Computational Economics **10:** 267–277, 1997.
© 1997 *Kluwer Academic Publishers. Printed in the Netherlands.*

A Search for Hidden Relationships: Data Mining with Genetic Algorithms

GEORGE G. SZPIRO
Israeli Centre for Academic Studies, Kiriat Ono, Israel
(Author's address: POB 6278, Jerusalem, Israel. e-mail: george@netvision.net.il)

(Received: November 1996)

Abstract. This paper presents an algorithm that permits the search for dependencies among sets of data (univariate or multivariate time-series, or cross-sectional observations). The procedure is modeled after genetic theories and Darwinian concepts, such as natural selection and survival of the fittest. It permits the discovery of equations of the data-generating process in *symbolic* form. The genetic algorithm that is described here uses parts of equations as building blocks to breed ever better formulas. Apart from furnishing a deeper understanding of the dynamics of a process, the method also permits global predictions and forecasts. The algorithm is successfully tested with artificial and with economic time-series and also with cross-sectional data on the performance and salaries of NBA players during the 94–95 season.

Key words: data mining, forecasting, genetic algorithms

1. Introduction

The explanation and the prediction of natural phenomena, be it to forecast the future or to elicit the hidden laws that underlie unknown dynamics, are the ultimate aims of science. To achieve these aims, data need to be collected and analyzed. After formulating tentative explanations for a given phenomenon and collecting the other information deemed relevant, the researcher may wish to see if the data suggest additional cause-and-effect relations, perhaps hinted at by unexpected correlations between different variables. In the case of time-series, the issues in question are compounded since lagged observations must also be considered, and additional problems like serial correlations, seasonalities, etc. arise.

One way to search for underlying laws is to 'mine' the data for hidden dependencies among the variables. Plotting data in two dimensions and visually inspecting it is usually the first such attempt. Unfortunately, except in the most trivial cases, not much information is gained in this manner. At some point, the frustrated researcher may turn to experiments with various combinations of variables (joining them by, say, the four arithmetic operators). However, data mining as a means for scientific discovery is very much frowned upon, and rightly so. A blind search is usually hopeless, since a model with, say, only five explanatory variables and four lags already gives a total of 20 arguments that can then be combined, possibly multiple times, in any of four ways. Stories of researchers stumbling upon a fundamental law in such high-dimensional search-spaces belong to science fiction, and scientific inquiry without an ex ante theory is usually not considered good practice. However, similar remarks are made about other techniques, which to all appearances border on the superstitious, for instance the so-called technical analysis of the stock market, i.e., the search for supposedly recurring patterns in the movement of stock prices. In spite of much ridicule by financial professionals for such a naïve investment approach, some researchers have not been deterred from applying more refined, high-dimensional versions of this methodology, for example, the so-called nearest neighbor technique (Farmer and Sidorovich, 1987), which performs technical analysis in higher dimensions. In light of some recent results (e.g., Scheinkman and LeBaron, 1989; Hsieh, 1991; LeBaron, 1992, 1993; Brock, Lakonishok, and LeBaron, 1992; Allen and Karjalainen 1993), it remains possible that this generally belittled approach could yield explanations about factors that influence market movements. Unfortunately, it is unlikely that such influence could be used for personal gain since market efficiencies will quickly remove any profit opportunities.

Similarly, it may not be unreasonable to expect a sophisticated search method to turn up hitherto unknown equations and formulas that underlie natural phenomena. We propose an algorithm that uses the concepts of evolutionary development, to breed equations whose performance improves in each generation.

We describe only the first part of a three-step program that should be followed when investigating and analyzing data. This first step consists in applying the genetic algorithms to find models that fit the data. In a second step, one ascertains that the equation that was found does not represent some spurious relationship between variables, and in the final step, the researcher must explain why it is that the formula describes the phenomenon.

