Optimizing Trading Strategies through Genetic Programming

N.G. Pavlidis[1], E.G. Pavlidis[2], M.N. Vrahatis[1]

[1]Computational Intelligence Laboratory, Department of Mathematics, University of Patras Artificial Intelligence Research Center (UPAIRC), University of Patras, GR-26110 Patras, Greece,

[2]Department of Economics, Lancaster University Management School, Lancaster LA1 4YX, UK

npav@math.upatras.gr, e.pavlidis@lancaster.ac.uk, vrahatis@math.upatras.gr

Abstract: This paper investigates the performance of genetically programmed trading rules applied to the daily Euro US Dollar exchange rate. Our findings suggest that significant profit opportunities exist especially during the period of the sharp depreciation of the dollar. However, the performance of these rules deteriorates as the time horizon increases. The application of the residual based and wild bootstraps indicates that genetic programming can identify patterns in the data which cannot be explained by standard statistical models.

Introduction

Since the inception of floating exchange rates in the early 1970s, economists have attempted to predict the movements of exchange rates. While exchange rate determination models based on macroeconomic fundamentals appear to explain the fluctuations of major exchange rates in the long run, and in economies experiencing hyperinflation, the empirical literature suggests that their performance is poor over shorter horizons. These models also fail in out-of-sample forecasting being unable to beat a naive random walk forecast (Meese and Rogoff, 1983). On the other hand, Technical Analysis (TA) focuses on the identification of price patterns and trends, as well as, the use of mechanical rules to generate valuable economic signals (see Sullivan *et al.* (1999) for a thorough description of a number of simple trading rules). Recent surveys (Cheung and Chinn, 2001) suggest that TA has been a major constituent of financial practice in foreign exchange markets. Moreover, a number of empirical and theoretical studies (e.g. Le Baron, 1999; De Grauwe and Grimaldi, 2006) during the last three decades suggest that the application of TA in the foreign exchange market can yield substantial profits. These findings raise doubts on the validity of the efficient market hypothesis. Olson (2004), however, argues that abnormal profit opportunities arise due to temporary inefficiencies which are in accordance with an evolving market. He further argues that the returns of simple trading rules over recent periods have declined, if not completely disappeared.

In this study, instead of examining the performance of a set of simple rules, we exploit the ability of Genetic Programming (GP) to identify profitable trading strategies. GP is an extension of Genetic Algorithms (GAs) that explores a space of computer programs to identify the one that addresses a given task most effectively (Koza, 1992). Like GAs, GP employs genetic operators inspired from Darwinian evolution in order to evolve a

population of candidate solutions towards more promising regions of the search scape. The advantages of this approach for the identification of trading strategies are twofold. Firstly, GP allows the construction of rules of arbitrary complexity and, therefore, it can be argued that it resembles the behavior of an optimizing market agent more adequately than a simple trading rule. Secondly, by avoiding the ex post specification of the trading rules, it circumvents a basic, but rarely addressed issue in the literature, namely *data-snooping* (Sullivan *et al.*, 1999). Neely *et al.* (1997) use GP to identify profitable trading rules in several daily foreign exchange rate series over a long span of data. Their results suggest that trading rules obtained through GP substantially outperform simple trading rules.

We employ GP for the daily exchange rate of the Euro against the US Dollar over the period 1/1/1999 to 30/12/2005. Our findings support the view that GP strategies can result in substantial profits. Moreover, the application of the standard residual based and wild bootstraps indicates that the ability of GP to generate profits cannot be explained by either a random walk, or an autoregressive model. However, the performance of the GP-identified trading rules deteriorates as they are applied to data further into the future which raises doubts about the ability of the rules to generate profits in the long run.

Genetic Programming

GP is an extension of Genetic Algorithms (GAs) in which the potential solutions, called *individuals*, are computer programs, rather than fixed length strings. Individuals in GP are expressed as *syntax trees.* The internal nodes of a syntax tree are *function nodes*, while the leaves constitute the *terminal nodes* (Koza, 1992). A function node applies a function to the outputs of its children. Terminal nodes, on the other hand, return as output the value of a constant, an input variable, or a zero-argument function. Examples of GP individuals are illustrated in Figs. 2 and 3.

