Submitted April 24, 1994 for the book *Intelligent Systems for Finance and Business*. Total of 6,465 words.

GENETIC PROGRAMMING FOR ECONOMIC MODELING

John R. Koza

Computer Science Department

Stanford University

Stanford, California 94305-2140 USA

EMAIL: Koza@Cs.Stanford.Edu

PHONE: 415-941-0336

FAX: 415-941-9430

ABSTRACT

The problem of finding an econometric model expressing the mathematical relationship between empirically observed econometric variables can be viewed as a search of the space of possible computer programs for a program that produces the desired output for given inputs. This computer program can be found via the recently developed genetic programming paradigm which breeds populations of computer programs in a Darwinian competition using genetic operations such as fitness proportionate reproduction and crossover (sexual recombination). In this paper, we rediscover the well-known non-linear "exchange equation" $P=\frac{MV}{Q}$ relating the money supply, price level, gross national product, and velocity of money in an economy.

1. INTRODUCTION

The problem of discovering the mathematical relationship between the empirically observed variables measuring a system is an important problem in economics and other areas of science (Langley et. al. 1987). In practice, the observed data may be noisy and there may be no known way to express the relationships involved in a precise way. Problems of this type are sometimes called *symbolic system identification* problems, *black box* problems, *data mining* problems, or *modeling* problems. When the model that is discovered is used in predicting future values of the state variables of the system, the problem is called a *forecasting* problem.

Mathematical relationships can be expressed by means of formulae and ordinary mathematical equations; however, computer programs offer greater flexibility in the way that they express an arbitrary mathematical relationship. Computer programs start with one or more inputs (the independent variables of the system), perform various operations on the variables (including arithmetic operations and conditional decision-making operations), and produce certain outputs (the dependent variables). Thus, the problem of finding a mathematical model for empirical data can be viewed as a problem of finding a computer program that produces the observed value of the single dependent variable as its output when given the values of the independent variables as input. We call problems of this type *symbolic regression* because we are seeking a mathematical expression, in symbolic form, that fits, or approximately fits, a given sample of data.

Symbolic regression differs from conventional linear regression, quadratic regression, exponential regression, and other types of regression wherein the nature of the model is specified in advance by the user. In conventional linear regression, for example, 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 expression involving the independent variable(s) that minimizes some measure of error (such as the square root of the sum of the squares of the differences) between the values of the dependent variable(s) computed with the linear expression and the given values for the dependent variable(s). Similarly, in quadratic regression the goal is to discover a set of numerical coefficients for a quadratic expression that minimizes the error. In employing conventional regression technique, the user must select, as a preparatory step, whether to try to fit to a linear model, a quadratic model, or some other model. But often, the real problem is deciding what type of model most appropriately fits the data, not merely computing the appropriate numerical coefficients after the model has been preselected. Symbolic regression searches for both the functional form and the appropriate numeric coefficients that go with that functional form. Finding the functional form and the appropriate numeric coefficients of a 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.

The desired computer program can be found by means of the recently developed genetic programming paradigm originally developed for solving problems of artificial intelligence, automatic programming, and machine learning. In genetic programming, populations of computer programs are bred using Darwinian competition and genetic operations. The Darwinian competition is based on the principle of survival and reproduction of the fittest. The genetic crossover (sexual recombination) operator is genetically mates 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 may be a satisfactory solution to the problem.

In this chapter, we illustrate the process of formulating and solving a problem of econometric modeling using genetic programming. We focus on the non-linear econometric exchange equation $P=\frac{MV}{Q}$ relating the price level, gross national product, money supply, and velocity of money in an economy. The problem of modeling requires finding the computer program, in symbolic form, that fits given numeric data. The numerical value of the velocity of money is one of the elements of the to-be-discovered computer program.

In genetic programming, the individuals in the population are compositions of functions and terminals appropriate to the particular problem domain. The set of functions used typically may include arithmetic operations, mathematical functions, conditional logical operations, and other functions appropriate to the problem domain at hand. The set of terminals typically includes the inputs to the as-yet undiscovered computer program and various numerical constants. The search space is the space of all possible computer programs that can be recursively composed of the available functions and terminals. The crossover operation appropriate for mating two parents from this space of programs creates new offspring programs by exchanging sub-trees between the two parents.

As will be seen, the computer program required to solve the problem described above will emerge from a simulated evolutionary process. This simulated evolutionary process will start with an initial population of randomly generated computer programs composed of functions and terminals appropriate to the problem domain.