Genetic algorithms were pioneered by Holland (1975) and Goldberg (1989) and have already been used in economics (Palmer et al., 1994) and in finance (Allen and Karjalainen, 1993). However, so far they have mostly been applied in areas such as engineering, chemistry, optimization, acoustics, and pattern recognition. The closely related method of evolutionary programming has been employed to elicit some basic physical laws from experimental data, for example, Kepler’s third law and Ohm’s law (Koza, 1992).

The next section, which borrows from a previous article (Szpiro, 1996), introduces and explains genetic algorithms and the manner in which we shall use them. In Section 3 we present some examples – numerical experiments, with and without noise, as well as data drawn from economics. Section 4 summarizes the results.

2. Genetic Algorithms

The basic tenet of this paper is that mathematical equations are just strings of symbols that conform to a simple grammar: two arguments are combined by an arithmetic operator and the resulting expression is enclosed in parentheses. The arguments are either real numbers, variables, or lagged variables, or are themselves self-contained expressions enclosed in a pair of parentheses. The algorithm that we propose builds up equations from such building blocks and then breaks the strings up again. The resulting expressions, super-expressions, and compounded expressions form the material on which the Darwinian processes operate: natural selection and survival of the fittest cause suitable blocks to tend to combine with other useful blocks while useless parts disappear. The genetic algorithm consists of a few, simple routines, which we now describe.

(a) *Initialization*. The 'agents', j, of the populations are initially endowed with simple equation-strings of the form

$$\text{Equation-string}_j = ((\text{A Op B})\ \text{Op}\ (\text{C Op D})), \tag{1}$$

where A, B, C, and D are variables of a multivariate series, or lagged values of a time-series, or real numbers, and where 'Op' stands for one of the four arithmetic operators, addition, subtraction, multiplication, and division.[1] The operators and arguments are assigned at random.

(b) *Computing the fitness* The fitness of each equation-string is assessed by how well it explains the variance of the training set, i.e., by the R^2-value. In order to discourage the genetic algorithm, however, from creating ever longer strings, we perform a modification to the value of R^2 in the following manner

$$r^2 = 1 - (1 - R^2)\frac{n-1}{n-k}, \tag{2}$$

where n is the length of the training period and k is the number of variables or lagged data that appear in the string.[2] We use r^2 as the fitness measure in the algorithm, but report the familiar R^2-value, since explained variance is a more customary concept.

(c) *Ranking of agents*. The agents are ranked in descending order of their fitness.

(d) *Choice of mates*. In this subroutine the members of the population are organized into pairs. The fittest agent is the first to choose a mate. It makes its choice from among the remaining agents, the probability of any one of them being chosen as a partner being proportional to its fitness. Then the next fittest chooses, etc., until half the agents have formed pairs. The remaining half disappear.

[1] Actually in order to avoid division by very small numbers or by zero, we use a 'protected' division, which, while preserving the sign, is defined as the division by Max(0.001, |B|). Other mathematical operators are conceivable, for instance the *Log* and *Exp* functions (with one argument), '*if – then – else*' statements (with three arguments), or Boolean operators.

[2] This new fitness measure r^2, which confers an advantage on short strings, is similar to the *corrected* R^2, as it is known from statistics, the difference being that multiple occurrences of the same variable are counted separately: compositions like A* A/A increase k by three, and B/B or C−C increase k by two, even though in the latter cases there are no new variables.

(e) *Reproduction and crossover*. This routine is the heart of the algorithm. Each pair of agents has four offspring. The first two are identical to their parents. The equation-strings of the two other offspring are formed as recombinations of their parents' equation-strings: randomly chosen, self-contained parts of the parents' equation-strings (either a randomly chosen datum or real number, or a randomly chosen opening parenthesis, and all the contents of the equation-string until the corresponding closing parenthesis) are interchanged. Consider, for example, parent equation-strings given as follows

$$\begin{aligned} &\text{Parent}_1\text{:} \quad (A^*B)/C, \\ &\text{Parent}_2\text{:} \quad (D - (E/F)). \end{aligned} \tag{3}$$

Offspring$_1$'s and Offspring$_2$'s equation-strings are identical to their parents. Then – by crossover of the building blocks B and (E/F) – the two other offspring's equation-strings could be of the form

$$\begin{aligned} &\text{Offspring}_3\text{:} \quad (A^*(E/F))/C, \\ &\text{Offspring}_4\text{:} \quad (D - B). \end{aligned} \tag{4}$$

(f) *Mutation*. A small percentage of the equation-strings' basic elements, single arguments or operators, is mutated at random. The top ranked equation-strings are exempted from mutation, so that their information is not lost inadvertently. Mutation helps prevent agents from getting stuck on a local maximum.