Like GAs, GP is an evolutionary search algorithm that employs a set of individuals, named *population*, and updates them over consecutive iterations, called *generations*, using genetic operations. The central GP operators are *selection*, *crossover*, and *mutation*. A flowchart of the operation of GP is provided in the Figure 1. All the selection schemes proposed in the literature on GAs are also applicable to GP. At present we employ *proportionate selection*. According to proportionate selection, the probability, p_i, of selecting individual *i* of the current generation to constitute a parent for an individual of the next generation (*offspring*) is equal to:

$$p_i = E_i \Big/ \sum_{j=1}^{N} E_j ,$$

where *E* is the function that is to be maximized, and E_i is the function value that corresponds to the *i*-th individual. The reproduction operator inserts a copy of the parent individual to the population of the next generation.

The primary GP search operator is crossover. Crossover operates on two parent individuals and yields two offsprings. Standard crossover randomly selects a node in each parent tree and then swaps the subtrees rooted at these nodes (Koza, 1992). Koza suggests to use a 90% probability of selecting as crossover point a function node and to select a terminal node with probability 10%.Standard crossover tends to produce offsprings that frequently inherit most of their code from one parent, and also favors local adjustments near the leaves of syntax trees (Poli and Langdon,1998). To overcome

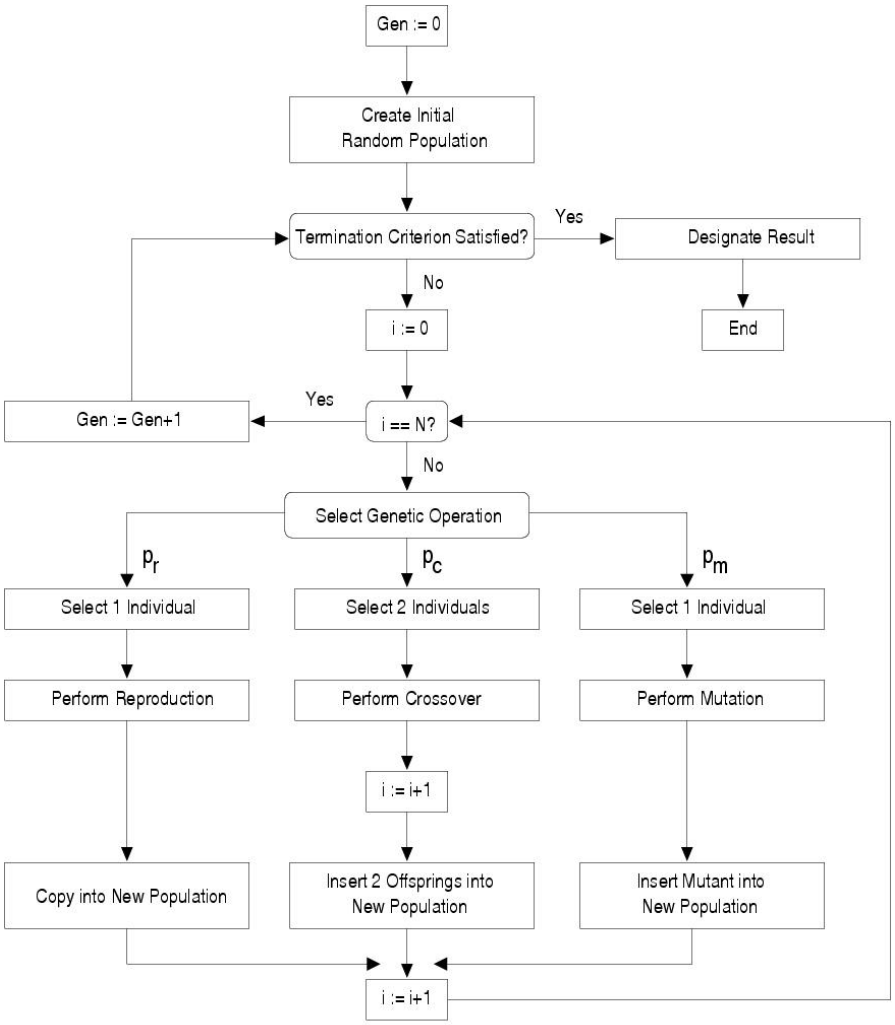

Figure 1: Flowchart of the operation of Genetic Programming

these limitations, Poli and Langdon proposed the *uniform crossover operator* (GPUX), inspired from the homonymous operator in GAs. At early stages of the algorithm GPUX favors global search by swapping large subtrees near the root of the syntax trees. As the population converges the operator becomes more and more