A GENETIC APPROACH TO ECONOMETRIC MODELING

John R. Koza
Computer Science Department
Stanford University
Stanford, CA 94305 USA
Koza@Sunburn.Stanford.Edu
415-941-0336

ABSTRACT

An important problem in economics and other areas of science is finding the mathematical relationship between the empirically observed variables measuring a system. In many conventional modeling techniques, one necessarily begins by selecting the size and shape of the mathematical model. Because the the vast majority of available mathematical tools only handle linear models, this choice is often simply a linear model. After making this choice, one usually then tries to find the values of certain coefficients and constants required by the particular model so as to achieve the best fit between the observed data and the model.

But, in many cases, the most important issue is the size and shape of the mathematical model itself. That is, one really wants first to find the functional form of the model that best fits observed empirical data, and, only then, go on to find any constants and coefficients that happen to be needed. Some techniques exist for doing this.

We suggest that finding the functional form of the model can productively be viewed as being equivalent to searching a space of possible computer programs for the particular individual computer program which produces the desired output for given inputs. That is, one is searching for the computer program whose behavior best fits the observed data. Computer programs offer great flexibility in the ways that they compute their output from the given inputs.

The most fit individual computer program can be found via a new "genetic programming" paradigm originally developed for solving artificial intelligence problems. This new "genetic programming" paradigm genetically breeds populations of computer programs in a Darwinian competition using genetic operations. The Darwinian competition is based on the principle of survival and reproduction of the fittest. The genetic crossover (sexual recombination) operator is designed for genetically mating computer programs so as to create potentially more fit new offspring programs. The best single individual computer program produced by this process after many generations can be viewed as the solution to the problem.

In this paper, we illustrate the process of formulating and solving problems of modeling (i.e. symbolic regression, symbolic function identification) with this new "genetic programming" paradigm using hierarchical genetic algorithms. In particular, the "genetic programming" paradigm is illustrated by rediscovering the well-known multiplicative (non-linear) "exchange equation" M=PQ/V relating the money supply, price level, gross national product, and velocity of money in an economy.

1. INTRODUCTION AND OVERVIEW

An important problem in economics and other areas of science is finding the mathematical relationship between the empirically observed variables measuring a system (Langley and Zytkow 1989). In many conventional modeling techniques, one necessarily begins by selecting the size and shape of the mathematical model. Because the vast majority of available mathematical tools only handle linear models, this choice is often simply a linear model. After making this choice, one usually then tries to find the values of certain coefficients and constants required by the particular model so as to achieve the best fit between the observed data and the model.

But, in many cases, the most important issue is the size and shape of the mathematical model itself. That is, one really wants first to find the functional form of the model that best fits observed empirical data, and, only then, go on to find any constants and coefficients that happen to be needed.

We suggest that finding the functional form of the model can be viewed as being equivalent to searching a space of possible computer programs for the particular individual computer program which produces the desired output for given inputs. That is, one is searching for the computer program whose behavior best fits the observed data. Computer programs offer great flexibility in the ways that they compute their output from the given inputs.

The most fit individual computer program can be found via a new “genetic computing” paradigm originally developed for solving artificial intelligence problems. This new “genetic computing” paradigm genetically breeds populations of computer programs in a Darwinian competition using genetic operations. The Darwinian competition is based on the principle of survival and reproduction of the fittest. The genetic crossover (sexual recombination) operator is designed for genetically mating computer programs so as to create potentially more fit new offspring programs. The best single individual computer program produced by this process after many generations can be viewed as the solution to the problem.

In this paper, we illustrate the process of formulating and solving problems of modeling (which may also be called empirical discovery, symbolic regression, or symbolic function identification) with this new “genetic computing” paradigm using hierarchical genetic algorithms. In particular, the problem of modeling requires finding a function, in symbolic form, that fits given numeric data points representing some observed system. Finding such an empirical model for a system can also be used in forecasting future values of the state variables of the system.

In this paper, we focus on the simple non-linear econometric “exchange equation” PQ=MV relating the price level, gross national product, money supply, and velocity of money in an economy.

We claim that the process of solving the problems of the type described above can be reformulated as a search for a most fit individual computer program in the space of possible computer programs. In our research we use the LISP programming language so in particular, the search space is the hyperspace of LISP “symbolic expressions” (called S-expressions) composed of various terms (called atoms in LISP) along with standard arithmetic operations, standard programming operations, standard mathematical functions, and various functions peculiar to the given problem domain.

For example, the standard arithmetic function of multiplication is relevant when we are attempting to discover the econometric “exchange equation” PQ=MV. In general, the objects that are manipulated in our attempts to build computer programs are of four types. These objects include functions of various number of arguments, such as multiplication mentioned above; variable atoms; constant atoms, such as 0, 1, etc.; and control structures such as If-Then-Else, Do-Until, etc.

In this recently developed new genetic algorithm paradigm, the individuals in the genetic population are compositions of functions (i.e. LISP S-expressions). In particular, the individuals in the population are LISP S-expressions created from compositions of functions and atoms appropriate to the particular problem domain. The set of functions used typically includes arithmetic operations, mathematical functions, conditional logical operations, and functions appropriate to the problem domain at hand. The set of atoms used typically includes various constants and particular inputs appropriate to the problem domain. The search space is the hyperspace of all possible LISP S-expressions that can be recursively composed of the available functions and atoms. The crossover operation appropriate for mating two parents from this hyperspace of LISP S-expressions creates new offspring S-expressions by exchanging sub-trees (i.e. sub-lists) between the two parents. The results of this process are inherently hierarchical.

As will be seen, the LISP S-expression required to solve the problem described above will emerge from a simulated evolutionary process using a new “genetic computing” paradigm using a hierarchical genetic algorithm.

In each case, this simulated evolutionary process will start with an initial population of randomly generated LISP S-expressions composed of functions and atoms appropriate to the problem domain.

The fitness of each individual LISP S-expression in a population at any stage of the process will be measured in a simple, natural, and consistent way. Simply stated, fitness will measure how well the individual performs in the particular problem environment. In particular, fitness will be measured by the sum of the squares of the distances (taken for all the environmental cases) between the point in the solution space created by the S-expression for a given set of arguments and the correct point in the solution space. The closer this sum is to zero, the better the S-expression.

Predictably, the initial random individual S-expressions will have exceedingly poor fitness. Nonetheless, some individuals in the population will be somewhat more fit than others. And, in the valley of the blind, the one-eyed man is king.

Then, a process based on the Darwinian model of reproduction and survival of the fittest and genetic recombination will be used to create a new population of individuals. In particular, a genetic process of sexual reproduction among two parental S-expressions will be used to create offspring S-expressions. At least one of two participating parental S-expressions will be selected in proportion to fitness. The resulting offspring S-expressions will be composed of sub-expressions ("building blocks") from their parents. Finally, the new population of offspring (i.e. the new generation) will replace the old population of parents and the process will continue.

At each stage of this highly parallel, locally controlled, and decentralized process, the state of the process will consist only of the current population of individuals. Moreover, the only input to the algorithmic process will be the observed fitness of the individuals in the current population in grappling with the problem environment.

As will be seen, this process will produce populations which, over a period of generations, tend to exhibit increasing average fitness in dealing with their environment, and which, in addition, can robustly (i.e. rapidly and effectively) adapt to changes in their environment.

The solution produced by this process at any given time can be viewed as the entire population of disjunctive alternatives (typically with improved overall average fitness), or, more commonly, as the single best individual in the population at that time ("winner take all").

The hierarchical character of the computer programs that are produced by the genetic computing paradigm is an essential aspect of the “genetic computing” paradigm. The results of this genetic computing paradigm process are inherently hierarchical. And, in many cases, the results contain default hierarchies which often solve the problem in a relatively parsimonious way.

The dynamic variability of the computer programs that are developed along the way to a solution is also an essential aspect of the “genetic computing” paradigm. In each case, it would be difficult and unnatural to try to specify or restrict the size and shape of the eventual solution in advance. Moreover, the advance specification or restriction of the size and shape of the solution to a problem narrows the window by which the system views the world and might well preclude finding the solution to the problem.

We believe that problems of the type described above can be expeditiously solved only if the flexibility found in computer programs is available. This flexibility includes the ability to perform alternative computations conditioned on the outcome of intermediate calculations, to perform computations on variables of many different types, to perform iterations and recursions to achieve the desired result, and to define and subsequently use computed values and sub-programs.

2. BACKGROUND ON GENETIC ALGORITHMS

Genetic algorithms are highly parallel mathematical algorithms that transform populations of individual mathematical objects (typically fixed-length binary character strings) into new populations using operations patterned after (1) natural genetic operations such as sexual recombination (crossover) and (2) fitness proportionate reproduction (Darwinian survival of the fittest). Genetic algorithms begin with an initial population of individuals (typically randomly generated) and then iteratively (1) evaluate the individuals in the population for fitness with respect to the problem environment and (2) perform genetic operations on various individuals in the population to produce a new population.

John Holland of the University of Michigan presented the pioneering formulation of genetic algorithms for fixed-length character strings in *Adaptation in Natural and Artificial Systems* (Holland 1975). Holland established, among other things, that the genetic algorithm is a mathematically near optimal approach to adaptation in that it maximizes expected overall average payoff when the adaptive process is viewed as a multi-armed slot machine problem requiring an optimal allocation of future trials given currently available information.

In this work, Holland demonstrated that a wide variety of different problems in adaptive systems (including problems from economics, game theory, pattern recognition, optimization, and artificial intelligence) are susceptible to reformation in genetic terms so that they can potentially be solved by the highly parallel mathematical “genetic algorithm” that simulates Darwinian evolutionary processes and naturally occurring genetic operations on chromosomes.

Genetic algorithms differ from most iterative algorithms in that they simultaneously manipulate a population of individual points in the search space rather than a single point in a search space.

Genetic algorithm superficially seem to process only the particular individual binary character strings actually present in the current population. However, Holland's 1975 work focused attention on the fact that they actually also automatically process, in parallel, large amounts of useful information concerning unseen Boolean hyperplanes (called schemata) representing numerous additional similar individuals not actually present in the current population. Thus, genetic algorithms have a property of "intrinsic parallelism" which enable them to create individual strings for the new population in such a way that the hyperplanes representing these unseen similar other individuals are all automatically expected to be represented in proportion to the fitness of the hyperplane relative to the average population fitness. Moreover, this additional computation is accomplished without any explicit computation or memory beyond the population itself. As Schaffer (1987) points out, "Since there are very many more than N hyperplanes represented in a population of N strings, this constitutes the only known example of the combinatorial explosion working to advantage instead of disadvantage."

In addition, Holland established that the seemingly unprepossessing genetic operation of crossover in conjunction with the straight-forward operation of fitness proportionate reproduction causes the unseen hyperplanes (schemata) to grow (and decay) from generation to generation at rates that are mathematically near optimal when the process is viewed as a set of multi-armed slot machine problems requiring an optimal allocation of trials to maximize payoff given the information available at the time.

Holland's l975 work also highlighted the relative unimportance of mutation in the evolutionary process. In this regard, it contrasts sharply with numerous other efforts to solve adaptive systems problem by merely “saving and mutating the best”, such as the 1966 *Artificial Intelligence through Simulated Evolution* (Fogel et. al.) and other work using only asexual mutation (Dawkins 1987).

The current increasing interest in genetic algorithms stems from their intrinsic parallelism, their mathematical near optimality in solving problems, and the availability of increasing powerful computers.

An overview of genetic algorithms can be found in Goldberg's *Genetic Algorithms in Search, Optimization, and Machine Learning* (1989). Recent work in genetic algorithms and genetic classifier systems can be surveyed in Davis (1987) and Schaffer (1989).

Representation is a key issue in genetic algorithm work because genetic algorithms directly manipulate the coded representation of the problem and because the representation scheme can severely limit the window by which the system observes its world. Fixed length character strings present difficulties for some problems — particularly problems in artificial intelligence where the desired solution is hierarchical and where the size and shape of the solution is unknown in advance. The need for more powerful representations has been recognized for some time (De Jong 1985).

The structure of the individual mathematical objects that are manipulated by the genetic algorithm can be more complex than the fixed length character strings. Smith (1980) departed from the early fixed-length character strings by introducing variable length strings, including strings whose elements were if-then rules (rather than single characters). Holland's introduction of the classifier system (Holland 1986) continued the trend towards increasing the complexity of the structures undergoing adaptation. The classifier system is a cognitive architecture into which the genetic algorithm is embedded so as to allow adaptive modification of a population of string-based if-then rules (whose condition and action parts are fixed length binary strings). See also Holland (1990).