The fitness of each individual computer program in a population at any stage of the process will be measured as to how well the program grapples with its problem environment (i.e., of producing the observed outputs from the given inputs). In particular, fitness will be measured by the sum of the squares of the distances (taken for all the fitness cases) between the point in the solution space created by the program for a given set of inputs and the correct point in the solution space. The closer this sum is to zero, the better the program.

Predictably, the initial random individual programs will have exceedingly poor fitness. Nonetheless, some individuals in the population will be somewhat more fit than others. Then, a process based on the Darwinian principle 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 programs will be used to create offspring programs. Both participating parental programs are selected in proportion to fitness. The resulting offspring programs will be composed of subexpressions (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 dynamic variability of the computer programs that are developed along the way to a solution is also an essential aspect of genetic programming. We do not specify the size and shape of the eventual solution in advance. 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.

2. 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 natural genetic operations such as sexual recombination (crossover) and fitness proportionate reproduction (Darwinian survival of the fittest).

Genetic algorithms begin with an initial population of individuals (typically randomly generated) and then, iteratively, evaluate the individuals in the population for fitness with respect to the problem environment and perform genetic operations on various individuals in the population to produce a new population.

John Holland 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 reformulation 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. The current increasing interest in genetic algorithms stems from their intrinsic parallelism, their mathematical near optimality in solving problems, and the availability of increasing parallel computers. An overview of genetic algorithms can be found in Goldberg's *Genetic Algorithms in Search, Optimization, and Machine Learning* (1989). Additional information on current work in genetic algorithms can be found in Forrest (1993), Davis (1987, 1993), Michalewicz (1992), Bauer (1994), Whitley (1992), Maenner and Manderick (1992), Schaffer and Whitley (1992), Albrecht, Reeves, and Steele (1993), and Langton (1994).

The classifier system (Holland 1986) 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). 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. Holland (1990) discusses the global economy as an adaptive system.

3. GENETIC PROGRAMMING

Genetic programming is an extension of the genetic algorithm in which the genetic population consists of computer programs (that is, compositions of primitive functions and terminals).

The book *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (Koza 1992) demonstrated a surprising and counterintuitive result, namely that computers can be programmed by means of natural selection. Specifically, genetic programming is capable of evolving a computer program for solving, or approximately solving, a surprising variety of problems from a wide variety of fields. To accomplish this, genetic programming starts with a primordial ooze of randomly generated computer programs composed of available programmatic ingredients and genetically breeds the population using the Darwinian principle of survival of the fittest and an analog of naturally occurring genetic crossover (sexual recombination) operation. In other words, genetic programming provides a way to search the space of possible computer programs to find a program which solves, or approximately solves, a problem.

Genetic programming is a domain independent method that genetically breeds populations of computer programs to solve problems by executing the following three steps:

(1) Generate an initial population of random computer programs composed of the primitive functions and terminals of the problem.

(2) Iteratively perform the following substeps until the termination criterion has been satisfied:

   (a) Execute each program in the population and assign it a fitness value according to how well it solves the problem.

   (b) Create a new population of programs by applying the following two primary operations. The operations are applied to program(s) in the population selected with a probability based on fitness (i.e., the fitter the program, the more likely it is to be selected).

      (i) *Reproduction*: Copy an existing program to the new population.

      (ii) *Crossover*: Create two new offspring programs for the new population by genetically recombining randomly chosen parts of two existing programs. The genetic crossover (sexual recombination) operation (described below) operates on two parental computer programs and produces two offspring programs using parts of each parent.

(3) The single best computer program in the population produced during the run is designated as the result of the run of genetic programming. This result may be a solution (or approximate solution) to the problem.

The use of automatically defined functions in genetic programming is discussed in Koza (1994). Recent advances in genetic programming are described in Kinnear (1994). A videotape visualization of numerous applications of genetic programming can be found in Koza and Rice (1992) and Koza (1994).

3.1 Preparatory Steps in Applying Genetic Programming

In applying genetic programming with automatic function definition to a problem, there are five major preparatory steps. These steps involve determining

(1) the set of terminals,
(2) the set of functions ,
(3) the fitness measure,
(4) the parameters and variables for controlling the run, and
(5) the criterion for designating a result and terminating a run.

3.2 Terminal Set and Function Set

The set of terminals and functions are the ingredients from which the yet-to-be-evolved computer program will be composed.

Consider the well-known econometric exchange equation $P=\frac{MV}{Q}$, 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.

• GNP82 is annual rate for the United States Gross National

Product in billions of 1982 dollars.

• GD is the Gross National Product Deflator (normalized to 1.0 for 1982).

• M2 is the monthly value of the seasonally adjusted money stock M2 in billions of dollars, averaged for each quarter.

• FYGM3 is the monthly interest rate yields of 3-month Treasury bills, averaged for each quarter.