local, in the sense that the offsprings it produces are progressively more similar to their parents. GPUX starts by identifying the common region between two syntax trees. Each node that lies in the common region is considered for crossover with a constant probability. For nodes that lie in the interior of the common region GPUX swaps the nodes without affecting the subtrees rooted at these nodes. On the contrary, for nodes on the boundary of the common region the subtrees rooted at these nodes are swapped. An example of GPUX is provided in Fig. 2. Finally, subtree mutation operates on a single parent individual by substituting a randomly selected subtree with a new one. Fig. 3 illustrates the workings of uniform crossover and subtree mutation.

A critical component of GP is the random-tree creation algorithm(s) it employs. The standard GP initialization technique, *ramped half and half*, relies on two tree-generation algorithms, *GROW* and its full-tree variant, *FULL* (Koza, 1992). GROW chooses randomly between function nodes and terminal nodes as long as the depth of the syntax

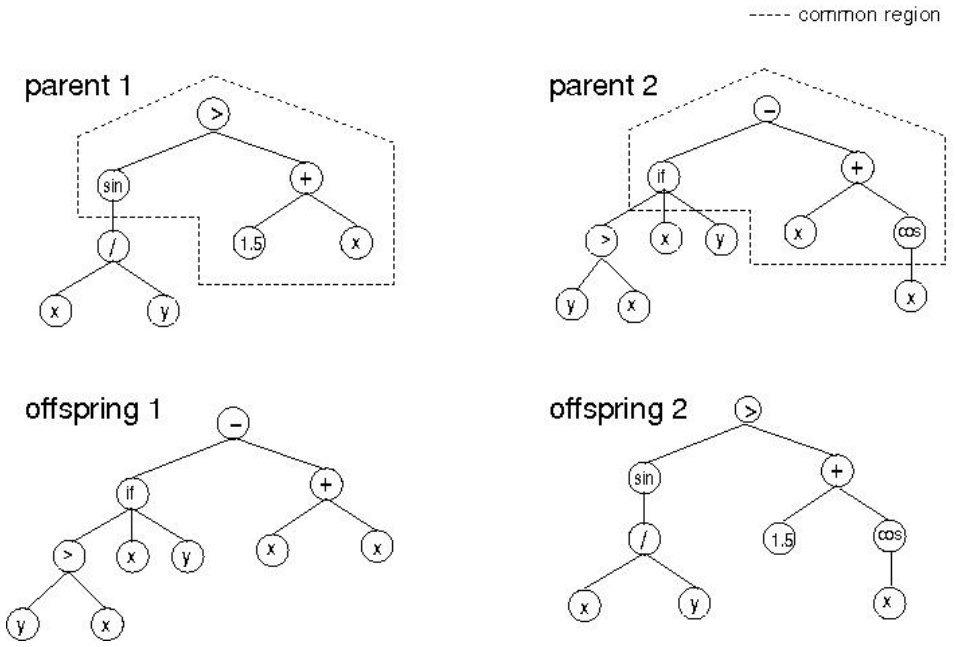

Figure 2: Genetic programming uniform crossover

tree along the current branch is smaller than the maximum depth threshold, *D*. If *D* is reached, a terminal node is always inserted. FULL can be seen as a full-tree variant of GROW that always inserts a function node if the depth of the branch is smaller than *D*. GROW is by far the most commonly employed tree generation algorithm, not only for the initialization of the population, but also for subtree mutation (Luke, 2000). A critical shortcoming of GROW is that depending on the size of function set, *F*, and the terminal set, *T*, the expected size of the trees can become infinite. Alternatively, GROW can produce a large number of small trees. To promote the diversity of individuals in the population the PTC2 algorithm (Luke, 2000) was presently employed.