Marimon, McGrattan, and Sargent (1990) have applied genetic classifier systems to describe the emergence of a commodity in a simulated trading environment as a medium of exchange among artificially intelligent agents .

3. BACKGROUND ON GENETIC PROGRAMMING PARADIGM

The new "genetic computing" paradigm described in this paper continues the above trend in the field of genetic algorithms towards increasing the complexity of the structures undergoing adaptation. In the new "genetic computing" paradigm, populations of entire computer programs are genetically bred to solve problems.

We have recently shown that entire computer programs can be genetically bred to solve problems in a variety of different areas of artificial intelligence, machine learning, and symbolic processing (Koza 1989, 1990a). In this recently developed "genetic programming" paradigm, the individuals in the population are compositions of functions and terminals appropriate to the particular problem domain. The set of functions used typically includes arithmetic operations, mathematical functions, conditional logical operations, and domain-specific functions. Each function in the function set must be well defined for every element in the range of any other function in the set. The set of terminals used typically includes inputs (sensors) appropriate to the problem domain and various constants. The search space is the hyperspace of all possible compositions of functions that can be recursively composed of the available functions and terminals. The symbolic expressions (S-expressions) of the LISP programming language are an especially convenient way to create and manipulate the compositions of functions and terminals described above. These S-expressions in LISP correspond directly to the "parse tree" that is internally created by most compilers.

This new genetic algorithm paradigm has been successfully applied (Koza 1989, 1990a) to example problems in several different areas, including (1) machine learning of functions (e.g. learning the Boolean 11-multiplexer function, (2) planning (e.g. developing a robotic action sequence that can stack an arbitrary initial configuration of blocks into a specified order), (3) automatic programming (e.g. discovering a computational procedure for solving pairs of linear equations, solving quadratic equations for complex roots, and discovering trigonometric identities), (4) sequence induction (e.g. inducing a recursive computational procedure for the Fibonacci and the Hofstadter sequences), (5) pattern recognition (e.g. translation-invariant recognition of a simple one-dimensional shape in a linear retina), (6) optimal control (e.g. centering a cart and balancing a broom on a moving cart in minimal time by applying a "bang bang" force to the cart) (Koza and Keane 1990a, Koza and Keane 1990b), (7) symbolic "data to function" regression, symbolic "data to function" integration, and symbolic "data to function" differentiation, (8) symbolic solution to functional equations (including differential equations with initial conditions, integral equations, and general functional equations), (9) empirical discovery (e.g. rediscovering Kepler's Third Law), and (10) simultaneous architectural design and training of neural networks.

In this section we describe this new “genetic computing” paradigm using hierarchical genetic algorithms by specifying (1) the nature of the structures that undergo adaptation in this paradigm, (2) the search space of structures, (3) the initial structures, (4) the environment, (5) the fitness function which evaluates the structures in their interactions with the environment, (6) the operations that are performed to modify the structures, (7) the procedure for using the information available at each step of the process to select the operations and structures to be modified, and (8) the method for terminating the algorithm and identifying its output.

3.1. THE STRUCTURES UNDERGOING ADAPTATION

The structures that undergo adaptation in the “genetic computing” paradigm are hierarchically structured computer programs. This is in contrast to the one-dimensional linear strings (whether of fixed or variable length) of characters (or other objects) used in previous work in the genetic algorithm field.

In order to be able to successfully manipulate and modify entire computer programs using operations patterned after genetic operations appearing in nature, we must work in a computer programming language that is unusually flexible. Various “functional programming” languages (e.g. FORTH) might be suitable for accomplishing the work described in this paper. However, the LISP programming language (first developed by John McCarthy in the 1950’s and frequently used in artificial intelligence and symbolic processing applications) is especially well-suited to our needs here. LISP is especially suitable for complex compositions of functions of various types, handling hierarchies, recursion, logical functions, self-modifying computer programs, self-executing computer programs, iterations, and structures whose size and shape is dynamically determined (rather than predetermined in advance). The LISP programming language is especially appropriate when the structures to be manipulated are hierarchical structures. Moreover, <u>both</u> programs and data have the <u>same form</u> in LISP. In particular, the Common LISP dialect of the LISP programming language is especially suitable and convenient for our purposes here.

Thus, the structures that undergo adaptation in “genetic computing” are Common LISP computer programs. LISP computer programs are called “symbolic expressions” (that is, S-expressions). As previously mentioned, both programs and data have the same form in LISP so that it is easy to first modify a computer program and then execute it.

In the LISP programming language, everything is expressed in terms of “functions” operating on some arguments. In LISP S-expressions, the function appears just inside an opening (left) parenthesis and is then followed by its arguments and a closing (right) parenthesis. Thus, for example, (+ 1 2) calls for the function of addition (+) to be applied to the arguments 1 and 2. In other words, the LISP S-expression (+ 1 2) is equivalent to “1+2” in ordinary mathematics and evaluates to 3. In LISP, any argument can itself be an S-expression. For example, (+ 1 (* 2 3)) calls for the addition function to be applied to the argument 1 and the argument (* 2 3). That is, the addition function is to be applied to 1 and the result of applying the multiplication function (*) to the arguments 2 and 3. The result is 7.

The LISP S-expression (+ 1 (IF (> TIME 10) 2 3)) demonstrates the freedom which LISP allows in combining functions of different types (in contrast to most programming languages). In this S-expression, the “function” of “greater than” (>) is applied to the argument TIME and the argument 10. TIME is a variable atom and 10 is a constant atom. The result is either T (True) or NIL depending on what time it is. The resulting value (T or NIL) is the first argument of the conditional “function” IF. The function IF evaluates to either its second argument (2) or its third argument (3) according to whether its first argument is T or NIL, respectively. That is, the IF function evaluates to 2 or 3 depending on whether TIME is greater than 10. The addition function (+) is then applied to 1 and the result of the IF function. Thus, the entire S-expression thus evaluates to either 3 or 4 depending on whether TIME is greater than 10. Note that the functions in this S-expression were +, IF, and >. The LISP programming language has “functions” which perform all of the operations found in other programming languages.

The set of possible S-expressions for a particular domain of interest depends on the functions and atoms that are available in the domain. The possible S-expressions are those that can be composed recursively from the available set of n functions $F = \{f_1, f_2, \ldots, f_n\}$ and the available set of m atoms $A = \{a_1, a_2, \ldots, a_m\}$. Each particular function f in F takes a specified number z(f) of arguments $b_1, b_2, \ldots, b_{z(f)}$.

As a specific example, consider the well-known econometric “exchange equation” PQ=MV, which relates the money supply M, price level P, gross national product Q, and the velocity of money V of an economy.

In particular, suppose we are given the 120 quarterly values (from 1959:1 to 1988:4) of four econometric time series. The first time series is “GNP82” (i.e. the annual rate for the United States gross national product in billions of 1982 dollars). The second time series is “GD” (i.e.the gross national product deflator normalized to 1.00 for 1982). The third series is “FYGM3” (i.e. the monthly interest rate yields of 3-month Treasury bills, averaged for each quarter). The fourth series is “M2” (i.e. the monthly values of the seasonally adjusted money stock M2 in billions of dollars, averaged for each quarter).

In attempting to rediscover the “exchange equation” using hierarchical genetic algorithms, we might use the function set F = {+, -, *, %, RLOG, EXP, GNP82, GD, FM2, FYGM3} and an empty atom set A.

Note that the modified division operation % produces a result of zero if division by zero is attempted. Note that the restricted logarithm function RLOG is the logarithm of the absolute value and is equal to zero for an argument of zero. This definition allows arbitrary compositions of the functions in the function set.

Note that GNP82, GD, FM2, and FYGM3 are functions of time. Time runs in quarters from 1959:1 to 1988:4.

The actual long-term historic postwar value of the M2 velocity of money in the United States is relatively constant and is approximately 1.6527 (Hallman et. al. 1989, Humphrey 1989). Thus, a "correct" solution for the price level P is terms of M, V, and Q is the multiplicative (non-linear) relationship

$$P = \frac{MV}{Q}$$

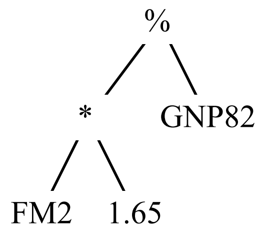
or, alternately,

$$GD(T) = \frac{(M2(T) * 1.6527)}{GNP82(T)}$$

Thus, in LISP, one correct LISP S-expression for prices in terms of the “exchange equation” would be

```
(% (* FM2 1.6527) GNP82).
```

Any LISP S-expression can be depicted graphically as a rooted point-labeled tree in a plane whose internal points are labeled with functions, whose external points (leaves) are labeled with atoms, and whose root is labeled with the function appearing just inside the outermost left parenthesis. The tree corresponding to the LISP S-expression above for the “exchange equation” is shown below:



In this graphical depiction, the 2 internal points of the tree are labeled with functions (% and *). The 3 external points (leaves) of the tree are labeled with atoms. The root of the tree is labeled with the function appearing just inside the outermost left parenthesis of the LISP S-expression (i.e.division %). Note that two lines emanate from the multiplication function * and the division function %. because they each take two arguments. Note also that no lines emanate from the atoms at the external points (leaves) of the tree.

3.2. THE SEARCH SPACE

The search space for hierarchical genetic algorithms is the hyperspace of valid LISP S-expressions that can be recursively created by compositions of the available functions and available atoms for the problem. This search space can, equivalently, be viewed as the hyperspace of rooted point-labeled trees in the plane having internal points labeled with the available functions and external points (leaves) labeled with the available atoms.

3.3. THE INITIAL RANDOMLY GENERATED STRUCTURES

The process of generating the initial random population begins by selecting one of the functions from the set F at random to be the root of the tree. Whenever a point is labeled with a function (that takes k arguments), then k lines are created to radiate out from the point. Then, for each line so created, an element is selected at random from the entire combined set C to be the label for the endpoint of that line. If an atom is chosen to be the label for any point, the process is then complete for that portion of the tree. If a function is chosen to be the label for any such point, the process continues. The probability distribution over the atoms and functions in the combined set C and the number of arguments required for each function determines an average size for the trees generated by this process.

Examples of randomly generated individuals that appeared in the initial generation (generation 0) for the “exchange equation” problem are (RLOG GNP82), (+ FYGM3 (EXP -0.92)), (RLOG (+ 0.27 (EXP 0.65))), etc.

3.4. THE ENVIRONMENT

The environment is a set of cases which provides a basis for evaluating particular S-expressions. For example, for the “exchange equation”, the environment is set of 120 cases listing, for each quarter between 1959:1 and 1988:4, the values of GNP82, FM2, and FYGM3 along with the associated value of GD.

3.5. THE FITNESS FUNCTION

Each individual in a population is assigned a fitness value as a result of its interaction with the environment. Fitness is the driving force of Darwinian natural selection and genetic algorithms.

The "raw fitness" of any LISP S-expression is the sum of the squares of the distances (taken over all the environmental cases) between the point in the solution space (which is real-valued here) returned by the individual S-expression for a given set of arguments and the correct point in the solution space. In particular, the raw fitness r(h,t) of an individual LISP S-expression h in the population of size M at any generational time step t is

$$r(i,t) = \sum_{j=1}^{N_e} |S(i,j) - C(j)|^2$$

where V(h,j) is the value returned by S-expression h for environmental case j (of Ne environmental cases) and where S(j) is the correct value for environmental case j.

The closer this sum of distances is to zero, the better the S-expression.

Thus, the raw fitness of an individual LISP S-expression for the "exchange equation" problem is computed by accumulating, over each of the 120 values of time T from 1959:1 to 1988:4, the sum of the squares of the differences between the actual value of GD and whatever value the individual LISP S-expression produces for that time.

Each raw fitness value is then adjusted (scaled) to produce an adjusted fitness measure a(h,t). The "adjusted fitness" value is

$$a(i,t) = \frac{1}{(1+r(i,t))},$$

where r(h,t) is the raw fitness for individual h at time t. Unlike raw fitness, the adjusted fitness is larger for better individuals in the population. Moreover, the adjusted fitness lies between 0 and 1.

Each such adjusted fitness value a(h,t) is then normalized. The "normalized fitness" value n(h,t) is

$$n(i,t) = \frac{a(i,t)}{\sum_{k=1}^{M} a(k,t)}$$

The normalized fitness not only ranges between 0 and 1 and is larger for better individuals in the population, but the sum of the normalized fitness values is 1. Thus, normalized fitness is a probability value.

