**
A fractional value between 0 and 1 controlling the rate at which mutation occurs during a chromosomal crossover operation. When set to 0, mutation never occurs. When set to 1, every gene in a chromosome will mutate. In nature, mutation occurs at a very low frequency. A small amount of mutation is beneficial to the evolutionary process, causing Optimax to discover fitness terrain features it might not otherwise find. A high probability of mutation is destructive to the mating process and will prevent convergence from ever occurring.
Genetic Complement Number
The number of genetic complements that will be generated in each generation. A very small number of complements benefit the evolutionary process by increasing the genetic diversity of the mating pool in quantum increments. A large proportion of genetic complements in a generation cause chaos during mating.

The genetically engineered complement's fitness is unknown, so Optimax gives it a fighting chance by always assigning to it the median fitness of the current population.

A value of 1 is common.

Convergence Length
The number of generations over which convergence is measured.

Convergence Std Dev
The standard deviation value used when calculating convergence.

2.2.2.5 Price Modulation
Back testing has one great pitfall - the data never changes. It has no surprises. You can add code to your strategy and modify your inputs until you have managed to handle the price series and generate a nice smooth equity curve. But then what? How can you know if it will work in the future? Even if you had the superhuman patience to paper-trade it for a year, how would you know it would continue to work after that? That, in strategy development, is the one million dollar question. There are no tools or calculations that you can use to tell you how much you have curve-fit your solution. At this point, it's mostly deduction based upon your testing skill balanced with a measure of gut instinct.

Some investors will use multiple stocks to test the same strategy, and that can help. But each stock, like each person, has its own unique personality. A strategy that works on a volatile high-tech stock like KLAC would probably not be suitable for a low-volatility blue-chip stock like GM. To take the same approach to all stocks would be extremely difficult, and it doesn't help you at all to create strategies that exploit the unique characteristics of a stock to obtain maximum gains.

The biggest contributor to the problem is that when you are testing, the data never changes. How can you know what would happen if the data were just a little bit different, or even a whole lot different? And the biggest question is the most obvious: the market is always changing - why would you back test against static data?

Because, until now, you didn't have any choice. There were no tools that enabled you to test against changing data. And that is where price modulation comes in.
In Optimax, the price modulator modifies the price series *dynamically during optimization*. This is an radically new and powerful technique to stress-test your strategy and perform what-if scenarios *all at the same time as you are optimizing*! There has never been such a invaluable and time-saving feature like this available before for TradeStation.

You control price modulation on the Optimization Settings screen, as shown below.

These settings control the depth and character of the price series modulation.

**Waveforms**
The price modulator is composed of two low-frequency oscillators. Optimax uses the output of the oscillators to modulate two aspects of the series: the prices points and the range of the bars. On the left, in the Price Osc Type frame, you select the waveform you want an oscillator to produce. (The Noise Amplitude must be set to zero to see a pure waveform.) In the middle, under the Price Modulation and the Range Modulation tabs, you control the characteristics of each oscillator's waveform. On the right, you see two sample graphs: the left graph shows the pure output from an oscillator, and the right shows a sample bar chart with the waveform applied. When the Price Modulation tab is selected, the Price Osc Type controls the Price Modulation oscillator and the graphs display the output from it. When the Range Modulation tab is selected, the Price Osc Type applies to that oscillator, as do the sample graphs.

**Oscillator Frequency**
The internal resolution of the oscillator is 3600 bars. Another way to state this is that, at an output frequency of 1 Hz, one cycle will last for 3600 bars before repeating. At the oscillator's maximum output frequency of 10 Hz, each cycle lasts for 360 bars.

To calculate the effective duration of the modulation, divide the number of bars per day into 3600. For example, on 5-minute bars with a 1 Hz modulation frequency, one cycle of modulation will last for 3600 / 78 = 46.15 days. At 10 Hz, there will be 10 cycles in that same period and so one cycle will span 4.6 days.

To look at other examples, at 1 Hz on 1-minute bars, the modulation range is
3600 / 390 = 9.2 days; on daily bars, it's 3600 / 1 = 14 years.

**Oscillator Amplitude**
When the output amplitude of the oscillator is zero, the left-hand sample graph will show a flat line crossing at 0, and no modulation will occur. This is useful when you want to modulate only one aspect of the series: either price or range.

At 10%, Optimax will modulate the prices and/or bar ranges by a maximum of +/-10% at the peaks and valleys of the waveforms.

**DC Bias**
Using a non-zero DC bias (direct current bias) adds or subtracts a constant value to the prices and/or bar ranges. When used in conjunction with an amplitude less than 5%, you can create a pure increase or decrease in volatility.

**Noise Amplitude**
Setting a non-zero Noise Amplitude adds a random amount of modulation to the waveform, thus making it "noisy." Without noise, the price series will be statically modulated, meaning that the original, historical series will be modulated, but every trial during the optimization will see the same modulated price series. To use the full potential of price modulation, you must add noise to the modulation. This causes a varying, random amount of modulation to be added to the price series for each individual test, and thus every individual will see a different price series. The amount of difference each individual sees depends on the amount of noise you add to the signal. If you use the maximum noise amplitude of 10%, each individual will see the most difference possible for each test. Using less amplitude will cause each individual to see price series values that are more similar to each other.

The pseudo-random function that Optimax uses is repeatable, meaning that for a given generation/individual number the pseudo-random values will always be the same. This enables Optimax to exactly recreate the price series if you repeat the test, or most importantly when you want to review the trades and/or view the price series in TradeStation the price series will be recreated exactly as it was during the test. Remember, it is not important that the function be truly random; rather, it is important to modify the series in such a way that each test within a single optimization is performed on a differing price series. The repeatable nature of the randomness function ensures that Optimax meets both objectives admirably.

**Noise Frequency**
The noise frequency setting determines how many bars within a cycle are modulated by noise. At 1%, one in 100 bars are modulated, at 10%, one in 10 are modulated, and at 100%, every bar is modulated. Not all bars need to be modulated in order to produce the desired effect of randomizing the price series. When this value is less than 100%, the modulation values is added to each bar's original prices. When the frequency is at 100%, then every bar is modulated by
noise, the values are cumulative, and a random walk effect occurs. This means that the amount of noise modulation applied to each bar is additive, and the series begins to "walk" in random directions.

**Modulation On**
By selecting Modulation On, you enable price modulation for your optimization, and Optimax will use the output from both oscillators to modulate the price points and bar ranges. The right-hand sample graph displays a sample result of the modulation over one quarter of the effective length of the modulation. With Modulation On deselected, no price modulation is used during the optimization, and the right-hand graph shows an unmodulated sample price series.

The sample series is simply a static set of values used by Optimax for a sample display of the effect of the modulation. It is not real data.

**Manual Price Modulation**
When you are not optimizing, you can view the effects of your modulation settings directly within TradeStation using the Modulate Prices function.

To use this function:
- In Optimax first turn Modulation On and then click the Modulate Prices button, shown above.
- Within TradeStation, insert the same symbol into a chart for both datastreams one and two.
- Within that chart, insert the OMX_PriceModX strategy.

The price series on datastream 1 will then exhibit the modulation characteristics you specified in the optimization settings.
**Quick Tip:**
Your reference datastream doesn't also have to be data2 - just change the StreamNum input on the OMX_PriceModX strategy to equal your reference datastream number. For example, if your chart already uses data1 through data4, insert the same symbol as data1 again into data5, and set the StreamNum input to 5.

We use data2 in the documentation only as an example, making it easier to read.

For example, here you see a price modulation applied to APOL in datastream one. Below it is the same datastream, unmodulated.

*Comparing a modulated with an unmodulated price series.*

When you compare datastream one (modulated) with datastream two (unmodulated), you can precisely see the effects of the modulation. Most of the effects in this chart are subtle, but if you look closely you can see the differences.
In the highlighted rectangles you can see some easily distinguishable modulation effects. The degree of these changes is entirely under your control - they can be as subtle or as violent as you choose; that depends solely on the settings you specify in the optimization settings window. You can adjust the settings until you achieve the effects you desire.

**Important Note:**
When Optimax is in Price Modulation mode, each time you change the modulation settings, you need to toggle modulation mode off and back on again, and then toggle the OMX_PriceModX strategy off and on again in order to observe the effects of the change on the TradeStation chart.

The settings shown below were the ones that we used to create the effect you see above.

These price modulation settings created the effect shown above.

These range modulation settings created the effect shown above.

After adjusting the modulation settings and verifying their effects using the above procedure, you can run the optimization with confidence that the price modulation will perform in a manner that you consider appropriate for your purposes.

When you turn the OMX_PriceModX strategy off (or remove it,) restore data1 to
its' original state by reloading the chart in TradeStation using View, Refresh, Reload.

When you have performed an optimization with price modulation enabled and after the optimization is complete, you can view the effects of the price modulation for any single individual. To do this, simply enter the generation and individual number into the inputs for the strategy. The price series and trade series will be reconstructed within TradeStation and will display in datastream one. Optimax must be up and in the "optimization stopped" mode to accomplish this.

**Refresh Timer On**

When adding random noise, the modulation effect becomes different for every individual in an optimization. To get a feel for how the datastream will vary between individuals, select Refresh Timer On. The sample graphs will refresh continuously while varying the random elements of the modulation as they would from individual to individual during an optimization.

**Sample Settings**

Here are some sample settings for you to try.

<table>
<thead>
<tr>
<th>Instrument</th>
<th>Effect</th>
<th>Application</th>
<th>Bar Length</th>
<th>Price Modulation</th>
<th>Osc Type</th>
<th>Osc Freq</th>
<th>Osc Amp</th>
<th>DC Bias</th>
<th>Noise Amp</th>
<th>Noise Freq</th>
<th>Range Modulation</th>
<th>Osc Type</th>
<th>Osc Freq</th>
<th>DC Bias</th>
<th>Noise Amp</th>
<th>Noise Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>@ES</td>
<td>Mild volatility expansion</td>
<td>- general curve-fitting avoidance - mild stress-test for swing-trade strategies, especially those with pyramiding</td>
<td>1 minute</td>
<td>One of triangle (line = longer consolidation times, triangle = reversal)</td>
<td>0.5 to 3.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20 to 99%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Summary**

Now that it is available, price modulation should become an essential part of all of your back testing. Using static data during a back test is now considered archaic as it easily leads to curve-fitting and provides an unrealistically static test bed for basing your strategy evaluations on.

2.2.2.5.1 Modulating Multiple Streams

You can modulate data streams other than data1 by changing the code in the prolog. A sample is shown here:
Above we have added calls to PriceModX2 and X3, thus causing data streams 2 and 3 to also be modulated in addition to data 1. This can be useful when you have a strategy that uses more than one data stream and you wish to modulate multiples streams at the same time. For example, if you have $ADV and $DECL on data 2 and 3, you could use this approach to add noise to the $ADV and $DECL data streams as well as whatever you have on data1.

You do not have to modulate all your streams. For example, you could choose to modulate only the 2nd data stream by calling only the OMX_PriceModX2 function.

You will need one reference data stream for each different stream you wish to modulate. For example, if you have @ES on data1, $ADV on data2 and $DECL on data3 and you want to modulate them all, you would insert @ES again on data4, $ADV again on data5 and $DECL on data6. You would then set the StreamNum inputs as shown in the sample code above.

2.2.3 Running Optimizations

When you are ready to run the optimization, follow these steps:

- Insert your strategy into a chart.
- Add another datastream, and make it identical to Data1. (This additional stream is used by Optimax as the price modulation reference stream.) In your strategy, change the OMX_StreamNum input to be the number of the datastream you just inserted. For example, if you just inserted data4, set OMX_StreamNum to 4.
- In Optimax, click the Optimize button.
Clicking the Optimize button in Optimax.

Wait until you see the message "Waiting for TradeStation to send data...". This may take a minute if you have a slow machine; this is especially true if you are short on RAM.

- In TradeStation, optimize your strategy, varying OMX_IterationNum from 1 to a Stop number, with an increment of 1.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OMX_IterationNum</td>
<td>0</td>
</tr>
<tr>
<td>OMX_Generation</td>
<td>0</td>
</tr>
<tr>
<td>OMX_Individual</td>
<td>0</td>
</tr>
<tr>
<td>OMX_EquityResolution</td>
<td>100</td>
</tr>
<tr>
<td>L.Len1</td>
<td>15</td>
</tr>
<tr>
<td>L.Len2</td>
<td>20</td>
</tr>
</tbody>
</table>

Varying the IterationNum parameter in TradeStation.

Do not optimize any other inputs. They will all be controlled from Optimax.

To determine the Stop number, multiply the number of individuals per generation by the number of generations you want to process. For example if you have set the Initial Population Size parameter to 100 and you want to process 50 generations, the Stop should be at least 500. You can set it higher, because you can stop processing at any time you wish. (See below.)

You must click the Optimize button in Optimax before clicking the Optimize button in TradeStation. Starting the optimization in TradeStation first will give unpredictable results.

- What happens then is this: at the first bar of each iteration within TradeStation, the EL code asks Optimax for a set of input values and evaluates the strategy using them. At the last bar, the EL calls Optimax functions to write the strategy results to a file and then starts a new iteration, obtaining the next set of inputs from Optimax. This is repeated until you stop the process, until convergence is reached, until the Max Generations parameter is reached, is or until the last iteration number. Usually, it is easiest to just use a very high iteration number and wait for convergence or stop it manually.

Note: When either convergence or Max Generations is reached, TradeStation
will "spin" through the remaining Iteration numbers until it reaches the Stop value. These iterations do not do anything useful; you can click Abort during this time and not affect the results.

- Periodically, the Optimax user interface checks for new strategy result files and from them creates graphs and updates indicators. You can vary the refresh rate - that is the number of seconds before Optimax looks for new files and generates new graphs.

![Adjusting the refresh rate in Optimax.](image)

Each refresh requires computer resources to perform. Ideally, you want to set this high enough so you can watch your optimization in progress, but low enough so that it doesn't noticeably slow down TradeStation. If you have two CPUs, then TradeStation will use one CPU and Optimax will use the other, and Optimax will not impact TradeStation's speed at all. In this case you can choose a high refresh rate - but not higher than the time taken to perform a single refresh.

At the other end of the scale, if you have a single CPU on a slow machine and want to give as many cycles as possible to TradeStation, choose Manual. Optimax will then wait for you to click the refresh button before generating graphs.

![Clicking the Refresh button in Optimax.](image)

You can even turn off the Animation icon in the top right-hand corner of the window by clicking Stop Animation. When you do this, Optimax is in
maximum conservation mode and only consumes the cycles necessary to
generate inputs required for optimization. It does not give any outward
indication that it is doing anything.

The default refresh rate is 15 seconds, visible in the status bar at the bottom
of the window.

- You can stop the optimization at any time. In TradeStation click Abort,
  Restore Original Inputs. Then in Optimax, click Stop Optimization.

