

VIENNA UNIVERSITY OF TECHNOLOGY

BACHELOR THESIS

How do naturally inspired search algorithms help to find the optimal trading strategy?

Oliver Wesely
(1325558)

Bachelor programme Statistics and Mathematics in Economics
(033203)

supervised by
Ao.Univ.Prof. Dipl.-Ing. Mag.rer.nat. Dr.techn. Wolfgang Scherrer

co-supervised by
em.o.Univ.Prof. Dr. Wolfgang Janko
Dipl.-Ing.Mag. Riccardo Gismondi

March 31, 2017

Abstract

Lots of studies have showed that in the context of financial markets, technical analysis is a very useful tool for forecasting trends. Looking at financial literature, many different rules exist to make buy and sell decisions on a daily basis. The simplest ones are moving average rules which are often used with traditional moving average lengths, but these are not always the optimal input parameters. In order to improve their performance, we optimized the inputs of the trading strategy with the Genetic Algorithm (GA) and showed how changes in the GA design affect the result. Looking at the multitude of different rules, we realized that there is still no optimal one, so in the second part we tried to optimize the trading strategy itself with Grammatical Evolution (GE). Beside these two experiments we generally described the theoretical background of the GA and GE and showed each performance in one example.

1 Introduction

Found in the literature, the main types of search methods are: calculus-based, enumerative and random. Calculus-based methods can be subdivided into two main search class methods: indirect and direct. Indirect methods use the gradient method by setting the gradient of the objective function equal to zero while direct methods use hill climbing to find the local optima. But both methods have the big disadvantage that they are local in scope and depend on the existence of derivatives. Enumerative schemes focus on looking at objective function values of every point in the space, but many practical spaces are simply too large to search one at a time. The remaining method is the one of random search algorithms with an increasing popularity due to its performance and robustness on optimizing complex problems. These random search algorithms include the GA and GE which are heuristic algorithms and do not guarantee a globally optimal solution, yet get pretty close to one. [Goldberg, 1989]

The aim of the work described in this paper is to investigate how genetic algorithms and grammatical evolution can be used to optimize the parameters of a given trading strategy or the strategy itself and show how changes in the design of GA and GE can affect the quality of the solution.

This paper is structured in the following way: in section 2 the GA is presented and two examples of its performance and robustness including the optimiza-

tion of the MA trading rule are given. In section 3 the GE is presented and two application examples are shown: the first compares the multiple regression model with the GE and the second one optimizes the trading rule for a given time series. The conclusion and some future work can be found in section 4 and the MATLAB code in appendix A.

The complete analysis in this paper has been made with MATLAB R2016b academic version computed on the CPU of an ALIENWARE laptop with Intel(R) Core (TM) i7, 2.9 GHz, 32 GB RAM, 64-Bit.

2 Genetic Algorithm

The Genetic Algorithm (GA) belongs to the family of algorithms known as Evolutionary Computation (EC). The idea behind these algorithms is inspired by Darwin's theory of evolution by natural selection. GAs are heuristic algorithms, which implies that they don't discover the 'best' solution but want to find a close to the optimal one, and are designed to search linear functions or even highly nonlinear spaces for global optima.

To find a solution for complex problems the GA works by modifying an initial population, which includes randomly generated potential solutions to the problem of interest. Traditionally each potential solution is encoded as a binary string and to measure the quality of any particular solution a problem-specific fitness function is used. Over a fixed number of generations the GA improves the fitness value of the current population by performing selection, crossover and mutation. The final solution of the GA will be the best one of the last population. [Brabazon and O'Neill, 2006, Mahfoud and Mani, 1996]

2.1 Genotype Encoding

The genotype represents all the information stored in the chromosome while the phenotype describes the outward appearance of an individual. In order to measure the utility of a specific genotype, a genotype-phenotype mapping or encoding is needed. The simplest method for binary encoding is to convert the binary string into an integer value (Table 1). A binary genotype of length 8 for example can encode integer values from 0 to $2^8 - 1 = 255$. [Brabazon and O'Neill, 2006, Rothlauf, 2006]

Table 1: Binary Encoding examples

Candidate	Binary Code	Integer
A	00000111	7
B	00110010	50
C	11100010	226
D	01011100	92

2.2 Selection

After measuring the fitness for each phenotype using a fitness function, the selection operator tries to improve the population by mating only structures with higher fitness and delete the others. The selection method we are using is the random tournament selection, where two different chromosomes are chosen randomly, depending on the selection rate, and are going to be mated. Exemplary we are looking at the example before (Table 1) and use the sinus function applied on the integer value as the fitness of a chromosome. With the random tournament selection and a selection rate of 0.25, which means 25% of the population of the worse fitness measures will be newly generated by mating with 2 of the rest of the population. Therefore in the example the chromosomes are sorted due to their fitness measure and two parents will be randomly selected. In our example the randomly selected parents are candidates A and B out of A,B and C. (Table 2) [Brabazon and O'Neill, 2006, Mahfoud and Mani, 1996]

Table 2: Selection example

Candidate	Binary Code	Integer	Fitness
A	00000111	7	0.6570
C	11100010	226	-0.1934
B	00110010	50	-0.2624
D	01011100	92	-0.7795

2.3 Crossover

Crossover itself primarily serves to specify the search regions to regions of better solutions to the problem of interest due to the fitness measure. Examples for Crossover methods would be a single or two point crossover or a uniform crossover. The two point crossover selects two positions randomly and the segments in between those two are exchanged. In addition the selection rate doesn't need to remain constant during the GA. In our example points 3 and 6 are chosen for two point crossover and with parents A, B we get the new candidate

E due to the used crossover method. (Table 3) [Brabazon and O'Neill, 2006]

Table 3: Two point crossover

Candidate	Binary Code	Integer	Fitness
A	00000111	7	0.6570
C	11100010	226	-0.1934
B	00110010	50	-0.2624
E	00100110	38	0.2964

2.4 Mutation

The mutation operator is very useful to ensure that the search process doesn't stop in the case of population convergence due to the crossover operator. The aim is to find a rate of mutation which doesn't destroy good potential solutions but generates useful novelty. Additionally we use elitism in our GA, which means that the currently best chromosome of a population can't mutate. With a mutation rate of 10% in the example follows the mutation of $24 * 0.1$ allele binaries. (Table 4) [Brabazon and O'Neill, 2006]

Table 4: Mutation

Candidate	Binary Code	Integer	Fitness
A	00000111	7	0.6570
C'	11100 <u>1</u> 10	230	-0.6161
B	00110010	50	-0.2624
E'	00 <u>0</u> 00110	6	-0.2794

GAs derive their power of finding a close to the optimal solution mainly from the crossover and mutation operators, which lead to a highly effective search in the searched area, even though it is not easy to choose the 'right' parameters like the mutation and crossover rate and method. [Mahfoud and Mani, 1996]

To analyze the relationship between these values and the final 'best' solution of the GA, the following two examples will help us. The first example is about finding the minimum of the 2-dimensional Rastrigin's function where the global minimum is 0 in $(x, y) = (0, 0)$. In the second one the input parameters of a predefined trading strategy on the GBPUSD exchange rate are going to be optimized due to the return with a GA.

2.5 Example 1: Rastrigin Function

The Rastrigin function is a n -dimensional NP hard problem and often used as a performance test problem for optimization algorithms.

[Maheshwari et al., 2016] In order to illustrate this we used a 2-dimensional Rastrigin function. (Figure 1 and 2)

$$f(x) = 10 * n + \sum_{i=1}^n x_i^2 - 10 * \cos(2 * \pi * x_i)$$

$$x_i \in [-5.12, 5.12]$$

$$\min_x f(x) = 0$$

2.5.1 Methodology

In this example we used a GA with random selection, mating-heuristic crossover - which means the phenotype real values of the chromosomes are combined, 2 offspring per parents and only different parents are allowed, elitism and the selection and mutation rate stay the same over all generations. To show the convergence and differences through the generations in a GA figure 3 - 6 show the results of a GA with 30 generations, population 50, selection rate 0.3 and mutation rate 0.1.

2.5.2 Results

In order to see the effect that changes in the GA input parameters would bring to the final 'best' solution, we evaluated the GA for different values of the population (10, 15, 20, ..., 100), generations (10, 15, 20, ..., 100), mutation (0.01, 0.05, 0.1, 0.2, 0.3) and selection rate (0.1, 0.3, 0.5, 0.7) three times for each combination. The total computational time was 03 : 18 (mm:ss) with the corresponding MATLAB code in appendix A.1. To truncate outliers we took the mean of the three results of each combination. As expected our test showed that a higher population and generation number leads to a better final 'best' solution (Figure 7) and needs longer to be computed (Figure 8). Regarding mutation and selection rate it is not so easy to calculate their effect on the GA. In figures 9-11 we tried to show the importance of the combination between selection and mutation rate. From these three figures you might conclude that a low mutation rate (0.01) performs better, relative to the other mutation rates, only if the selection rate is low (0.3). If the selection rate is higher (0.5, 0.7) the performance of the low mutation rate is worse relative to the others and the performances of the other two mutation rates differ only a bit and converge with a higher population number to the real optimal solution which is zero. In total

it seems that a higher selection rate combined with a normal mutation rate tends to perform best and is robust at a high population number. To test if the 'best' optimal solution converges to the real optimal solution we evaluated the GA with $pop = 500, gen = 1000, sel = 0.3, mut = 0.1$ and 2 out of 3 times we received zero as the optimal solution and one time a value like 10^{-10} , following that the GA of the Rastrigin function converges to the real optimal solution with a high population and generation number.