3.6. THE GENETIC OPERATIONS

The two primary genetic operations for modifying the structures undergoing adaptation are Darwinian fitness proportionate reproduction and crossover (recombination). They are described below.

3.6.1. THE FITNESS PROPORTIONATE REPRODUCTION OPERATION

The operation of fitness proportionate reproduction for hierarchical genetic algorithms is the basic engine of Darwinian reproduction and survival of the fittest. It is an asexual operation in that it operates on only one parental S-expression. The result of this operation is one offspring S-expression. In this operation, if $s_i(t)$ is an individual in the population at generation t with fitness value $f(s_i(t))$, it will be copied into the next generation with probability

$$\frac{f(s_i(t))}{\sum_{j=1}^{M} f(s_j(t))}$$

Note that the operation of fitness proportionate reproduction does not create anything new in the population. It increases or decreases the number of occurrences of individuals already in the population. To the extent that it increases the number of occurrences of more fit individuals and decreases the number of occurrences of less fit individuals, it improves the average fitness of the population (at the expense of the genetic diversity of the population).

3.6.2. THE CROSSOVER (RECOMBINATION) OPERATION

The crossover (recombination) operation for hierarchical genetic algorithms is a sexual operation that starts with two parental S-expressions. Typically the first parent is chosen from the population with a probability equal to its normalized fitness and the second parent is chosen from the population using a random probability distribution. The result of the crossover operation is two offspring S-expressions. Unlike fitness proportionate reproduction, the crossover operation creates new individuals in the populations.

Every LISP S-expression can be depicted graphically as a rooted point-labeled tree in a plane whose internal points are labeled with functions, whose external points (leaves) are labeled with atoms, and whose root is labeled with the function appearing just inside the outermost left parenthesis. The operation begins by randomly and independently selecting one point in each parent using a specified probability distribution (discussed below). Note that the number of points in the two parents typically are not equal. As will be seen, the crossover operation is well-defined for any two S-expressions. That is, for any two S-expressions and any two crossover points, the resulting offspring are always valid LISP S-expressions. Offspring contain some traits from each parent.

The "crossover fragment" for a particular parent is the rooted sub-tree whose root is the crossover point for that parent and where the sub-tree consists of the entire sub-tree lying below the crossover point (i.e. more distant from the root of the original tree). Viewed in terms of lists in LISP, the crossover fragment is the sub-list starting at the crossover point.

The first offspring is produced by deleting the crossover fragment of the first parent from the first parent and then impregnating the crossover fragment of the second parent at the crossover point of the first parent. In producing this first offspring the first parent acts as the base parent (the female parent) and the second parent acts as the impregnating parent (the male parent). The second offspring is produced in a symmetric manner.

Because entire sub-trees are swapped, this genetic crossover (recombination) operation produces syntactically and semantically valid LISP S-expressions as offspring regardless of which point is selected in either parent.

For example, consider the parental LISP S-expression:

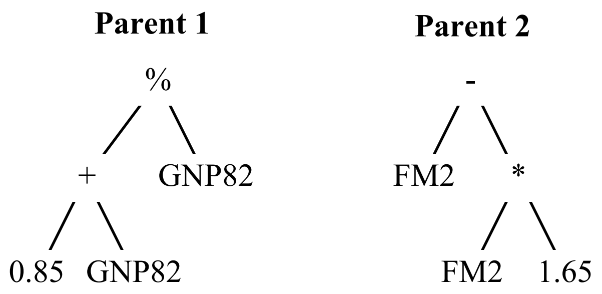
```
        (% (+ 0.85 GNP82) GNP82)
```

The "%" function above is the division function defined so that division by zero delivers zero as its result. Now, consider the second parental S-expression below:

```
        (- FM2 (* FM2 1.65))
```

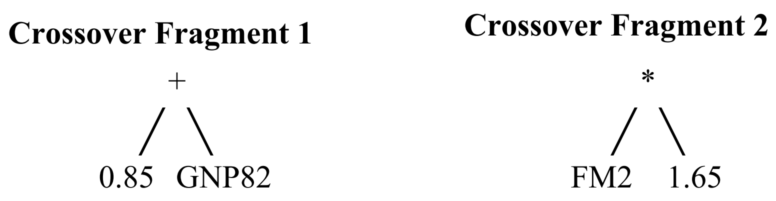
These two LISP S-expressions can be depicted graphically as rooted, point-labeled trees with ordered branches.

The two parental LISP S-expressions are shown below:

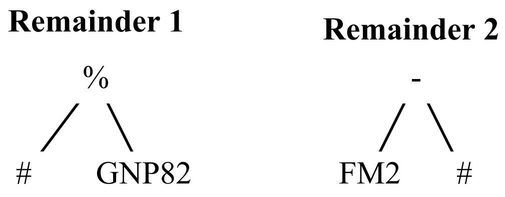


Assume that the points of both trees are numbered in a depth-first way starting at the left. Suppose that the second point (out of 6 points of the first parent) is randomly selected as the crossover point for the first parent and that the third point (out of 6 points of the second parent) is randomly selected as the crossover point of the second parent. The crossover points are therefore the "+" in the first parent and the "*" in the second parent.

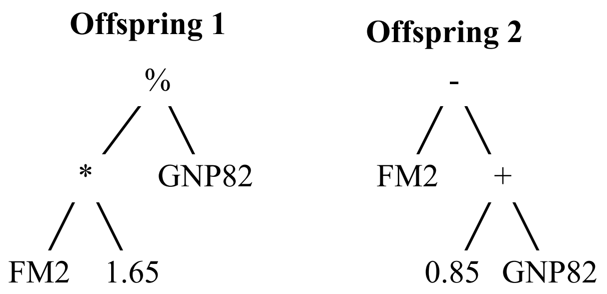
The two crossover fragments are two sub-trees shown below:



The remainders are shown below:



These two crossover fragments correspond to the bold, underlined sub-expressions (sub-lists) in the two parental LISP S-expressions shown above. The two offspring resulting from crossover are shown below.



Note that the first offspring above is a perfect solution for the exchange equation, namely

```
        (% (* FM2 1.65) GNP82).
```

Details of this crossover process can be found in Koza (1990a).

3.7. THE METHOD FOR SELECTING OPERATIONS

The algorithm is controlled by various parameters, including three major parameters, namely, the population size, the number of individuals in the population undergoing fitness proportionate reproduction, and the number of individuals in the population undergoing crossover.

In this paper, the population size was 500. Crossover was performed on 90% of the population. That is, if the population size is 500, then 175 pairs of individuals from each generation were selected (with reselection allowed) from the population with a probability equal to their normalized adjusted fitness. In addition, fitness proportionate reproduction was performed on 10% of the population on each generation. That is, 50 individuals from each generation were selected (with reselection allowed) from the population with a probability equal to their normalized adjusted fitness. Note that the parents remain in the population and can often repeatedly participate in other operations during the current generation. That is, the selection of parents is done with replacement (i.e. reselection) allowed.

Several minor parameters are used to control the computer implementation of the algorithm. In this paper, a maximum depth of 15 was established for S-expressions. This limit prevented large amounts of computer time being expended on a few extremely large (and usually highly unfit) individual S-expressions. Of course, if we could execute all the individual LISP S-expressions in parallel (as nature does) in a manner such that the infeasibility of one individual in the population does not disproportionately jeopardize the resources needed by the population as a whole, we would not need this kind of limit. Thus, if a crossover between two parents would create an individual whose depth exceeded this limit, the crossover operation is simply aborted. In effect, the contemplated crossover operation is replaced with fitness proportionate reproduction for the two parents. The choice of a number such as 15 appears to be liberal in practice inasmuch as running counts monitoring such aborted crossovers indicate that this situation is only rarely encountered. Similarly, a maximum depth of 4 was established for the random individuals generated for generation 0.

3.8. IDENTIFYING THE RESULTS AND TERMINATING THE ALGORITHM

The solution produced by this process at any given time can be viewed as the entire population of disjunctive alternatives (presumably with improved overall average fitness) or, more commonly, as the single best individual in the population at that time ("winner takes all"). The process can be terminated when either a specified total number of generations have been run or when some performance criterion is satisfied. For example, if a solution can be recognized if it is discovered, the algorithm can be terminated at that time and the single best individual can be considered as the output of the algorithm.

4. THE GENERAL CONCEPT OF "DATA TO FUNCTION" SYMBOLIC REGRESSION

In ordinary linear regression, one is given a set of values of various independent variable(s) and the corresponding values for the dependent variable(s). The goal is to discover a set of numerical coefficients for a linear combination of the independent variable(s) which minimizes some measure of error (such as the sum of the squares) between the given values and computed values of the dependent variable(s). Similarly, in quadratic regression, the goal is to discover a set of numerical coefficients for a quadratic expression which similarly minimizes error. In Fourier “regression”, the goal is to discover a set of numerical coefficients for Sine and Cosine functions of various periodicities which similarly minimizes error.

Of course, it is left to the researcher to decide whether to do a linear regression, quadratic regression, or a higher order polynomial regression or whether to try to fit the data points to some non-polynomial family of functions (e.g. sines and cosines of various periodicities, etc.). But, often, <u>the</u> issue is deciding what type of function most appropriately fits the data, not merely computing the numerical coefficients after the type of function for the model has already been chosen. In other words, the problem is <u>both</u> the discovery of the correct functional form that fits the data and the discovery of the appropriate numeric coefficients.

We call such problems "data to function" "symbolic regression because we seek to find the functional form and coefficients that fits a given sample of data."

4.1. SIMPLE EXAMPLE OF SYMBOLIC REGRESSION

Suppose we are given a sampling of the numerical values from an unknown curve over 20 points in some domain, such as the interval [-1, +1]. That is, we are given 20 pairs of points (x_i, y_i). These points might include pairs such as (-0.4, -0.2784), (0.5, 0.9375). (0.25, 0.33203), etc. The goal is to find a function, in symbolic form, that is a perfect fit or good fit to the 20 pairs of numerical data points. (The unknown curve happens to be $x^4+x^3+x^2+x$ for this example).

The solution to this problem of finding a function in symbolic form that fits a given sample of data can be viewed as a search for a function from a hyperspace of functions that can be composed from a set of candidate functions and arguments. The set of terminals for this particular problem consists of just the independent variable X. That is T = {X}. The set of available functions might include addition (+), subtraction (-), multiplication (*), the restricted division function (%), the sine function SIN, the cosine function COS, the exponential function EXP, and the restricted logarithm function RLOG. The restricted division function returns zero when division by zero is attempted and is used so that any composition of functions from the function set and terminals from the terminal set always produces a valid result. The restricted logarithm function RLOG returns 0 for an argument of 0 and otherwise returns the logarithm of the absolute value of the argument. These two functions are restricted in this manner so that they always return a valid floating point number. The function set is therefore F = {+, -, *, %, SIN, COS, EXP, RLOG}.

In one run with a population of 300, the worst single individual S-expression in the initial random population (generation 0) was

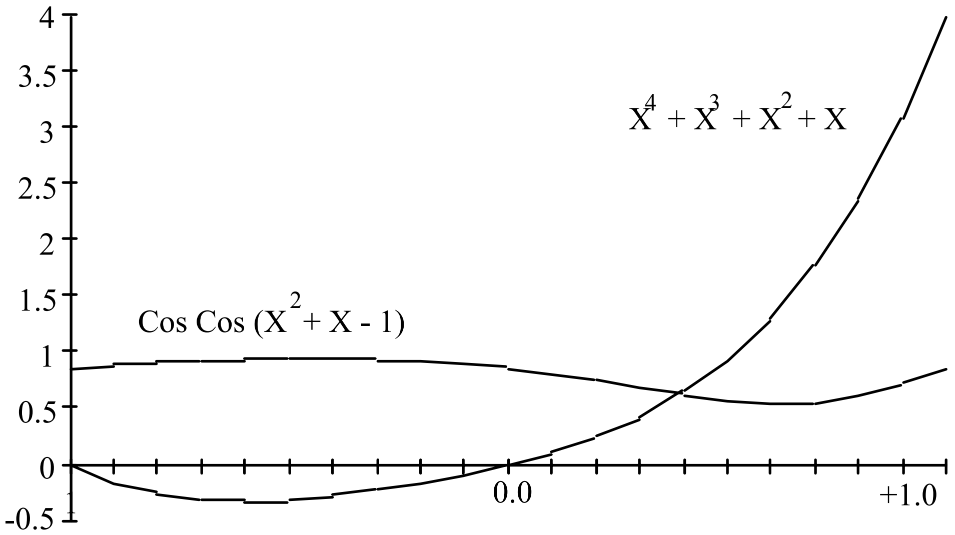
```
        (EXP (- (% X (- X (RSIN X))) (RLOG (RLOG (* X X))))).
```

The sum of the absolute values of the differences between this worst single S-expression and the 20 data points was approximately 10^{23}.