Stopping the Optimization in Optimax.

When you stop the optimization in TradeStation, do not click Accept Best
Solution So Far. It will not give you the best solution so far; it causes the last
iteration number to be sent to Optimax out of sequence and the results are
unpredictable.

Once you have pressed the Stop button in Optimax, the optimization cannot
be restarted. Pressing Start will start a new optimization.

2.2.3.1 Evaluating Results

Once you have stopped the optimization, you can evaluate the results using both
Optimax and TradeStation. The following sections describe in detail the functions
within Optimax for this all-important phase of your optimization.

The Equity Grid
The Inputs Grid
The Search Space Indicator
The Input Space Indicators
The Equity Detail Report
Displaying Trades in TradeStation
Covering the Terrain
Evaluation In a Nutshell

2.2.3.1.1 The Equity Grid

The equity grid contains a graph for every individual (combination of inputs) in
the optimization. Each graph displays the equity curve for a particular individual,
as shown below.
The equity grid displays equity curves for all individuals in an optimization.

Clicking on an equity graph displays the detailed information for that graph at the bottom of the window. Graphs without an equity curve (such as G1-2 and G1-4 above,) indicate a combination of inputs that did not generate any trades.

You can drag the split bar at the bottom of the equity grid up or down to display more or fewer rows.

**Controlling Thumbnails**
Clicking the thumbnail size buttons shown below switches the size of the graphs in the equity grid, known as *thumbnails*, to small, medium or large.

By selecting small, medium or large thumbnails you change the size of the graphs in the equity grid.

The first time you display a thumbnail, Optimax creates a GIF image in a cache folder on your PC. After that, when it needs to display the graph again, it looks for the cached GIF and, if there, displays it directly. This improves the speed of scrolling in the equity grid.
You can set the characteristics of each thumbnail size separately using the thumbnail settings menus shown below.

Selecting thumbnail options for the next optimization run.

Choose the menu item For This Historical Run to change the characteristics of the run you are currently viewing. Choose the menu item For The Next Run to cause the thumbnail settings to affect your next optimization and not any of the existing runs.

For information on managing your thumbnail cache, click here.

2.2.3.1.2 The Inputs Grid

The Inputs Grid, shown below, presents all input combinations tested during the optimization in a spreadsheet-like format.

Using the inputs grid, you can see all of the inputs for all individuals in a spreadsheet-like format.

Clicking a graph in the equity grid highlights the inputs grid row associated with it. Duplicate values within columns are suppressed, so you can easily see when a value changes. The combination of inputs in each row of the equity grid is
guaranteed to be unique within a generation.

Right-click and select Copy (or press CTRL+C) to copy the contents to the clipboard; from there you can paste it into a spreadsheet.

To display all inputs and all performance numbers combined into a single grid, right-click and select Open Detail Canvas, as shown below.

This will open a window in which you can perform detailed numerical analysis and create graphs of your input and performance results.

2.2.3.1.2.1 The Inputs Grid Detail Canvas

To display all inputs in a single grid with all performance figures, right-click the Inputs Grid in the main window and select Open Detail Canvas, as shown below.

Selecting the menu to display the Inputs Grid Detail Canvas.

The Inputs Grid Detail Canvas, also simply called the IDC, displays all inputs and performance results in a single grid. Here you can sort and graph your results, and look for correlations between inputs and performance numbers.
Using the IGrid Detail Canvas, you can see all input combinations with the performance results in a spreadsheet-like format.

A key feature of the IDC is the ability to see the associated equity curves on every row. Optimax uses the small thumbnails to display these grids. To change the size or characteristics of these graphs, adjust the small thumbnail settings for that historical run.

When you click any row, Optimax displays the detailed performance report for that individual in the main window.

You can sort by any column by clicking the heading. Clicking again sorts in the opposite direction. The sort is stable, enabling you to easily sort by multiple columns. To do this, perform your sort on the columns in the reverse order. For example, if you want your final sort sequence to be Total # of Trades then Fitness, first click on the Fitness heading to sort it, then click on Total # of Trades.

You can move a column by dragging it to a new position.

Ease-of-use Tip
To easily re-order columns, first maximize the window and then select Fit All Columns to Window so that all columns are visible at once. Hover over each column, watching the status bar at the bottom of the window to observe the name of the column, then drag it to the new position.

Resize columns by pointing at that headings and dragging the separator bar.
between them. Resize rows by pointing at any cell in the left-most column and dragging the horizontal separator bar. You may also use the menu functions to resize all columns to fit the values, to fit the headings, or to the greater of both - this is called the Fit All Columns to Max function in the menu.

Select the Functions menu or right-click the canvas to display a list of functions you can perform.

Selecting the Functions menu.

The Functions menu only contains functions that you can perform on all columns at once. Right-click the canvas to select single-column versions of these
functions.

**Hiding and Showing Columns**
Select the Hide All Columns function to hide all columns except the first column of data. Select Show All Columns to display them all again. To hide or show a single column, select that column from the Columns menu. The check-mark beside its' name indicates its' current state; checked indicates that the column is visible, unchecked means it is hidden. The first column of data can't be hidden, so you are not able to select or deselect it in the menu.

If you have a lot of columns, the Column menu will extend beyond the vertical range of the screen. At both ends will be an arrow, enabling you to scroll up or down in the menu.

**Ease-of-use Tip**
To easily scroll in the Columns menu, open the menu, then hold down the up and down arrow keys to scroll. Press Enter to select the item with the highlight.

**Ease-of-use Tip**
Hold down the CTRL key and right-click any column to hide it.

**Graphing Columns**
Another key feature of the grid is the ability to graph within the cells of the grid, as show below.
In-cell graphing enables you to see the raw numbers, the graphs and the equity curves all at the same time.

Positive numbers are colored green and extend into each cell from the left; negative numbers are red and extend into the cell from the right.

Reducing the size of the rows enables you to see many more rows at one time, and when the row height is small enough Optimax will create a continuous graph effect. An easy way to do this is to select the Fit All Rows To Window function, and then resize the window until you see the continuous graph effect, as shown below.

The continuous graph effect enables you to easily spot correlations between column values.

When hovering over any cell, Optimax displays the value from that cell in the status bar at the bottom of the window, along with row statistics such as the...
minimum, maximum and standard deviation of the values in that column. Note that Optimax only displays statistics for columns that have been graphed.

You can increase or decrease the size of the rows (jog them up or down) in small increments by holding down the Alt key and pressing the up-arrow or down-arrow keys.

The size of the equity thumbnails does not change as you change the size of the rows, only the visible portion of the graph that is displayed. When the rows are very small the graphs cannot be easily seen so no new images will be loaded into empty cells; this increases performance when paging up or down.

When you select the Graph All Columns function, hidden columns are not graphed to increase performance.

**Flooding Columns**

In addition to in-cell graphs, you can flood the background color of a column with a gradient of values, as illustrated below.

Optimax computes a gradient value from green to red corresponding to the maximum to minimum values in the first column following the equity curves. It then copies the gradient from the first column to the second (and subsequent columns, if you are using an All Columns function) as shown below.
This helps you to visualize how the values in the first column are distributed and how they relate to the values in other columns. It is most useful when you use it simultaneously with in-cell graphing, as show below.

The main purpose of this is to enable you to easily visualize how values from the first column change as the value from another column changes. It is most useful when you want to examine relationships between two columns. By sorting a secondary column, you can more easily see the relationship between the change in values between the two columns.
Here you can see the relationship between Fitness in column 1 and Len1.

In this example, we have flood-graphed with fitness in the first column, then sorted by the Len1 input. This gives us a clear depiction of how Len1 affects fitness. The smallest Len1 values give the lowest fitness values, and the highest Len1 values yield better fitnesses, with some exceptions as shown by the red and orange lines cutting through the upper section of values.

By moving a different column into the first position and then performing the flood-graph operation again, you can cause the background gradient across all columns to reflect any column of information.

When you select the Flood All Columns function, hidden columns are not flooded to increase performance.

**Scaling Types**
There are two types of scaling, linear and standard deviation, selected as shown below.
When graphing using the linear method, Optimax considers the maximum positive value within the column to be +100% and draws a full green bar at that value, and the minimum negative value to be -100% and draws a full red bar at that value. At 0, no bar is drawn. This is useful for values that behave in a linear fashion, such as columns containing input values.

When graphing using the Standard Deviation method, +/-100% is set equal to +/-3 times the standard deviation of the column. Values above or below 3 times the standard deviation are drawn at 100%. This enables you to graph columns containing outliers without the outliers disrupting the scale of the graph. This method can be useful when graphing performance numbers, which often contain outliers.

When flooding using the linear method, Optimax considers the maximum value within the first column to be bright green, and the minimum to be bright red. It then internally creates a linear scale between the two extremes and locates each value in the column on that scale, drawing each as a color on the linear gradient between bright green and bright red. A number that is on the mean between the maximum and minimum is pure yellow. Notice that when using this method red does not always represent a negative number, it only represents the minimum value. Likewise, green is not always positive, it is the maximum. This method is useful for visualizing the distribution of values between maximum and minimum. Yellow indicates values at the mean.

When flooding using the standard deviation method, bright green corresponds to 3 times the standard deviation of the column values, and bright red corresponds to -3 times the standard deviation. Yellow corresponds to 0. Using this method, green will always represent a positive value, and red always represents a negative value. Outliers will not disturb the gradient as they would using the linear method, so this method is appropriate for performance results, while the linear method is more useful for input values.

Select the Clear All Columns function to clear all graphing and flooding. Optimax does not clear hidden columns to increase performance. Right-click and select Clear This Column to clear a single column.

**Decimals**

By default, Optimax displays the grid with values rounded to 0 decimal places. Using the Set Decimals function, you can set the number of decimals for any or all columns. Values are rounded for display to the number of decimals you specify. To set the number of decimals for a single column, right-click that column and select the Set Decimals For This Column function.

**Controlling Images**

By default, the IDC uses image auto-load, meaning it automatically loads images each time you scroll so that images are only loaded on an as-needed basis. This gives a good balance between performance and convenience. You can turn this
feature off by selecting the menu function Image Auto-load Off, enabling you to scroll more quickly. You can then load the images manually for the current page using the menu function Load Images For This Page.

Alternatively, you can select Load All Images and Optimax will generate and load all images into the grid in a single operation. Once all images have been loaded, scrolling in the grid is very fast. With all images loaded Optimax will also enable scroll tracking, meaning that rows will scroll as you drag the scroll bar, without waiting for you to release it.

If you hide the image column, images will not be loaded until you show it again. This is another way you can increase scrolling speed.

**During An Optimization**
You can display the IDC both while an optimization is in progress and after it is complete. Your grid format changes will not be saved while an optimization is in progress. However, once the optimization is complete, Optimax remembers all of your grid settings such as column order, width and graphing when you close the window. The next time you open the grid it will appear as it was when you closed it.

**Saving To A File**
Use the Save As Excel menu function to save the grid data and formatting to an Excel-97 XLS file. Excel does not support as much functionality as the IGrid, so some formatting will not be saved, such as equity curve images and in-cell graphing.

Use the Save As CSV menu function to save the raw grid data to a file on your hard drive as a text file with values separated by commas. This type of file is readable in most spreadsheet programs. No formatting is saved to this type of file.

2.2.3.1.3  The Search Space Indicator

The Search Space indicator, also known as the SSpace Indicator is a rectangular area near the center of the main window, as show below.
The SSpace indicator lies near the center of the Optimax main window.

This indicator is designed to display how much of the search space has been tested during an optimization, and how well distributed the tests have been over the fitness landscape. The base rectangle represents the entire possible search space (all possible input combinations.) Optimax draws a vertical line for each individual test it performs. Each line is drawn at an offset from the left of the rectangle, the value of which is calculated from the input combination values. Each possible input combination can be represented by a unique vertical line in the entire space. If the screen resolution were high enough to allow all permutations to be displayed and the entire space were solid purple, that would indicate that all possible combinations had been tested. If it were entirely white, that would indicate that no combinations have been tested. A display such as that shown above indicates that the search space has been covered fairly evenly, with some concentration in several areas. This is a normal display for a completed genetic optimization. An exhaustive optimization would cause the indicator to act like a traditional progress bar, because all possible input combinations would be tested in sequence until they are exhausted, as shown below.

The SSpace indicator as it would appear in a partially complete exhaustive optimization.

Internally, the width of the indicator is wide enough to contain a single vertical line for every permutation. If the number of permutations is greater than the visible width of the indicator in pixels, then the indicator is scaled to fit and some degree of approximation occurs when rendered on the screen. In this situation, it would be possible for the indicator to be entirely white, and yet some tests could have occurred. At the other end of the scale, it would be possible for the
indicator to be entirely filled in purple without all possible permutations having been covered. This is due to the approximation necessary to display the indicator in an external resolution less than the internal resolution.

Right-click and select Copy if you wish to copy the image to the clipboard.

2.2.3.1.4 The Input Space Indicators

The Input Space, or ISpace indicators, display a more detailed representation of the inputs that were tested and simultaneously represent the results of each test. These indicators are shown below.

The input space indicators reside in the main window just below the inputs grid. This optimization is approximately half-way through the first generation.

These indicators play a vital role in understanding the topography of the fitness landscape. Above you see several indicators - one black box each. Each indicator sits beneath a column of the inputs grid, and displays the results for that input. Resizing an inputs grid column width will resize the associated ISpace indicator.

If an input is static (max = min for that input in the optimization settings window) then the associated ISpace box will be completely black.

Clicking an indicator box, as shown below, displays the indicator in a detail window. Or, right-click and select Copy if you wish to copy the image to the clipboard.

Clicking an ISpace indicator opens a detail window for it.

The detail canvas window enables you to closely inspect the effect the input
values had during the optimization. Clicking anywhere on the detail canvas displays crosshairs and detail information about each cell.

The crosshairs are hovering over individual 11 and input value 43.

Above we see a input space detail canvas for an input named Len1 after the optimization has been completed. The individuals proceed from left to right across the X-axis, and the input values from bottom to top on the Y-axis.

Below, we see another example of input space detail canvas, this one for an input named Len2.
The crosshairs are hovering over individual 40 and input value 8.

As the optimization proceeds, cells are drawn from left to right as Optimax tests each individual of a generation. The color of the cell reflects the fitness of each individual. During the optimization, Optimax keeps track of the best and worst fitness values and dynamically creates a color gradient from red to green, with pure red representing the worst fitness value so far, and green representing the best. The result is a series of colored cells varying from red through orange through green with colors reflecting how fit each individual is and vertical position indicating the value tested.

Once the first generation is complete, Optimax begins drawing cells again from the left to right, overlaying cells that are already drawn. This creates set of virtual layers, built up from the first to the last generation, each one laying on top of the previous, allowing cells to show through to lower layers if they are not overlayed on a higher layer.