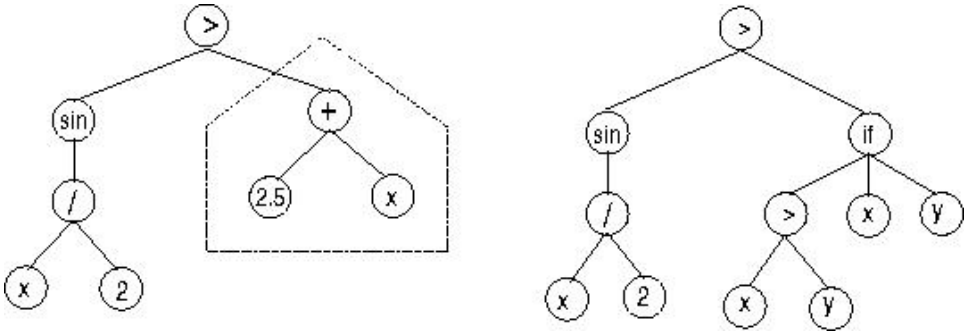

Figure 3: Subtree mutation

Bootstrapping Methodology

To test whether the performance of the GP rules is due to standard statistical properties of the data we conduct three bootstrap exercises. The suitability of this approach in the present context, lies in the fact that bootstrapping utilizes the empirical distribution function of the data and, therefore, it addresses a number of stylized facts concerning exchange rate returns, such as conditional heteroskedasticity and leptokurtosis. We

choose two potential candidates for the Data Generating Process (DGP), a random walk and an autoregressive (AR) model. The random walk model enables us to test if the GP profits are due to chance. The use of an AR model is based on the fact that any linear invertible time series process can be approximated by an AR(∞) process. Therefore, it can shed light on whether GP trading strategies exploit the linear dependence of exchange rate returns over short periods of time. For each of the three bootstrap exercises we simulate 500 data series so as to test the null hypotheses that the generated returns can be explained by these two models. The *p*-values for these tests are defined as the number of times that the GP rule achieves returns on the simulated series that are higher from the ones on the original data set.

The simulation of the random walk model was based on the nonparametric procedure, firstly proposed by Efron (1982), called *resampling*. This procedure amounts to drawing each observation of a bootstrap sample randomly and with replacement from the observed return series. In this case, returns are independent and identically distributed by construction, while the simulated price series follow a random walk with the same drift as the original series.

For the second model we employ two bootstrap approaches, the standard residual based and the fixed-design wild bootstrap. Both of these approaches use the residuals and the underlying fitted parameters of the same AR(4) model to generate new simulated data sets. The order of the model was chosen by the Akaike Information Criterion. The residual based bootstrap generates samples recursively from the equation:

$$r_t^* = \widehat{\alpha} + \sum_{p=1}^{4} \widehat{\beta}_i r_{t-i}^* + \widehat{\varepsilon}_t^*,$$

where, $\widehat{\varepsilon}_t^*$ are the resampled and centered residuals from the original model. A major drawback of this approach is that it doesn't take into account the conditional heteroskedasticity as it treats the regression error as independently and identically distributed. Gonçalves and Kilian (2004) establish the asymptotic validity of the fixed-design WB for stationary autoregressions with known finite lag order when the error term exhibits conditional heteroskedasticity of unknown form. Their results cover as special cases the N-GARCH, t-GARCH and asymmetric GARCH models, as well as, stochastic volatility models. The fixed-design wildbootstrap DGP takes the form:

$$r_t^* = \widehat{\alpha} + \sum_{p=1}^{4} \widehat{\beta}_i r_{t-i} + \widehat{\varepsilon}_t \eta_t,$$

where, $\widehat{\varepsilon}_t$ are the residuals from the AR(4) model and η_t is a random variable which follows the Rademacher distribution *F*. That is, η_t takes the value of 1 with probability *p*=0.5, and -1 with probability 1-*p.*