The median (150th best) individual in the initial random population was

```
        (COS (COS (+ (- (* X X) (% X X)) X))).
```

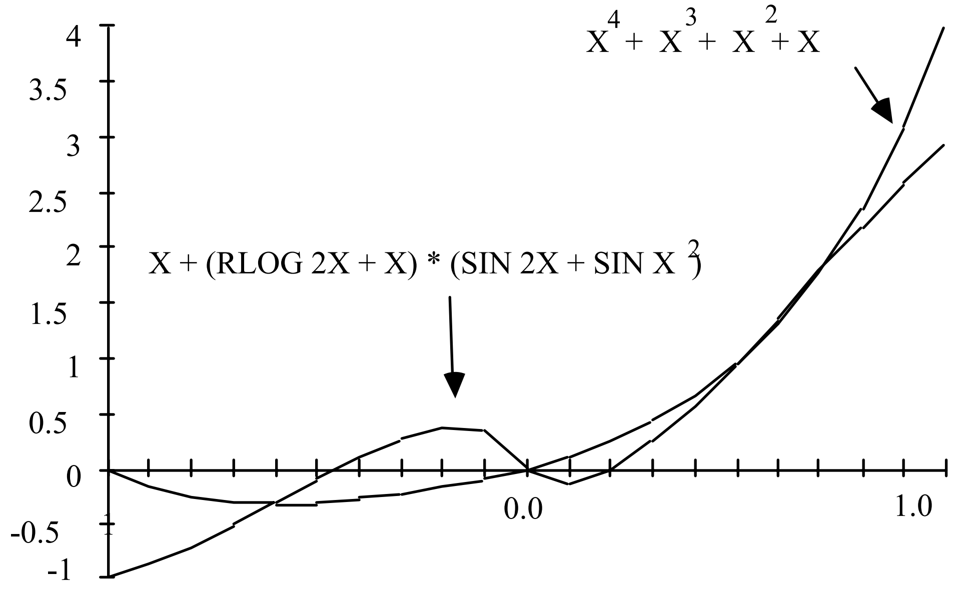
This individual is equivalent to Cos [Cos (x2 + x -1)] and has a raw fitness of this was 23.67. That is, the average distance between the curve for this individual function and the curve for $x^4+x^3+x^2+x$ for the 20 points is about 1.2.



The second best individual in the initial random population was

```
      x + [RLog 2x + x] * [Sin 2x + Sin x²]
```

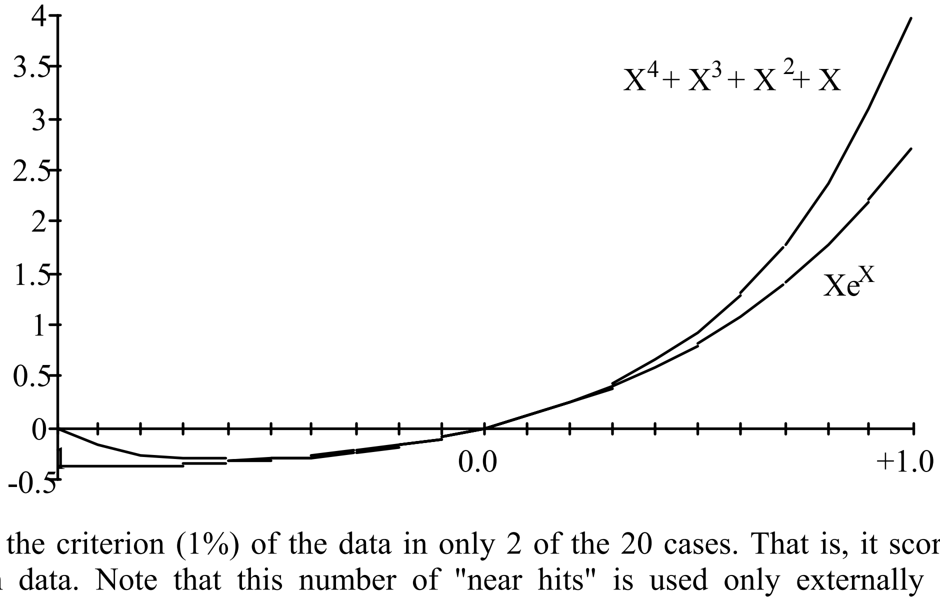
This individual has a raw fitness of 6.05. That is, the average distance between the curve for this function and the curve for $x^4+x^3+x^2+x$ for the 20 points is about 0.3.



The best single individual in the population at generation 0 was the S-expression

```
      (* X (+ (+ (- (% X X) (% X X)) (SIN (- X X))) (RLOG (EXP (EXP X))))).
```

This S-expression is equivalent to xe^x. The sum of the absolute value of the differences between this S-expression and the 20 data points was 4.47 (i.e. an average difference of about 0.23 per data point).



It came within the criterion (1%) of the data in only 2 of the 20 cases. That is, it scored 2 "near hits" with the given data. Note that this number of "near hits" is used only externally for purposes of monitoring and describing the process; it is not used by the genetic algorithm.

Although xe^x is not a good fit, much less a perfect fit, to $x^4+x^3+x^2+x$, it is nonetheless better than the worst individual, better than the median individual, and, in fact, better than all the other 299 randomly created S-expressions. When graphed, it bears some similarity to the target S-expression. In addition, it has some of the gross characteristics of the target function. For example, both xe^x and $x^4+x^3+x^2+x$ are zero when X is zero. Also, when X approaches +1.0, xe^X approaches 2.718 while $x^4+x^3+x^2+x$ approaches 4.

By generation 2, the best single individual improved to

```
(+ (* (* (+ X (* X (* X (% (% X X) (+ X X))))
         (+ X (* X X))
       X)
    X)
```

which is equivalent to $x^4 + 1.5x^3 + 0.5x^2 + x$. The raw fitness of this best individual improved to 2.57. It scored 5 "near hits." This S-expression bears greater similarity to the target S-expression than its predecessors. It is, for example, a polynomial of the correct order (i.e. 4). Moreover, the coefficients of two of the four terms are correct and the incorrect coefficients (1.5 and 0.5) are not too different from the correct coefficients (1.0 and 1.0).

Notice that even though no numerical coefficients were explicitly provided in the terminal set, the fractional coefficient 0.5 was created by the process by first creating $\frac{1}{2}x$ (by dividing $\frac{x}{x}$ by x+x). The coefficient 1.5 was similarly created.

By generation 4, the raw fitness of the best single individual in the population attained the perfect value of 0.0. This individual also scored 20 "near hits." This individual was

```
(+ X (* (+ X (* X (+ X (* X X)))) X)).
```

This S-expression is equivalent to $x^4+x^3+x^2+x$. This is the desired solution.

4.2. SYMBOLIC REGRESSION WITH CONSTANT CREATION

Discovery of the appropriate numeric coefficients is a problem that must be addressed in order to successfully do symbolic regression in general. In the foregoing simple example of symbolic regression, the terminal set consisted only of the independent variable X. There was no explicit facility for "constant creation," although the constant 1 was created by the algorithm by dividing X by X. We now address the issue of "constant creation."

The problem of "constant creation" can be solved by extending the terminal set by one ephemeral element (called “R”) during the generation of the initial random population. Thus, the terminal set would be enlarged for this particular problem and become T = {X, R}. Whenever the ephemeral atom "R" is chosen for any point of the tree during the generation of the initial random population, a random number in a specified range is generated and attached to the tree at that point. In a real-valued problem such as this problem, the random constants were floating point numbers between -1.0 and +1.0. Note that this random generation is done anew for each point where the ephemeral “R” atom is encountered so that the initial random population contains a variety of different random constants. Once generated and inserted into the S-expressions of the population during the initial random generation, these constants remain constant.

After the initial random generation, the numerous different random constants arising from the ephemeral “R” atoms will then be moved around from tree to tree by the crossover operation. These random constants will become embedded in various sub-trees that then carry out various arithmetic operations on them. This "moving around" of the random constants is not at all haphazard, but, instead, is driven by the overall goal of achieving ever better levels of fitness. For example, a symbolic expression that is a reasonably good fit to a target function may become a better fit if a particular constant is, for example, decreased slightly. A slight decrease can be achieved in several different ways. For example, there may be a multiplication by 0.90, a division by 1.10, a subtraction of 0.8, or an addition of -0.004. Notice that if a decrease of precisely 0.09 in a particular value would produce a perfect fit, a decrease of 0.08 may be more fit than a decrease of only 0.07. Thus, the relentless pressure of the fitness function in the natural selection process determines both the direction and magnitude of the numerical adjustments.

The process of combining random constants to achieve a desired goal is not always so direct as the simple example above where a decrease of 0.9 was desired. In one particular problem $\pi/2$ (about 1.57) was needed to solve the problem. In that problem, $\pi/2$ was found by first finding $2 - \pi/2$ (about 0.429). The value $2-\pi/2$ was found by a succession of decreasing approximations in 11 steps. Starting with the available constant 1 and the available function SIN, (SIN 1) was first computed (0.841). Then the SIN of 0.841 was taken to obtain a still smaller number, namely 0.746. This result was then squared to obtain a still smaller number, namely 0.556. Then the SIN function was successively applied six more times to obtain a succession of still smaller numbers. The last one was 0.433. That is, the composition `(SIN (SIN (SIN (SIN (SIN (SIN (* (SIN (SIN 1)) (SIN (SIN 1)))))))))))` was used to compute the constant 0.433. Each successive step in this 11 step process produced a constant closer to what was needed. Each successive S-expression had slightly improved fitness relative to its predecessor.

To illustrate symbolic regression with constant creation, suppose we are given a sampling of the numerical values from the unknown curve $2.718x^2+3.1416x$ over 20 points in some domain, such as the interval

[-1, +1]. In one run, the best individual S-expression in the population in generation 41 was

```
(+ (- (+ (* -.50677 X) (+ (* -.50677 X) (* -.76526 X))))
   (* (+ .11737) (+ (- X (* -.76526 X)) X))).
```

This S-expression is equivalent to $2.76X^2 + 3.15X$.

Symbolic regression has been similarly successfully performed on a large number of target expressions, including expressions such as $\sin X + \cos X + X^2 + X$.

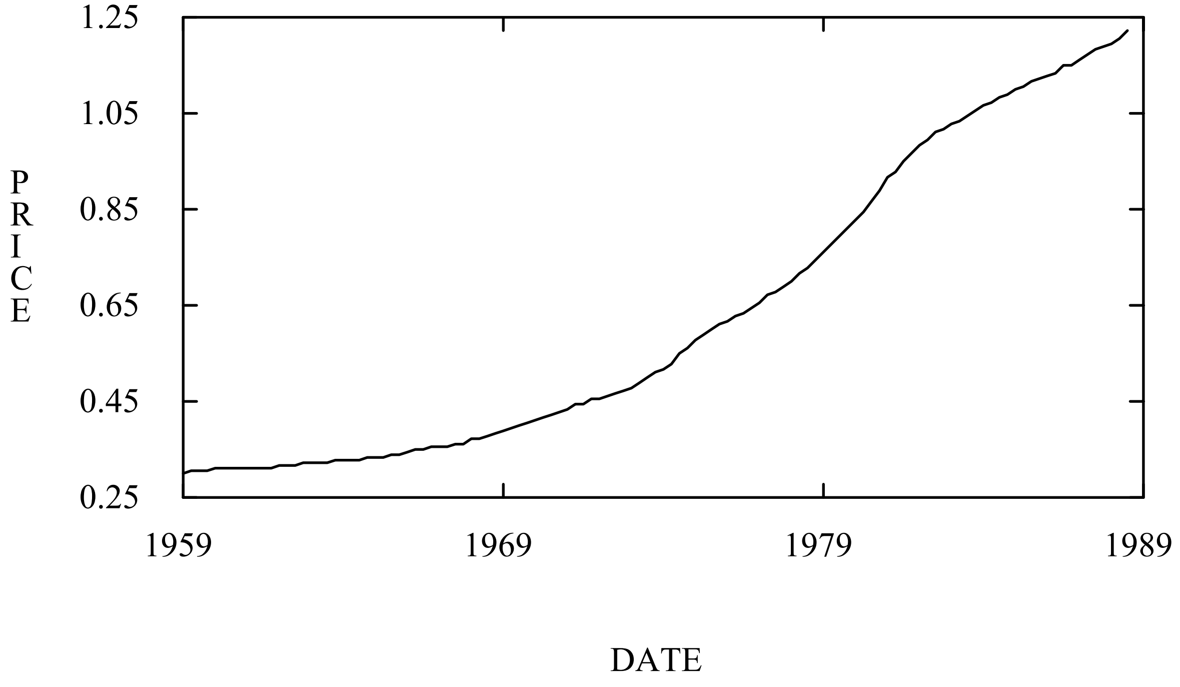
5. REDISCOVERING THE "EXCHANGE EQUATION" FROM EMPIRICAL TIME SERIES DATA

An important problem area in virtually every area of science is finding the empirical relationship underlying observed values of the variables measuring a system (Langley and Zytkow 1989). In practice, the observed data may be noisy and there may be no known way to express the relationships involved in a precise way.