The end result is a type of heat map that displays color bands clustering around input values having similar fitnesses, making it very easy to recognize favorable and unfavorable input ranges.

Without these indicators, you would need to analyse all of the inputs in the inputs grid to determine the most favorable combinations and their stabilities - this would not be such an easy task. With these Input Space indicators this is not
necessary; at a glance you can see how well all input values did for each input. These innovative labour-saving indicators were developed by us specifically for Optimax; they cannot be found in any other product.

**More Details**

On a detail canvas, there are three buttons and a slider enabling you to control how Optimax draws the cells. These are shown below. These buttons act like radio buttons; only one of them can be selected at a time.

![ISpace Detail Canvas toolbar control its' drawing modes.](image)

When you select the Frame-By-Frame button, moving the slider displays each generation separately, one at a time, erasing the previous drawing each time. This mode enables you to inspect the evolution of the inputs and their fitness by isolating one generation at a time; you can easily see which combinations were tested and the fitness obtained for each.

With Build-up, when you drag the slider, drawing occurs one generation at a time, keeping and overlaying earlier generations. In this mode, you can observe the progress of the evolution at each stage and watch the bands build up over time. It is possible in this mode to uncover earlier results and view them without being covered up by later generations.

The third button - Overlay All - draws all layers one on top of another, as a final product. You cannot change the slider in this mode, since it is a static view of the final result. This mode is useful when you are in either of the other two modes and you wish to quickly compare a given view with the final result. This is the same drawing mode as is used to display the indicator normally, as originally discussed when we first introduced these indicators.

In the first two modes, the left-most position of the slider displays all generations simultaneously and displays the same picture as the Overlay All mode.
The Main Window ISpace Slider
On the Optimax Main Window, there is also an ISpace generation slider, shown below.

The generation slider on the main window affects all of the ISpace indicators on the main window simultaneously.

Optimax displays this slider after the optimization is finished. It behaves like the one in the ISpace Canvas window in Frame-By-Frame mode, but it affects the ISpace indicators on the main window all at the same time. When dragging it, you will see a pop-up displaying the generation numbers. When you stop, it will draw that single generation in all of the indicators to the right. With it, you can watch the evolution of all inputs simultaneously. When it is positioned at the top of the slider, it is in its normal mode and will display all frames as a final result.

You can enlarge these ISpace indicators by dragging the split bar immediately above them up and down, and by resizing the widths of the columns of the Inputs Grid.

2.2.3.1.5 The Equity Detail Report

For each graph in the equity grid, you can click and display the performance details, as shown below.
Clicking a cell in the equity grid displays the detail information below.

On the bottom left, you can see the equity curve in detail. Beneath the primary curve you will see the equity curves for both parents of the individual. Generation 1 individuals do not have parents, so the parent graphs will be blank for those individuals. The split bars above and to the right of the equity curves enable you to control the size of the graphs. When there is enough room, the parent graphs will display below the primary one. If you cannot see them, adjust the split bars.

Right-clicking and selecting Copy copies a graph to the clipboard. This action copies both the picture and the data; if you paste into a paint program, it pastes a picture. If you paste into Excel, it pastes the data so that you can analyze it or redraw it using Excel graphing capabilities.

You can control the resolution of these equity graphs. For more information on this topic, click here.

On the right is the performance detail report. This report contains all of the important performance numbers from each test, as well as lineage and other
Optimax User Guide

optimization information, as shown below.

The performance report contains all of the key figures you need to evaluate an individual's performance.

The first column, Performance, contains a list of variables sent from TradeStation during the optimization. For your reference, the title for this column displays the run, generation and individual numbers to identify the information displayed. Negative values in the body of the report appear in red. You can add as many more of these values as you like, by performing your own calculations in EasyLanguage and sending the result to Optimax. For more information on how to do this, see Enhancing the Epilog.

The second column, Lineage, informs you of the parents of this individual, and any gene modifications that Optimax performed during mating. If one of the parents was a genetic complement, the parent number will be preceded by a tilde (~). The Embryonic Transform field identifies the changeling method used, if any, during the clone prevention phase of mating. The Crossover Count and Mutation Count fields tell you how many crossover operations and mutations were performed.

The Inputs column displays the input values that form the individual's makeup. The From Mother/From Father columns display the inputs of the mother and
father, with the mother's values displayed in green and the father's in blue. The color of each child's input matches the color of either the mother or father, depending upon which one is the donating parent. This makes it easy for you to see where the input value came from. If both the mother and father values are equal for a given input, the color of the child's input value will be black. If the value did not come from either the mother or father, then it is the result of a mutation, and is displayed in red.

You can highlight and press CTRL+C to copy cells, or right-click and use the context menu to copy the entire performance report to the clipboard. You can then paste it into a spreadsheet.

2.2.3.1.6 Displaying Trades in TradeStation

After an optimization is complete you can display, within TradeStation, the trade series for any individual. To do so, you can type in the input values by copying them from Optimax. If there are many inputs, simply type the generation and individual into the strategy inputs with iteration = 0, as shown below.