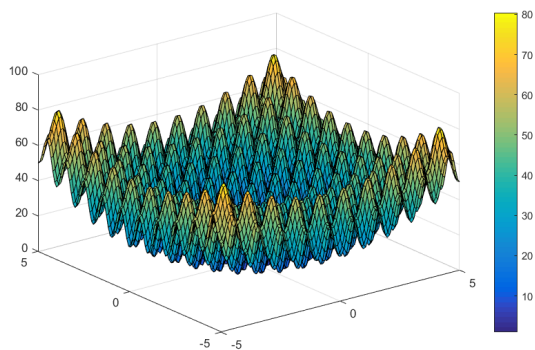


Figure 1: Rastrigin function

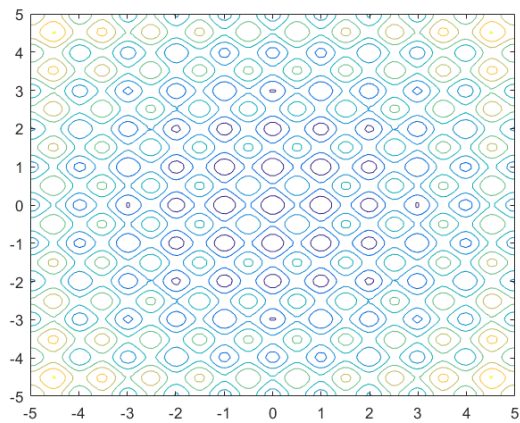


Figure 2: Rastrigin function contour plot

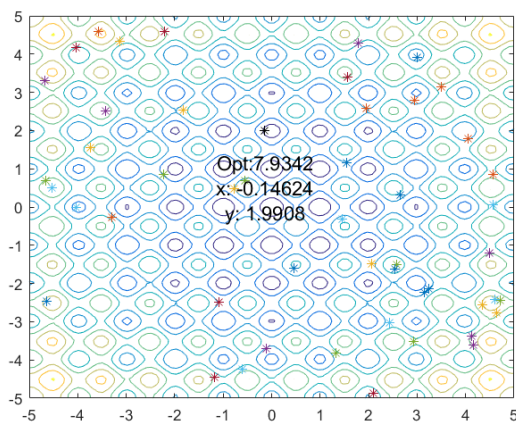


Figure 3: First generation

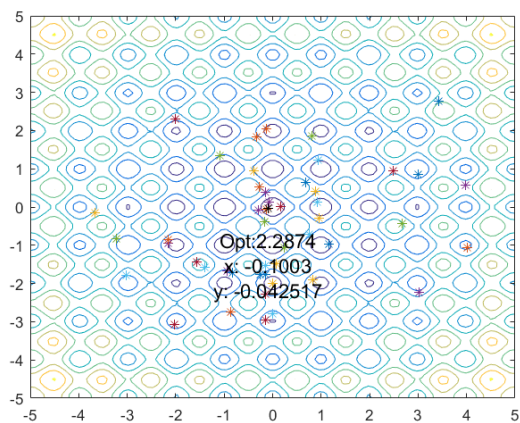


Figure 4: 10th generation

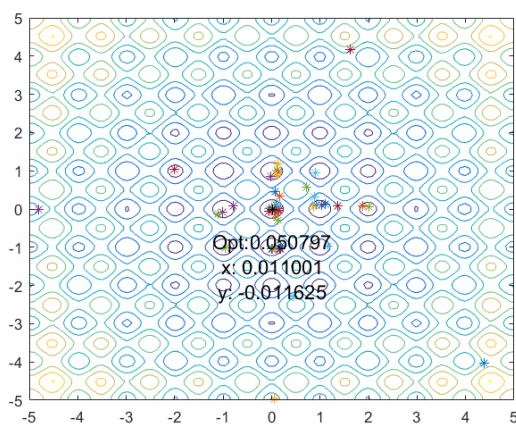


Figure 5: 20th generation

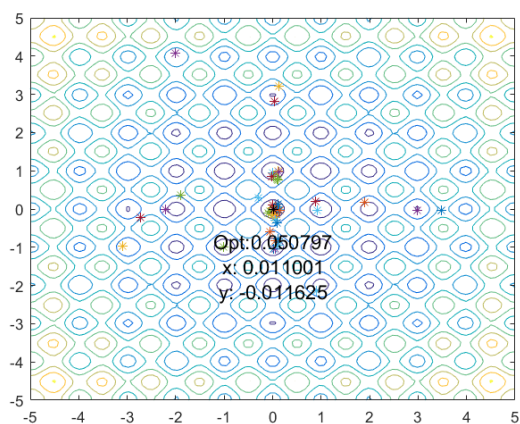


Figure 6: 30th generation

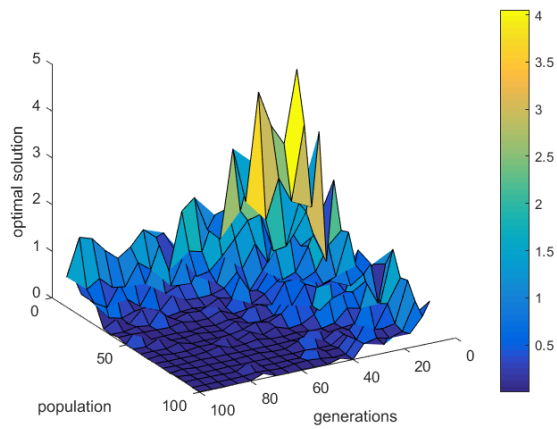


Figure 7: GA Performance

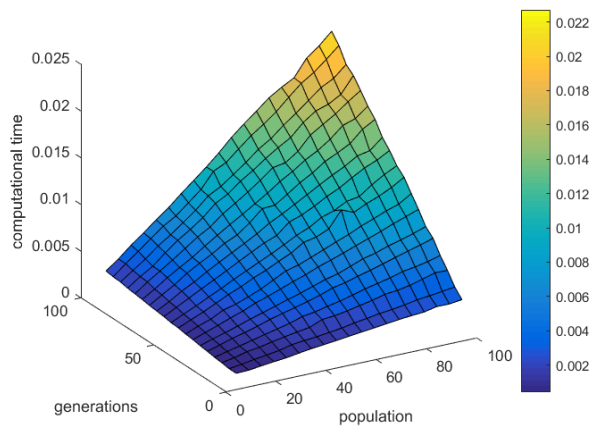


Figure 8: GA Performance

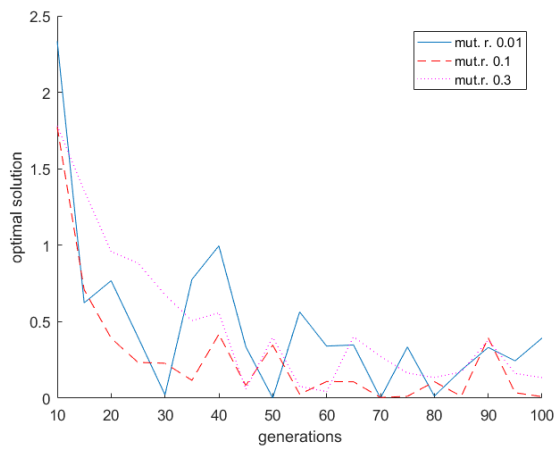


Figure 9: GA Performance, Pop=50, sel=0.3

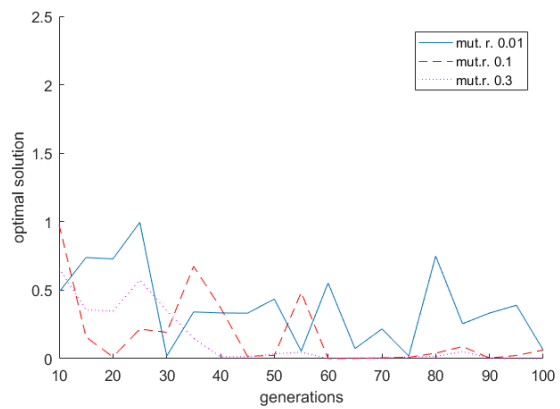


Figure 10: GA Performance, Pop=50, sel=0.5

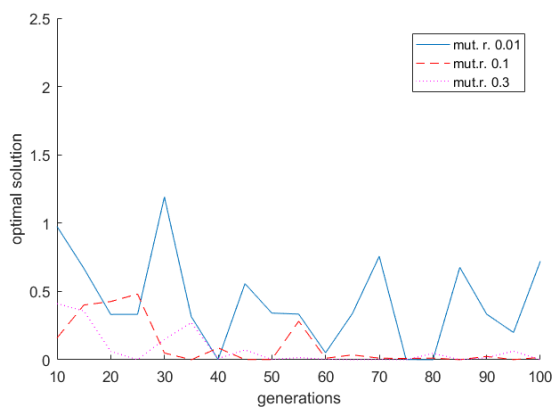


Figure 11: GA Performance, Pop=50, sel=0.7

2.6 Example 2: Optimizing input parameters of the Moving Average Strategy

Technical analysts focus on price and return of stocks in contrast to fundamental analysts who look at earnings, dividends, new products and R&D to indicate profits of stocks. The aim of any technical analyst is to identify price patterns and market trends using historical prices to forecast stock price movements which comes into conflict with the efficient market hypothesis stating that financial asset prices fully reflect all available information [Fama, 1970]. With this hypothesis future movements are not predictable but due to some academic research in the nineties this theory has been rejected, as risk premiums are varying over time and depend on business cycles. Another argument is that agents are not fully rational in their behavior. Based on these arguments technical analysts aim of identifying price patterns is not impossible. [Bruder et al., 2011, Hong and Satchell, 2011] To show that forecasting price movements is possible we are using time series methods. The white noise process is the simplest kind of time-series model which makes forecasting impossible. But due to much academic research stocks aren't white noise processes which implies that formulating trading strategies and using technical analysis is useful. [Hamilton, 1994]

2.6.1 Methodology

In this example we are going to use one of the simplest trading strategies based on technical analysis which is called the Moving Average (MA) rule. This strategy determines buy and sell signals when a short moving average crosses a long moving average. This crossing indicates price trends the MA rule takes advantage of, which are related to the idea of the profitable price momentum strategies. When the short moving average exceeds the long moving average an upward trend is expected to be initiated and the trader takes a long position and vice versa at each crossing. The trading signal will always be evaluated at the closing price and executed at the opening price the day after. In each trade we invest the whole capital with 100 as starting capital. The moving average of a time period t is defined as $MA(t) = \frac{\sum_{i=1}^t P(t-i+1)}{t}$. [Hong and Satchell, 2011] With a trailing stoploss percentage included into the trading strategy we are trying to improve the performance of it. In contrast to a simple stoploss which is evaluated at the point of long buying or short selling and stays the same over one trade, a trailing

stoploss will be actualized daily on the closing price depending if it is a long or short position. If price movements hit the stoploss the current trade will be closed at the stoploss price and a new position will be opened at the next crossing of the moving averages. [Clare et al., 2012]

The aim in this example is to find the optimal combination of long and short moving average and trailing stop loss in terms of maximizing the return with a given MA and stoploss interval and backtesting time period with the GA. After the optimization with the GA the optimal trading strategy will be validated on the remaining time period.

The GA used in this example is an enlarged version of the GA in section 2. Now you can decide if the selection and mutation rate stays the same or if it decreases linearly or negative logarithmically over all generations (figure 12).