The problem of discovering such empirical relationships from actual observed data is illustrated by the well-known econometric “exchange equation” PQ=MV, which relates the price level P, money supply M, the velocity of money V, and the gross national product Q of an economy. Suppose that our goal is to find the relationship between quarterly values of the price level P and the three other elements of the equation. That is, our goal is to rediscover that P=MV/Q from the actual observed time series data.

In particular, suppose we are given the 120 quarterly values (from 1959:1 to 1988:4) of four econometric time series. The first time series is the annual rate for the United States Gross National Product in billions of 1982 dollars. This series is conventionally called “GNP82." The second time series is the Gross National Product Deflator (normalized to 1.0) for 1982 (called “GD”). The third series is the monthly interest rate yields of 3-month Treasury bills, averaged for each quarter (called “FYGM3”). The fourth series is the monthly values of the seasonally adjusted money stock M2 in billions of dollars, averaged for each quarter (called “M2”). The time series used here were obtained from the CITIBASE data base of machine-readable econometric time series (Citibank 1989). The CITIBASE™ data was accessed by an Apple Macintosh II™ computer using software provided by VAR Econometrics Inc. (Doan 1989).

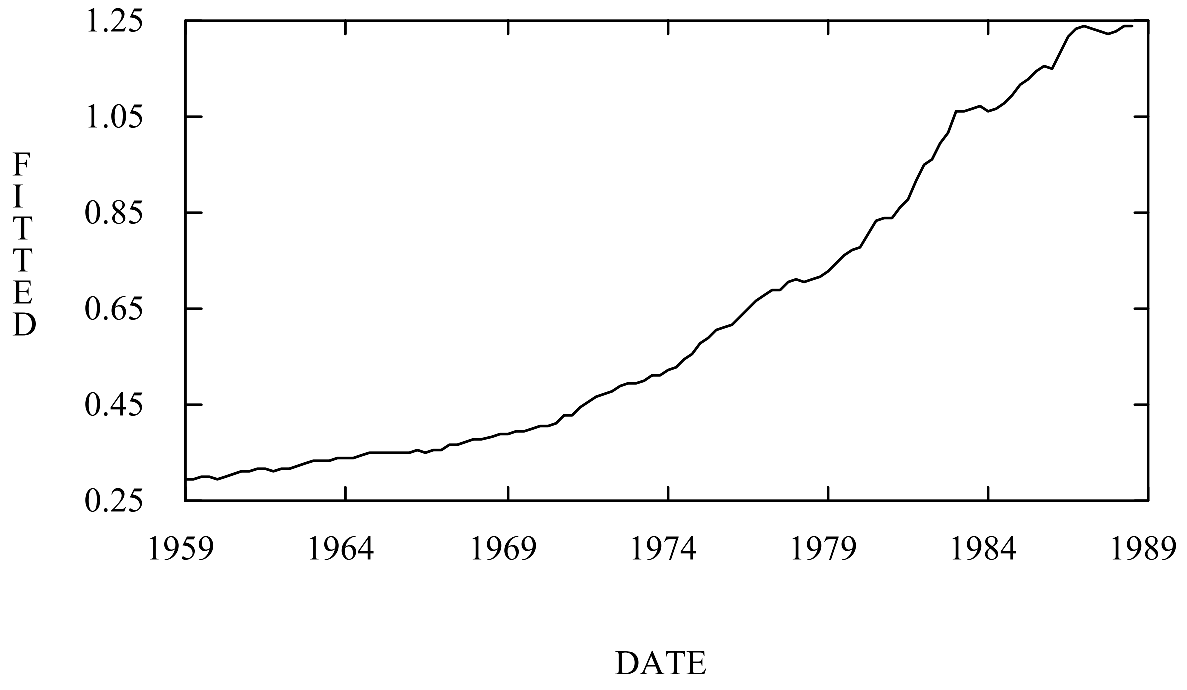
The graph below shows the gross national product deflator GD from 1959:1 to 1988:4.



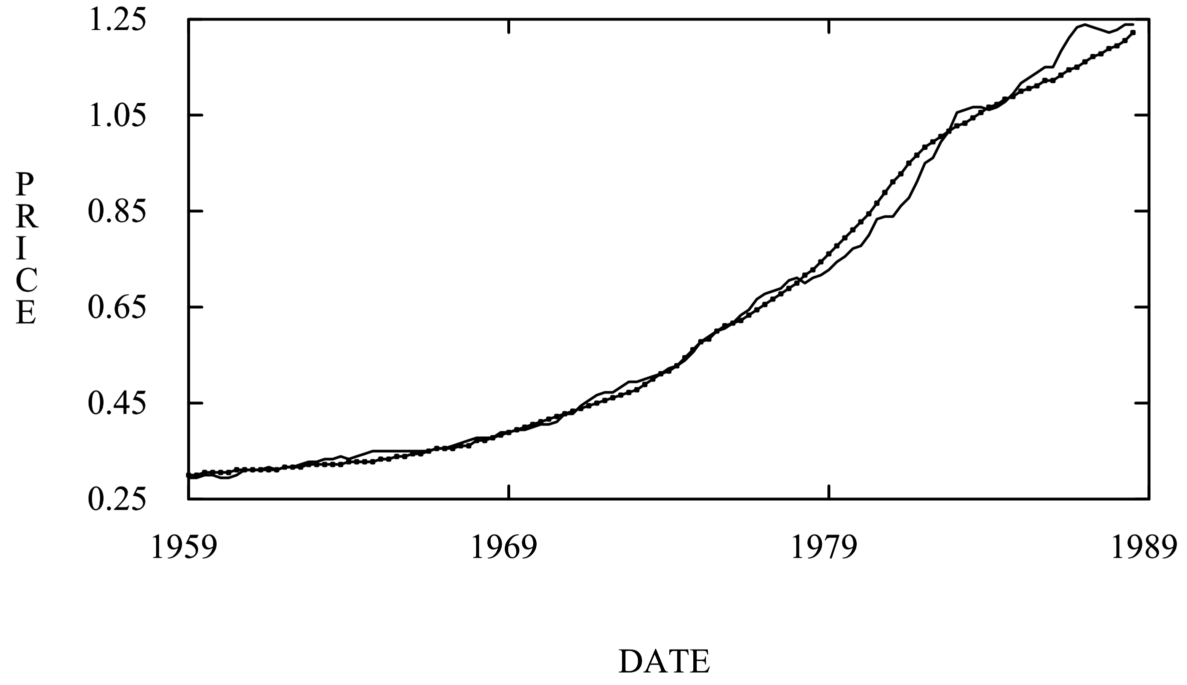
The actual long-term historic postwar value of the M2 velocity of money in the United States is 1.6527 (Hallman et. al. 1989, Humphrey 1989). Thus, the “correct” "exchange equation" is the multiplicative (non-linear) relationship

$$GD = \frac{(M2 * 1.6527)}{GNP82}$$

The graph below shows the fitted GD series calculated from the above model for 1959:1 to 1988:4.

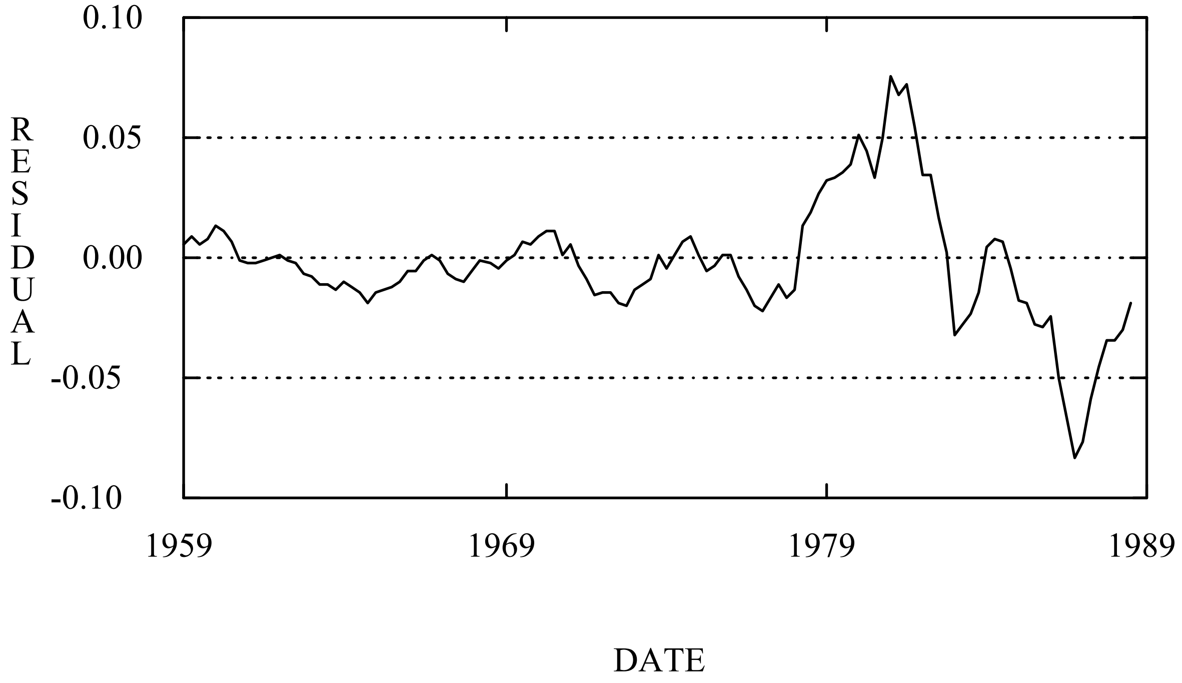


The graph below shows both the gross national product deflator GD from 1959:1 to 1988:4 and the fitted GD series calculated from the above model for 1959:1 to 1988:4. The actual GD series is shown as a line with dotted points. The fitted GD series calculated from the above model is a simple line.



The sum of the squared errors over the entire 30-year period involving 120 quarters (1959:1 to 1988:4) was 0.077193 and the R^2 value was 0.993320.

A plot of the corresponding residuals from the fitted GD series calculated from the above model for 1959:1 to 1988:4 is shown below:



5.1. MODEL DERIVED FROM FROM FIRST TWO-THIRDS OF DATA

We first divide the 30-year, 120-quarter period into a 20-year, 80-quarter "in-sample" period running from 1959:1 to 1978:4 and a 10-year, 40-quarter "out-of-sample" period running from 1979:1 to 1988:4.

The set of functions available for this problem is F = {+, -, *, %, EXP, RLOG}.

The set of terminals available for this problem is T = {GNP82, FM2, FYGM3, R}, where "R" is the ephemeral random constant atom allowing various random floating point constants to be inserted at random as constant atoms amongst the initial random LISP S-expressions. The "terminals" GNP82, FM2, and FYGM3 provide access to the values of the time series. In effect, these "terminals" are functions of the unstated, implicit time variable.

Notice that we are not told *a priori* whether the functional relationship between the given observed data (the three independent variables) and the target function (the dependent variable GD) is linear, multiplicative, polynomial, exponential, logarithmic, or otherwise. Notice also that we are not told that the addition, subtraction, exponential, and logarithmic functions as well as the time series for the 3-month Treasury bill rates (FYGM3) are irrelevant to the problem.

A population size of 500 individuals was used.

The initial random population (generation 0) was, predictably, highly unfit. In one run, the sum of squared errors between the single best S-expression in the population and the actual GD time series was 1.55. R^2 was 0.49.

After the initial random population is created, each successive new generation in the population is created by applying the operations of fitness proportionate reproduction and genetic recombination (crossover).