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OMQ_IterationNum</td>
<td>0</td>
</tr>
<tr>
<td>OMQ_Generation</td>
<td>4</td>
</tr>
<tr>
<td>OMQ_Individual</td>
<td>67</td>
</tr>
<tr>
<td>OMQ_EquityResolution</td>
<td>100</td>
</tr>
<tr>
<td>_Len1</td>
<td>42</td>
</tr>
<tr>
<td>_Len2</td>
<td>0</td>
</tr>
</tbody>
</table>
```

Setting the generation and individual numbers in TradeStation to see the trade series.

In this state, your strategy calls Optimax to obtain the input values, and displays the trade series that created the equity curve. If the curves do not match, ensure that your chart uses the same begin and end dates as it did during the optimization, and that the data hasn't changed from TradeStation.

2.2.3.1.7 Covering the Terrain

Due to the nature of genetic searches, your results may vary with each optimization run. This does not mean that they are invalid - there can be many satisfactory sets of inputs for a given strategy - it only means that the parameters you used for the search did not cover the entire fitness landscape each time. Whether your search finds all maximums depends to a large extent on the number of individuals in the initial generation. Try increasing this number for highly complex landscapes.
Genetic searches search by approximation - there is a tradeoff between speed and thoroughness. In most cases the tradeoff is more than satisfactory, since this is the only way possible to obtain representative samples from many parts of the fitness landscape in a practical length of time. Genetic optimizations are not capable of discovering a best-case single case solution isolated on the fitness landscape without some degree of gradually approaching fitnesses. If there are not many clues pointing in that direction, it probably won't find it. Ultimately, those single-maxima solutions are not desirable in strategy optimizations anyway, since they are inherently unstable and are usually a curve-fitted result, not a result of real-world relationships exploited by your strategy.

Running the optimization several times and obtaining similar results each time can give you a level of comfort that you have adequately scanned the fitness landscape.

2.2.3.1.8 Evaluation In a Nutshell

Hindsight is always 20/20. Creating a strategy that gives good results from historical data is only the start of your task - your strategy must continue to perform in the future with data it has never before seen in order for you to make money. If the stock market changes - and it will - this will affect your results, usually for the worse. To avoid nasty surprises in the future, here is a checklist of things to look for while optimizing.

Evaluation Checklist
Stable Inputs
Look for green bands in the ISpace indicators which indicate areas of input stability. The wider the band, the more stable is the area of values and the smoother the change will be from a winning to a loosing strategy. Remember, if you change the inputs a little, the equity curve should only change a little. If it changes a lot, this indicates that you are probably curve-fitting. The more drastic the change, the more likely it is that your strategy will fail in the future when it encounters new market conditions. Look for the green bands to avoid this pitfall.

Consistent Returns
Look at the equity curves - if they look like a straight line drawn at a 45-degree angle then the returns are consistent. The absolute net profit means little without consistency. Using OMX_F_MSRatio or a similar measure of consistency for your fitness will increase your chances of finding consistent returns. Remember, if your strategy's returns are consistent from period to period, then it is much more likely to continue to perform consistently in the future. If the results vary widely from period to period, then it is likely to vary even more widely in the future. Look for straight-line equity curves to side-step this common error.

Convergence, Not Divergence
Look for the similarity in equity curves as the optimization progresses, indicating the optimization is coming to a natural end. These curves should become more and more like each other. This is called convergence. If they stay widely diverse
or become more diverse over time, then the settings are not appropriate for your strategy and you have not successfully used Optimax to assess your strategy.

**Even Search Space Coverage**
Look for a uniform coverage of the SSpace indicator, indicating that the search space has been well-covered and no major input combinations remain unsearched. Big gaps of white indicate that some part of the terrain was not covered. Coverage will never be perfectly uniform - there will always be some clustering of purple - because Optimax concentrates its search in the higher fitness areas.

**Few Bands**
Look for few bands within the ISpace indicator panels. The fewer the bands, the less probable it is that you are curve-fitting. Many bands indicate a complex fitness terrain and a higher probability of curve-fitting and thus future failure of the strategies.

**Consistent Results**
Run the same optimization multiple times and look for consistent results. If the banding occurs in the same places at the end of each optimization, then you know that Optimax has adequately covered the entire search space each time. But if the results are different each time, then the space was not adequately searched. To correct this, increase the Initial Population Size setting, or decrease one of these parameters: Genetic Complement number, Mutation Rate, Elite Percent Taper, Maximum Life Span, Tournament Group Size, Roulette Scaling Factor, or increase Top Percent.

2.2.3.2 Managing Optimizations

Optimax keeps track of the history of all of your optimizations. These appear as a list in the Run History pane of the main window, shown below.

![Optimax Visual Optimizer](image)

*The Run History pane enables you to go back and review the results of previous optimizations.*
You can delete historical runs by highlighting a row, then by using a right-click and selecting Delete Run on the context menu, as shown below:

Selecting a row and right-clicking displays a context menu in the Run History pane.

The data for these historical runs are stored in a folder on your PC - in Program Files/Optimax/Run History - as shown below. Deleting a run within Optimax deletes the associated numerical folder.

Optimax stores run history in folders on your PC.

You can also delete run history folders manually if you wish; be sure that Optimax is not running if you choose to manually work with the Run History folders.
You can also protect and or unprotect a run using the associated context menu options. This sets the associated folder attribute to read-only, and will protect you from accidentally deleting a run you wish to keep. To then delete it, you must unprotect it first.

**Notes & Comments**

By expanding the Run History pane, you can type in comments for each run to the right, as shown below.

Here you can type free-form comments.

To begin typing, double-click the desired cell, or highlight it and press F2. Pressing Enter saves the comment; pressing Escape cancels the changes. The maximum comment length is 128 characters.

If you want to store longer comments or more complex information, right-click the row as shown below.
Here you can select from two types of documents.

Each of the first two menu items will open a document for you to store additional notes in. If you select the RTF item, Optimax will create a file named N#.RTF where # is the run number, and open it using the default program for RTF files (Word, for example.) This file is stored in the run history folder associated with that run.

By selecting TXT, Optimax will create a similar file, but with the TXT extension, and open it with the default program associated with TXT files - usually Notepad.

These can be used for storing any type of notes you like; you could even cut & paste your strategy into the notes file. This way you will always have a copy of the EasyLanguage that you used in that run (in case you change it later,) and you will have the basis for version management for your strategies.

Remember, Optimax never transmits your information to the Internet, so you don't need to worry about us being able to see your source code.

Note:
When you start Optimax it transmits your Customer Key to our servers for the sole purpose of verifying your level of access to Optimax functionality. No other information is ever transmitted to the Internet.

Thumbnail & Indicator Cache
The thumbnails displayed in the equity grid and the indicators displayed beneath it are cached in the run's associated GIF subfolder, shown above. If you keep a lot of old optimization runs, this cache can start to take up a lot of space. You can safely delete GIF subfolders if you want to keep save space. The next time
you display a run, Optimax will recreate the GIF subfolder and rebuild the thumbnails within it.

If the thumbnails or indicators become corrupt for some reason, you can force Optimax to rebuild them by shutting it down, deleting the associate GIF subfolder, starting it back up and displaying the run again. This procedure should clear up any problems you have should you encounter display corruption.

Archiving
You can archive your old optimization data by moving the numerical run history folders to another location - to CD for example. You can copy this data back at any time and the next time you run Optimax it will find and display these runs. We recommend that you always leave the most recent optimization run folder in place, since Optimax begins numbering by finding the highest existing number and adding 1 to it. If you remove the highest numbered folder, Optimax will begin sequencing from number 000001 and you will end up with duplicate folder names. Although you can rename folders (be sure to use an entirely numeric name,) this does make run management more difficult for you, and so is not recommended.

2.2.3.3 Restrictions

Currently, Optimax has the following restrictions:

- The chart you are optimizing can contain only one active strategy. For example you can't use a separate stops and targets strategy - all of the code must be within one strategy, and only that strategy can be active on the chart when you are optimizing.
- You cannot run multiple instances of Optimax while optimizing. Doing so can cause the run history to become corrupt.
- We do not recommend optimizing while online to TradeStation during market hours. Doing so will usually cause the charts to change during optimization and could invalidate the results. If you need to run an optimization spanning several days and you want to use TradeStation market analysis during the day, we recommend setting up the optimization on a separate computer with TradeStation running offline.

2.2.4 Optimizing Sample 1

In this example we step through the process of optimizing the OMX_Sample1 strategy. This is a simple moving average crossover strategy. The code for it is shown here:
2.2.4.1 Preparing The Strategy

In this step you will create take the code from OMX_Sample1_A, convert it, and create a new strategy OMX_Sample1_B with the converted code.

1. In TradeStation, open the EasyLanguage document OMX_Sample1_A. Highlight all code and copy it to the clipboard.
2. In Optimax, click the Optimization Settings button, then click Convert EL. Paste the code into the left pane and click Convert.
3. Highlight all of the code in the right pane, and copy it to the clipboard. Switch back to TradeStation, create a new strategy named OMX_Sample1_B, paste the code and verify the strategy.

OMX_Sample1_B is now ready to be optimized with Optimax.

Note
OMX_Sample1_C contains pre-converted code for your convenience. You can view the application of this strategy in the sample workspace OMX_Sample1.tsw.

2.2.4.2 Setting Optimization Parameters

Set up your optimization parameters as illustrated in the figure below.
In addition, set the range modulation tab options as follows:

Click the Validate button to ensure your settings are correct. Note that the Estimates calculator updates itself whenever you click Validate.

**Note**
These options are a simple example of settings you can use for an optimization. You are encouraged to explore and try other options to see how they affect your results.

### 2.2.4.3 Running The Optimization

1. In TradeStation, create a daily bar chart for APOL from 12/7/1994 to 11/28/03. Insert APOL twice - once on data1 and again on data2. Use the same date range for both.
2. Insert your strategy into the chart and set the properties as follows:
• Commission $0.012 per share
• Slippage $0.01 per share
• Initial Capital 100,000
• Interest Rate 0
• Maximum number of bars study will reference: 330
• Fixed Shares: 100

3. Click Start Optimization in Optimax
4. In TradeStation, optimize the iteration number from 1 to 9999.
5. After about 15 to 30 generations, you should see a pattern developing in the ISpace indicators, and the graphs in the equity grid should be fairly uniform within each generation. When you see this, stop the optimization by clicking Abort, Restore Original Inputs in TradeStation, then click Stop Optimization in Optimax.

2.2.4.4 Evaluating the Results

In our run, looking at the Search Space indicator, we can see that the search space was covered fairly evenly, and probably with enough penetration to discover the major features of the fitness landscape. Wider purple rectangles indicate areas in which the search was concentrated.

*The search space indicator shown here indicates fairly even search space coverage.*

With the ISpace indicators, we can see that favorable values for Len1 cluster around 42, and values for Len2 cluster around 8. To see an indicator clearly, click it as shown here:

*ISpace indicators display a heat map of the evolutionary process.*

Here we see the input space detail canvas for the Len1 input.
The Input Space indicator for the Len1 input indicates a primary clustering of favorable inputs around value 43.

Here we see the input space detail canvas for the Len2 input.
The Input Space indicator for the Len2 input indicates a solitary clustering of favorable inputs around value 8.

Returning to TradeStation and plugging Len1=43 and Len2=8 into the inputs for the strategy (while setting OMX_IterationNum, OMX_Generation and OMX_Individual to 0) yields the following equity curve:
Several trials using nearby values for Len1 and Len2 also yield similar equity curves. This means that Optimax has identified a stable area in the fitness landscape with smooth approaches to a maximum.

**Note:**
Due to the nature of genetic searches, it is possible that your results will not be exactly the same as ours. You should, however, be able to obtain comparable results. Running the optimization several times may yield different results due to the randomness inherent in the search process. In any event, you should be able
to obtain satisfactory results when Optimax is used correctly.

2.2.5 Next Steps

Now that you’ve successfully performed your first optimization, you'll want to try Optimax with your own strategies. If you follow the same procedure as you did for Sample 1, you should be well on your way. You may encounter some bumps along the road; this chapter should help you to smooth them out as you travel on your road to financial freedom.

Using the Run History Pane

After you have performed an optimization, make notes about it by double-clicking the User Notes column, and by right-clicking it to create more detailed notes. This will help you remember what each run was for. For example, you might type "Sample 1 test" beside your first optimization. You can even right-click and paste your EL into the TXT or RTF documents so that you will always have a copy of it to go along with the run, in case you change it in the future. That way, you can always reproduce the same equity curve if you change something and worsen your returns.

When you want to start another optimization, simply highlight the run history row that contains the optimization settings you want to use. You can modify them first before starting the optimization. Modifying them does not change the settings for that historical run - instead it sets them up for the next run. When you close the Optimization Settings window, the settings are ready to use. Click the Start Optimization button, start optimizing in TradeStation, and those settings will now be saved with the new run. Clicking an old run always restores the original settings from history again.

Learning the Ropes

After you’ve successfully optimized a strategy, here are a few things to try:

- Run your first optimizations with price modulation turned off. (In the Optimax Optimization Settings window, clear the Modulation On checkbox.)
- Run the same optimization three times, once using each fitness function to compare how the fitness function affects the optimization results. To do this, simply uncomment each function in turn in the Epilog.
- Turn on price modulation, and insert the OMX_PriceModX strategy into a new chart. Play with the price modulation settings to get a feel for how they affect the price series in TradeStation.
- Re-optimize the strategy with price modulation enabled to see how this affects the performance. Remember, at this point you aren’t trying to improve the returns - you’re stress-testing the strategy to see how quickly it will fail. Expect lower returns; if your strategy continues to perform successfully then you will have determined that your returns have a higher probability of success in the future as well.
**Rules of Thumb**
To avoid errors, you will need to follow these guidelines when working with Optimax:

- If you have multiple versions of TradeStation installed, close Optimax before using a different version of TradeStation, the start Optimax back up again.
- If you get an error in Tradestation during a run - for example a Max Bars Back violation - you must stop the optimization in both TradeStation and Optimax and then start a brand new optimization as per normal. Optimax will remember your settings, so you only have to click Start in Optimax, and then optimize the IterationNum input in TradeStation to start again.
- Always start the optimization in Optimax first, then in TradeStation.
- Always stop the optimization first in Tradestation, then in Optimax.
- When stopping the optimization in TradeStation always select Cancel Optimization And Restore Original Inputs.
- Always save your workspace with the IterationNum, Generation and Individual number inputs set to 0. If they are set to any other number, you will get an error when you open the workspace.
- Do not defragment or back up your hard drive during an optimization. Doing so can prevent your results from being sent to Optimax. Before starting an optimization, turn off any programs that do this automatically, such as Diskeeper or Second Copy.

### 2.2.6 DLL Errors
There are some situations that can cause you to receive a DLL error message. These are listed below:

- if you start the optimization in TradeStation before Optimax. (You must start the optimization in Optimax first.)
- if you stop the optimization in Optimax before TradeStation (You must stop the optimization in TradeStation before Optimax.)
- if you call a OMX_GetParm function with a name that doesn't exist in the Inputs section of the Optimization Settings window. (Make sure the names match, and there are the same number of Inputs in both places)
- if you stop the optimization in TradeStation and select "Accept best solution so far." (Select "Restore original inputs" instead.)
- if you have a value other than zero in the IterationNum input in your strategy without having Optimax in "optimization running" mode
- if you have a value other than zero in the Generation/Individual inputs in your strategy without having, in Optimax, the associated run highlighted in the run history pane
- if you have a value other than zero in all three input fields IterationNum/Generation/Individual at once
2.2.7 FAQ

When running an optimization I get the following event error in TradeStation: Floating point invalid numbers in calculation. What's wrong?

Check your optimization settings in Optimax and ensure that their names match those used on the GetParm function calls in the OMX prolog.

What bar lengths does Optimax support?

Optimax supports all bar lengths from monthly down to 1-tick bars. Note that for tick, volume or point-and-figure charts you may need to create your own fitness functions.

What does it mean when I see a No Pic message in the equity grid?

If you see the No Pic message, it means one of two things:

- the computer was too busy when Optimax tried to draw the picture and it wasn't able to complete the drawing. In this case, scrolling the cell out of view and back into view should cause the cell to draw correctly. This happens more frequently when your graphics subsystem is slowed down, for example when you are using remote viewing software at the same time as running Optimax.

- the OMX_BarInfo function wasn't called from within your strategy during the optimization for that individual. If this is the case, is possible that the that means that the OMX_BarInfo function was never called from your strategy during the optimization for that individual. This function is called from the epilog. The following are some possible reasons for this problem:
  1. you specified more bars back in the strategy properties than are contained within the chart
  2. the EasyLanguage code is structured incorrectly such that the function is never called
  3. the OMX_Init variable is not = 1 while the OMX_IterationNum is > 0

Can I include more calculations in the performance detail report?

Yes! You can send as many values as you like from your strategy to the performance detail report in Optimax. You do this in the epilog code within your strategy. Click here for more information.

Can I change the resolution of the equity curves in Optimax?

Yes, you can control the resolution of the equity graph using the EquityResolution parameter. Click here for additional information.

The equity curve in TradeStation doesn't match the one in Optimax. How can I fix this?

If the curves do not match:

- ensure that your chart uses the same begin and end dates as it did during
the optimization
• ensure that the data in the chart hasn't changed. Often, TradeStation updates the information to correct bad ticks. If this has happened, your charts won't match.
• ensure that the optimization was not performed while online during market hours. If it was, the data can change during the optimization, causing the charts not to match.

2.2.8 Glossary

Chromosome
A set of related genes. You define chromosomes in the Optimization Settings window.

Complement
The theoretical genetic opposite of an individual. Genetic complements are made from the least fit individuals in the hopes of creating a fit individual. The number of complements added per generation can be controlled via the optimization settings.

Father
An individual which contributed genetic material for the purpose of creating another individual. In Optimax, there is no difference between a mother and a father.

Fitness
A floating-point number from -1E300 to +1E300 representing the desirability of a strategy's equity curve. It is usually the result of a user calculation within the strategy. You pass this value from the strategy to Optimax at the end of each iteration. Positive values are considered fit; negative values are unfit.

Gene
An input used during an optimization. You define genes in the Optimization Settings window.

Generation
A set of individuals generated during an optimization run.

Individual
One particular combination of inputs for a strategy. Optimax generates individuals during an optimization run and evaluates the fitness of each by passing the inputs to TradeStation.

**Input Space**

The conceptual mathematical area covered by all possible permutations of a set of inputs as described in the input options for an optimization run. Numerically, it is expressed as a one-dimensional scalar value. Graphically, it is displayed by the Input Space Indicator using a set of parallel vertical lines whose offset within the indicator indicate the relative position of the input combination within the entire input space.

**Input Space Indicator**

An indicator which graphically displays the distribution of input combinations evaluated so far within one optimization run. See also: Input Space.

**ISpace**

See: Input Space.

**ISpace Indicator**

See: Input Space Indicator.

**Lineage**

The information tracing the ancestry of an individual.

**Mother**

An individual which contributed genetic material for the purpose of creating another individual. In Optimax, there is no difference between a mother and a father.

**OMX**

A short form for the word "Optimax."

**Roulette Selection**

A mating selection method in which individuals are chosen with at random with a probability proportional to each individual's fitness.

**Search Space**

In purpose it is similar to an Input Space, but has a more complex expression.
Numerically, it is expressed as a two-dimensional value and graphically it is expressed using a combination heat map/scatter plot. The X/Y coordinates of the plot represent individuals and generations, and the color of each point represents the fitness.

**Search Space Indicator**

An indicator which graphically displays the distribution of input combinations evaluated within one optimization run, graphically depicting input combinations having the highest fitness values. See also: Search Space.

**SSpace**

See: Search Space.

**SSpace Indicator**

See: Search Space Indicator.

**Tournament Selection**

A mating selection method in which a number of individuals are chosen at random to compete in a tournament. The individual with the highest fitness wins the tournament and goes on to mate.

### 2.3 The Optimax API

Optimax provides a published interface enabling developers to extend and enhance it's interface capabilities and the functionality of it's optimization. Using these calls, one could possibly use Optimax to perform genetic optimizations for previous versions of TradeStation, or other products such as Wealth-Lab, MetaStock, and even non-stock related applications. Optimax also provides exits from the user interface so that you can add your own functionality.

**Optimax Architecture**

Optimax controls and communicates with TradeStation via Optimax.dll located in the Windows\System32 directory, as shown below.
Because TradeStation does not publish a remote-control interface, the strategy optimization is initiated from TradeStation by iterating over a simple iteration counter. The Optimax DLL receives the iteration numbers and various other pieces of information and then determines what needs to be done with each call.

### 2.3.1 Easylanguage Functions

Optimax includes several open-source Easylanguage functions used to communicate with the Optimax interface before, during and after optimizations. Although possible, we recommend that you do not modify these functions. Instead, if you deem a change desirable, make a copy and modify the copy.

#### 2.3.1.1 Fitness Functions

Use the following fitness functions within the strategy's Epilog as a way of reducing the strategy's performance to a single numerical value.
Because all optimization decisions are based on the output of the fitness function, the correct choice and use of a fitness function is paramount in achieving your optimization goals. Toward this end, you may want to consider trying OMX_F_GPF. This function is a product of our own research and development - it was created specifically for Optimax and is the most comprehensive fitness function by far that we know of. Because of its complexity, we recommend you start with a simpler function first, then as you gain experience and require more control from your optimization, try OMX_F_GPF.

We also encourage you to create your own fitness functions and to experiment as there are as many ways to evaluate fitness as there are people creating strategies.

2.3.1.1.1 OMX_F_GPF

**Summary:**
OMX_F_GPF is a customizable fitness function that measures and combines several fitness components into a compound value. Call this function in the Epilog to compute a fitness value.

**Ease-of-use Tip**
Sample code for using this function is available within TradeStation in strategies OMX_CutAndPaste and OMX_Sample1_C. Simply open one of those strategies and cut and paste the sample code into your own strategy to use.

**Prototype:**