In attempting to rediscover the exchange equation using genetic programming paradigm, we might use the function set for the problem

F = {+, -, *, %, RLOG, EXP}.

The first four functions in this function set are arithmetic operations. The protected division operation % produces a result of one if division by zero is attempted. RLOG and EXP are protected versions of the mathematical functions for the logarithm and exponential. The protected logarithm function RLOG is the logarithm of the absolute value and is equal to zero for an argument of zero. These definitions allow arbitrary compositions of the functions in the function set.

GNP82, GD, FM2, and FYGM3 are functions of time, *T*, measured in quarters. However, if the time variable, T, is made implicit, these four functions become zero-argument functions and are regarded as terminals (i.e., endpoints of the parse tree) so that the terminal set for the problem

T = {GNP82, GD, FM2, FYGM3, ←}.

Here ← is the ephemeral random constant allowing various random floating point constants to be inserted at random amongst the initial random computer programs as described in detail in Koza 1992. 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.

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,

$$P = \frac{MV}{Q} \tag{1}$$

or, alternately,

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

As it happens, the monthly interest rate yields of 3-month Treasury bills (FYGM3) is extraneous in this equation.

In this chapter, we will present programs in the LISP programming language. In LISP, everything is expressed in terms of functions operating on some arguments. In computer programs, 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 computer program (+ 1 2) is equivalent to "1+2" in ordinary mathematics and evaluates to 3. In LISP, any argument can itself be an program. 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.

Thus, in LISP, one correct computer program for the price level of an economy in terms of the exchange equation would be

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

Any computer program 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 terminals, and whose root is labeled with the function appearing just inside the outermost left parenthesis. The tree corresponding to the computer program above for the exchange equation is shown in figure 1.

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

It is not necessary to use the LISP programming to use genetic programming. In fact, most implementations of genetic programming employ other computer languages. However, LISP makes it clear that all computer programs are applications of functions to arguments and can be represented as a parse tree such as figure 1.

3.3 The Fitness Measure

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.

Fitness cases provide a basis for evaluating a particular program. For example, for the exchange equation, the fitness cases consist of the 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.

The raw fitness of any computer program is the sum, over the fitness cases, of the squares of the distances (taken over all the fitness cases) between the point in the solution space (which is real-valued) returned by the individual program 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 computer program *h* in the population of size *M* at any generational time step *t* is

$$r(i,t) = \sum_{j=1}^{N_e} |S(i,j) - C(j)|^2 \tag{3}$$

where *V(h,j)* is the value returned by program *h* for fitness case *j* (of N_e fitness cases) and where *S(j)* is the correct value for fitness case *j*. The closer this sum of distances is to zero, the better the program.

Thus, the raw fitness of an individual computer program 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 computer program 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

$$a(i,t) = \frac{1}{(1+\mathrm{r}(\mathrm{i},\mathrm{t}))} \tag{4}$$

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

$$n(i,t) = \frac{\mathrm{a}(\mathrm{i},\mathrm{t})}{\sum_{\mathrm{k}=1}^{\mathrm{M}} \mathrm{a}(\mathrm{k},\mathrm{t})} \tag{5}$$

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.

3.4 Parameters for Controlling Runs

The population size is 500. The maximum number of generations, *G*, to be run is 51 (i.e., an initial random population, called generation 0, and 50 additional generations). Crossover is performed on 90% of the population. That is, if the population size is 500, then 175 pairs of individuals from each generation are selected (with reselection allowed) from the population with a probability equal to their normalized adjusted fitness. In addition, fitness proportionate reproduction is performed on 10% of the population on each generation. That is, 50 individuals from each generation are 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. Several minor parameters are used to control the computer implementation of the algorithm as described in Koza 1992.

3.5 Result Designation and Termination Criterion

The single individual with the best value of fitness over all the generations (the so-called best-so-far individual) is designated as the result of a run. Each run is terminated after running $G = 51$ generations.

3.6 The Reproduction and Crossover Operations

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

The operation of fitness proportionate reproduction for the genetic programming paradigm 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 program. The result of this operation is one offspring program. 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

$$\frac{f(s_i(t))}{\sum_{j=1}^{M} f(s_j(t))} \tag{6}$$

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).

The crossover (recombination) operation for the genetic programming paradigm is a sexual operation that starts with two parental programs. Both parents are chosen from the population with a probability equal to its normalized fitness. The result of the crossover operation is two offspring programs. Unlike fitness proportionate reproduction, the crossover operation creates new individuals in the populations.

Every computer program 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 terminals, 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 programs. That is, for any two programs and any two crossover points, the resulting offspring are always valid computer programs. Offspring contain some traits from each parent.