Experimental Results and Discussion

The dataset presently employed is the daily closing prices for the Euro against the US Dollar exchange rate from Barclays Bank International provided by Datastream. The 1825 observations cover the period from January 4th 1999 to December 30th 2005. The data are normalized by dividing each observation with the 250-day moving average. Each imput pattern contains the 50 previous normalized observations. Therefore, after normalization and embedding 1525 patterns were available. The first 500 patterns were

assigned to the training set, while the remaining 1025 were assigned to the test set. The time series is plotted in Fig. 4.

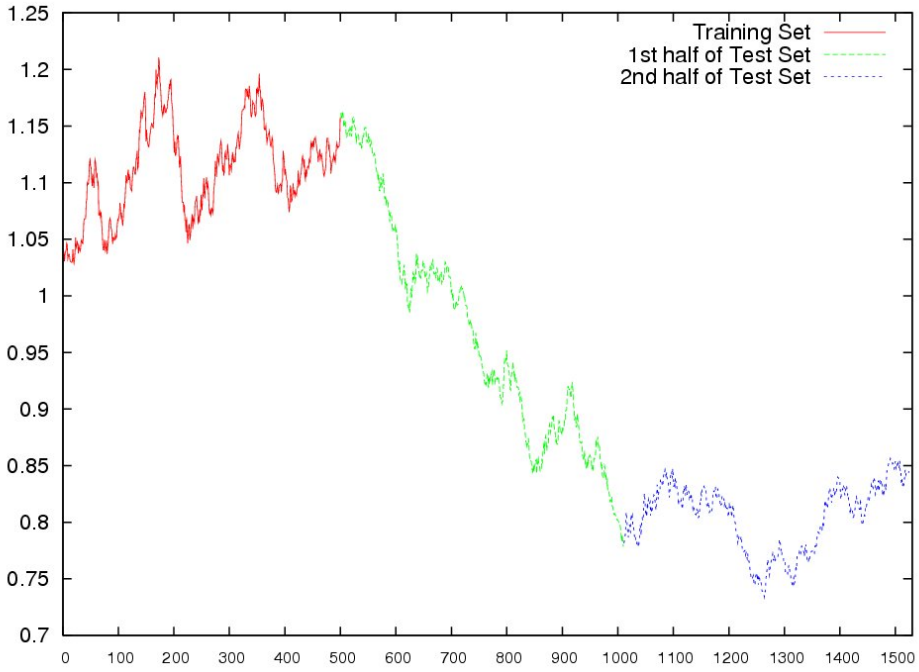

Figure 4: Time series of the Euro/US Dollar.

Computational experiments were performed using an object-oriented GP implementation based on the C++ interface for the creation of expressions described in (Koenig and Moo, Chap. 8, 1996) and the GNU compiler collection (gcc) version 4.0.3. The terminal set, *T*, consisted of,

$$T = \{x_t^n, x_{t+1}^n, \ldots, x_{t+50}^n, rand\},$$

where, x_t^n stands for the normalized exchange rate at date *t*, and `rand` denotes a random real constant in the interval [-1,1]. The function set, *F*, contained the following functions:

- Ternary functions: if then else,
- Binary functions: $+, -, *, /, >, \geq, <, \leq,$ and, if, or, xor, equal,
- Unary functions: sin, cos, exp, ln, sqrt, max, min, average.

A positive evaluation of an individual over a pattern is assumed to signal that the current holdings should be held in the base currency (in this case US Dollars), and vice versa. In particular, if the system at date *t*, holds US Dollars and the evaluation of the individual over the corresponding input pattern is positive then all the available funds are converted to Euros. On the contrary, if the system holds Euros and the individual evaluation is non-positive, then the amount is converted to US Dollars. In all other cases, the holdings do not change currency at date *t*. The last observation of the series is always employed to convert the final holdings to the base currency. A one-way transaction cost equal to 0.5% and 0.05% for the training, and test periods, respectively, was used. The choice of 0.05% is in accordance with the cost faced by large institutional investors. A larger transactions cost was imposed during training to penalize rules that trade very frequently

(Neely *et al.*, 1997). The fitness function returns the overall rate of return over the dataset.