```c
OMX_Fitness = OMX_F_GPF(
    OMX_IterationNum,          { The Optimax Iteration Number
    PrintInfo,                 { 1=print detail info to the output window, 2=\GPF.txt
          0=don't      }
    SendVarsToOMX,             { true=call OMX_SendVar for each of the intermediate values
    RejectNegNetProfit,        { Always reject a negative net profit value?
    PeriodType,                { 1 = day, 2 = week, 3 = month
    InvestmentCapital,         { Available Investment Capital (in $)
    ATP_Tgt,                   { --- Avg Number of Trades Per Period
    ATP_Wgt,                   { Target # of trades per period
    ATP_Rng,                   { Weighting factor
    ATP_Rng,                   { Acceptable +/- Range
    ATP_Bal,                   { Balance between SSDev and Proximity (1 to 0)
    ATP_TLT,                   { Time-line Scaling Type (1=Linear, 2=Exponential)
```
ATP_TLW,                   { Time-line Scaling Weighting factor (0 to +/-N.N) }
ATP_Pos,                   { Use positive values only? (Ignore negatives?) }
\(--\) PPT_Tgt,                   { Target Pct Profitable trades per period 1 = 1%, 80 = 80% }
PPT_Wgt,                   { Weighting factor }
PPT_Rng,                   { Acceptable Range }
PPT_Bal,                   { Balance between SSDev and Proximity (1 to 0) }
PPT_TLT,                   { Time-line Scaling Type (1=Linear, 2=Exponential) }
PPT_TLW,                   { Time-line Scaling Weighting factor (0 to +/-N.N) }
PPT_Pos,                   { Use positive values only? (Ignore negatives?) }
\(--\) ABT_Tgt,                   { Target number of bars per trade }
ABT_Wgt,                   { Weighting factor }
ABT_Rng,                   { Acceptable +/- Range }
ABT_Bal,                   { Balance between SSDev and Proximity (1 to 0) }
ABT_TLT,                   { Time-line Scaling Type (1=Linear, 2=Exponential) }
ABT_TLW,                   { Time-line Scaling Weighting factor (0 to +/-N.N) }
ABT_Pos,                   { Use positive values only? (Ignore negatives?) }
\(--\) BTR_Tgt,                   { Target ratio of bars per winning trade to bars per losing trade } 
BTR_Wgt,                   { Weighting factor }
BTR_Rng,                   { Acceptable +/- Range }
BTR_Bal,                   { Balance between SSDev and Proximity (1 to 0) }
BTR_TLT,                   { Time-line Scaling Type (1=Linear, 2=Exponential) }
BTR_TLW,                   { Time-line Scaling Weighting factor (0 to +/-N.N) }
BTR_Pos,                   { Use positive values only? (Ignore negatives?) }
\(--\) ACI_Tgt,                   { Target avg % of capital invested }
ACI_Wgt,                   { Weighting factor }
ACI_Rng,                   { Acceptable +/- Range }
ACI_Bal,                   { Balance between SSDev and Proximity (1 to 0) }
ACI_TLT,                   { Time-line Scaling Type (1=Linear, 2=Exponential) }
ACI_TLW,                   { Time-line Scaling Weighting factor (0 to +/-N.N) }
ACI_Pos,                   { Use positive values only? (Ignore negatives?) }
\(--\) MCI_Tgt,                   { Target max % of capital invested }
MCI_Wgt,                   { Weighting factor }
MCI_Rng,                   { Acceptable +/- Range
MCI_Bal,                   { Balance between SSDev and Proximity (1 to 0)
MCI_TLT,                   { Time-line Scaling Type (1=Linear, 2=Exponential)
MCI_TLW,                   { Time-line Scaling Weighting factor (0 to +/-N.N)
MCI_Pos,                   { Use positive values only? (Ignore negatives?)

--- Average Win / Avg Loss Ratio - maximized
AWL_Wgt,                   { Weighting factor
AWL_TLT,                   { Time-line Scaling Type (1=Linear, 2=Exponential)
AWL_TLW,                   { Time-line Scaling Weighting factor (0 to +/-N.N)
AWL_Pos,                   { Use positive values only? (Ignore negatives?)

--- Profit Factor - maximize
PRF_Wgt,                   { Weighting factor
PRF_TLT,                   { Time-line Scaling Type (1=Linear, 2=Exponential)
PRF_TLW,                   { Time-line Scaling Weighting factor (0 to +/-N.N)
PRF_Pos,                   { Use positive values only? (Ignore negatives?)

--- NetProfit to MaxDrawdown Ratio
NMR_Wgt,                   { Weighting factor
NMR_Pos,                   { Use positive values only? (Ignore negatives?)

bDoComputation,            { Only compute when this value is True. usually LastBarOnChart
PeriodReturn,              { Equity Return in each period
PeriodTrades,              { Number of trades in each period
PeriodWins,                { Number of wins in each period
PeriodLosses,              { Number of losses in each period
PeriodPctPft,              { Percent profitable by period
PeriodGrossProfit,         { Gross profit by period
PeriodGrossLoss,           { Gross loss by period
PeriodProfitFactor,        { Profit Factor by period
PeriodEvenBars,            { Total bars in even trades by period
PeriodWinBars,             { Total bars in winning trades by period
PeriodLosBars,             { Total bars in losing trades by period
PeriodTotalTrades,         { Total number of trades by period
PeriodABT,                 { Period average bars per trade
PeriodAWL,                 { Avg Win/Loss per period
PeriodBTR,                 { AB Win/AB Loss ratio per period
PeriodACI,                 { Avg % of capital invested per period
PeriodMCI ;                { Max % of capital invested per period

©2004 Modus Novus Inc.
Parameters:
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Possible Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OMX_IterationNum</td>
<td>NumericSimple</td>
<td>Any integer</td>
<td>The Optimax iteration number</td>
</tr>
<tr>
<td>PrintInfo</td>
<td>NumericSimple</td>
<td>0,1,2</td>
<td>When 1, this function prints detailed information about the internal computations to the TradeStation Output Print Log window. This information is useful for tuning the weighting parameters and for determining the optimum size of the input arrays. When 2, the same information is printed to file \GPF.txt (in the root directory,) instead. When 0, no information is printed. When True, this function sends information about internal computations to the Optimax Performance Detail Report. This information is useful for tuning the weighting parameters. When True, equity curves which end with a negative Net Profit will always have a negative fitness value. When False, only the potential for generating positive equity across the entire curve determines the sign of the return value, not the final value of the equity.</td>
</tr>
</tbody>
</table>
Function Type: Series
This is a series function - it executes at the end of the every bar.

Returns (Double)
This function returns a floating-point value which can be negative or positive, with values above 0 indicating desirable fitness and values 0 or below indicating undesirable fitness.

Description and Use
This function creates a compound, comprehensive fitness value from several discrete components. It is compatible with any type of strategy. Use it when you want to achieve fitness objectives other than a simple maximum net profit.

Notes
- Within this document the term GPF refers to this function and the output from it.
- Be sure to read the Period Arrays section below before attempting to use this function.

Introduction
This is not a "black box" fitness function as most are. This function is parametric, meaning it requires you to specify several input parameters. The output value depends upon the input parameter settings. In order to use this function successfully and to obtain the desired results, you are required to understand the correct use of the inputs. So instead of using many different fitness functions that contain components in a predefined static configuration, this one is flexible in that you can customize it by specifying parameter combinations to emphasize varying aspects of fitness and so customize the function according to the type of chart, type of strategy and your personal fitness goals.

Parameters and Components
Within this function are discrete internal computations, known as internal components. Input parameters control this internal fitness components, and are as follows:
Component (per period) | Specifiable Parameters
--- | ---
Number of Trades Per Period | Target value, Weight, Range, Timeline Scaling, Pos-Only
Percent Profitable Trades | Target value, Weight, Range, Timeline Scaling, Pos-Only
Number of Bars per Trade | Target value, Weight, Range, Timeline Scaling, Pos-Only
Win/Loss Trade Bars Ratio | Target value, Weight, Range, Timeline Scaling, Pos-Only
Avg % of Capital Invested | Target value, Weight, Range, Timeline Scaling, Pos-Only
Max % of Capital Invested | Target value, Weight, Range, Timeline Scaling, Pos-Only
Avg Win/Avg Loss Ratio | Weight, Timeline Scaling, Pos-Only (value is always maximized)
Profit Factor | Weight, Timeline Scaling, Pos-Only (value is always maximized)
NetProfit to Max IDD Ratio | Weight, Timeline Scaling, Pos-Only (value is always maximized)

The final GPF value is the sum of the internal component values, as shown here:

\[
GPF = ATP + PPT + ABT + AWL + BTR + ACI + MCI + PRF + NMR
\]

where:
- ATP is Average Number of Trades per period
- PPT is Percent Profitable Trades
- ABT is Average number of bars per trade
- AWL is Average Win/Average Loss ratio
- BTR is Win Bars Per Trade/Loss Bars per Trade Ratio
- ACI is Avg % of Capital Invested
- MCI is Max % of Capital Invested
- PRF is Profit Factor
- NMR is NetProfit to Max IDD Ratio
- GPF is the output value from this fitness function

**Subcomponents**
Each component is built from several subcomponents, as follows:

- The raw component value
- The weighting factor you specify for the component
- The proximity to the target you specify for the component
- The measure of uniformity of the raw component values on a per-period basis (This is referred to the SSDev value in the subsequent documentation.)
GPF combines subcomponent values together to obtain a final component value, according to the following equation:

\[
C_{\text{Final}} = \left( (C_{\text{Raw}} \times (1-C_{\text{Bal}}) \times C_{\text{Proximity}}) + (C_{\text{Raw}} \times C_{\text{Bal}} \times C_{\text{SSDev}}) \right) \times C_{\text{Weight}}
\]

where:
- \( C_{\text{Final}} \) is the final value for the component
- \( C_{\text{Raw}} \) is the raw (average per period) subcomponent value
- \( C_{\text{Bal}} \) is the balance between the importance of the SSDev and Proximity subcomponents
- \( C_{\text{Proximity}} \) is the proximity to target subcomponent value
- \( C_{\text{SSDev}} \) is the SSDev (measure of uniformity) subcomponent value
- \( C_{\text{Weight}} \) is the weighting specified by you for the subcomponent

For example, to obtain the Percent Profitable Trades component final value:

\[
PPT_{\text{Final}} = \left( \text{XOrNegProduct}(PPT_{\text{Raw}} \times (1-PPT_{\text{Bal}}), PPT_{\text{Proximity}}) + \text{XOrNegProduct}(PPT_{\text{Raw}} \times PPT_{\text{Bal}}, PPT_{\text{SSDev}}) \right) \times PPT_{\text{Weight}}
\]

This calculation takes a portion of the Raw value and weights it by proximity. It then takes the remaining portion of the Raw value and weights it by uniformity per period. It adds the two weighted portions together and then weights the entire sum by the weighting factor you specified for the subcomponent. A perfect score for a subcomponent is 1, provided the maximum weighting value you use is 1.

If you specify a balance (PPT_Bal) value of 0.5, then the proximity to target is equally as important as the uniformity per period. You can adjust this by altering the balance value. For example, setting balance to 0.8 causes 80% of the importance to be placed on the uniformity, and only 20% on the proximity to target. Setting balance to 1 causes only uniformity to be considered. Setting it to 0 causes only proximity to target to be considered.

If either the Raw or the Proximity value or BOTH are negative, the result of the multiplication is negative. This is accomplished via the XOrNegProduct function. This also holds true for the Raw and SSDev values. This unique treatment of the product of values ensures that a negative portion of the component remains negative; if it didn't it would incorrectly increase the value of the subcomponent.

If you have specified Positive Only = True for the component and the component's final value is negative, GPF sets the final value to 0. This causes negative values to be ignored instead of subtracting them from the final GPF. Use this option when you have fewer equity curves with positive values and you want to increase the diversity of the initial search space. If you have a majority of curves with positive GPF values, using Positive Only = False helps the GPF to concentrate on the most promising areas and should increase the
rate of convergence.

**Component Weights**
When you call OMX_F_GPF, you specify weighting values for each component. These values can range from 0 to any number. If you set all weights to the same value (e.g. if you set them all to 1) then all components will be considered equal in weight. Usually, you would weight the profit factor highest, then percent profitable and/or NetProfit to MaxIDD ratio, then the rest below those. You can cause a component to be ignored completely by specifying a weighting factor of 0 for that component.

Note that you will have to experiment with the weights in order to determine appropriate values. You will find that raw values can vary quite a bit from component to component, so you must use the weights to compensate for differences in raw values as well as to apply relative strength to each.

**Target Value and Range**
The target value enables you to specify what you consider to be an ideal value for one component. The range specifies the acceptable range of values from the target. If a value exceeds the target +/- the range, then the fitness will become negative and the individual will be rejected. The closer the value is to the target, the higher the resulting fitness value will be. Values at the edges of the range result in lower fitness values.

For example, an ideal target value for Number of Bars per Trade on a 1-minute chart may be 10, within an acceptable range of +/-5, meaning that a test that results with Number of Bars Per Trade of 10 will have the highest fitness value. Tests that have values down to 6 and up to 14 will result in a positive fitness. Values at the edge of and outside of that range will result in a 0 or negative fitness.

The GPF function computes the proximity of the a given value to the target value as follows:

- a value that is 50% of the way to the edge of the range is a 50% (0.5) match
- a value that is 25% of the way to the edge of the range is a 75% (0.75) match
- a value that is equal to the target value is a perfect match (1.0)
- a value that is 100% of the way to the edge of the range is a 0 match
- a value that is greater than 100% away is a negative match

So for example if the target is 10 bars per trade with a +/-5 range, then an 8-bar trade is a 0.6 match, a 6-bar trade is 0.2, a 10-bar trade is a 1 (perfect) match, a 12 is a 0.6 match, 14 is 0.2, 15 is 0, 16 is -0.2 etc.

A range of < 1 presents special cases. Specifying a range of < 1 on any value
that can only be an integer will result in the highest selectivity and will only pass a single value because any value that equals the target will a 100% match, and any other value will be a negative match.

One may think that there should not be a specifiable target for Percent Profitable but rather it should always be maximized. That is fine - if you wish to always maximize it, set the target to 100. However, in real life it is impossible to achieve 100% profitable trades and any back-test that indicates 100% profitable trades should thus be suspected of displaying strongly curve-fit results. A more realistic target would be 80% profitable trades. Specifying a target of 80% will discount performance results about 80 and the closer they are to 100 the more they will be discounted, thus penalizing results according to their probability of being curve-fit. Doing this should yield more accurate optimization results.

To minimize a value such as the Average Bars Per Trade, specify a target value of 1 and a Range equal to the maximum acceptable Average Bars Per Trade plus one.

The RejectNegNetProfit Parameter
If you specify the parameter RejectNegNetProfit = True, you can cause the GPF fitness to always be negative whenever the Net Profit of the back test is negative. This is a traditional approach because it seems intuitive that if the test lost money, then it would be undesirable. However, it could also be argued that if the test made money at some point, even though at the end it lost, then it contains some potential for making money and the fact that it lost at the end was only a co-incidence of the end-date for the test and a negative portion of the equity curve. To rephrase, if your end-date for the test had been different, then the NetProfit could have been positive and the result would not have been rejected. So using this GPF function you have the option to also consider curves that end in a negative NetProfit value that yet exhibit positive traits throughout much of the test period.

Periodicity
Periodicity is a key concept in this fitness function. It is built in to every component except the NMR component.

Instead of using a single period for computing each value as most fitness functions do (that period being the duration of the back-test,) this function computes values on a per-period basis, and then averages them to obtain the raw subcomponent per component. You can specify that the period be daily, weekly or monthly using the PeriodType parameter, as follows:
<table>
<thead>
<tr>
<th>Period</th>
<th>Type</th>
<th>Period Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>day</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>week</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>month</td>
<td></td>
</tr>
</tbody>
</table>

GPF stores all historical values into separate bins for each period and computes the consistency of the series of bin values. This measure of consistency is called the SSDev subcomponent. GPF uses the SSDev subcomponent value to weight the component in addition to using your weighting as well. For example, assume that in a particular back-test over 6 months there were 42 wins and 18 losses. The percent profitability is 70%. The ideal distribution of those trades is:

**Distribution 1**

<table>
<thead>
<tr>
<th>Period</th>
<th>Wins</th>
<th>Losses</th>
<th>% Profitable</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>3</td>
<td>70</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>3</td>
<td>70</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>3</td>
<td>70</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>3</td>
<td>70</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>3</td>
<td>70</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>3</td>
<td>70</td>
</tr>
</tbody>
</table>

---

42 18 70%  

**Total % Profitable Trades**

The above distribution is obviously more desirable than the one below, which also results in exactly 70% profitable trades, but only averages 20% per period.

**Distribution 2**

<table>
<thead>
<tr>
<th>Period</th>
<th>Wins</th>
<th>Losses</th>
<th>% Profitable</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>4</td>
<td>91</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

---

42 18 70%  

**Total % Profitable Trades**

20%  

**Average % Profitable Trades**
Most fitness functions will see both of the above results as equally desirable, because they only look at the totals, not the distributions. The GPF function sees them differently. First instead of computing the percent profitable from the totals only, it computes percent profitable for each period separately and averages them. So the average percent profitable for the first series is 70%, whereas the average percent profitable for the second series is only 20%. GPF uses the average as the Raw subcomponent.

Secondly, the GPF function also penalizes uneven distributions in addition to any penalty incurred from the averaging. This is done because averaging alone does not always penalize uneven distributions, as shown below.

<table>
<thead>
<tr>
<th>Distribution 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>Total</td>
</tr>
<tr>
<td>Average</td>
</tr>
</tbody>
</table>

Here we can see that a less-than-ideal distribution can actually result in a higher average than the simple total computation, and so needs to be penalized due to its less-than ideal distribution. To penalize uneven distributions, GPF always computes a proprietary SSDev factor as shown here:

$$SSDev = 1 - \left( \frac{\text{StdDev of PctProfitable per period}}{\text{Avg PctProfitable per period}} \right)$$

SSDev is the standard deviation of the percent variability of the value, standardized on a scale from 1 to -infinity, where 1 is no variability 0 is 100% variability, and -infinity is infinite variability.

GPF multiplies the average value by the SSDev factor. If the distribution is perfect (as it was in Distribution 1,) then the SSDev factor will be 1, and so there is no penalty after multiplication. The more uneven the distribution, the lower the SSDev factor will be, and thus the higher the penalty.

For example, computing the SSDev factor for Distribution 1 yields no penalty: $\text{StdDev} = 0$
Avg = 70
SSDev = 1 - ( 0 / 70 ) = 1
Result: No penalty

For Distribution 2, there is a heavy penalty:
StdDev = 19.1
Avg = 70
SSDev = 1 - ( 19.1 / 70 ) = -0.625
Result: Heavy penalty. The final GPF will be negative and so this individual will be entirely discarded.

For Distribution 3, there is a minor penalty:
StdDev = 32.5
Avg = 70
SSDev = 1 - ( 32.5 / 20 ) = 0.73
Result: Minor penalty. The Percent Profitable component of the final GPF will be multiplied by 0.73 and so reduced by only 27%.

GPF computes the average and SSDev values for all components of the GPF except the NMR (NetProfit to Max IDD Ratio.)

When using this function as compared to other fitness functions, you will find that this function emphasizes consistency in all of the measurements. This should result in less scattering on the ISpace indicators and both fewer and narrower bands than with other fitness functions. The end result is a more clear indication of the input combinations that yield the targeted fitness values.

**Period Arrays**
GPF uses arrays to store the period bin values - one array element per bin. The speed of the function is directly proportional to the size of the arrays - the bigger the arrays, the slower the function will be. In order to enable you to maximize the speed of GPF, you define these arrays in your strategy and pass them to GPF. If you get an Array Out Of Bounds message from TradeStation when using GPF, then you need to increase the size of the arrays.

Compute the necessary array size as follows:

\[
\text{ArraySize} = \text{Number of Periods} + 1
\]

So for example, if you are using daily periods and your are back testing over 100 days, then define the arrays as [101]. Set all GPF arrays to the same size.

You can also determine how big the arrays should be by setting the PrintInfo parameter to True; GPF will print a message telling you the optimal array size.
size. (See the section below on PrintInfo for more information on using this parameter.)

**Important Note on the use of Periods**

For period processing to be entirely accurate, you should ensure that the first period processed by your strategy begins on a period boundary, and the last period ends on a period boundary. For example, for type 1 (daily processing), you would ensure that the first bar processed by your strategy is the first bar of a day and the last bar is the final bar of a day. If your strategy starts processing the first day in the middle or ends in the middle then the period calculation for the first and/or final period will be incorrect, because the first and/or final period is not a full day. The same holds true for types 2 and 3, (weekly and monthly periods.) For type 2, you should ensure that your strategy processing begins on a Monday (if you're using weekly periods) and ends on a Friday or for type 3, on the 1st of a month and ends on the last day of a month.

Starting the first period in the middle or at the end primarily affects the computation of the SSDev value (the uniformity measurement.) The extent of the effect is determined by the number of periods you process. The greater the number of periods you process, the less it will matter. For example, if you choose daily periods and have 200 of them in your back test, then you have at most only 1% error in your performance calculations if the first and last periods are not entire periods. If on the other hand you choose weekly periods and you process only 4 periods and the first one begins on a Friday, then you have introduced a 25% error margin into the fitness computation.

A simple way to ensure that the first and last periods are fully processed to the boundaries you intend is to add code to your strategy to prevent processing before and/or after those points in time and then add an extra period to the beginning and end dates of the chart.

**Time-line Scaling**

GPF enables you to weight the importance of the results from recent periods more heavily than for older periods. For example, when you look at an equity curve, if the first part of the curve is good but the last part looses money you might say to yourself, "It would have been a good strategy a year ago, but lately it isn't doing well. The market has changed so I don't like this curve as much as one where performance was not as good a year ago, and is doing well lately." To this end, GPF includes *Time-line scaling*, meaning that you can cause measurements from recent periods to be more heavily weighted than measurements from earlier periods. You can specify the scaling type as linear or exponential, and a scaling weight factor. The higher the weight factor, the more important GPF considers recent period measurements.

GPF applies time-line scaling to the Raw average value of the component, and to the SSDev value of the component, so both average and uniformity are
scaled with time according to the scaling type and weight factor you specify.

When you specify linear scaling, period values are scaled according to the following algorithm:

\[
\text{for } i = 1 \text{ to } \text{NumPeriods} \{
\text{Weight} = \text{Weight} + ((i - 1) \times \text{AbsValue(WeightFactor))}) ;
\text{WeightedSum} = \text{WeightedSum} + (\text{Weight} \times \text{PeriodValues[i]}) ;
\text{Divisor} = \text{Divisor} + \text{Weight} ;
\}
\text{Result} = \frac{\text{WeightedSum}}{\text{Divisor}} ;
\]

where:
- WeightFactor is the Time-line scaling weight factor you specify
- NumPeriods is the number of periods in the back-test
- PeriodValues is an array of the period values in the back-test

When you specify exponential scaling, period values are scaled according to the following algorithm:

\[
\text{for } i = 1 \text{ to } \text{NumPeriods} \{
\text{Weight} = \text{Weight} \times \text{AbsValue(WeightFactor))}) ;
\text{WeightedSum} = \text{WeightedSum} + (\text{Weight} \times \text{PeriodValues[i]}) ;
\text{Divisor} = \text{Divisor} + \text{Weight} ;
\}
\text{Result} = \frac{\text{WeightedSum}}{\text{Divisor}} ;
\]

where:
- WeightFactor is the Time-line scaling weight factor you specify
- NumPeriods is the number of periods in the back-test
- PeriodValues is an array of the period values in the back-test

For example, if you were to specify a scaling type of 1 (linear) and a factor of 1, recent periods would be considered more important than earlier periods. With a factor of 2 or 4, even more so. With a factor of 0.5, recent periods are considered only slightly more important. With a factor of 0, all periods would be evenly weighted and no time-line scaling would occur.

If you were to specify a scaling type of 2 (exponential) instead, the scaling becomes exponential and tapers off much more quickly as you go backward in time. When using exponential scaling, a factor of 2 weights more recent results much more than for linear scaling. A factor of 1 causes even weighting to occur across all periods.

To get a feel for the effects of time-line scaling, the following table illustrates some sample results.

Assume there are 4 periods in the back test with period values of 1, 2, 3, 4:
In the above chart, WSSDev is the weighted SSDev value. (The weighted SSDev value is used for all computations within the GPF function.)

For another example, assume there are 10 periods in the back test with period values of 100, 90, 80, 70, 60, 50, 40, 30, 20, 10:

<table>
<thead>
<tr>
<th>Type</th>
<th>TLS Weight Factor</th>
<th>Average</th>
<th>Scaled Avg</th>
<th>Std Dev</th>
<th>Scaled Std Dev</th>
<th>WSSDev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>-32</td>
<td>2.5</td>
<td>1.51</td>
<td>1.12</td>
<td>1.20</td>
<td>0.52</td>
</tr>
<tr>
<td>Linear</td>
<td>-16</td>
<td>2.5</td>
<td>1.52</td>
<td>1.12</td>
<td>1.20</td>
<td>0.52</td>
</tr>
<tr>
<td>Linear</td>
<td>-8</td>
<td>2.5</td>
<td>1.55</td>
<td>1.12</td>
<td>1.20</td>
<td>0.52</td>
</tr>
<tr>
<td>Linear</td>
<td>-4</td>
<td>2.5</td>
<td>1.59</td>
<td>1.12</td>
<td>1.20</td>
<td>0.52</td>
</tr>
<tr>
<td>Linear</td>
<td>-2</td>
<td>2.5</td>
<td>1.67</td>
<td>1.12</td>
<td>1.19</td>
<td>0.52</td>
</tr>
<tr>
<td>Linear</td>
<td>-1</td>
<td>2.5</td>
<td>1.79</td>
<td>1.12</td>
<td>1.18</td>
<td>0.53</td>
</tr>
<tr>
<td>Linear</td>
<td>-0.5</td>
<td>2.5</td>
<td>1.94</td>
<td>1.12</td>
<td>1.17</td>
<td>0.53</td>
</tr>
<tr>
<td>Linear</td>
<td>0</td>
<td>2.5</td>
<td>2.50</td>
<td>1.12</td>
<td>1.12</td>
<td>0.55</td>
</tr>
<tr>
<td>Linear</td>
<td>0.5</td>
<td>2.5</td>
<td>3.06</td>
<td>1.12</td>
<td>1.17</td>
<td>0.53</td>
</tr>
<tr>
<td>Linear</td>
<td>1</td>
<td>2.5</td>
<td>3.21</td>
<td>1.12</td>
<td>1.18</td>
<td>0.53</td>
</tr>
<tr>
<td>Linear</td>
<td>2</td>
<td>2.5</td>
<td>3.33</td>
<td>1.12</td>
<td>1.19</td>
<td>0.52</td>
</tr>
<tr>
<td>Linear</td>
<td>4</td>
<td>2.5</td>
<td>3.41</td>
<td>1.12</td>
<td>1.20</td>
<td>0.52</td>
</tr>
<tr>
<td>Linear</td>
<td>8</td>
<td>2.5</td>
<td>3.45</td>
<td>1.12</td>
<td>1.20</td>
<td>0.52</td>
</tr>
<tr>
<td>Linear</td>
<td>16</td>
<td>2.5</td>
<td>3.48</td>
<td>1.12</td>
<td>1.20</td>
<td>0.52</td>
</tr>
<tr>
<td>Linear</td>
<td>32</td>
<td>2.5</td>
<td>3.49</td>
<td>1.12</td>
<td>1.20</td>
<td>0.52</td>
</tr>
<tr>
<td>Exponential</td>
<td>-32</td>
<td>2.5</td>
<td>1.03</td>
<td>1.12</td>
<td>1.48</td>
<td>0.41</td>
</tr>
<tr>
<td>Exponential</td>
<td>-16</td>
<td>2.5</td>
<td>1.07</td>
<td>1.12</td>
<td>1.46</td>
<td>0.42</td>
</tr>
<tr>
<td>Exponential</td>
<td>-8</td>
<td>2.5</td>
<td>1.14</td>
<td>1.12</td>
<td>1.42</td>
<td>0.43</td>
</tr>
<tr>
<td>Exponential</td>
<td>-4</td>
<td>2.5</td>
<td>1.32</td>
<td>1.12</td>
<td>1.33</td>
<td>0.47</td>
</tr>
<tr>
<td>Exponential</td>
<td>-2</td>
<td>2.5</td>
<td>1.73</td>
<td>1.12</td>
<td>1.20</td>
<td>0.52</td>
</tr>
<tr>
<td>Exponential</td>
<td>-1</td>
<td>2.5</td>
<td>2.50</td>
<td>1.12</td>
<td>1.12</td>
<td>0.55</td>
</tr>
<tr>
<td>Exponential</td>
<td>-0.5</td>
<td>2.5</td>
<td>3.27</td>
<td>1.12</td>
<td>1.20</td>
<td>0.52</td>
</tr>
<tr>
<td>Exponential</td>
<td>0</td>
<td>2.5</td>
<td>1.00</td>
<td>1.12</td>
<td>1.50</td>
<td>0.40</td>
</tr>
<tr>
<td>Exponential</td>
<td>0.5</td>
<td>2.5</td>
<td>1.73</td>
<td>1.12</td>
<td>1.20</td>
<td>0.52</td>
</tr>
<tr>
<td>Exponential</td>
<td>1</td>
<td>2.5</td>
<td>2.50</td>
<td>1.12</td>
<td>1.12</td>
<td>0.55</td>
</tr>
<tr>
<td>Exponential</td>
<td>2</td>
<td>2.5</td>
<td>3.37</td>
<td>1.12</td>
<td>1.50</td>
<td>0.52</td>
</tr>
<tr>
<td>Exponential</td>
<td>4</td>
<td>2.5</td>
<td>3.68</td>
<td>1.12</td>
<td>1.33</td>
<td>0.47</td>
</tr>
<tr>
<td>Exponential</td>
<td>8</td>
<td>2.5</td>
<td>3.86</td>
<td>1.12</td>
<td>1.42</td>
<td>0.43</td>
</tr>
<tr>
<td>Exponential</td>
<td>16</td>
<td>2.5</td>
<td>3.93</td>
<td>1.12</td>
<td>1.46</td>
<td>0.42</td>
</tr>
<tr>
<td>Exponential</td>
<td>32</td>
<td>2.5</td>
<td>3.97</td>
<td>1.12</td>
<td>1.48</td>
<td>0.41</td>
</tr>
</tbody>
</table>
Notice that exponential scaling affects the scaled standard deviation much more than linear scaling.

Also notice that for weight factors between 0 and 1, exponential scaling causes earlier periods to be weighted more heavily than recent periods, but with linear scaling later periods are weighted more heavily than earlier periods. To invert the results for both scaling types, use a negative weighting factor. Negative weight factors cause earlier periods in time to be weighted more heavily than recent periods in the exact inverse proportion to the positive weighting factor of the same value.

To turn time-line scaling off, use a Scale Type of 1 (linear) and a Factor of 0, or a Scale Type of 2 (exponential) and a Factor of 1. These settings cause measurements within all periods to be equally weighted.

## Capital Components
The ACI component (Avg % of Capital Invested) and MCI component (Max % of Capital Invested) are useful for optimizing strategies that scale into and/or out of a position, or vary the number of shares invested. If you always use a fixed number of shares per trade and do not pyramid into or out of a position, set the ACI_Wgt and MCI_Wgt values to 0 to disable these components.

However, if your positions can be different sizes, using ACI during optimization enables Optimax to favor input combinations within a target range of average % of capital invested. For example, if you want to discover inputs that favor an average of 80% of your capital invested at all times, you might try specifying ACI_Tgt = 80 and ACI_Rng = 30 with an appropriate weighting, such as 0.25 to target 80% capital invested on average. Or, to favor input combinations that lower your average exposure, you might specify ACI_Tgt = 25 and ACI_Rng = 30 to target 25% capital invested on average.

The MCI component (Max % of Capital Invested) is useful for optimizing strategies that take advantage of margin. For example, setting MCI_Tgt to 200 and the range to 200 causes this function to target an average maximum of 200% of capital invested, and all input combinations that result in an average maximum of over 400% to be rejected as unfit.

When using these components, set the InvestmentCapital parameter equal to the initial investment capital you will use in real life.

**The PrintInfo and SendVarsToOMX Parameters**

Use these two parameters to display the intermediate results computed within GPF for all components and internal subcomponents. This is a very useful aid to assist you in adjusting the parameters to obtain the desired results.

Setting the PrintInfo function to 1 will print detailed information about the fitness calculation to the EL Output window. To display this information within TradeStation, select View, EasyLanguage Output Bar, and click the Print Log tab at the bottom of the window. This information will display during an optimization, or if you copy the call to the GPF function to the end of your strategy then it will display during testing. (Be sure to remove or comment out the copy before beginning an optimization.)

Setting this parameter to 2 will send the same information to a file instead. The file will be located in your root directory and it will be named GPF.txt.

Setting it to 0 will turn this functionality off.

Setting SendVarsToOMX to True will display the computation details within Optimax in the Detailed Performance Report pane.
GPF Tuning
To tune your weighting factors, run an optimization with SendVarsToOMX = True. Click sample equity curves in the Equity Grid and view the Detail Performance report at the bottom of the Optimax window. Examine the fitness values to determine which components are contributing the most to the GPF value, then adjust your weights to further emphasize or de-emphasize the components as desired.

Note that you will have to experiment with the weights in order to determine appropriate values. You will find that raw values can vary quite a bit from component to component, so you will need to use the weights to compensate for differences in raw values as well as to apply relative strength to each.

For example, you may find that one of the Raw values is extremely high compared to the rest, and so you may reduce the weighting on that component to bring it’s contribution to the GPF into line with the rest. Or conversely, you may find that the Raw value for one of the components is very low compared to the rest. In that case, you may need to increase the weighting for that component to compensate for this imbalance.

Examples
The following call computes the GPF based on monthly measurements. It handles up to 129 months in the back test, writes subcomponent information to \GPF.txt and to the performance report, always rejects a negative net profit, considers profit factor of highest importance, considers uniformity and proximity to target to be of equal importance for all subcomponents, and ignores both negative component values and capital components.