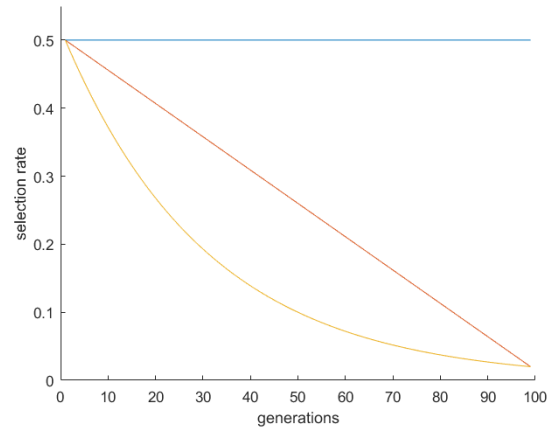


Figure 12: selection rate changes over generations

2.6.2 Data

The data series used is the exchange rate between the British Pound and the US Dollar (GBPUSD) in the time period 01/2000–02/2017 on a daily basis, in total 4464 days. [Tel,] (figure 13 and 14) In the figures two big abnormalities can be identified, in figure 13 you see a big loss between day 2100 and 2300 in our time period which represents the financial crisis in 2008/09 and in figure 14 a abnormal downward peak on the return at *July 24, 2016 (day 4280)* which was the day after the Brexit voting.

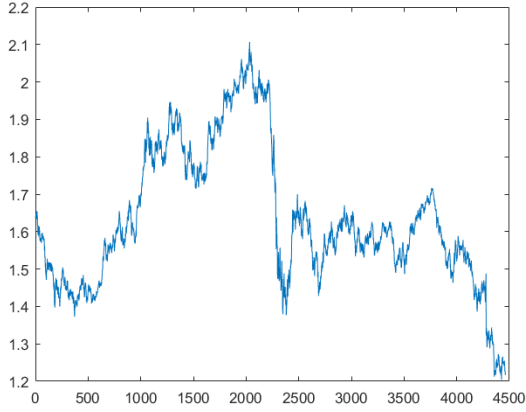


Figure 13: GBPUSD 01/00-02/17 daily closing prices

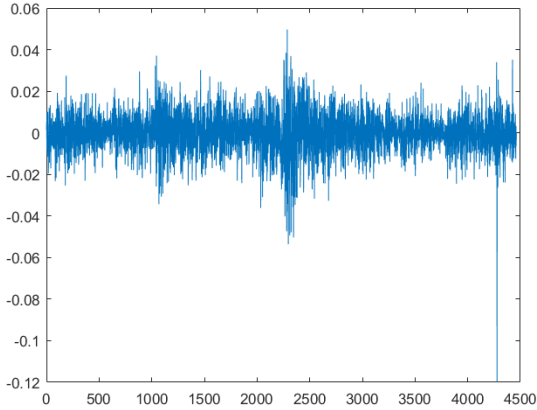


Figure 14: GBPUSD 01/00-02/17 daily returns

2.6.3 Results

In this section we would like to analyze the effect of different input parameters of the GA, especially the effect of different backtesting time periods. But for this purpose we need to make the results of using different time periods comparable. This can be done by using the ratio of daily returns in the validation time period to daily returns in the backtesting time period. This ratio indicates if the return in the validation time period fits well to the one in the backtesting time period, it fits perfect if the ratio is 1.

We evaluated the GA for different values of the population (50, 100, 150), generations (20, 40, 60, ..., 200), mutation (0.1, 0.2) and selection rate (0.3, 0.5), MA upper boundary (100, 300), backtesting time peri-

ods (500, 1000, 1500, ..., 4000) and each of these 1920 combinations for three times. The other input parameters have been the same for all combinations: stoploss interval [0.01, 0.1] and the selection and mutation rate decreases negative logarithmically. The total computational time was 16 : 05 : 50 (hh:mm:ss) with the corresponding MATLAB code in appendix A.2.

On figure 15 you can see the daily return ratios as already described above, but with that line plot you cannot analyze exactly. In order to make the ratios comparable we used boxplots as you see in figure 16 and 17 and furthermore differentiated between the MA upper boundaries, figure 16 involves only combinations with a MA upper boundary of 100 and figure 17 of 300. Between these two figures there are only small differences but ratios on the first figure tend to be a bit higher than those on the other one. Additionally you can slightly guess the typical under- and overfitting curve. Combinations with a lower time period don't have enough information (underfitting) and those with a higher time period have too much information (overfitting) to fit the validation time period well. Time periods like 2000 and 2500 have the best fitting therefore we are going to look more precisely on these combinations.

Figure 18 shows these two time periods with a MA upper boundary of 100 and is split in boxplots with all possible combinations of selection and mutation rate. The values at the y axis show the corresponding sum of backtesting time, selection and mutation rate. You can see that it might be better to use a selection rate of 0.3 and in this case the usage of 0.2 as a mutation rate is only a little better. But you definitely cannot say if the time period 2000 or 2500 is the better one. After this analysis we tried to improve the results for these specific input parameters by increasing the number of population and generations. In figure 19 you find the best output of the GA due to the return and ratio of daily returns with 2500 as the backtesting time period, population 2000 and 5000 generations. The best received combination was a short MA of 6 days and a long MA of 8 days with a stoploss of 1.5486%. With this trading strategy the return in the backtesting time is 84.3669% with a maximal draw-down duration of 649 days, a maximal draw-down percentage of 17.28% and 196 trades (buy and sell together is one trade). And in the validation time this trading strategy reached a return of 4.67% with maximal draw-down duration of 1247 days, a maximal draw-down percentage of 16.21% and 145 trades.