Regarding the parameters of the GP algorithm, the maximum tree depth, *D*, at initialization was set to 5, while in subsequent generations *D* was equal to 10. The maximum number of nodes in any syntax tree was 100. Population size was 1000 and the maximum number of generations was 1000. The reproduction, mutation, and crossover probabilities were $p_r = 0.1$, $p_m = 0.3$, and $p_c = 0.6$, respectively. Finally, the probability of performing uniform crossover at each node of the common region was 0.5. The stopping criterion for the algorithm was to reach the maximum number of generations. The GP output was the individual with the highest fitness encountered during its execution.

A total of 150 experiments were performed using the aforementioned configuration. In all cases, highly profitable rules were identified in the training set. Despite the fact that the performance of the majority of these rules on the test set was poor, the best rule yield an averaged return of 5% per annum by making a total of 43 transactions over the 1025 observations comprising the test set. The bootstrap results indicate that the returns of this rule are not consistent with the random walk and the AR benchmarks. The *p*-value corresponding to the null hypothesis that the generated returns can be explained by the random walk is 0.07. For the AR(4) the GP-identified rule achieved higher returns on the simulated series than on the observed data set only in eight and six percent of the cases when the residual based and the fixed design wild bootstrap were used as DGPs, respectively.

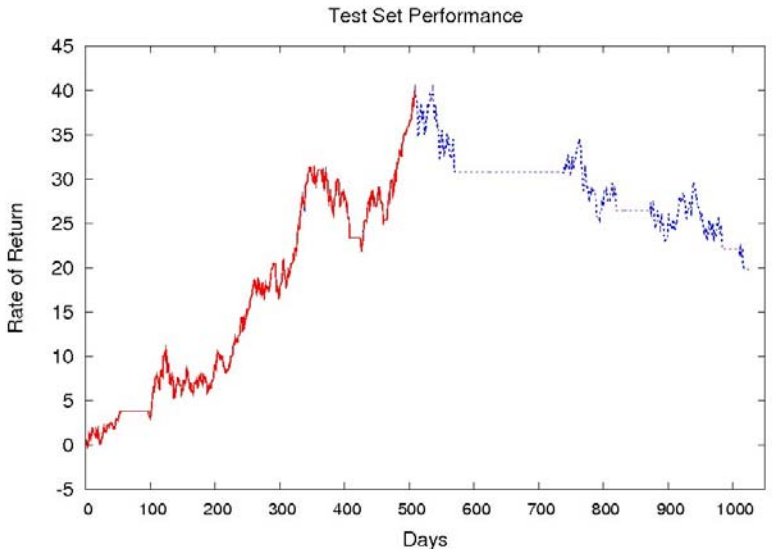

Figure 5: Overall rate of return of the best performing trading rule on the test set (solid red: first 510 days; dashed blue line: second part of the test set, 515 days)

To investigate the impact of the time horizon on generalization, the test set was segmented into two sets, with 510 and 515 patterns respectively. The estimated average annual return of the rule was 20.4% and -7.17% respectively. During the first part of the test set, the rule performs 25 trades, while in the latter part 19. Clearly the performance of the trading strategy is remarkable in the first period, but it deteriorates drastically in the second period yielding substantial losses. The evolution of the overall rate of return

obtained through this rule with respect to time is illustrated in Fig. 5. In this figure, the rate of return on the first half of the test set is depicted with a solid red line, while the dashed blue line shows the overall rate of return over last 515 days of the test set. A similar behavior was exhibited by several trading rules that GP produced.