```
Arrays:
PeriodReturn[130](0),
PeriodTrades[130](0),
PeriodWins[130](0),
PeriodLosses[130](0),
PeriodPctPft[130](0),
PeriodGrossProfit[130](0),
PeriodGrossLoss[130](0),
PeriodProfitFactor[130](0),
PeriodEvenBars[130](0),
PeriodWinBars[130](0),
PeriodLosBars[130](0),
PeriodTotalTrades[130](0),
PeriodABT[130](0),
PeriodAWL[130](0),
PeriodBTR[130](0),
PeriodACI[130](0),
PeriodMCI[130](0);
OMX_Fitness = OMX_F_GPF(
  OMX_IterationNum, { Optimax Iteration number
  2, { Write details to the Print Log or GPF.txt? }
  true, { send variables to performance report }
  true, { Always reject a negative net profit value? }
  3, { Period Type: 1 = day, 2 = week, 3 = month }
)
100000, { Available Investment capital } 
(--Weight, Target, Range, Balance, TLT, TLW, PosOnly --) { TLT = Timeline weighting type, TLW = Timeline weighting scale factor }
0.10, 30, 30, 0.5, 2, 2, true, { # of trades per period: }
1.00, 80, 80, 0.5, 2, 2, true, { Pct Profitable trades per period (1 to 100%) }
0.10, 1, 10, 0.5, 2, 2, true, { Average bars per trade }
0.50, 2, 4, 0.5, 2, 2, true, { Bars/winning trade:bars/losing trade ratio }
0.00, 80, 80, 0.5, 2, 2, true, { Avg % of capital invested }
0.00, 200, 200, 0.5, 2, 2, true, { Max % of capital invested }
0.25, { Win/Loss ratio (always maximized) }
1.00, { Profit factor (always maximized) }
0.05, true, { NetProfit to MaxDrawdown Ratio (always maximized) }
LastBarOnChart, { When to perform the computation }
{ All Arrays }
PeriodReturn, { Equity Return in each period }
PeriodTrades, { Number of trades in each period }
PeriodWins, { Number of wins in each period }
PeriodLosses, { Number of losses in each period }
PeriodPctPft, { Percent profitable by period }
PeriodGrossProfit, { Gross profit by period }
PeriodGrossLoss, { Gross loss by period }
PeriodProfitFactor, { Profit Factor by period }
PeriodEvenBars, { Total bars in even trades by period }
PeriodWinBars, { Total bars in winning trades by period }
PeriodLosBars, { Total bars in losing trades by period }
PeriodTotalTrades, { Total number of trades by period }
PeriodABT, { Period average bars per trade }
PeriodAWL, { Avg Win/Loss per period }
PeriodBTR, { AB Win/AB Loss ratio per period }
PeriodACI, { Avg % of capital invested per period }
PeriodMCI ) ; { Max % of capital invested per period }
**Bar Validity**
Valid on intraday, daily, weekly and monthly bars. Invalid on tick, volume and point & figure charts.