The crossover fragment for a particular parent is the rooted subtree whose root is the crossover point for that parent and where the subtree consists of the entire subtree 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 sublist 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 subtrees are swapped, this genetic crossover (recombination) operation produces syntactically and semantically valid computer programs as offspring regardless of which point is selected in either parent.

For example, consider the parental computer program:

`(%` **`(+ 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 program below:

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

These two computer programs can be depicted graphically as rooted, point-labeled trees with ordered branches.

The two parental computer programs are shown in figure 2.

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 subtrees shown in figure 3. The places from which the crossover fragments were removed are identified with a #.

The remainders are shown in figure 4.

These two crossover fragments correspond to the bold, underlined subexpressions (sublists) in the two parental computer programs shown above. The two offspring resulting from crossover are shown in figure 5.

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

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

4. REDISCOVERING THE EXCHANGE EQUATION

The problem of discovering empirical relationships from actual observed data is illustrated by the well-known econometric exchange equation $P = \frac{MV}{Q}$, 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 the multiplicative (non-linear) relationship

$$\mathrm{GD} = \frac{(\mathrm{M2} * 1.6527)}{\mathrm{GNP82}} \tag{7}$$

from the actual observed time series data given the 120 quarterly values (from 1959:1 to 1988:4) of the four econometric time series GNP82, GD, FYGM3, and M2.

The four time series were obtained from the CITIBASE™ data base of machine-readable econometric time series (Citibank 1989).

The sum, over the entire 30-year period involving 120 quarters (1959:1 to 1988:4), of the squared errors between the actual 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 was 0.077193. 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 in figure 6.

4.1 Model Derived 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.

We are not told *a priori* whether the functional relationship between the given observed data (the independent variables) and the target function (the dependent variable GD) is linear, multiplicative, polynomial, exponential, logarithmic, or otherwise. 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) is irrelevant to the problem.

The initial random population (generation 0) was, predictably, highly unfit. An example of a randomly generated individual that appeared in generation 0 is

```
(RLOG GNP82), (+ FYGM3 (EXP -0.92)), (RLOG (+ 0.27 (EXP 0.65)))
```

In one run of the genetic programming paradigm, the sum of squared errors between the single best program in the population and the actual GD time series was 1.55. The value of 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 is equivalent to

$$GD = \frac{(M2 * 1.634)}{GNP82} \tag{8}$$

This individual can be graphically depicted as a rooted, point-labeled tree with ordered branches as shown in figure 7.

Table 1 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.

Table 1. The sum of the squared errors and R^2 for the 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

Figure 8 shows both the gross national product deflator GD from 1959:1 to 1988:4 and the fitted GD series calculated from the above genetically produced 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.

A plot of the residuals from the fitted GD series calculated from the above model for 1959:1

to 1988:4 is shown in figure 9.

4.2 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 is shown in figure 10.

This individual is equivalent to

$$GD = \frac{(M2 * 1.6565)}{GNP82} \tag{9}$$

Table 2 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.

Table 2. The sum of the squared errors and R^2 for the 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

Figure 11 shows both the actual gross national product deflator GD from 1959:1 to 1988:4 and the fitted GD series calculated from the above genetically produced 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 in figure 12.

5. CONCLUSION

We have shown how genetic programming 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. The genetically evolved program representing the exchange equation included an appropriate evolved numerical value for the velocity of money.

6. 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 Stanford University helped in analyzing the results.

7. REFERENCES

Albrecht, R. F., Reeves, C. R., and Steele, N. C. 1993. *Artificial Neural Nets and Genetic Algorithms*. Springer-Verlag.

Bauer, R. J., Jr. 1994. *Genetic Algorithms and Investment Strategies*. John Wiley.

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

Davis, L. (editor). 1987. *Genetic Algorithms and Simulated Annealing.* Pittman, London.

Davis, L. 1991. *Handbook of Genetic Algorithms* Van Nostrand Reinhold.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading.

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

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

Holland, J. H. 1986. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (editors). *Machine Learning: An Artificial Intelligence Approach*. Volume II, Pages 593-623. Los Altos, CA: Morgan Kaufman.

Holland, J. H. 1990. The global economy as an adaptive system. In *Santa Fe Institute Studies in the Sciences of Complexity: The Economy as an Evolving Complex System.*(P. W. Anderson, K. J. Arrow, and D. Pines editors.). Addison-Wesley, Redwood City.

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

Kinnear, K. E. Jr. (editor). 1994. *Advances in Genetic Programming*. Cambridge, MA: The MIT Press.