Parts (b) to (f) of the algorithm are re-run for γ generations.

Other evolutionary schemes are, of course, also possible. For example, the procedure for the choice of a mate, or the method of passing 'genetic information' from one generation to the next, may be varied, and obviously the numerical values of the parameters can be changed as well. It is interesting to note, though, that the algorithm is quite robust to such changes. The most efficient algorithm must be found through experimentation. It should be noted that the genetic algorithm does not necessarily find the simplest version of an expression. For instance the number 1.0 could be stated as A/A, zero as $B-B$, 2^*C as $C^*(D+D)/D$, etc. Also, equations need not be expressed in terms of the most recent lags, they could be recursively defined in terms of previous lags.

3. Examples

For the examples that are presented in this section, the algorithm is implemented with a population size of 400 and a maximal string length of 600. If equation-strings become longer than that, the string is re-initialized, when they are shorter, the remaining slots are filled with null symbols. Whenever an operator needs to be chosen at random – at initialization or at mutation –, the probabilities of getting a number or a variable are 0.25 and 0.75, respectively. Numbers are uniformly distributed in $[0, 10]$ with a precision of one decimal. In each generation 240 mutations occur among the 95 percent lower ranked equation-strings. We typically let the algorithm run for 200 or 400 generations.

As a first example of the power of genetic algorithms, we investigate a time-series of the so-called Hénon attractor [Hénon 1976]

$$x_0 = 1 - 1.4x^2_{-1} + 0.3x_{-2}. \tag{5}$$

Series produced by this process are known to be chaotic and are notorious for their behavior, which often cannot be distinguished from random data by conventional means. We create a series of length 100 and run the algorithm with a maximum of 10 lags. In two typical examples, the genetic algorithm 'breeds' the following equation-strings

$$\begin{aligned} \text{A0} \;=\; & ((((6.5 - ((7.8^*\text{A1})^*\text{A1}))/ \\ & ((3.6 - \text{A10}) + \text{A10})) + (((6.5 - (((9.7 + 0.7)^*\text{A1})^*\text{A1}))/ \\ & (3.6 + ((6.5^*\text{A3})^*(\text{A6}^*((6.5 - 6.5)/ \\ & ((3.6 - \text{A10}) + \text{A10})))))) + \text{A2}))/3.6), \end{aligned}$$

and

$$\begin{aligned} \text{A0} \;=\; & ((((5.1 - ((\text{A1}^*(4.5 + ((\text{A3} - A2)/((\text{A1}^*(4.5 + ((\text{A3} - \text{A1})/ \\ & (\text{A3} - \text{A2}))))^*\text{A1}))))^*\text{A1})) - 1.0) - (\text{A3}^*\text{A3}))/3.3), \end{aligned}$$

with $R^2 = 0.999$ and 0.995. The edited versions of these equation-strings are

$$x_0 = 1.0031 - 1.4043x^2_{-1} + 0.2778x_{-2} \tag{6}$$

and

$$\begin{aligned} x_0 \;=\; & 1.2434 - 1.3636x^2_{-1} - 0.30303x^2_{-3} \\ & +0.30303\frac{(x_{-2} - x_{-3})^2}{x_{-1} + 4.5x_{-2} - 5.5x_{-3}}, \end{aligned} \tag{7}$$

respectively. To the uninitiated reader the result is quite surprising: these equations are extremely similar to the form of the true data-generating process. Obviously, out-of-sample, one-period forecasts using either one of the above two strings are near perfect. Note that in the first of theses strings, the lagged variables A3 and A6 appear, but since they are multiplied by zero $(6.5 - 6.5)$ they disappear from the edited result. The variable A10 also emerges but disappears after being added to $-$A10.