**Notes**
Do not pass the constant `True` as the last parameter. Doing so will cause the calculation to be computed on every bar and significantly slow down your optimization.

### 2.3.1.1.2 OMX_F_MSRatio

**Summary:**
OMX_F_MSRatio is a fitness function that measures the consistency of returns over time. Call this function in the Epilog to compute a fitness value.

**Prototype:**

```plaintext
OMX_F_MSRatio = OMX_F_MSRatio( PeriodType, PeriodsBack, bDoComputation ) ;
```

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Possible Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PeriodType</td>
<td>NumericSimple</td>
<td>1, 2, 3</td>
<td>The type of period used. 1=daily, 2=weekly, 3=monthly</td>
</tr>
<tr>
<td>PeriodsBack</td>
<td>NumericSimple</td>
<td>1-10,000</td>
<td>The number of periods to use. If less than 2 it uses all periods in the back test.</td>
</tr>
<tr>
<td>bDoComputation</td>
<td>TrueFalseSimple</td>
<td>True/False</td>
<td>When true the computation is performed; usually passed the value LastBarOnChart</td>
</tr>
</tbody>
</table>

**Function Type: Series**
This is a series function - it executes every bar.

**Returns (Double)**
This function returns a floating-point value from 0 to 1, with values approaching 1 indicating consistency and approaching 0 meaning inconsistency in returns.

**Description and Use:**
This is a modified Sharpe Ratio function. Like the original Sharpe Ratio, this is a time-based measure of consistency of equity returns. It will assess the
consistency of your gains over a period of days, weeks or months. Like the original Sharpe, the higher the return value from this function, the more consistent the returns are over the given set of periods. The lower the value, the more inconsistent the returns are.

Use this function when you want your strategy to achieve consistent returns and it does not exhibit large draw downs (for example, due to tight stops.)

**Formula:**

\[ M_R = \frac{\sum_{I=1}^{N} R_I}{N} \]

\[ SD = \left( \frac{\sum_{I=1}^{N} (R_I - M_R)^2}{(N - 1)} \right)^{\frac{1}{2}} \]

\[ \text{Sharpe Ratio} = \frac{M_R - R_{RF}}{SD} \]

where

- \( R_I \): Return for period \( I \)
- \( M_R \): Mean of return set \( R \)
- \( N \): Number of Periods
- \( SD \): Period Standard Deviation
- \( R_{RF} \): Period Risk Free Return

**Examples:**

On the last bar of the chart, compute the MSRatio based on daily returns over all periods in the back test:

\[ \text{OMX Fitness} = \text{OMX}_F\text{MSRatio}(1, 0, \text{LastBarOnChart}) ; \]

On the last bar of the chart, compute the MSRatio based on weekly returns for the last 52 weeks in the back test:

\[ \text{OMX Fitness} = \text{OMX}_F\text{MSRatio}(2, 52, \text{LastBarOnChart}) ; \]

On the last bar of the chart, compute the MSRatio based on monthly returns for all months in the back test:

\[ \text{OMX Fitness} = \text{OMX}_F\text{MSRatio}(3, 0, \text{LastBarOnChart}) ; \]

**Bar Validity:**

Valid on intraday, daily, weekly and monthly bars. Invalid on tick, volume and
point & figure charts.

Notes:
This Sharpe calculation is a variant of the original - it is not the standard Sharpe ratio. Since this number is to be used as a fitness value, the objective of this calculation is only to determine how consistent the returns are over a given period of time. It purposefully leaves out the risk-free return portion of the original Sharpe calculation, as that is much less relevant to the requirement for determining fitness, and is only an approximation and varies from month to month in reality.

Do not pass the constant True as the last parameter. Doing so will cause the calculation to be computed on every bar and significantly slow down your optimization.

2.3.1.1.3 OMX_F_VanTharp

Summary:
OMX_F_VanTharp is a fitness function that returns a measure of the profit-to-risk ratio standardized over one year. Call this function in the Epilog to compute the fitness value.

Prototype:

OMX_Fitness = OMX_F_VanTharp ;

Parameters:
None.

Function Type: Series
This is a series function - it executes every bar.

Returns (Double)
This function returns a floating-point value from negative to positive infinity, with higher values indicating higher quality returns and lower values meaning lower quality returns.

Description and Use:
This is a modified Van Tharp function. Like the original Van Tharp, this is a time-standardized measure of the profitability of returns. It will assess the quality of your gains over a period of days, weeks or months. Like the original Van Tharp, the higher the return value from this function, the higher quality the returns are over a year. The lower the value, the lower the quality.

Use this function when you want to maximize your returns with a trade-off of moderate drawdowns for higher returns.

Formula:
\[ \text{OpportunitiesPerYear} = \text{TotalTrades} \left( \frac{365}{\text{StudyDays}} \right) \]

\[ PW = \frac{\text{TotalWins}}{\text{OpportunitiesPerYear}} \]

\[ PL = \frac{\text{TotalLosses}}{\text{OpportunitiesPerYear}} \]

\[ \text{Expectancy} = \frac{\text{AW} \times PW + \text{AL} \times PL}{|AL|} \]

\[ \text{Quality} = \text{Expectancy} \times \text{OpportunitiesPerYear} \]

where
- \text{AW} Average Win
- \text{PW} Probability of winning
- \text{AL} Average Loss
- \text{PL} Probability of losing

**Example:**

\[ \text{OMX_Fitness} = \text{OMX_F_VanTharp} ; \]

**Bar Validity:**
Valid on intraday, daily, weekly and monthly bars. Invalid on tick, volume and point & figure charts.

**Notes:**
This Van Tharp calculation is a more conservative variant of the original - it is not the standard Van Tharp quality value. This function uses the average losing trade as the standard of risk instead of the minimum losing trade.

### 2.3.2 DLL Function Reference

The Optimax DLL contains a set of functions that are called from TradeStation during and after an optimization. This DLL is written in C and the functions all use the \_stdcall calling convention. All parameters use standard c-type data types, and all character strings are null-terminated.

The functions must be called in an appropriate sequence. Out-of-sequence calls can result in incorrect optimization results and assertion errors. For the correct sequence to use, see [Adding Optimax Calls](#).

**Note:**
When you use these functions, the directory containing the TradeStation program files must be on the path. Optimax tries to find it automatically when it starts up and dynamically adds it to the path. In TradeStation 7 this should work fine. If you try to use these calls with a prior TradeStation version, you may need to
modify you path manually.

2.3.2.1 OMX_BARINFO1

**Prototype:**

```c
int OMX_BARINFO1( 
    float Dt, float Tm, Double OpenEquity, Double OpenValue ) ;
```

**Purpose:**
This function stores the passed values for Optimax to later display as an equity graph.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dt</td>
<td>an EasyLanguage date variable</td>
</tr>
<tr>
<td>Tm</td>
<td>an EasyLanguage time variable</td>
</tr>
<tr>
<td>OpenEquity</td>
<td>current value of the portfolio</td>
</tr>
<tr>
<td>OpenValue</td>
<td>current value of the open position; pass 0 if there is no open position</td>
</tr>
</tbody>
</table>

**Returns:**
This function always returns 0.

**Use:**
Call this function each time you want to store a data point for the equity curve. The value is stored as a "B" record in the G####I####.csv file.

**Notes:**
Storing a data point for every bar on an intra day is not advisable, since this can severely degrade the performance of the optimization in TradeStation and in Optimax, and the full level of detail isn't necessary in Optimax.

2.3.2.2 OMX_ENDPASS1

**Prototype:**

```c
int OMX_ENDPASS1( long IterationNum ) ;
```

**Purpose:**
This function tells Optimax to complete processing for the current individual.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IterationNum</td>
<td>The current iteration number during the optimization</td>
</tr>
</tbody>
</table>

**Returns:**
This function returns 1 if successful. It returns 0 if the call fails. Failure can occur
because:
- the preceding call to OMX_InitPass1 failed (user error) or,
- the IterationNum is different from the one given to the OMX_InitPass1
  function for this pass (user error) or,
- the shared memory map could not be accessed (internal system error)

Use:
Call this function on the last bar of the strategy to complete processing for the
current individual.

Notes:
This function appends the inputs, detail report variables and lineage information
in the G#I#.csv file. It also updates the information in FileInfo.ini.

2.3.2.3 OMX_FIRSTDATE1
Prototype:

long OMX_FIRSTDATE1( void ) ;

Purpose:
This function returns the first date of the optimization period.

Parameters:

| none | this function accepts no parameters |

Returns:
This function returns the first date in the optimization period in EasyLanguage
format.

Use:
None - not currently implemented.

Notes:
This function is scheduled to be used in walk testing, to be implemented in a
coming release. It is not necessary to use this function at this time.

2.3.2.4 OMX_GETPARM1
Prototype:

float OMX_GETPARM1( char *ParmName ) ;

Purpose:
This function gets an input value for an optimization pass.
Parameters:

| ParmName        | the name of an input to be optimized, entered on the optimization options screen in Optimax |

Returns:
This function returns a value to be used as an input to the strategy.

Use:
Call this function on the first bar of the strategy to get the value to be used for an input.

Notes:
The name passed here should match the name of an input entered into the Optimization Settings window. If the name does not match any known input names, 0 is returned.

2.3.2.5 OMX_INITPASS1

Prototype:

```c
int OMX_INITPASS1( 
    Long IterationNum, Long GenerationNum, Long IndividualNum, char *Symbol, float PriceScale, Long BarNum, Long BarType, Long BarInterval, Long BarDate, Long BarTime, Long LastCalcDate, Long LastCalcTime ) ;
```

Purpose:
This function initializes an optimization pass from within the strategy stores the passed values for Optimax to later display as an equity graph.

Parameters:
### IterationNum
A positive integer or 0. This tells Optimax which pass to process. Iteration numbers must be non-repeating sequential integers. When 0, this tells Optimax that an optimization is not in process.