Koza, J. R. 1990. A Genetic Approach to Econometric Modeling. Paper presented at Sixth World Congress of the Econometric Society, Barcelona, Spain.

Koza, J. R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, MA: The MIT Press.

Koza, J. R. 1994. *Genetic Programming II*: *Automatic Discovery of Reusable Programs*. Cambridge, MA: The MIT Press.

Koza, J. R., and Rice, J. P. 1992.*Genetic Programming: The Movie*. Cambridge, MA: The MIT Press.

Koza, J. R. 1994. *Genetic Programming II Videotape: The Next Generation*. Cambridge, MA: The MIT Press.

Langley, P., H. A. Simon, G. L. Bradshaw and J. M. Zytkow. 1987. *Scientific Discovery: Computational Explorations of the Creative Process.* MIT Press, Cambridge.

Langton, C. G. (editor). 1994. *Artificial Life III.* Addison-Wesley.

Maenner, R., and Manderick, B. (editors). 1992. *Proceedings of the Second International Conference on Parallel Problem Solving from Nature*. North Holland.

Marimon, R., E. McGrattan and T. J. Sargent. 1990. Money as a medium of exchange in an economy with artificially intelligent agents. *Journal of Economic Dynamics and Control.* 14 329-373.

Michalewicz, Z. 1992. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag.

Schaffer, J. D. and Whitley, D. (editors). 1992. *Proceedings of the Workshop on Combinations of Genetic Algorithms and Neural Networks 1992*. Los Alamitos, CA: The IEEE Computer Society Press.

Whitley, D. (editor). 1992. *Proceedings of Workshop on the Foundations of Genetic Algorithms and Classifier Systems, Vail, Colorado 1992*. San Mateo, CA: Morgan Kaufmann Publishers Inc.

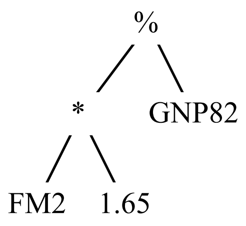

Figure 1 The exchange equation represented parsimoniously as a tree.

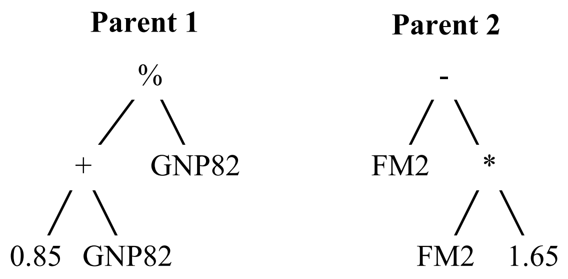

Figure 2. Two parental Computer programs for the crossover operation.

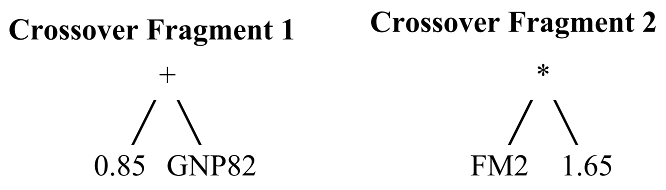

Figure 3. The two crossover fragments.

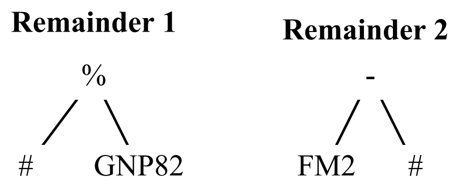

Figure 4. The remaining Computer programs.

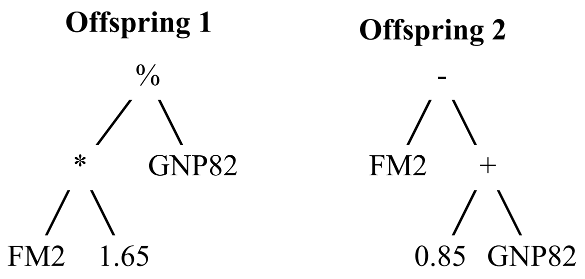

Figure 5. The two offspring from the crossover operation.