For our second example, we generate an artificial data set, a_t, by constructing a time-series from the lagged components of a multivariate series,

$$a_0 = \frac{b_{-1}c_{-4}}{d_{-6}}. \tag{8}$$

The series b_t, c_t, and d_t are composed of pseudo-random numbers that are uniformly distributed in [0, 1], [1, 2], and [2, 3], respectively. A series of length 100 is generated, and a maximum of ten lags is specified. In two typical runs, after 400 generations we obtain

$$\begin{aligned} \text{A0} &= (\text{B1}/((\text{B4}/((\text{D1} + 2.7) + \text{C6})) + (\text{D6}/\text{C4}))), \quad \text{and} \\ \text{A0} &= ((\text{B1}^*\text{C4})/(\text{D6} + (\text{B2}/\text{D5}))), \end{aligned}$$

with R^2-values of 0.991 and 0.966, respectively. The edited versions of these strings have the following forms

$$a_0 = \frac{b_{-1}c_{-4}}{d_{-6} + \dfrac{b_{-4}c_{-4}}{c_{-6} + d_{-1} + 2.7}} \quad \text{and} \quad a_0 = \frac{b_{-1}c_{-4}}{d_{-6} + \dfrac{b_{-2}}{d_{-5}}}, \tag{9}$$

which are again fairly similar to the true data-generating process.

We now complicate the genetic algorithm's task somewhat by adding noise to the data-generating process

$$a_0 = \frac{b_{-1}c_{-4}}{d_{-6}} + \varepsilon_0. \tag{10}$$

The noise term ε_0 is normally distributed with zero mean and standard deviation 0.1. In three typical runs of 400 generations each, the genetic algorithm breeds the following strings

$$\text{A0} = ((\text{C4}^*\text{B1})/\text{D9}),$$

$$\text{A0} = (\text{B1}/(\text{D6} - (\text{C4}/2.2))), \quad \text{and}$$

$$\text{A0} = ((\text{B1}/((\text{D6} + \text{D5}) - \text{B5}))^*\text{D3}),$$

with R^2-values over the training set of 0.711, 0.756, and 0.704, respectively. One-period forecasts for out-of-sample time-series of length 100 typically give R^2-values of 0.640 to 0.750, 0.650 to 0.800, and 0.520 to 0.720, respectively, for the three strings. When assessing the quality of these equation-strings, note that the true, noiseless series typically result in R^2-values of not more than approximately 0.720 to 0.840 when used as a predictor for out-of-sample noisy data.

Let us now turn to examples with data drawn from economics. The series that are used are available on the Internet, whence they were downloaded. [gopher: //una.hh.lib.umich.edu: 70/11/ebb/indicators/BCIH]. As emphasized in the introduction, when applying genetic algorithms, the correct procedure first lets the algorithm find an equation-string that fits the training data, and then checks this string's performance on out-of-sample data. However, this procedure is actually a compound test, since it verifies (a) whether genetic algorithms produce good results, and (b) whether the equation-strings that are bred are appropriate theoretical models for the underlying data. Since this paper's purpose is to show whether, and how well, the genetic algorithms perform, we limit ourselves to the first part of the test. Determining whether models that are bred are appropriate representations of the data-generating models (and whether they allow forecasts) is left to researchers analyzing the specific data-set. Hence, we make no claim that the equation-strings in the following examples represent anything but spurious correlation. We do claim, however, that the algorithm finds equations that may fit the data better than, say, OLS regressions.