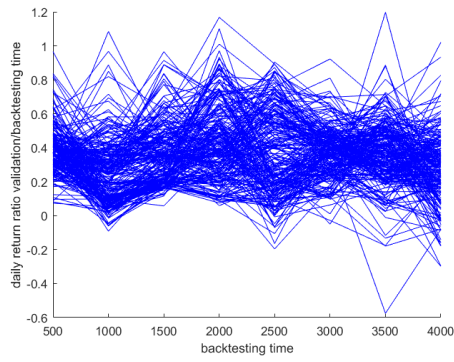


Figure 15: daily return ratio

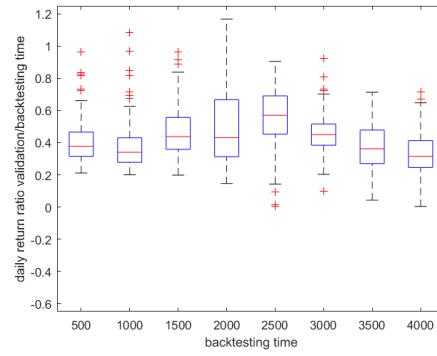


Figure 16: boxplot, MA max. 100

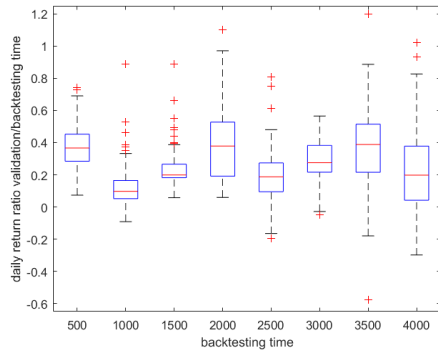


Figure 17: boxplot, MA max. 300

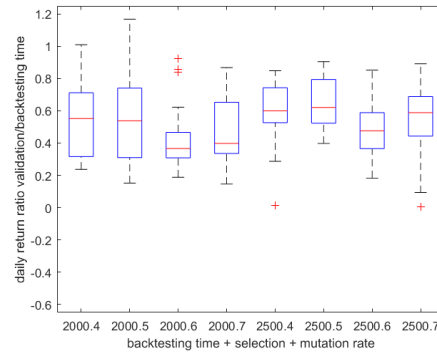


Figure 18: boxplot, MA max. 100, backtest 2000/2500

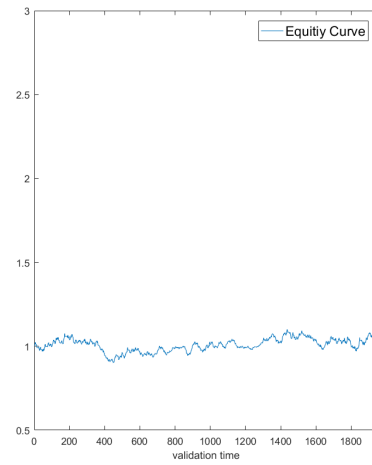
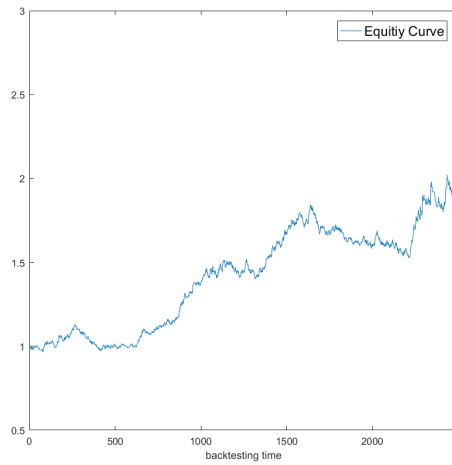


Figure 19: SMA 6 days, LMA 8 days, stoploss 1.5486%, validation time return 4.67%

3 Grammatical Evolution

The Genetic Programming (GP) is another important evolutionary algorithm which belongs to Evolutionary Computation (EC). Grammatical Evolution (GE) is a grammar-based form of GP which improves dealing with dynamic environments. One of the key features of GE to solve problems in dynamic environments is the genotype-to-phenotype mapping process.

3.1 Mapping Process

In GE binary strings are used grouped into codons of eight bits. The integer values defined by these codons are used to select a production rule from a grammar defined using the Backus-Naur form (BNF) via a mapping function. The BNF definition describes the output language which is going to be produced by the system.

BNF grammars are made up of the tuple $\{N, T, S, P\}$, where N is the set of non-terminal symbols, T the set of terminal symbols, S the start symbol and a combination of members of T and P a set of production rules that maps N to T . The generated phenotypic program will only consist of elements of the terminal set T .

The mapping process starts with the start symbol S and will select the leftmost non-terminal symbol as long as non-terminal symbols are in the current output. For each non-terminal the following mapping function will be used to replace this non-terminal with one of its available choices:

$$Rule = c \bmod r$$

where c is a codon integer and r the number of choices of the current non-terminal. This will form a complete program, which is only comprised from elements of T , in a developmental approach. The codons are chosen out of the genome code beginning from the left-hand side and generating the integer value.

But under special conditions it is possible that an incomplete mapping could occur. Therefore in standard GE the mapping process determines if one of the following conditions is met:

- A complete program is generated before the end of the genome is reached.
- The end of the genome is reached before a completed program is generated, then its genome-reading frame returns to the left-hand side of the genome again.

- If a threshold on the number of wrapping events is reached and the individual is still not completed, then the mapping process is halted and the individual is assigned the worst possible fitness value.

[Dempsey et al., 2009, Brabazon and O'Neill, 2006]

3.1.1 Mapping Example

Consider the following genome, represented as a series of integer-valued codons:

50 7 115 240 86 181 42 201 17,

and a BNF Grammar:

$$\begin{aligned} N &= \{ \langle expr \rangle, \langle op \rangle, \\ &\quad \langle opvar \rangle, \langle var \rangle \} \\ T &= \{ 1, 2, 3, +, -, x, y \} \\ S &= \{ \langle expr \rangle \} \end{aligned}$$

$P :$

$$\langle expr \rangle := \langle expr \rangle \langle op \rangle \langle expr \rangle \quad (0)$$

$$\langle opvar \rangle \quad (1)$$

$$\langle op \rangle := + \quad (0)$$

$$- \quad (1)$$

$$\langle opvar \rangle := 1 \quad (0)$$

$$2 \quad (1)$$

$$3 \quad (2)$$

$$\langle var \rangle \quad (3)$$

$$\langle var \rangle := x \quad (0)$$

$$y \quad (1)$$

Beginning with the start symbol $\langle expr \rangle$ there are two possible replacements of it. The codon reading starts on the left-hand side with 50 and is mapped to the number of available expressions. Therefore $50 \bmod 2 = 0$ and $\langle expr \rangle$ will be replaced by $\langle expr \rangle \langle op \rangle \langle expr \rangle$. Taking the left-most non-terminal which is again $\langle expr \rangle$ and the next codon 7 with $7 \bmod 2 = 1$ implies the replacement of $\langle expr \rangle$ by $\langle opvar \rangle$. Then $\langle opvar \rangle$ is replaced with $\langle var \rangle$, as $115 \bmod 4 = 3$, and $\langle var \rangle$ with x , as $240 \bmod 2 = 0$. The following left-most non terminal of $x \langle op \rangle \langle expr \rangle$ is $\langle op \rangle$ which will be replaced by $+$ due to $86 \bmod 2 = 0$. With the remaining codons the final phenotypic program is $x + 3$. The two leftover codons are not used and are simply ignored in the mapping process.

3.2 Crossover and Mutation in GE

The standard genetic operators like selection, crossover and mutation (Section 2.2-2.4) can be applied to the underlying genotypic representation. Due to the mapping process, the effect of crossover and mutation on the phenotype can be a bit more complex. But the mechanics of the operators are the same, so mutation changes a bit or an integer randomly, and the one or two point crossover swaps sections of the genetic code between the parents. [Dempsey et al., 2009, Brabazon and O'Neill, 2006]

3.3 Grammatical Constant creation

One of the most important adaption to the standard GE is the generation and usage of constants. One method is to evolve constants using the Digit Concatenation. One example of a grammar using Digit Concatenation is given with the following grammar:

$$\begin{aligned} \langle real \rangle &:= \langle int \rangle \langle dot \rangle \langle int \rangle \mid \langle int \rangle \\ \langle int \rangle &:= \langle int \rangle \langle digit \rangle \mid \langle digit \rangle \\ \langle digit \rangle &:= 0|1|2|3|4|5|6|7|8|9 \\ \langle dot \rangle &:= . \end{aligned}$$

which could produce constants of any size or floating-point precision with evolution determining their usefulness. [Dempsey et al., 2009]

To prevent the creation of not useful constants in we used a modified version of the Digit Concatenation which is given by:

$$\begin{aligned} \langle r \rangle &:= \langle x \rangle . \langle x \rangle & (0) \\ \langle x \rangle &:= \langle x \rangle \langle x \rangle \langle x \rangle & (1) \\ &\vdots \\ \langle x \rangle &:= \langle x \rangle \langle x \rangle \langle x \rangle . \langle x \rangle \langle x \rangle \langle x \rangle & (8) \end{aligned}$$

but instead of using terminal symbols for $\langle x \rangle$ the GE replaces them by the codons integer values to create constants.

3.4 Example 1: Multiple Regression with Grammatical Evolution

In this example we want to compare the forecasting potentials of the multiple regression model and the

GE with 3 explanatory variables and one explained variable.