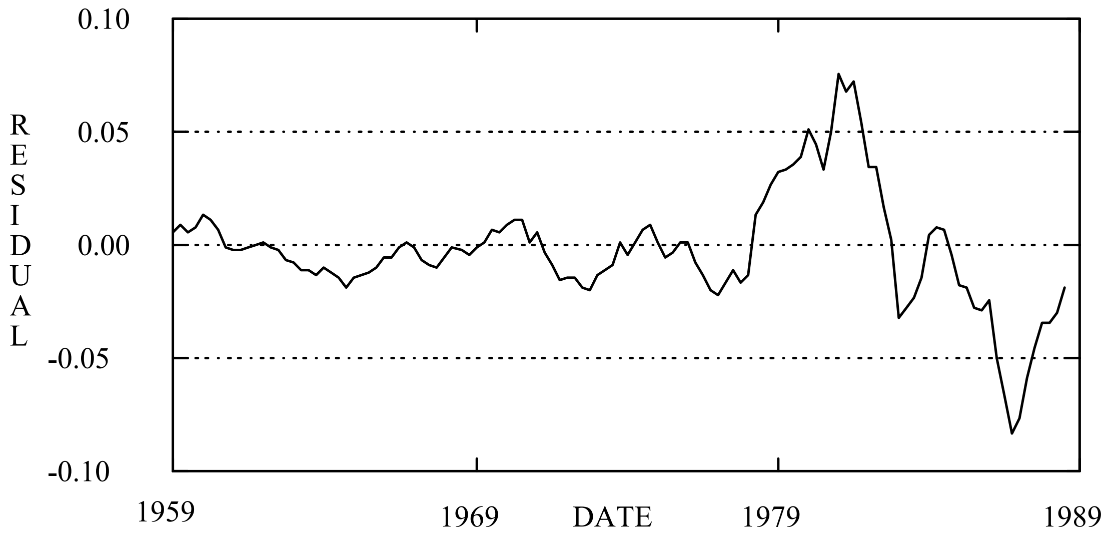

Figure 6. The corresponding residuals from the fitted GD series calculated from the above model for 1959:1 to 1988:4.

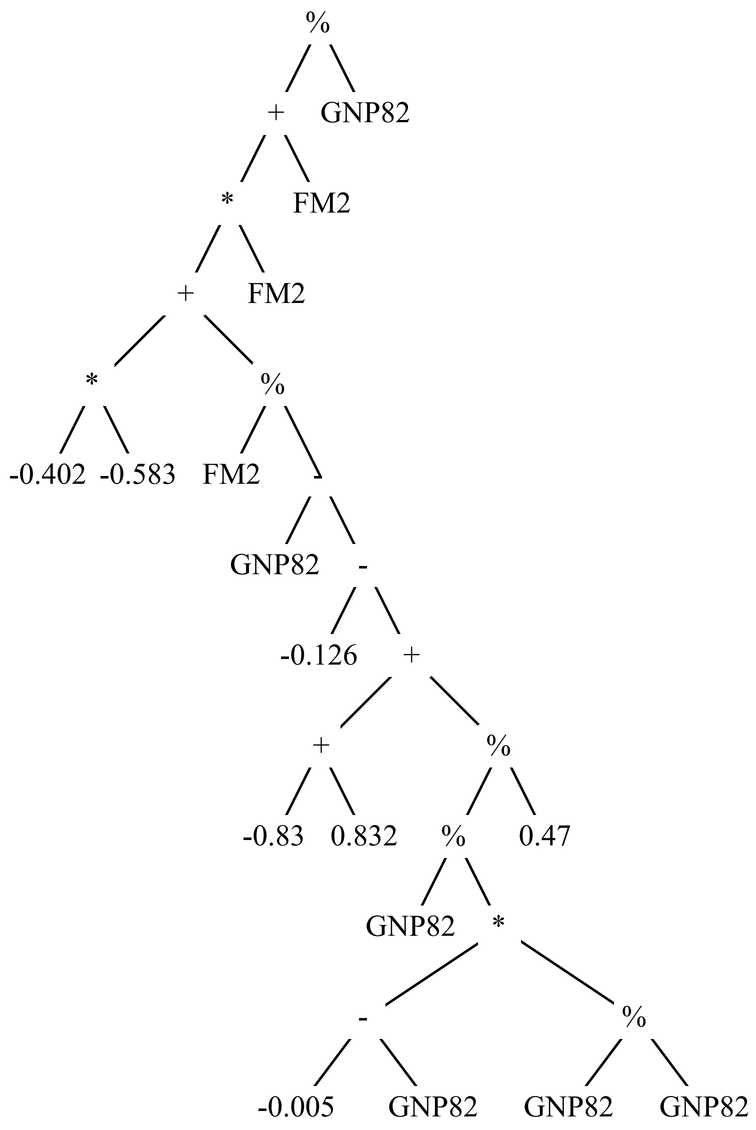

Figure 7. A typical best individual depicted as a tree.