We first analyze monthly US figures for unemployment (U, expressed in thousands) for the period January 1968 to October 1995 as a function of the prime rate (PR, in percent), the money supply ($M1$, billions of 1987-dollars), the smoothed change in manufacturer's unfilled orders (UO, billions of 1987-dollars), the capital utilization rate in manufacturing (CU, in percent), the working age population (WA, measured in 100,000s), and the index of net business formation $(BF, 1967 = 100)$. First differences are taken of all series and the 333 observations are fed into the genetic algorithm. We limit our investigation to the first lags of the series. As a benchmark, we present the results of an OLS regression of current unemployment (U_0) on the first lags of the other series

$$\begin{aligned} U_0 = 539 &- \underset{[-1.97]}{5.79PR_{-1}} - \underset{[-0.39]}{7.53M1_{-1}} - \underset{[-5.33]}{82.79UO_{-1}} + \underset{[0.86]}{5.79CU_{-1}} \\ &- \underset{[-3.19]}{20.01WA_{-1}} - \underset{[-0.71]}{13.56BF_{-1}}. \end{aligned} \tag{11}$$

(The t-statistics are given in square brackets.) The regression results in an R^2 of 0.168. The genetic algorithm – also restricted to first lags – gives, in two typical examples, the following equation-strings

$$\begin{aligned} \mathrm{A0} = {} & (((6.0^*\mathrm{D1})^*((-7.5 + -2.9) + (6.0^*\mathrm{D1}))) \\ & - (\mathrm{G1} - ((-7.4^*\mathrm{F1}) + (((\mathrm{B1}^*(-4.5 + (-9.5^*\mathrm{G1}))) \\ & + (-9.8^*\mathrm{F1})) - 6.0)))), \end{aligned}$$

$$\begin{aligned} \mathrm{A0} = {} & ((\mathrm{D1} - 6.2)^*(((\mathrm{B1}^*\mathrm{G1}) - ((\mathrm{E1} - 6.2)^*\mathrm{F1})) \\ & - (((\mathrm{D1} - 5.1) - 6.2)^*\mathrm{D1}))), \end{aligned}$$

which in edited form are

$$\begin{aligned} U_0 &= -6.0 - 4.5PR_{-1} - 56.4UO_{-1} - 17.2WA_{-1} \\ &\quad -BF_{-1} - 9.5PR_{-1}BF_{-1}, \end{aligned} \tag{12}$$

and

$$\begin{aligned} U_0 &= -70.06UO_{-1} + 6.2UO^2_{-1} - UO^3_{-1} - 38.44WA_{-1} \\ &\quad +PR_{-1}UO_{-1}BF_{-1} - UO_{-1}CU_{-1}WA_{-1} \\ &\quad +6.2(UO_{-1}WA_{-1} - PR_{-1}BF_{-} + CU_{-1}WA_{-1}). \end{aligned} \tag{13}$$

These equations result in R^2-values for the data-set of 0.227 and 0.217, respectively. The genetic algorithm finds strings that increase the explained variance by 35 and by 29 percent, relative to the regression results. It is noteworthy that the coefficients of UO_{-1} in the strings $(-56.4$ and $-70.06)$ are bred despite the fact the value of the numerical entries is limited to ± 10.0 and that in both runs the algorithm ignores $M1_{-1}$, whose t-values in the OLS regression also show a very low influence on U_0.

As a second example we investigate monthly observations on personal income less transfer payments (PI, billions of 1987 dollars) and money supply ($M2$, billions of 1987 dollars) for the period January 1968 to September 1995. The series are highly collinear, and we again feed the algorithm with the first differences. We check the explanatory power, if any, of the first four lags of $M2$ on PI. An OLS regression between PI_t and $M2_{t-j}$ $(j = -1 \textit{ to } -4)$ reveals no explanatory power, the goodness of fit being close to zero $(R^2 = 0.017)$. Conversely, in three typical runs, the genetic algorithm finds the strings

$$\begin{aligned} \text{A0} &= ((3.7 + (-8.4/(\text{B3} + (\text{B4}^*\text{B2})))) \\ &\quad + (((\text{B1} - (4.2 + (9.5/((\text{B2}^*\text{B4}) + \text{B3}))))/ \\ &\quad ((\text{B2} + 3.7) + (\text{B4}^*\text{B2})))/(\text{B1} + \text{B4}))), \end{aligned}$$