3.4.1 Methodology

The multiple regression model is one of the most important methods used for empirical analysis in economics and other social sciences, especially the method of ordinary least squares is popularly used for estimating parameters of a multiple regression model. The general multiple linear regression model is defined as:

$$y = \beta_0 + \beta_1 x_1 + \dots + \beta_k x_k + u$$

where β_0 is the intercept, $\beta_j, j = 1, \dots, k$ the parameters associated with x_k (slope parameters), x_k the independent variables, y the dependent variable and u the error term. The key assumption in this model is that the error term is uncorrelated with the explanatory variables:

$$\mathbb{E} u x_j = 0 \quad \forall j = 1, \dots, k$$

To estimate the parameters $\beta_j, j = 0, \dots, k$ the method of ordinary least squares chooses those estimates that minimize the sum of the squared residuals. [Wooldridge, 2006]

In our example the explanatory variables and explained variable are defined as:

$$\begin{aligned} y &:= pH \\ x_1 &:= caustic\ soda\ flow \\ x_2 &:= organic\ acid\ flow \\ x_3 &:= temperature \end{aligned}$$

With the GE our aim is to estimate a nonlinear function f , which minimizes the error term u :

$$y = f(x_1, \dots, x_k) + u$$

The used GE in this example consists of the following BNF grammar:

$N = \{ \langle e \rangle, \langle b \rangle, \langle u \rangle, \langle v \rangle \}$
 $T = \{ \sin, \cos, \exp, \log, \text{sqrt}, +, -, *, / \}$
 $S = \{ \langle e \rangle \}$

$P :$
 $\langle e \rangle := \langle e \rangle \langle b \rangle \langle e \rangle \mid (\langle e \rangle \langle b \rangle \langle e \rangle) \mid \langle u \rangle (\langle e \rangle) \mid \langle v \rangle$
 $\langle b \rangle := + \mid - \mid * \mid /$
 $\langle u \rangle := \sin \mid \cos \mid \exp \mid \log \mid \text{sqrt}$
 $\langle v \rangle := \text{var0} \mid \text{var1} \mid \text{var2} \mid \langle y \rangle$
 $\langle y \rangle := \langle x \rangle . \langle x \rangle \mid \dots \mid \langle x \rangle \langle x \rangle \langle x \rangle . \langle x \rangle \langle x \rangle \langle x \rangle$
 $\text{var0} := \text{caustic soda flow}$
 $\text{var1} := \text{organic acid flow}$
 $\text{var2} := \text{temperature}$

Furthermore it uses random selection, elitism, 2 offspring per parents and only different parents are allowed. You can also decide whether the selection and mutation rate stays the same or if it decreases linearly or negative logarithmically over all generations and which crossover method you want to use, a one or two point crossover. The non-terminal $\langle y \rangle$ represents the modified version of the Digit Concatenation as in section 3.3 defined.

Another important input parameter in the GE is the number of used binary codons per chromosome and the maximal number of used functions in a grammar, which should in computing with MATLAB be lower than 32, because MATLAB doesn't allow more than 32 nested functions.

The aim in this example is to find the optimal grammar in the backtest data, due to the sum of the absolute errors, and validate this optimal grammar with the remaining values. Especially the average ratio of the sum of absolute errors in the backtest to the validation data will be interesting, just as the comparison between the multiple regression model and the GE.

3.4.2 Data

The data used in this example are 4 vectors with size 100 of the parameters caustic soda flow, organic acid flow, temperature and pH [Neu,] and we are interested in the effect of the first three parameters on pH. (figure 20) We therefore tried to find a optimal grammar in the backtesting data with GE and then

validated the so found optimum with the remaining data.

3.4.3 Results

In this section we would like to analyze the effect of different input parameters of the GE, especially the effect of using different backtesting data. To make the results of using different backtesting data comparable, we use the same idea as in section 2.6.2 by using the ratio of average error in the validation data to average error in the backtesting data. This ratio indicates if the error in the validation time period fits well to the one in the backtesting time period and simultaneously prevents under- and overfitting in the model.

We evaluated the GE for different values of the population (100, 200, 300, 400), generations (100, 300, 500, 700, 900), selection and mutation rate (0.5/0.3, 0.3/0.2, 0.3/0.1), backtesting data (20, 35, 50, 65, 80, 95) and each of these 360 combinations for three times. The other input parameters have been the same for all combinations: one point crossover, 100 binary codons per chromosome, 30 as the maximal number of used functions in a grammar and the selection and mutation rate decreases negative logarithmically. The total computational time was 16 : 30 : 00 (hh:mm:ss) with the corresponding MATLAB code in appendix A.3.

On figure 21 you can see the average error ratios as already described above of all combinations. To make the ratios comparable we used boxplots as you see in figure 22 where the typical under- and overfitting curve is slightly observable, except the box with 50 as backtesting data is an outlier. Combinations with less backtesting data don't have enough information (underfitting), especially those with 20 are spread over all average error ratio values, and those with a higher time period have too much information (overfitting) to fit the validation time period well. Except to a view outliers the box with 80 as backtesting data seems to be the one with the lowest and most robustest error ratio, so we are going to look more precisely on these combinations.

Figure 23 shows all combinations with 80 as backtesting data split in boxplots with all possible combinations of selection and mutation rates. You can see that it might be best to use the combination of 0.3 and 0.2 as selection rate and mutation rate due to a robust and low error ratio.

After this analysis we tried to improve the results for these specific input parameters by increasing the

number of population and generations. In figure 24 you find the best output of the GE due to the error and ratio of errors found with population 2000 and 1000 generations. The optimal received grammar is:

$$\begin{aligned} & \text{sqrt}(\text{caustic soda flow} - \log(\text{organic acid flow}) \\ & * (\text{organic acid flow} + (\cos((\log(34100.164225500) \\ & - \text{temperature} / \text{caustic soda flow})) \\ & / (\text{organic acid flow} * \text{caustic soda flow})) \end{aligned}$$

With 94.15 as the sum of absolute errors in the back-testing data and 26.75 in the remaining validation data.

To compare these results with a multiple regression model we evaluated one on the first 80 data points in R and received the following regression model:

$$\begin{aligned} pH = & 2.04 + 0.17 * \text{caustic soda flow} \\ & - 0.42 * \text{organic acid flow} - 0.01 * \text{temperature} + u \end{aligned}$$

with caustic soda flow and organic acid flow statistically significant at the one percent level, which leads to 93.71 as the sum of absolute errors in the backtesting data and 25.33 in the remaining validation data. (figure 25)

Hence the GE is an acceptable alternative method to the multiple regression model in forecasting.