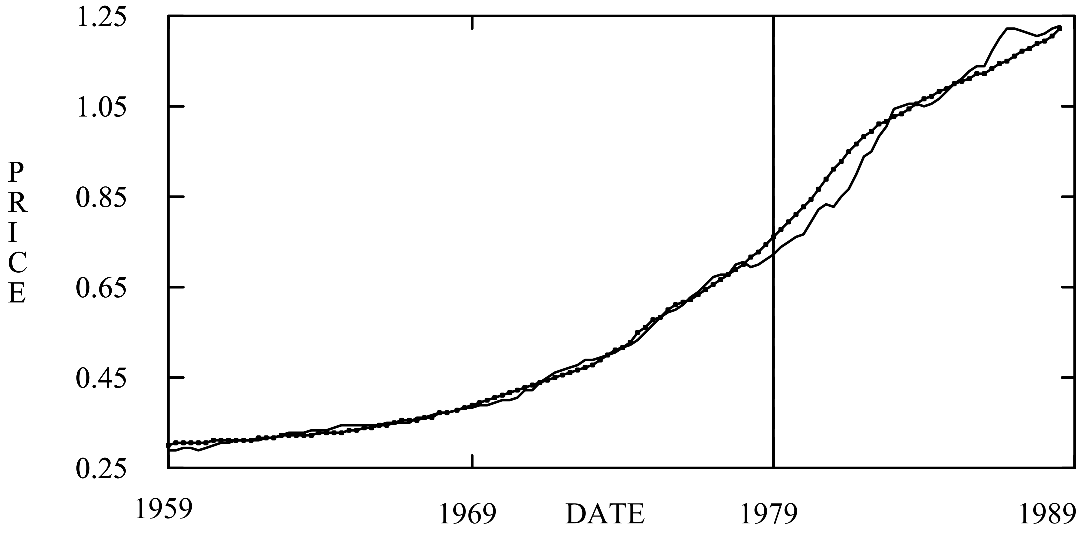

Figure 8. The gross national product deflator GD from 1959:1 to 1988:4 and the fitted GD series calculated from the above genetically produced model for 1959:1 to 1988:4. The actual GD series is shown as line with dotted points. The fitted GD series calculated from the above model is simple line.

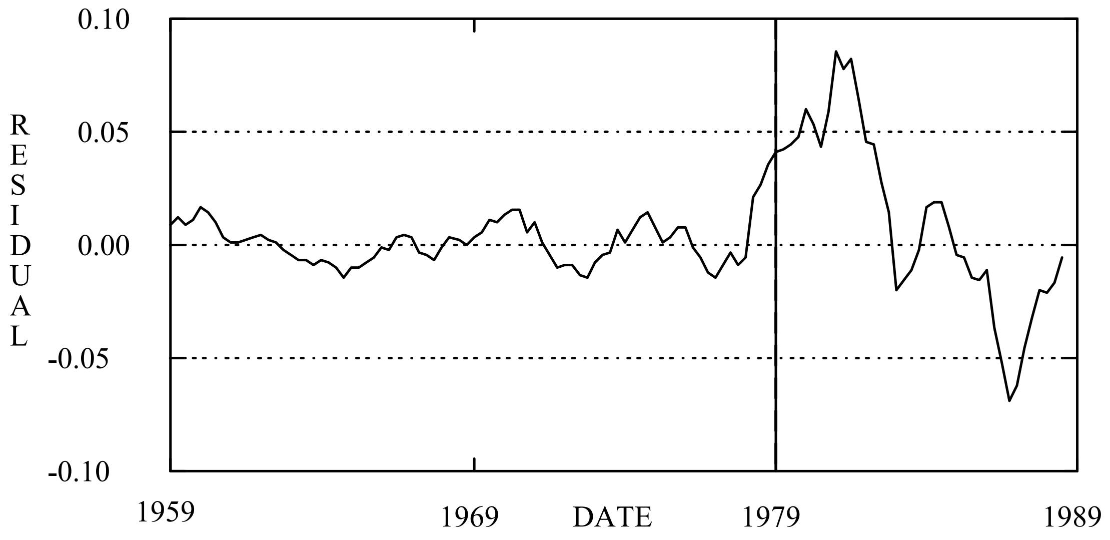

Figure 9. A plot of the residuals from the fitted GD series calculated from the above model for 1959:1 to 1988:4.

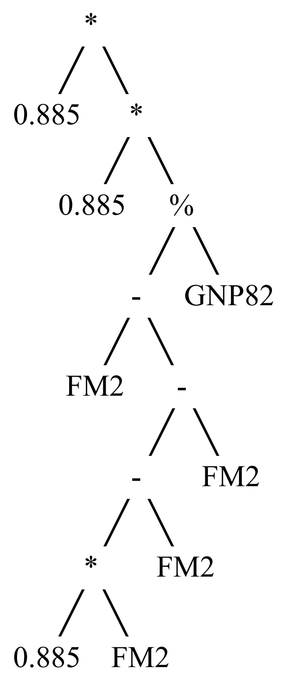

Figure 10. The individual graphically depicted as a rooted, point-labeled tree with ordered branches.