$$\begin{aligned} \text{A0} &= (((((\text{B2} + \text{B4})/((\text{B2}/1.4) + (\text{B4}^* - 7.7))) \\ &\quad + ((\text{B4}^* - 9.8) + B1)) + -7.0)/ \\ &\quad ((\text{B2}/1.4) + (\text{B4}^* - 7.7))), \end{aligned}$$

$$\begin{aligned} \text{A0} &= (((((-4.6/(\text{B3} + (\text{B2}^*(((6.6 + (\text{B3} + (\text{B4} + -6.3)))/ \\ &\quad - 2.4) + ((\text{B2} + \text{B2})/(\text{B3} + (\text{B4}^* - 9.9))))))) \\ &\quad + ((-1.0 + \text{B2})/(\text{B3} + (\text{B4}^* - 9.9)))) + \text{B1})/ \\ &\quad (\text{B3} + (\text{B4}^*\text{B2}))) + 6.6), \end{aligned}$$

with R^2-values of 0.551, 0.302, and 0.547, respectively! The edited versions of these rather complicated strings are

$$PI_0 = 3.7 - \frac{8.4}{M2_{-3} + M2_{-2}M2_{-4}} + \frac{M2_{-1} - 4.2 + \dfrac{9.5}{M2_{-3} + M2_{-2}M2_{-4}}}{\dfrac{3.7 + M2_{-2} + M2_{-2}M2_{-4}}{M2_{-1} + M2_4}}, \tag{14}$$

and

$$PI_0 = \frac{-7.0 + M2_{-1} - 9.8M2_{-4} + \dfrac{M2_{-2} + M2_{-4}}{0.714M2_{-2} - 7.7M2_{-4}}}{0.714M2_{-2} - 7.7M2_{-4}}, \tag{15}$$

and

$$PI_0 = \frac{6.6 + M2_{-1} + \dfrac{M2_{-2} - 1.0}{M2_{-3} - 9.9M2_{-4}} - \dfrac{4.6}{M2_{-3} - M2_{-2}(0.125 + 0.417M2_{-3} + 0.417M2_{-4}) + \dfrac{2M2^2_{-2}}{M2_{-3} - 9.9M2_{-4}}}}{M2_{-3} + M2_{-2}M2_{-4}} \tag{16}$$

(Recall, we are showing here only that the algorithm can find strings that – in evolutionary terms – are ‘fit’. No claim is made that the above formulas ‘explain’ the variance in personal income).

As a final example we apply the genetic algorithm to cross-sectional data. We investigate the relationship between the performance of NBA players and their salaries during the 1994–95 playing season. The data are taken from the WizWhy® demonstration diskette (WizSoft®, Framingham, MA, and Tel Aviv, Israel) and contain information on 248 players: salary (S, in millions of dollars), number of games (N), field goals (G), successful attempts at field goals (G%, in percent), free throws (T), successful attempts at free throws (T%, in percent), rebounds (R), and assists (A). After deleting the players with incomplete entries, we have data on 222 players, which are randomly divided into two groups of 111 players each. The first group serves as the training data for the genetic algorithm, and the second is used as an out-of-sample test set. An OLS regression on the first group results in an R^2-value of 0.548 (t-statistics in square brackets)

$$\begin{array}{llllll} S = 0.512 & - \; 0.020(N) & + \; 0.003(G) & + \; 0.098(G\%) & + \; 0.01(T) \\ & \quad [-3.28] & \quad [3.06] & \quad [0.08] & \quad [0.82] \\ & + \; 0.010(T\%) & + \; 0.001(R) & - \; 0.00002(A) & \\ & \quad [1.43] & \quad [2.45] & \quad [-0.03] & \end{array} \tag{17}$$

Applying this equation to the out-of-sample group gives a goodness of fit of 0.256.