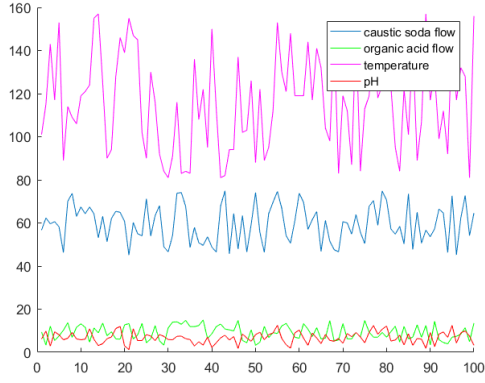


Figure 20: 4 chemical parameter vectors with size 100

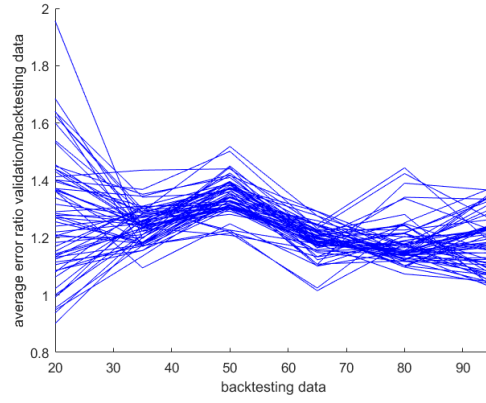


Figure 21: average error ratio

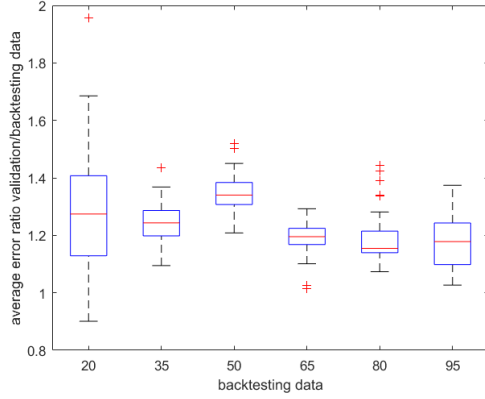


Figure 22: average error ratio boxplot

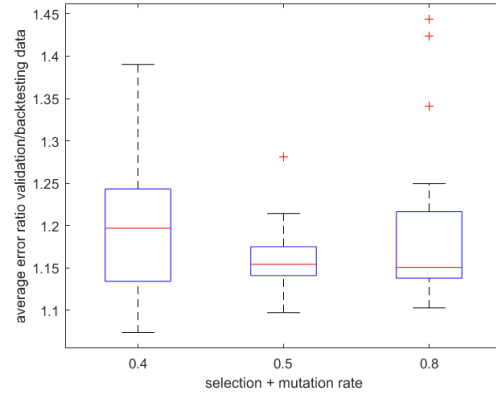


Figure 23: average error ratio boxplot, backtesting data 80

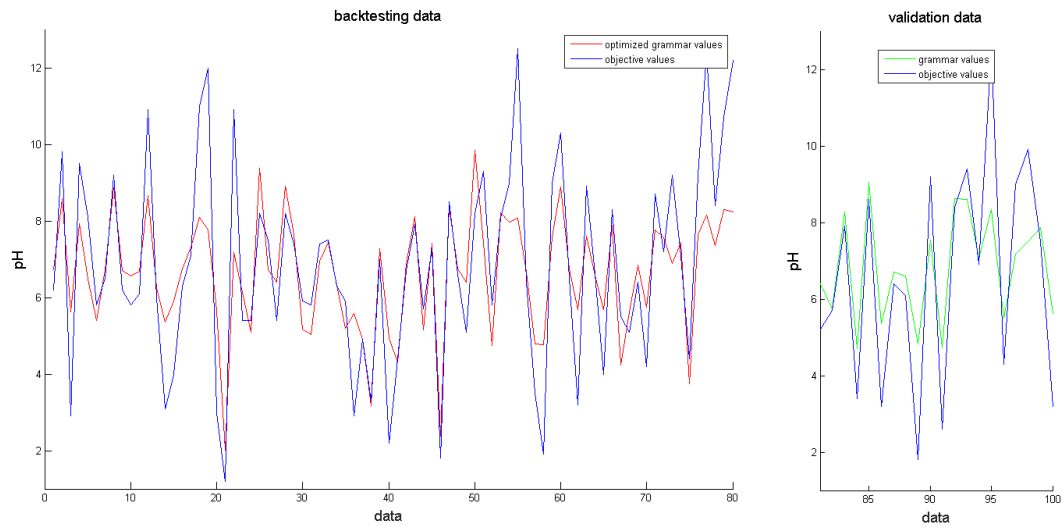


Figure 24: GE model on backtesting data 80

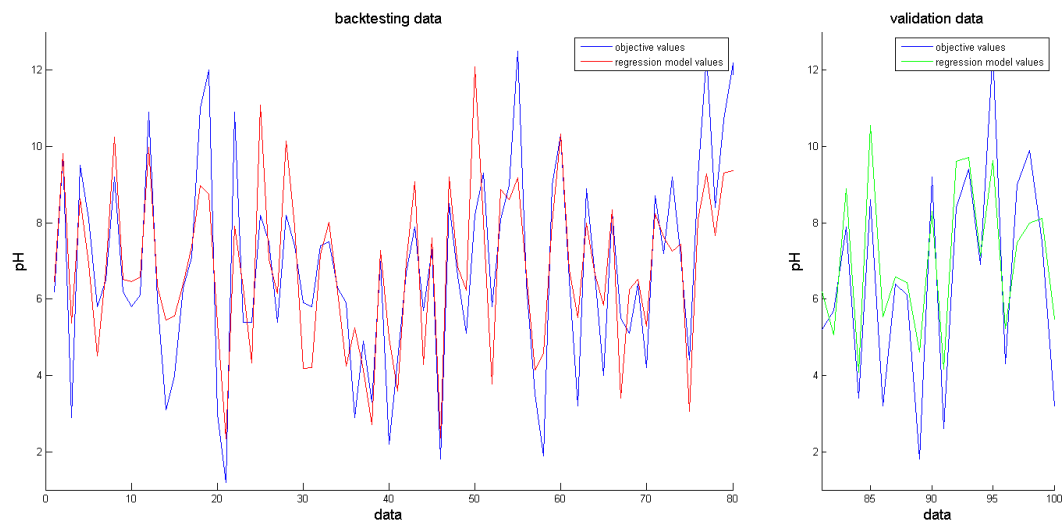


Figure 25: regression model on backtesting data 80

3.5 Example 2: Optimizing a Trading strategy with Grammatical Evolution

As already seen in section 2.6 formulating trading strategies and using technical analysis to forecast time series is useful.

3.5.1 Methodology

In contrast to section 2.6 now we don't use a predefined trading strategy and optimize its input parameters but optimize the trading strategy itself.

To find such an optimal trading strategy we are using the GE, where each chromosome represents a trading strategy, with the following self-developed BNF grammar:

```

N =    { < e >, < o >, < p >, ..., < l > }
T =    { &&, ||, ..., parb (1,2) }
S =    { if(< e >) tr(i + 1) = < t >;
        else tr(i + 1) = ±1; end
        stoploss = < l >; }

P :
< e >:=  < e > < o > < e >
        | < p > (< e >, < e >)
        | < y > < z > < y >
< o >:=  && ||
< p >:=  or | xor
< y >:=  < y > < b > < y >
        | (< y > < b > < y >)
        | < u > (< y >)
        | < v >
< z >:=  < | < = | > = | >
< b >:=  + | - | * | /
< u >:=  sin | cos | exp | log | sqrt
< v >:=  w(i - < j >, 4) | < k >
< k >:=  < d > . < d > | ...
        | < d > < d > < d > . < d > < d > < d >
< t >:=  +1 | -1
< j >:=  parb(2,1) | ... | parb(1,1)
< l >:=  parb(2,2) | ... | parb(1,2)

```

With a 2×2 matrix parb including the minimum and maximum of the past used values to calculate

a signal in column 1 and in column 2 the minimum and maximum of the stoploss used in the trading strategy. For example with $parb(2,1) = 100$ and $parb(1,1) = 0$ we will maximally include closing prices from 100 days ago until now within each trading strategy. In the start expression S a trading signal will be evaluated with $if(< e >)$ on each day i , using matrix w which consists of all open, high, low and close prices, the execution will be on day $i + 1$ at the opening price and the signals will be saved in vector tr . The choices of non-terminal $< t >$ are the values 1 or -1 and contrarily the else statement \pm will be replaced by the value -1 or 1. After that the GE will evaluate a stoploss $< l >$ out of the choices (0.01, 0.011, 0.012, ..., 0.1) if $parb(2,2) = 0.01$ and $parb(1,2) = 0.1$ for example. Furthermore the non-terminal $< e >$ guarantees at least one binary boolean or relational operator in the if-statement. The non-terminal $< k >$ represents the modified version of the Digit Concatenation as in section 3.3 defined. Except these characteristics the GE is the same as in section 3.3.

The aim of this example is to find the optimal trading strategy in terms of maximizing the return in the backtesting time period. After the optimization the trading strategy will be validated on the remaining time period.

3.5.2 Data

The data series used in this example is the exchange rate between the British Pound and the US Dollar (GBPUSD) as in section 2.6.2 already described.

3.5.3 Results

In this section we would like to analyze the effect of different input parameters of the GE, especially the effect of different backtesting time periods. To make the results of using different backtesting time periods comparable I use the same idea as in section 2.6.2 by using the ratio of daily returns in the validation data to daily returns in the backtesting data. This ratio indicates if the error in the validation time period fits well to the one in the backtesting time period and simultaneously prevents under- and overfitting in the model.

We evaluated the GA for different values of the population (20, 40, 60), generations (50, 75, 100), mutation (0.1, 0.2) and selection rate (0.3, 0.5), backtesting time periods (500, 1000, 1500, ..., 4000) and each of these 288 combination for two times. The other input

parameters have been the same for all combinations: one point crossover, 100 binary codons per chromosome, 30 as the maximal number of used functions in a grammar, stoploss interval $[0.01, 0.1]$, past used values $(-100, -99, \dots, 0)$ and the selection and mutation rate decreases negative logarithmically. The total computational time was 17 : 26 : 54 (hh:mm:ss) with the corresponding MATLAB code in appendix A.4.

On figure 26 you can see the daily return ratios as already described in section 2.6.3, but to make the ratios comparable we used boxplots as you see in figure 27. But it is not easy to find the best combination of input parameters. What you can see is a wide range of ratios with a backtesting time of 4000. To be able to analyze it we split the boxes within the different combinations of selection and mutation rate which you can see in figure 28 and 29. But now it's even more difficult to say which combination is best to use for a higher number of population and generations. But two combinations tend to be a bit better and more robust in comparison to all the others, which is the box with index 2000.5 on figure 28 which means a backtesting time of 2000, selection rate 0.3 and mutation rate 0.2 and the box with index 3000.6 on figure 29 with backtesting time 3000, selection rate 0.5 and mutation rate 0.1.

The main reason of these patchy findings was the high computational time on the CPU. Which leads to the idea to use a smaller sample size and a smaller number of population and generations, which are the decisive factors regarding the computational time and the results. Nevertheless with the used input parameters we have found two combinations which seem to be better than the others, and we then tried to improve the results for these specific input parameters by raising the number of population and generations.