In generation 1, the sum of the squared errors for the new best single individual in the population improved to 0.50.

In generation 3, the sum of the squared errors for the new best single individual in the population improved to 0.05. This is approximately a 31-to-1 improvement over the initial random generation. R^2 improved to 0.98. In addition, by generation 3, the best single individual in the population came within 1% of the actual GD time series for 44 of the 80 in-sample points.

In generation 6, the sum of the squared errors for the new best single individual in the population improved to 0.027. This is approximately a 2-to-1 improvement over generation 3. R^2 improved to 0.99.

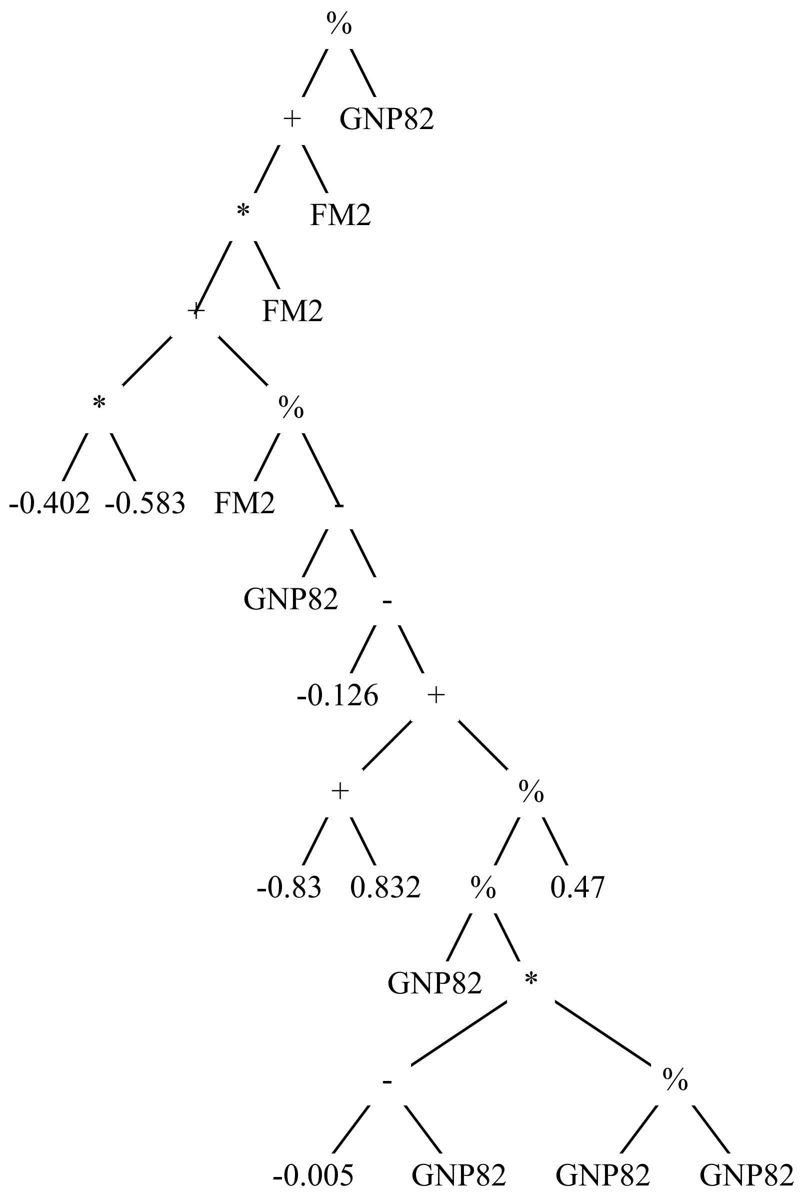
In generation 7, the sum of the squared errors for the new best single individual in the population improved to 0.013. This is approximately 2-to-1 improvement over generation 6.

In generation 15, the sum of the squared errors for the new best single individual in the population improved to 0.011. This is an additional improvement over generation 7 and represents approximately a 141-to-1 improvement over generation 0. R^2 was 0.99.

A typical best single individual from a late generation of this process had a sum of squared errors of 0.009272 over the in-sample period and is shown below:

```
(% (+ (* (+ (* -0.402 -0.583)
      (% FM2(- GNP82 (- 0.126
      (+ (+ -0.83 0.832)
      (% (% GNP82 (* (- 0.005 GNP82)
      (% GNP82 GNP82)))
      0.47)))))) FM2) FM2) GNP82).
```

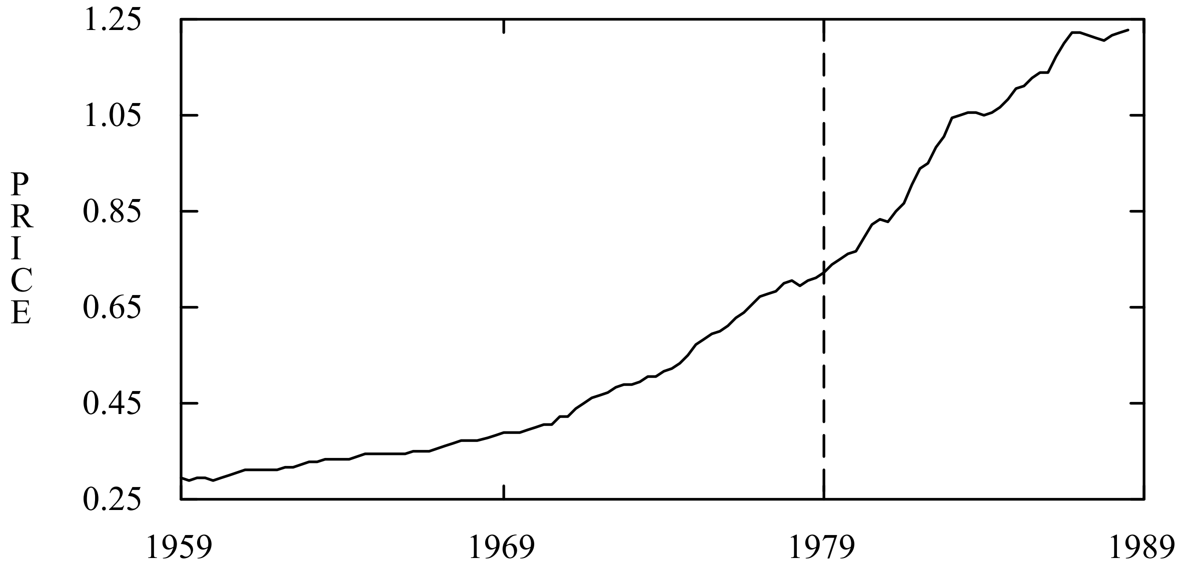
This individual can be graphically depicted as a rooted, point-labeled tree with ordered branches as shown below:



This individual is equivalent to

$$GD = \frac{(M2 * 1.634)}{GNP82}$$

The graph below shows the fitted values from this model (derived from the 80-quarter in-sample period) over all 120 quarters. The dotted line divides the 80-quarter in-sample period from the 40-quarter out-of-sample period.

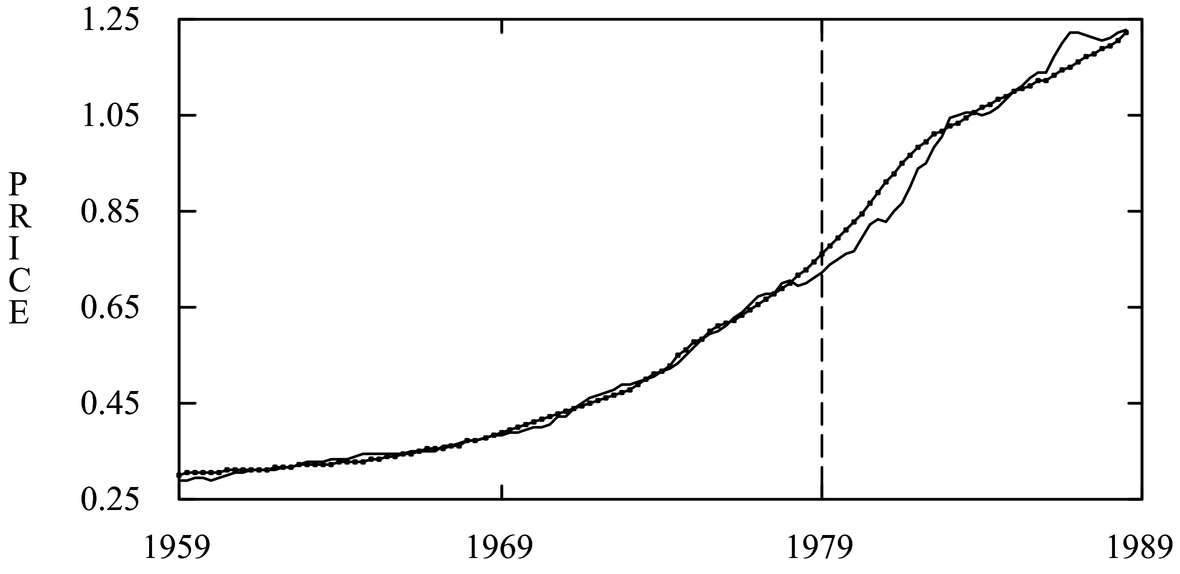


DATE

The table below shows the sum of the squared errors and R^2 for the entire 120-quarter period, the 80-quarter in-sample period, and the 40-quarter out-of-sample period:

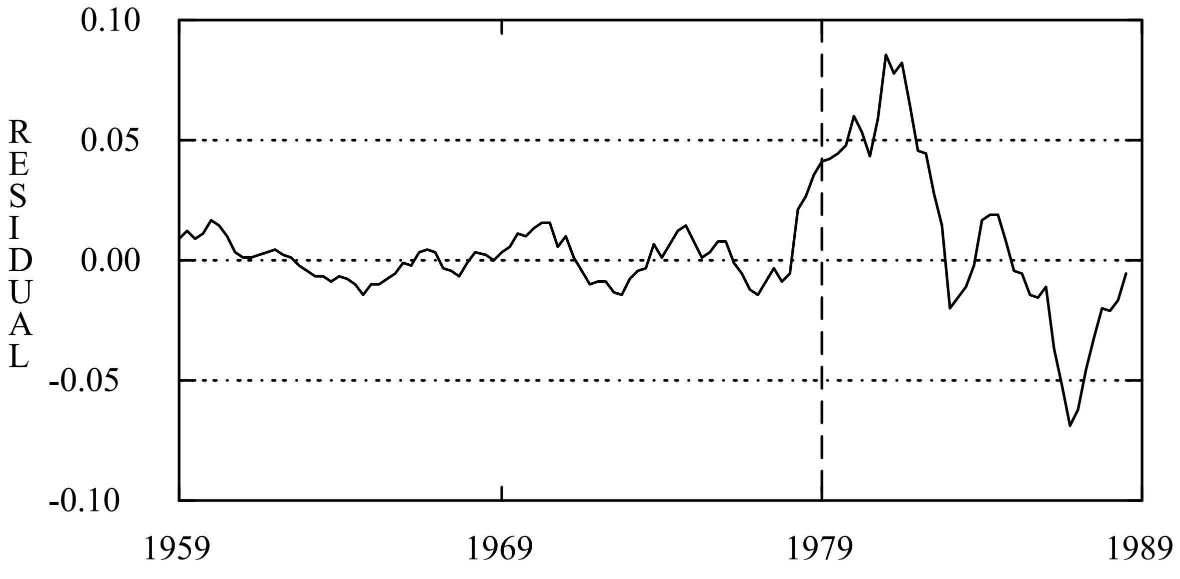
Data Range	*1- 120*	*1 - 80*	*81 - 120*
R^2	0.993480	0.997949	0.990614
Sum of Squared Error	0.075388	0.009272	0.066116

The graph below shows both the gross national product deflator GD from 1959:1 to 1988:4 and the fitted GD series calculated from the above model for 1959:1 to 1988:4. The actual GD series is shown as line with dotted points. The fitted GD series calculated from the above model is simple line.