### GenerationNum
The generation number of an individual to be processed. Always pass 0 during an optimization. Passing a non-zero value indicates that this is a setup request and the next calls will be GetParm requests for historic G/I input values.

### IndividualNum
The number of an individual within a generation to be processed. Always pass 0 during an optimization. Passing a non-zero value indicates that this is a setup request and the next calls will be GetParm requests for historic G/I input values.

### Symbol
The name of the current symbol

### PriceScale
The price scale of the current price series

### BarNum
The current bar number

### BarType
The bar type identified by TradeStation using the BarType variable.

### BarInterval
The bar interval identified by TradeStation using the BarInterval variable.

### BarDate
The date on the current bar in EasyLanguage format.

### BarTime
The time on the current bar in EasyLanguage format.

### LastCalcDate
The date on the last bar of the chart in EasyLanguage format.

### LastCalcTime
The time on the last bar of the chart in EasyLanguage format.

### Returns:

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Number of iterations has exceeded max number of permutations - normal end of run</td>
</tr>
<tr>
<td>0</td>
<td>the pass could not be initialized; possibly due to a function call sequence error or inability to open necessary output files.</td>
</tr>
<tr>
<td>1</td>
<td>success</td>
</tr>
</tbody>
</table>

### Use:
Call this function on the first bar of a strategy pass. It performs all initialization functions necessary for the optimization of that pass.

### Notes:
None.
2.3.2.6 OMX_LASTDATE1

Prototype:

long OMX_LASTDATE1( void ) ;

Purpose:
This function returns the last date of the optimization period.

Parameters:

| none | this function accepts no parameters |

Returns:
This function returns the last date in the optimization period in EasyLanguage format.

Use:
None - not currently implemented.

Notes:
This function is scheduled to be used in walk testing, to be implemented in a coming release. It is not necessary to use this function at this time.

2.3.2.7 OMX_LASTTIME1

Prototype:

long OMX_LASTTIME1( void ) ;

Purpose:
This function returns the last time of the optimization period.

Parameters:

| none | this function accepts no parameters |

Returns:
This function returns the last time in the optimization period in EasyLanguage format.

Use:
None - not currently implemented.

Notes:
This function is scheduled to be used in walk testing, to be implemented in a coming release. It is not necessary to use this function at this time.
2.3.2.8 OMX_PRICEMOD1

**Prototype:**

```c
int OMX_PRICEMOD1(
    int bUseFindAddr, Long IterationNum, Long GenerationNum, Long IndividualNum,
    LPLONG lpOpen1, LPLONG lpHigh1, LPLONG lpLow1, LPLONG lpClose1,
    LPLONG lpOpen2, LPLONG lpHigh2, LPLONG lpLow2, LPLONG lpClose2,
    char *Symbol, float PriceScale, Long BarNum, Long BarType, Long BarInterval,
    Long BarDate, Long BarTime, Long LastCalcDate, Long LastCalcTime
);
```

**Purpose:**
This function modulates OHLC values on datastream1 according to the parameters specified in the price modulation options of Optimax.

**Parameters:**
| **bUseFindAddr** | 0 = use the TradeStation FindAddress functions before modifying the prices.  
1 = do not use the FindAddress functions - use the passed addresses directly |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IterationNum</strong></td>
<td>The current iteration number during an optimization, or 0 if not optimizing.</td>
</tr>
<tr>
<td><strong>GenerationNum</strong></td>
<td>The generation number of the historic individual to be processed. Always pass 0 during an optimization. Passing a non-zero value indicates that this is a historic request and previous price values are returned.</td>
</tr>
<tr>
<td><strong>IndividualNum</strong></td>
<td>The individual number within a generation of the historic individual to be processed. Always pass 0 during an optimization. Passing a non-zero value indicates that this is a historic request and previous price values are returned.</td>
</tr>
<tr>
<td><strong>lpOpen1</strong></td>
<td>Address of the Open price on datastream1 for the current bar.</td>
</tr>
<tr>
<td><strong>lpHigh1</strong></td>
<td>Address of the High price on datastream1 for the current bar.</td>
</tr>
<tr>
<td><strong>lpLow1</strong></td>
<td>Address of the Low price on datastream1 for the current bar.</td>
</tr>
<tr>
<td><strong>lpClose1</strong></td>
<td>Address of the Close price on datastream1 for the current bar.</td>
</tr>
<tr>
<td><strong>lpOpen2</strong></td>
<td>Address of the Open price on a secondary datastream for the current bar.</td>
</tr>
<tr>
<td><strong>lpHigh2</strong></td>
<td>Address of the High price on a secondary datastream for the current bar.</td>
</tr>
<tr>
<td><strong>lpLow2</strong></td>
<td>Address of the Low price on a secondary datastream for the current bar.</td>
</tr>
<tr>
<td><strong>lpClose2</strong></td>
<td>Address of the Close price on a secondary datastream for the current bar.</td>
</tr>
<tr>
<td><strong>Symbol</strong></td>
<td>The name of the current symbol</td>
</tr>
<tr>
<td><strong>PriceScale</strong></td>
<td>The price scale of the current price series</td>
</tr>
<tr>
<td><strong>BarNum</strong></td>
<td>The current bar number</td>
</tr>
<tr>
<td><strong>BarType</strong></td>
<td>The bar type identified by TradeStation using the BarType variable.</td>
</tr>
<tr>
<td><strong>BarInterval</strong></td>
<td>The bar interval identified by TradeStation using the BarInterval variable.</td>
</tr>
<tr>
<td><strong>BarDate</strong></td>
<td>The date on the current bar in EasyLanguage format.</td>
</tr>
<tr>
<td><strong>BarTime</strong></td>
<td>The time on the current bar in EasyLanguage format.</td>
</tr>
<tr>
<td><strong>LastCalcDate</strong></td>
<td>The date on the last bar of the chart in EasyLanguage format.</td>
</tr>
<tr>
<td><strong>LastCalcTime</strong></td>
<td>The time on the last bar of the chart in EasyLanguage format.</td>
</tr>
</tbody>
</table>
Returns:
This function always returns 1.

Use:
Call this function on every bar during an optimization. This function accepts unmodulated prices from a second datastream and modulates the prices on datastream1. The second datastream values are used as reference values and are copied to the OHLC values on datastream1 (parameters 5 through 8) before modulation.

Notes:
This function can also be used for the manual price modulation function.

2.3.2.9 OMX_PRICEMODA

Prototype:

```c
int OMX_PRICEMODA(int bUseFindAddr, Long IterationNum, Long GenerationNum, Long IndividualNum, LPLONG lpOpen1, LPLONG lpHigh1, LPLONG lpLow1, LPLONG lpClose1, FLOAT Open2, FLOAT High2, FLOAT Low2, FLOAT Close2, char *Symbol, float PriceScale, Long BarNum, Long BarType, Long BarInterval, Long BarDate, Long BarTime, Long LastCalcDate, Long LastCalcTime ) ;
```

Purpose:
This function modulates OHLC values on datastream1 according to the parameters specified in the price modulation options of Optimax.

Parameters:
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bUseFindAddr</td>
<td>0 = use the TradeStation FindAddress functions before modifying the prices.</td>
</tr>
<tr>
<td></td>
<td>1 = do not use the FindAddress functions - use the passed addresses directly</td>
</tr>
<tr>
<td>IterationNum</td>
<td>The current iteration number during an optimization, or 0 if not optimizing.</td>
</tr>
<tr>
<td>GenerationNum</td>
<td>The generation number of the historic individual to be processed. Always pass 0 during an optimization. Passing a non-zero value indicates that this is a historic request and previous price values are returned.</td>
</tr>
<tr>
<td>IndividualNum</td>
<td>The individual number within a generation of the historic individual to be processed. Always pass 0 during an optimization. Passing a non-zero value indicates that this is a historic request and previous price values are returned.</td>
</tr>
<tr>
<td>lpOpen1</td>
<td>Address of the Open price on datastream1 for the current bar.</td>
</tr>
<tr>
<td>lpHigh1</td>
<td>Address of the High price on datastream1 for the current bar.</td>
</tr>
<tr>
<td>lpLow1</td>
<td>Address of the Low price on datastream1 for the current bar.</td>
</tr>
<tr>
<td>lpClose1</td>
<td>Address of the Close price on datastream1 for the current bar.</td>
</tr>
<tr>
<td>Open2</td>
<td>Value of the Open price on a secondary datastream for the current bar.</td>
</tr>
<tr>
<td>High2</td>
<td>Value of the High price on a secondary datastream for the current bar.</td>
</tr>
<tr>
<td>Low2</td>
<td>Value of the Low price on a secondary datastream for the current bar.</td>
</tr>
<tr>
<td>Close2</td>
<td>Value of the Close price on a secondary datastream for the current bar.</td>
</tr>
<tr>
<td>Symbol</td>
<td>The name of the current symbol</td>
</tr>
<tr>
<td>PriceScale</td>
<td>The price scale of the current price series</td>
</tr>
<tr>
<td>BarNum</td>
<td>The current bar number</td>
</tr>
<tr>
<td>BarType</td>
<td>The bar type identified by TradeStation using the BarType variable.</td>
</tr>
<tr>
<td>BarInterval</td>
<td>The bar interval identified by TradeStation using the BarInterval variable.</td>
</tr>
<tr>
<td>BarDate</td>
<td>The date on the current bar in EasyLanguage format.</td>
</tr>
<tr>
<td>BarTime</td>
<td>The time on the current bar in EasyLanguage format.</td>
</tr>
<tr>
<td>LastCalcDate</td>
<td>The date on the last bar of the chart in EasyLanguage format.</td>
</tr>
<tr>
<td>LastCalcTime</td>
<td>The time on the last bar of the chart in EasyLanguage format.</td>
</tr>
</tbody>
</table>
Returns:
This function always returns 1.

Use:
Call this function on every bar during an optimization. This function accepts unmodulated prices from a second datastream and modulates the prices on datastream1. The second datastream values are used as reference values and are copied to the OHLC values on datastream1 (parameters 5 through 8) before modulation.

Notes:
This function can also be used for the manual price modulation function.

2.3.2.10 OMX_SENDVAR1

Prototype:

```c
int OMX_SENDVAR1( char *VarName, Double VarValue ) ;
```

Purpose:
This function stores the passed values for Optimax to later display as part of the detail equity report.

Parameters:

```markdown
<table>
<thead>
<tr>
<th>VarName</th>
<th>a name to display in the detail report</th>
</tr>
</thead>
<tbody>
<tr>
<td>VarValue</td>
<td>a value to display beside the name in the detail report</td>
</tr>
</tbody>
</table>
```

Returns:
This function always returns 0.

Use:
Call this function during an optimization on the last bar of the chart. This stores a report variable for the detail equity report. You can call it multiple times and store multiple variables. The values are stored on the "R" record in the G#####I#####.csv file.

Notes:
None.

2.3.2.11 OMX_WRITELOG1

Prototype:

```c
int OMX_WRITELOG1( char *StringToWrite ) ;
```

Purpose:
This function writes the given string to the Optimax log for the current optimization run.
Parameters:
StringToWrite  a string to write to the Optimax log

Returns:
This function always returns 1.

Use:
Call this function during an optimization to write any desired information to the Optimax log. The log is stored in the Run History folder under the run number subfolder for the current run. You can call it as many times as you wish during a run.

Notes:
None.
Index

- A -
  adaptation 18
  adding optimax calls 24
  archiving 75
  automatic code converter 32

- B -
  barinfo 28

- C -
  cache 75
  calculator 35
  changeling methods 16, 38, 51
  chromosome 6
  clone check 16
  clone checking 38
  code conventions 32
  code converter 32
  complements 8
    genetic 8
  conventions 32
    code 32
  convergence 17
  Convergence Length 39
  Convergence Std Dev 39
  conversion procedure 33
  converted inputs 24, 26
  converting your strategy 32
  crossover 8, 51
  Crossover Rate 39
  curve fitting 18

- D -
  DC bias 41
  detailed equity 51
  detailed report 22

- E -
  EasyLanguage converter 32
  Elite Percent Taper 39
  elitism 11
  email 3
  embryonic transform 51
  EndPass 28
  enhancing the epilog 29
  epilog 24, 28
    enhancing 29
  equity detail 51
  equity grid 22, 51
  estimates 35
  evaluating results 51, 82
  evolution 6
  exhaustive search 38

- F -
  FAQ 88
  fitness 16, 28
  fitness landscape 20
  folders 75
  futher reading 22

- G -
  gene 6
  generation 16
  genetic complement 51
  Genetic Complement Number 39
  genetic optimization 5
  genetic options 39
  genetic search 38
  genetics 6
  GIF directory 75
  global maximum 20
  granularity 28

- H -
  hamming distance 11, 17
  history 75
incest filtering 11
increment 35
indicators 51
input space 51
ISpace 51
search space 51
SSpace 51
individual 6
Initial Population Size 39
input space canvas 22
input space indicator 51
input space map 22
inputs 24, 35
converted 26
inputs grid 22, 51
ISpace indicator 51
iteration number 48

landscape 20
fitness 20
lineage 51
local maximum 20

managing optimizations 75
manual price modulation 41
mate selection 11
mating 8, 11
mating pool 11
Max Generations 39
maximum 35
maximum life span 11, 39
max 35
Min nHamming Distance 39
minimum 35
Modulation On 41
mutation 16, 38, 51
Mutation Rate 39
MX_StreamNum( 24, 26

noise amplitude 41
noise frequency 41

OMX_BarInfo 28
OMX_BARINFO1 117
OMX_EndPass 28
OMX_Fitness 28
OMX_GetParm 35
OMX_IterationNum 48
OMX_PriceMod 41
OMX_SendVar 29
OMX_StreamNum 24, 26, 48
optimization 5
genetic 5
optimization parameters 34, 80
optimization results 20
optimizations 48, 81
running 48, 81
optimizing 22
oscillator amplitude 41
oscillator frequency 41

parameters 34, 80
performance report 51
preamble 24, 26
preparations 23
preparing the strategy 80
price modulation 18, 41
prolog 26
prologue 24
protecting runs 75

questions 3

random 16, 38
references 22
refresh rate 48
Refresh Timer On 41
restrictions 32, 79
results 51
evaluating 51
rhist grid 75
rival 16, 38, 51
roulette scaling factor 11, 39
roulette selection 11
run 75
protecting 75
unprotecting 75
run history 75
running optimizations 48, 81

- S -
sample 79
search space indicator 51
search space map 22
search types 38
SendVar 29
SSpace indicator 51
strategy 27
strategy conventions 32
stress testing 18
survival 11

- T -
thumbnail cache 75
thumbnails 51, 75
tilde 51
Top Percent 39
tournament group size 11, 39
tournament selection 11

- U -
unprotecting runs 75

- W -
walk-forward testing 18
waveforms 41
Endnotes 2... (after index)
Back Cover