In figure 30 you find the best output of the GA due to the return and ratio of daily returns with 2000 as the backtesting time period, population 500 and 1000 generations, with a computational time of 01:48:17 (hh:mm:ss). The optimal trading strategy can be found in appendix A.4.1, on the last page of this paper. With this trading strategy the return in the backtesting time is 136.72%, maximal draw-down duration of 221 days and maximal draw-down percentage of 9.09%. In the validation time this trading strategy reached a return of 11.91% with maximal draw-down duration of 776 days and maximal draw-down percentage of 15.63%.

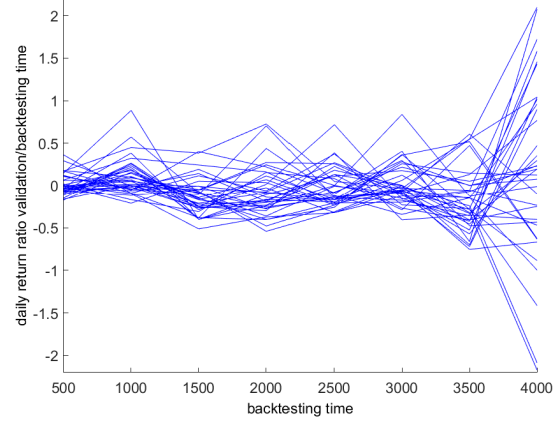


Figure 26: daily return ratio

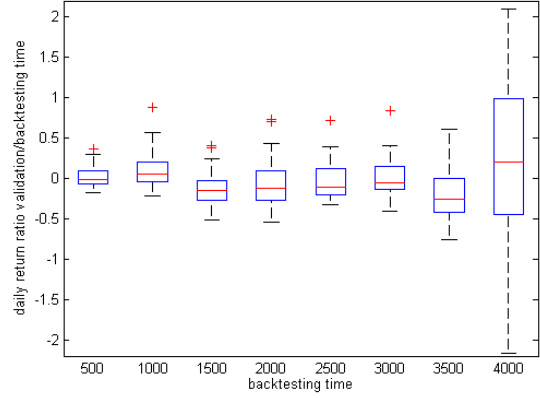


Figure 27: daily return ratio, boxplot

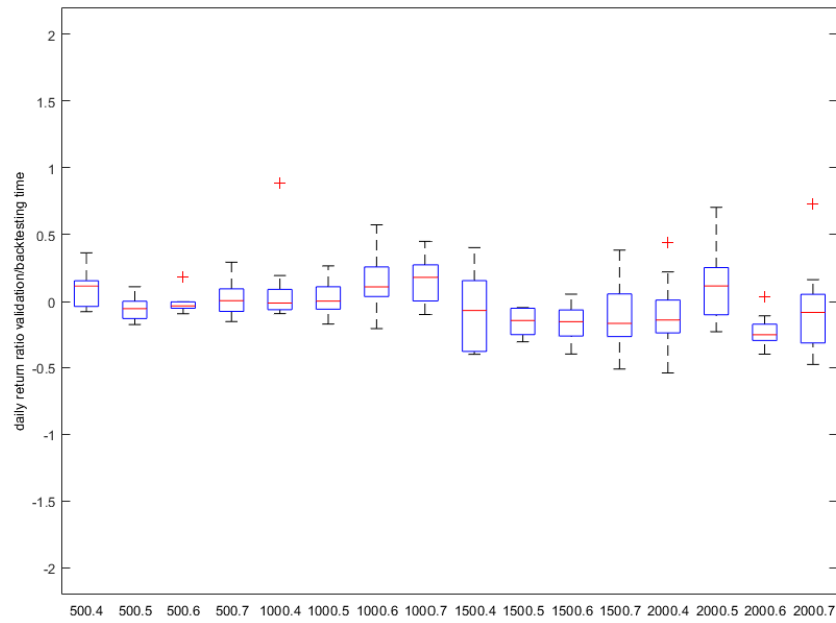


Figure 28: daily return ratio, boxplot, backtesting time 500-2000

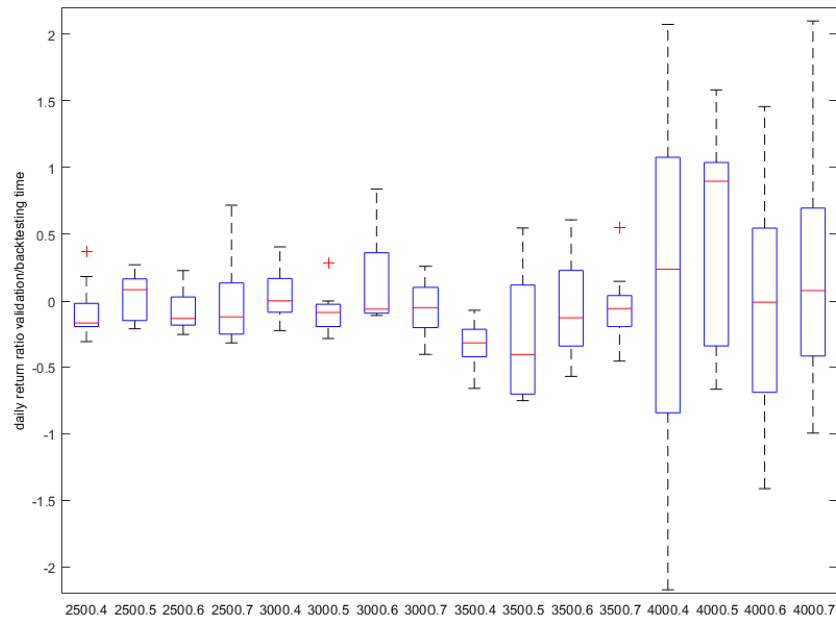


Figure 29: daily return ratio, boxplot, backtesting time 2500-4000

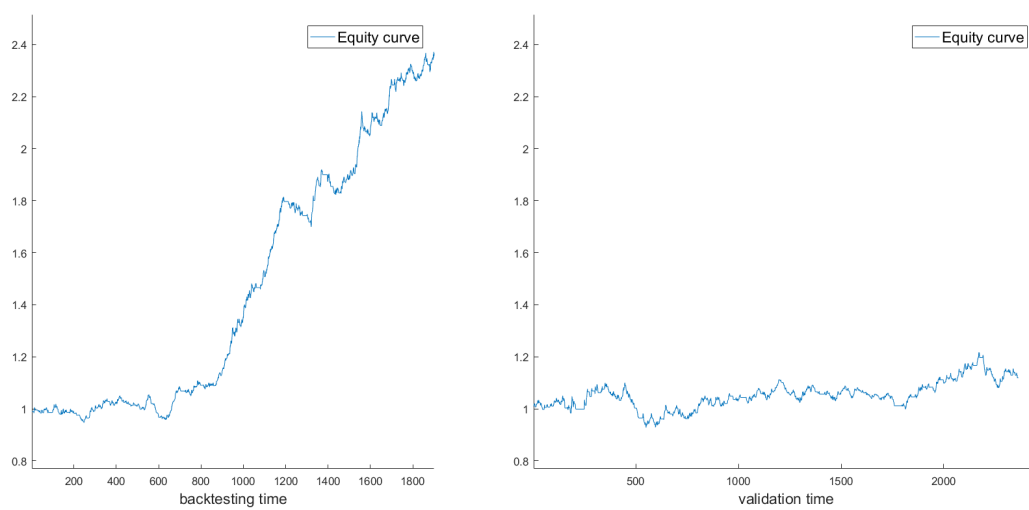


Figure 30: backtesting time 2000, stoploss 1%, validation time return 11.91%

4 Conclusion & Future work

The title of this paper is "How do naturally inspired search algorithms help to find the optimal trading strategy?" which I have tried to answer in different ways. The two used search algorithms were the genetic algorithm and the grammatical evolution and we tried to show their robustness on two standard examples of the literature, optimize the input parameters of a specific trading strategy and optimize the trading strategy itself. We made the following conclusions from our study:

- Results have shown that both the genetic algorithm and the grammatical evolution are two robust and well performing search algorithms for optimizing complex problems.
- The problem of finding the optimal input parameters for a given trading strategy, like the used moving average rule, can be solved by a genetic algorithm, in terms of finding a close to optimal solution, which is not known anyway, in a nonlinear space. But the returns in the validation time are not really satisfying with higher population and generations.
- The grammatical evolution is a good alternative to the multiple regression model in forecasting and finds similar solutions.
- In trying to find the optimal trading strategy itself we had the issue with the computational time on the laptop. The main reason for this was using only the CPU for computing, following that the found results have been a bit patchy but still show the possibilities and performance on optimizing complex problems.

Additionally future work, especially in optimizing the trading strategy, would include using the GPU instead of the CPU to reduce computational time and find even better solutions for this complex optimization problem in a highly nonlinear space.