An interpretation of these findings is provided by models in which heterogeneous agents use different forecasting techniques (Grimaldi and De Grauwe, 2003). According to these models as the exchange rate moves steadily in one direction, the use of extrapolative forecasting rules (chartism) becomes more profitable, thereby attracting more technical analysts in the market. The rise in the number of technical analysts reinforces the movement of the exchange rate in the same direction and therefore increases the profits of this type of traders. Thus, self-fulfiling dynamics in the profitability of TA arise. Grimaldi and De Grauwe (2003) argue that the limit of this dynamics is reached when almost everybody in the foreign exchange market has become a chartist. At this point, the self-reinforcing movement in the exchange rate and in profitability slows down, increasing the expected relative profitability of fundamentalists. This is so because as the bubble develops, the expected profits from fundamentalism increase. However, these are overwhelmed by the self-fulfilling profitability of chartism. When the latter tends to slow down, fundamentalism becomes attractive again. A small movement of the exchange rate can then trigger a fast decline in the share of chartism, back to its normal level of a tranquil market. As can be seen from Fig. 4, the US dollar undergoes a sharp depreciation during the first half of the test set. During this period the profitability of the trading rule is extremely high. Notice that this is not due to a buy and hold approach as the rule performs 25 trades in this period. As soon as the depreciation of the currency stops, the trading rule's profitability becomes negative.

Conclusions

In this work we investigate the performance of genetically programmed trading rules applied to the Euro US Dollar daily exchange rate. It can be argued that genetic programming firstly, resembles the behavior of an optimizing market agent more closely than a simple trading rule; and secondly, by avoiding the ex post specification of the trading rules, it circumvents the issue of data-snooping. The obtained experimental results suggest that such genetically programmed rules have the potential to identify significant profit opportunities especially during the period of the sharp depreciation of the dollar. The application of the residual based and wild bootstraps indicates that the considered rule can detect patterns in the data which cannot be explained by standard statistical models. The performance of the identified rules, however, deteriorates as the time horizon increases. This phenomenon is also encountered in theoretical models of the foreign exchange market in which heterogeneous agents use different forecasting techniques.

References

Y.-W. Cheung and M. D. Chinn, (2001). Currency traders and exchange rate dynamics: a survey of the US market, *Journal of International Money and Finance,* **20**(4), pp. 439-471.

B. Efron, (1982). *The jackknife, the bootstrap and other resampling*, SIAM, Philadelphia, PA.

S. Gonçalves, and L. Kilian, (2004). Bootstrapping autoregressions with conditional

heteroskedasticity of unknown form, *Journal of Econometrics* **123**(1), pp. 89-120.
P. De Grauwe and M. Grimaldi, (2006). Exchange rate puzzles. A tale of switching attractors, *European Economic Review*, **50**(1), pp. 1-33.
M. Grimaldi and P. De Grauwe , (2003). Bubbling and crashing exchange rates, Katholieke Universiteit Leuven, vol. CESifo 1045.
A. Koenig and B. Moo, (1996). *Ruminations on C++: A Decade of Programming Insight and Experience*. Addison-Wesley.
J. R. Koza, (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Press, Cambridge, MA, USA.
B. Le Baron, (1999). Technical trading rule profitability and foreign exchange intervention, *Journal of International Economics*, **49**(1), pp. 125-143.
S. Luke, (2000). Two fast tree-creation algorithms for genetic programming, *IEEE Transactions on Evolutionary Computation*, **4**(3), pp. 274-283.
R. A. Meese and K. Rogoff, (1983). The out-of-sample failure of empirical exchange rate models: Sampling error or misspecification? (Editor: J. Frenkel), Exchange Rates and International Macroeconomics, Chicago: University of Chicago Press, pp. 67-112.
C. J. Neely, P. A. Weller, and R. Dittmar, (1997). Is technical analysis in the foreign exchange market profitable? A genetic programming approach, *Journal of Financial and Quantitative Analysis*, **32**(4), pp. 405-426.
D. Olson, (2004). Have trading rule profits in the currency market declined over time? *Journal of Banking and Finance,* **28**(4), pp. 85-105.
R. Poli and W. B. Langdon, (1998). On the search properties of different crossover operators in genetic programming, (Editors: J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, R. Riolo), *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pp. 293-301.
R. Sullivan, A. Timmermann, and H. White, (1999). Data-snooping, technical trading rule performance, and the bootstrap, *Journal of Finance* **54**(5), pp. 1647-1691.