DATE

A plot of the residuals from the fitted GD series calculated from the above model for 1959:1 to 1988:4 is shown below:



DATE

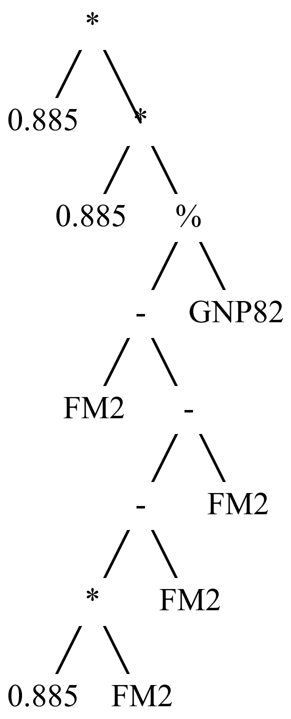
5.2. MODEL DERIVED FROM LAST TWO-THIRDS OF DATA

We now divide the 30-year, 120-quarter period into a 10-year, 40-quarter "out-of-sample" period running from 1959:1 to 1958:4 and a 20-year, 80-quarter "in-sample" period running from 1969:1 to 1988:4.

A typical best single individual from a late generation of this process had a sum of squared errors of 0.076247 over the in-sample period and is shown below:

```
(* 0.885 (* 0.885 (% (- FM2
            (- (- (* 0.885 FM2) FM2)
               FM2)) GNP82)))
```

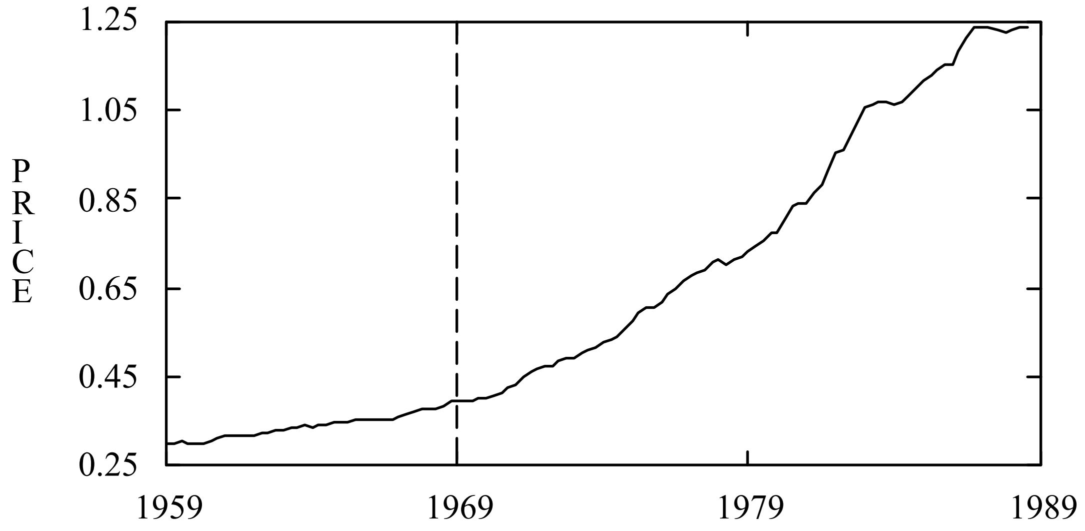
This individual can be graphically depicted as a rooted, point-labeled tree with ordered branches as shown below:



This individual is equivalent to

$$GD = \frac{(M2 * 1.6565)}{GNP82}$$

The graph below shows the fitted values from this model (derived from the 80-quarter in-sample period) over all 120 quarters. The dotted line divides the 40-quarter out-of-sample period from 80-quarter in-sample period.

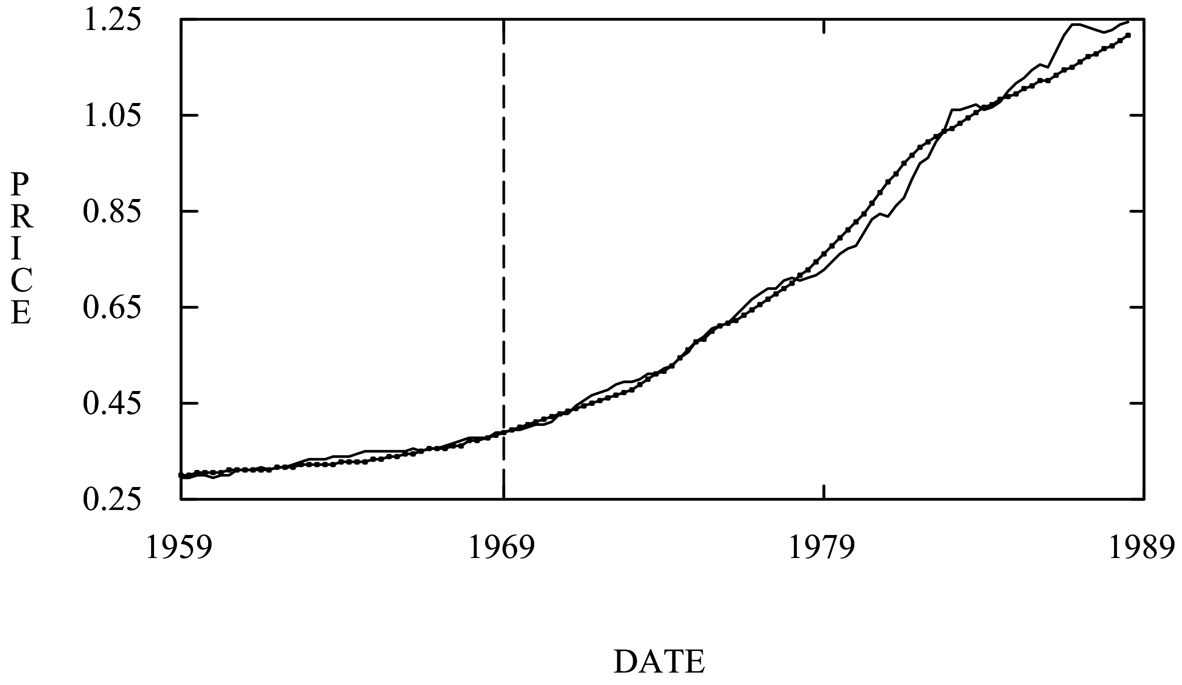


DATE

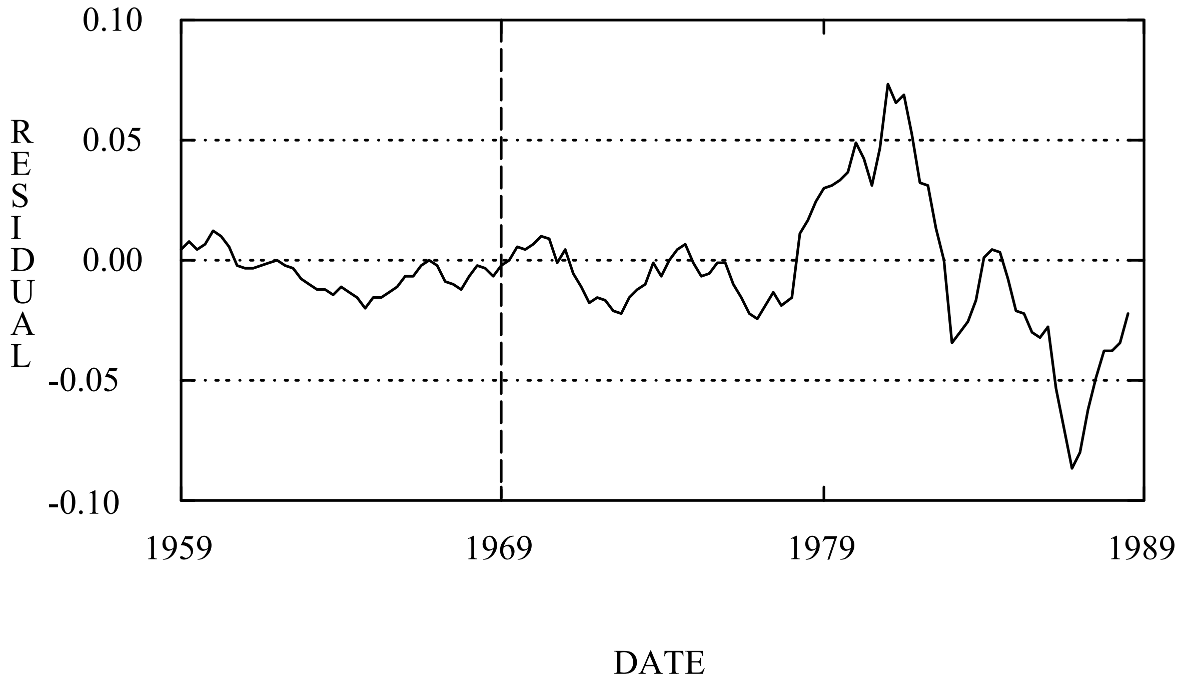
The table below shows the sum of the squared errors and R^2 for the entire 120-quarter period, the 40-quarter out-of-sample period, and the 80-quarter in-sample period, :

Data Range	*1- 120*	*1 - 40*	*41 - 120*
R^2	0.993130	0.999136	0.990262
Sum of Squared Error	0.079473	0.003225	0.076247

The graph below shows both the gross national product deflator GD from 1959:1 to 1988:4 and the fitted GD series calculated from the above model for 1959:1 to 1988:4. The actual GD series is shown as a line with dotted points. The fitted GD series calculated from the above model is simple line.



A plot of the residuals from the fitted GD series calculated from the above model for 1959:1 to 1988:4 is shown below:



6. ADDITIONAL EXAMPLES OF THE GENETIC PROGRAMMING PARADIGM

The "genetic programming" paradigm can be used to solve equations whose solution is in the form of a function that satisfies the given equation.

6.1. DIFFERENTIAL EQUATIONS

Differential equations are typically approached using analytic methods or numerical approximation methods. However, the problem of solving differential equations may be viewed as search in a hyperspace of compositions of functions and arguments for a particular composition which satisfies the equation and its initial conditions.

Consider the simple differential equation

$$\frac{dy}{dx} + y \text{ Cos } x = 0$$

having an initial value of y of 1.0 for an initial value of x of 0.0. The goal is to find a function which satisfies the equation, namely, $e^{-\text{Sin } x}$.

We start by generating 200 random values of the independent variable x_i over some domain, such as the unit interval [0, 1]. We call this set of values the "x-values." As the j-th individual candidate function f_j in the population is generated by the genetic algorithm, we evaluate $f_j(x_i)$ so as to obtain 200 pairs $(x_i, f_j(x_i))$. We call this 200 by 2 array of numbers the "unknown curve" or "candidate-curve."

We then numerically differentiate the "curve" $(x_i, f_j(x_i)$ with respect to the independent variable x_i to obtain the value of the derivative curve $(x_i, f_j'(x_i))$ for all 200 points. We then take the cosine of the 200 pairs of random values of x_i to obtain the curve consisting of the 200 values $(x_i, \text{Cos } x_i)$. We then multiply these 200 values by $f_j(x_i)$ so as to obtain 200 pairs $(x_i, f_j(x_i)*\text{Cos } x_i)$. We then perform the addition $f_j'(x_i) + f_j(x_i)*\text{Cos } x_i$ for all 200 points. To the extent that $f_j'(x_i) + f_j(x_i)*\text{Cos } x_i$ is close to zero for the 200 values of x_i, the unknown (candidate) function f_j is a good approximation to the solution to the differential equation.

To facilitate the input of the differential equations into the system, a toolkit has been written to allow a differential equation to be written out in terms of the independent variable x (called x-values), the unknown function being sought (called "candidate-function"), differentiation (with respect to a specified variable), and any other function. These functions can include addition, subtraction, multiplication, division, sine, cosine, etc. We apply a function to a "curve" by using the special "$" function and the name of the desired function. We adopt the convention that the right hand side of the equation is always zero. Thus, the left hand side of the differential equation involves the unknown (candidate) function. The "$" function can be applied to scalar constants as well as "curves." As an example, we would rewrite the left hand side of the above differential equation as

```
      ($'+ (differentiate candidate-function x-values)
           ($* candidate-function ($'cos x-values))).
```

This is interpreted as follows: The cosine function is applied to the independent variable (the x-values). Then, the result is multiplied by the unknown function y (the candidate function) to obtain an intermediate result. Then, the unknown function (candidate-function) is differentiated with respect to the independent variable (x-values) and the result added to the previously obtained intermediate result.

The sum of the absolute values of the differences between the left hand side of the equation and the right hand side of the equation (i.e. the zero function) is then computed. The closer this sum of differences is to zero, the better.

The fitness of a particular individual candidate (unknown) function is expressed in terms of two factors. The sum of absolute differences mentioned above represents 75% of the raw fitness of the function. The other 25% of the raw fitness is derived from the closeness of the candidate function to the initial condition, namely, the absolute value of the difference between the value computed by the candidate (unknown) function f_j for the initial condition and the given value for the initial condition.