The data of the first group is given to the genetic algorithm and, in one example, the following string is bred

$$\mathrm{A0} = (0.5 + (\mathrm{C0}/((4.7^*\mathrm{B0}) + (((\mathrm{G0} + 4.7) + \mathrm{E0})/ - 6.6)))),$$

whose edited version is

$$S = 0.5 + 6.6 \frac{G}{31.02N - 4.7 - T - R}, \tag{18}$$

with an R^2-value of 0.585. Hence for the in-sample data, the goodness of fit increased by about seven percent. However, the real power of the genetic algorithm becomes apparent when this string is used to predict the salaries of the out-of-sample players. Applying the above equation to the 111 players of the second group results in an R^2-value of 0.328. In comparison to the OLS-predictions, the genetic algorithm increases the explained variance by 28 percent. It is noteworthy that the algorithm in effect finds that the average number of field goals per game (G/N) is a useful predictor for the player's salary, while the OLS-regression linearizes this expression, by attaching a negative sign to the parameter of N.

Additional examples of equation-strings bred by genetic algorithms, for both clean and noisy series, as well as for the well-known sunspot series, can be found in (Szpiro, 1996).

4. Conclusions

Here we present an algorithm that searches for the structure of a data-generating process in symbolic form. We consider the equations that underlie such processes to be strings of symbols conforming to a simple grammar. The algorithm, modeled after genetic theories and Darwinian concepts, is fed the data and creates formulas by combining and discarding blocks of equations according to their usefulness. Examples show that genetic algorithms succeed surprisingly well with various univariate and multivariate time-series, as well as with cross-sectional data, even when noise is present. It must be emphasized, however, that the equation-strings that are found need not represent the actual data-generating process but could simply mimic spurious relationships between the variables. In real-world applications the equations that are bred by the genetic algorithm must be tested with out-of-sample data.

Acknowledgements

I would like to thank the editor of this Journal, Dave Belsley, for valuable comments, and Benny Csacskes for help with the computer code.

References

Allen, F. and Karjalainen, R. (1993). *Using genetic algorithms to find technical trading rules*, working paper, Rodney L. White Center for Financial Research, The Wharton School of the University of Pennsylvania.

Brock, W.A., Lakonishok, J. and LeBaron, B. (1992). Simple technical trading rules and the stochastic properties of stock returns, *J. of Finance*, **47**, 1731–1764.

Farmer, J.D. and Sidorovich, J.J. (1987). Predicting Chaotic Time Series, *Phys. Rev. Lett.*, **59**, 845–848.

Goldberg, D.E. (1989). *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley, Reading, MA.

Hénon, M. (1976). ‘A two-dimensional mapping with a strange attractor’, *Comm. Math. Phys.* **50**, 69–77.

Holland, J.H. (1975). *Adaptation in natural and artificial systems*, University of Michigan Press, Ann Arbor. 2nd ed 1992, MIT Press.

Hsieh, D.A. (1991). ‘Chaos and nonlinear dynamics: Applications to financial markets’, *J. of Finance*, **46**, 1839–1877.

Koza, J.R. (1992). *Genetic programming*, MIT Press, Cambridge.

LeBaron, B. (1992). ‘Nonlinear forecasts for the S&P stock index’, in Casdagli and Eubanks (eds), *Nonlinear modeling and forecasting*, Santa Fe Institute, Addison-Wesley, Reading, Mass.

LeBaron, B. (1993). ‘Nonlinear diagnostics and simple trading rules for high-frequency foreign exchange rates’, in Weigend and Gerschenfeld (eds.), *Time series prediction: Forecasting the future and understanding the past*, Santa Fe Institute, Addison-Wesley, Reading, Mass.

Palmer, R.G., Arthur, W.B., Holland, J.H., LeBaron, B. and Taylor, P. (1994). ‘Artificial economic life: a simple model of a stockmarket’, *Physica D*, **75**, 264.

Scheinkman, J.A. and LeBaron, B. (1989). ‘Nonlinear dynamics and stock returns’, *J. of Business*, **62**, 311–338.

Szpiro, G.G. (1997). ‘Forecasting chaotic time series with genetic algorithms’, *Phys. Rev. E*, **55**, 2557–2568.