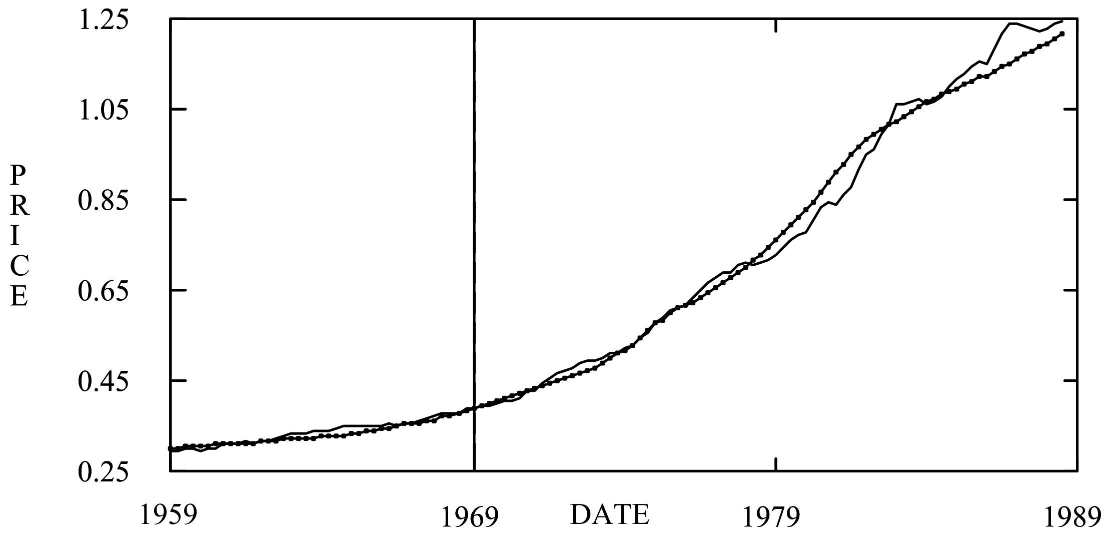

Figure 11. Both the gross national product deflator GD from 1959:1 to 1988:4 and the fitted GD series calculated from the above model for 1959:1 to 1988:4. The actual GD series is shown as a line with dotted points. The fitted GD series calculated from the above model is simple line.

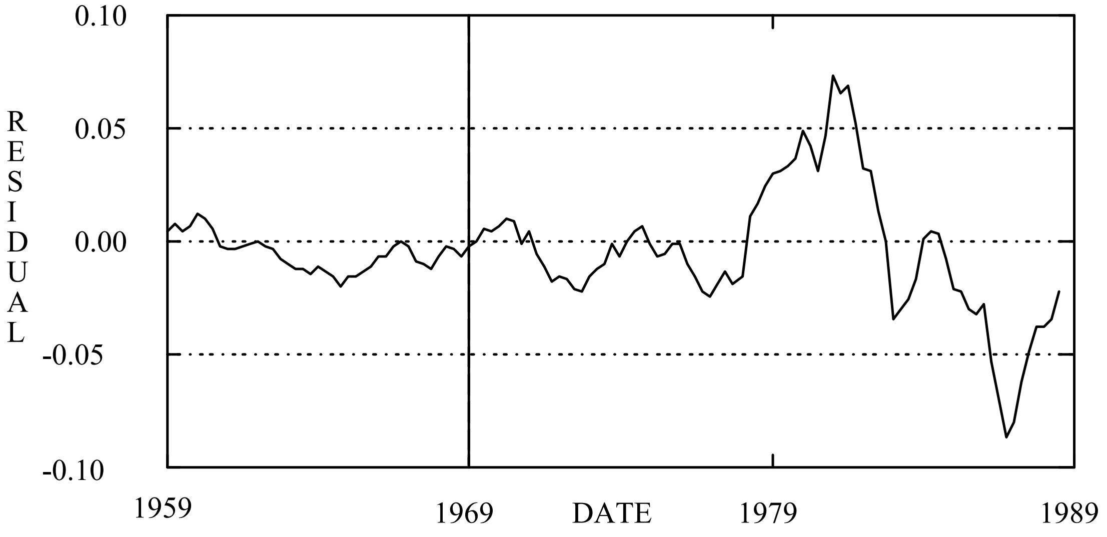

Figure 12. The residuals from the fitted GD series calculated from the above model for 1959:1 to 1988:4.