The combined set of functions and terminals for this problem is T = {+, -, *, SIN, COS, RLOG, EXP, X}.

In one run, the best individual in the initial random population (generation 0) was the function

$$e^{1-e^{x}}$$

Its raw fitness was 58.09 and only 3 of the 200 points were "near hits."

By generation 2, the best individual in the population was

$$e^{1-e^{\mathrm{Sin}\,x}}$$

Its raw fitness was 44.23 and only 6 of the 600 points were "near hits."

By generation 6, the best individual in the population was equivalent to

$$e^{-\mathrm{Sin}\,x}.$$

The raw fitness had dramatically improved (decreased) to only 0.057. Moreover, 199 of the 200 points were now "near hits." This function is, in fact, the exact solution to the differential equation.

A second example of a differential equation is

$$\frac{dy}{dx} - 2y + 4x = 0$$

with initial condition such that y = 4 when x = 1.

In one run, the best individual in the 28th generation was

```
(+ (* (EXP (- X 1)) (EXP (- X 1))) (+ (+ X X) 1)).
```

This is equivalent to

$$e^{-2}e^{2x} + 2x + 1,$$

which is the exact solution to the differential equation.

A third example of a differential equation

$$\frac{dy}{dx} = \frac{2 + \mathrm{Sin}\,X}{3\,(y-1)^2}$$

with initial condition such that y = 2 when x = 0.

In one run, the best individual in the l3th generation was

```
(- (CUBRT (CUBRT 1))
   (CUBRT (- (- (- (COS X) (+ 1 (CUBRT 1))) X) x))),
```

where CUBRT is the cube root function. This is equivalent to

$$1 + (2 + 2x - \mathrm{Cos}\,x)^{1/3},$$

which is the exact solution to the differential equation.

Note that when the initial condition of the differential equation involves only a value of the function itself (as is typically the case when the differential equation involves only first derivatives), any point in the domain of the independent variable (X) may be used for the initial condition. On the other hand, when the initial condition of the differential equation involves a value of a derivative of the function (as may be the case when the differential equation involves second derivatives or higher derivatives), it is necessary that the value of the independent variable (X) involved in the initial condition be one of the points in the random set of points x_i (and preferably an internal point of the domain). This is necessary to allow the first derivative (or higher derivative) of the unknown (candidate) function to be evaluated for the initial condition point.

6.2. INTEGRAL EQUATIONS

Integral equations are equations that involve the integral of the unknown function. Integral equations can be solved with the same general approach as above, except for the additional step of taking the integral of the candidate function.

An example of an integral equation is

$$y(t) - 1 + 2 \int_{r=0}^{r=t} \mathrm{Cos}(t-r)\, y(r)\, dr = 0$$

In one run, we found the solution to this integral equation, namely,

$$y(t) = 1 - 2te^{-t}$$

6.3. INVERSE PROBLEMS

Suppose we have a set of data consisting of various (x_i, y_i) pairs such as (9,6), (16,8), (25,10), (36,12), (2.25, 3.0), etc.

Symbolic regression would reveal that the dependent variable y_i is twice the square root of the independent variable x_i. That is,

$$y_i = 2 \sqrt{x_i}.$$

The problem of finding the inverse function involves a set of (x_i, y_i) pairs of data such as (6,9), (8,16), (10,25), (12,36), (3.0, 2.25), etc. and concluding that the dependent variable y_i is the square of half of the independent variable x_i. That is,

$$y_i = \frac{x_i^2}{2}$$

It will be seen that the problem of finding an inverse function for a given set of data is similar to the problem of symbolic regression discussed above, except for the additional step of switching the roles of the independent and dependent variables of the data set.

6.4. SOLVING FUNCTIONAL EQUATIONS

Consider the following functional equation:

$$f(2x) - 1 + 2\mathrm{Sin}^2 x = 0.$$

The goal is to solve this equation for the function f, which, when substituted into the equation, satisfies the equation.

As before, we begin by selecting a set of random points in a suitable domain. In particular, we select 50 points x_i in the domain of real numbers between -3.14 and +3.14. We store these 50 values in a vector. We then compute another vector of 50 values corresponding to the sine of each x_i. We then compute another vector of 50 values corresponding to the square of the sine of each x_i. We then compute another vector corresponding to twice the square of the sine of each x_i. Each of these computed vectors can also be viewed as a curve since we can think of the points for 2Sin^2x being plotted graphically on axes. Similarly, we set up a vector constant of 50 occurrences of the constant 1 (the "constant curve"). We then subtract this "constant curve" from the "curve" just computed for 2Sin^2x. Finally, we consider each of the S-expressions f_j in the current populations of individuals. Since the argument for the unknown function is 2x (instead of just x), we must first perform the step of multiplying the 50 x_i values by 2. We then compute the curve for f(2x) using the S-expression f_j.

If we happen to have the function f that exactly satisfies the equation, the new "curve" computed will consist of all zeroes. In any case, the value of the left hand side $f(2x) - 1 + 2\text{Sin}^2x$ corresponds to the fitness of the function.

In one run, the S-expression below emerged on generation 7 with a raw fitness of zero:

```
        (* 1 (COS (+ X X)).
```

This S-expression is equivalent of Cos 2x and solves the functional equation. That is, when Cos 2x is substituted into the equation

$$f(2x) - 1 + 2\text{Sin}^2 x = 0,$$

the equation is satisfied (i.e. the left hand side evaluates to zero for each random x_i).

6.5. SOLVING GENERAL MATHEMATICAL EQUATIONS FOR NUMERIC VALUES

An important special case of the process of solving functional equations occurs when the set of arguments (atoms) consists only of numeric constants. That is, there are no variable arguments (such as X) in the set of arguments used to construct the S-expressions. In this special case, the process can be used to solve a general mathematical equation for its numerical roots.

For example, consider the simple equation, with two identical roots of $\sqrt{2}$, which one would conventionally write as

$$x^2 - \sqrt{2}\, x + 2 = 0.$$

For our purposes here, we rewrite this equation as the functional equation

$$f^2(x) - \sqrt{2}\, f(x) + 2 = 0,$$

where the function f(x) is the unknown (instead of the variable x being the unknown).

We proceed by using a set of functions that contains functions such as addition, subtraction, multiplication, and division. The set of terminals (arguments), however, consists only of the ephemeral random constant atom ("R"). Note that x does not appear in this set of terminals (arguments). As a result, the set of S-expressions contains only random compositions of random constants. Typical S-expressions might be

```
        (+ 0.234 (* -0.685 0.478)) and
        (* (* 0.537 -1.234) (+ 1.467 0.899)).
```

As before, 50 random values of x_i are selected in a suitable domain (such as -2.0 to +2.0). A "curve" is then built up by squaring each x_i. Next, each x_i is multiplied by $\sqrt{2}$ and this result is subtracted from the square of each x_i. The constant value 2.0 is then added to each of the 50 values. The next step is to evaluate the fitness of each of the 300 individual S-expressions f_j in the population. Each S-expression in this problem has a particular numeric value because the initial population of S-expressions contained only constants. Its value does not depend on x_i. Thus, when each f_j is evaluated for fitness, the value is the same for all 50 cases (because the value f_j does not depend on x_i). As before, the sum of these 50 (identical) values is the fitness of the S-expression f_j. If the S-expression causes the left hand side of the equation (i.e. the raw fitness side) to be zero, that S-expression (which is, in fact, a numeric constant value) satisfies the equation.

In one run, the best individual S-expression in the 42nd generation evaluated to 1.41417. This is within 0.00004 of the value of |2, which is approximately 1.41421.

Note that this genetic approach to solving equations for numeric values produces quite precise values. This result is contrary to the conventional view that genetic algorithms are only good for searching for the general neighborhood of a correct answer in a large search space.

This view is perhaps correct when applied to conventional genetic algorithms operating on character strings whose length is fixed in advance. However, where the size and shape of the solution is allowed to vary dynamically as the problem is being solved, it is possible to search a large search space for the correct general neighborhood of the solution and then, by adaptively changing the representation scheme, converge closely onto the precise correct value.

7. FUTURE WORK

The techniques discussed in this paper should now be used to try to solve more complicated econometric modeling problems.

8. CONCLUSIONS

We have shown how the "genetic computing" paradigm can be used to create an econometric model by rediscovering the well-known non-linear econometric "exchange equation" relating the price level, gross national product, money supply, and velocity of money in an economy.

ACKNOWLEDGMENTS

James P. Rice of the Knowledge Systems Laboratory at Stanford University made numerous contributions in connection with the computer programming of the above.

Christopher Jones of the Engineering Economics Department at Stanford University analyzed the results of the genetic algorithms and produced the graphs in this article.

REFERENCES

Citibank, N. A. *CITIBASE: Citibank Economic Database (Machine Readable Magnetic Data File)*, 1946 - Present. New York: Citibank N.A. 1989.

Davis, L. (editor) *Genetic Algorithms and Simulated Annealing* London: Pittman l987.

Dawkins, Richard. The Blind Watchmaker. New York: W. W. Norton 1987.

De Jong, Kenneth A. Genetic algorithms: A l0 year perspective. *Proceedings of an International Conference on Genetic Algorithms and Their Applications.* Hillsdale, NJ: Lawrence Erlbaum Associates l985.

Doan, Thomas A. *User Manual for RATS - Regression Analysis of Time Series*. Evanston, IL: VAR Econometrics, Inc. 1989.

Fogel, L. J., Owens, A. J. and Walsh, M. J. Artificial Intelligence through Simulated Evolution. New York: John Wiley 1966.

Goldberg, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley l989.

Hallman, Jeffrey J., Porter, Richard D., Small, David H. *M2 per Unit of Potential GNP as an Anchor for* the Price Level. Washington,DC: Board of Governors of the Federal Reserve System. Staff Study 157, April 1989.

Holland, J. H. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press 1975.

Holland, John H. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In Michalski, Ryszard S., Carbonell, Jaime G. and Mitchell, Tom M. Machine Learning: An Artificial Intelligence Approach, Volume II. P. 593-623. Los Altos, CA: Morgan Kaufman l986.

Holland, J. H. The global economy as an adaptive system. In Anderson, Philip W., Arrow, Kenneth J., and Pines, David (editors) *Santa Fe Institute Studies in the Sciences of Complexity: The Economy as an Evolving Complex System.* Redwood City, CA: Addison_Wesley, 1990.

Humphrey, Thomas M. Precursors of the P-star model. *Economic Review*. Richmond, VA: Federal Reserve Bank of Richmond. July-August 1989. Pages 3-9.

Koza, John R. "Hierarchical Genetic Algorithms Operating on Populations of Computer Programs." In *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI)*. San Mateo: Morgan Kaufman 1989.

Koza, John R. *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems.* Stanford University Computer Science Department Technical Report STAN-CS-90-1314. June 1990. 1990a.

Koza, John R. and Keane, Martin A. "Cart Centering and Broom Balancing by Genetically Breeding Populations of Control Strategy Programs." In *Proceedings of International Joint Conference on Neural Networks*, *Washington, January 15-19, 1990*. Volume I. Hillsdale, NJ: Lawrence Erlbaum 1990.

Koza, John R. and Keane, Martin A. "Genetic Breeding of Non-Linear Optimal Control Strategies for Broom Balancing." In *Proceedings of the Ninth International Conference on Analysis and Optimization of Systems*. *Antibes, June, 1990.* Berlin: Springer-Verlag, 1990.

Langley, Pat and Zytkow, Jan M. Data-driven approaches to empirical discovery. *Artificial Intelligence*. 40 (1989) 283-312.

Langley, Pat, Simon, Herbert A., Bradshaw, Gary L., and Zytkow, Jan M. Scientific Discovery: Computational Explorations of the Creative Process. Cambridge, MA: MIT Press, 1987.

Marimon, Ramon, McGrattan, Ellen, and Sargent, Thomas J. Money as a medium of exchange in an economy with artificially intelligent agents. *Journal of Economic Dynamics and Control.* (14) 329-373 (1990).

Schaffer, J. D. Some effects of selection procedures on hyperplane sampling by genetic algorithms. In Davis, L. (editor) Genetic Algorithms and Simulated Annealing London: Pittman l987.

Schaffer , J. D. (editor) *Proceedings of the Third International Conference on Genetic Algorithms*. San Mateo, Ca: Morgan Kaufmann Publishers Inc. 1989.

Smith, Steven F. *A Learning System Based on Genetic Adaptive Algorithms*. PhD dissertation. Pittsburgh: University of Pittsburgh 1980.