References

- [Neu,] Neuroshell trader. <http://try.neuroshell.com/index/>. Accessed: 2016-11-07.
- [Tel,] Teletrader. <http://www.teletrader.com>. Accessed: 2017-03-10.
- [Brabazon and O'Neill, 2006] Brabazon, A. and O'Neill, M. (2006). *Biologically Inspired Algorithms for Financial Modelling*. Springer-Verlag Berlin Heidelberg.
- [Bruder et al., 2011] Bruder, B., Richard, J., Dao, T., and Roncalli, T. (2011). Trend filtering methods for momentum strategies.
- [Clare et al., 2012] Clare, A., Seaton, J., Thomas, S., and Smith, P. N. (2012). Breaking into the blackbox: Trend following, stop losses, and the frequency of trading: the case of the s and p500.
- [Dempsey et al., 2009] Dempsey, I., Brabazon, A., and O'Neill, M. (2009). *Foundations in Grammatical Evolution for Dynamic Environments*. Springer-Verlag Berlin Heidelberg.
- [Fama, 1970] Fama, E. (1970). Efficient capital markets: A review of theory and empirical work. *Journal of Finance*, pages 383–417.
- [Goldberg, 1989] Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Inc.
- [Hamilton, 1994] Hamilton, J. (1994). *Time Series Analysis*. Princeton University Press.
- [Hong and Satchell, 2011] Hong, K. and Satchell, S. (2011). Technical analysis, momentum and autocorrelation.
- [Maheshwari et al., 2016] Maheshwari, A., Garg, R., and Sharma, N. (2016). Implementation of dejong function (rastrigin) by genetic algorithm. *International Journal of Advanced Research in Computer Science and Software Engineering*, 6:425–427.
- [Mahfoud and Mani, 1996] Mahfoud, S. and Mani, G. (1996). Financial forecasting using genetic algorithms. *Applied Artificial Intelligence*, 10:543–565.
- [Rothlauf, 2006] Rothlauf, F. (2006). *Representations for Genetic and Evolutionary Algorithms*. Springer-Verlag Berlin Heidelberg.
- [Wooldridge, 2006] Wooldridge, J. (2006). *Introductory econometrics, a modern approach*. Thomson/South-Western.

A MATLAB code

A.1 Genetic Algorithm - Example 1

```

1 function [opt]=genalg (pop, sel, mut, gen, f, parb)
2 %-----Explanation of the function:-----
3 %The function genalg() uses a continuous genetic algorithm to minimize a
4 %k-dimensional function f using tournament selection, mating-heuristic
5 %crossover, 2 offspring per parents, only different parents and elitism.
6 %Input Variables:
7 %pop... population size, stays the same over generations
8 %sel... selection rate
9 %mut... mutation rate
10 %gen... generations
11 %f... k-dimensional function (via function handle)
12 %f.e. f=@(x,y) 20+x.^2+y.^2-10*cos(2*pi.*x)-10*cos(2*pi.*y);
13 %parb... parameter boundaries [upperboundary(1) upperboundary(2) ... ;
14 %lowerboundary(1) lowerboundary(2) ...], f.e. [5 5;-5 -5]
15 %-----
16 fpar=nargin(f);%number of function parameters
17 gesm=zeros(pop,gen*fpar); %total matrix with all generations
18 inpop=rand(pop,fpar);%random initial population, dimension: pop X fpar
19
20 %bring initial population random numbers inbetween the variable boundaries
21 for k=1:fpar
22     inpop(:,k)=inpop(:,k)*(parb(1,k)-parb(2,k))+parb(2,k);
23 end
24
25 %evaluate the fitness of each chromosome in the initial population
26 for k=1:pop
27     c=cell(1,fpar);
28     for j=1:fpar
29         c{j}=inpop(k,j);
30     end
31     inpop(k,fpar+1)=f(c{:});
32 end
33 gesm(:,1:fpar)=inpop(:,1:fpar); %initial population into total matrix
34
35 %sort population matrix due to the costs of the chromosomes (1st
36 %chromosome.. lowest costs) and delete chromosomes with higher costs
37 [s1,s2]=sort(inpop(:,fpar+1));
38 inpop=inpop(s2,:);
39
40 ofsnm=pop-ceil(pop*(1-sel)); %number of offspring
41 offs=zeros(ofsnm,fpar); %offspring place holder
42 parnum=ceil(ofsnm/2); %number of parents
43 parents=zeros(parnum,2); %parents place holder
44
45 %generation loop
46 for i=1:gen-1 % "-1" due to the initial generation
47
48     %random selection of parents

```

```

49  for k=1:parnum
50      tosu=zeros(1,2);
51      while tosu(1,1)==tosu(1,2) %only different parents are allowed
52          tosu=sort(ceil(rand(1,2)*ceil(pop*(1-sel))));
53      end
54      parents(k,:)=tosu(1:2);
55  end
56
57  %mating
58  for j=1:ofsnum
59      par=ceil(j/2); %parents
60      for k=1:fpar
61          count=0; %at least once a new random number should be generated
62          while offs(j,k)>parb(1,k) || offs(j,k)<parb(2,k) || count==0 %
63              values should be between boundaries
64              offs(j,k)=rand(1)*(inpop(parents(par,1),k)-inpop(parents(par,2),k))+inpop(parents(par,2),k);
65              count=1;
66          end
67      end
68      inpop((pop-ofsnum+1):pop,1:fpar)=offs(:,j); %add offspring to population
69
70  %calculate the costs of the offspring
71  for k=pop-ofsnum+1:pop
72      c=cell(1,fpar);
73      for j=1:fpar
74          c{j}=inpop(k,j);
75      end
76      inpop(k,fpar+1)=f(c{:});
77  end
78  %sort population matrix
79  [s1,s2]=sort(inpop(:,fpar+1));
80  inpop=inpop(s2,:);
81
82  %mutation
83  for j=1:ceil((pop-1)*fpar*mut)
84      %"-1" .. due to elitism, the "best" chromosome won't be mutated
85      rw=ceil(rand(1)*(pop-1))+1; %"best" one is in the first row
86      cw=ceil(rand(1)*fpar);
87      inpop(rw,cw)=rand(1)*(parb(1,cw)-parb(2,cw))+parb(2,cw);
88      %actualize the costs only of the mutated chromosome
89      c=cell(1,fpar);
90      for k=1:fpar
91          c{k}=inpop(rw,k);
92      end
93      inpop(rw,fpar+1)=f(c{:});
94  end
95
96  %sort population matrix
97  [s1,s2]=sort(inpop(:,fpar+1));

```

```

98     inpop=inpops(s2,:);
99
100     %put the population in the totalmatrix
101     gesm(:,1+i*fpar:fpar+i*fpar)=inpops(:,1:fpar);
102 end
103 opt=min(inpops(:,fpar+1)); %the optimal function value
104 end

```

A.2 Genetic Algorithm - Example 2

```

1 function [optval]=gatradsys(pop,sel,mut,gen,parb,btime,schem)
2 %-----Explanation of the function:-----
3 %Save the gatradsys.m,fMA.m and the GBPUSD.xlsx (open,high,low,close price) in
   the same folder.
4
5 %The function gatradsys() uses the continuous Genetic Algorithm to optimize a
   MA-Trading System with stoploss (Evaluation of the MA-Trading System with
   the fMA() function) using the GBPUSD values 01/2000–02/2017. The function
   will therefore evaluate for a predefined population size an initial
   population randomly, with respect to the input parameters (the parameter
   boundaries of the two MA's and of the stoploss). After the initial
   population, the Genetic Algorithm will iteratively try to find an optimal
   combination of short-,long-MA and stoploss to maximize the rate of return
   of the cost function (here: fMA() (Trading System))
6 %The Genetic Algorithm uses tournament selection, a mating-heuristic crossover
   , 2 offspring per parents, only different parents allowed and elitism (the
   combination with the currently optimal values will always stay in the
   population and is not going to mutate)
7 %You can choose if the selection and mutation rate stays the same over all
   iterations or if they get smaller (linearly or negative logarithmically).
8
9 %input Variables:
10 %-pop... population size, stays the same over generations (integer value)
11 %-sel... selection rate (between 0 and 1) (the percentage of population which
   is going to be changed each generation) (the selection rate itself will be
   changed or not, due to the used scheme. If yes the last/lowest value will
   be 1/pop.)
12 %-mut... mutation rate (between 0 and 1) (the percentage of population which
   will be mutated randomly) (the mutation rate itself will be changed or
   not, due to the used scheme. If yes the last/lowest value will be 1/pop.)
13 %-gen... generations (integer value)
14 %-parb... parameter boundaries [upperboundaryofMA1 upperboundaryofMA2
   upperboundaryofstoploss; lowerboundaryofMA1 lowerboundaryofMA1
   lowerboundaryofstoploss] (boundaries of the lower MA (MA1) and the higher
   MA (MA2) which will be used to generate the population – 2x3matrix); f.e.
   [100 100 1;1 1 0] (1<=MA1<=100, 1<=MA2<=100, 0<=stoploss<=1, boundaries of
   the MA's should be integers and the boundaries of the stoploss numbers
   inbetween 0 and 1, and take care that the upperboundary is higher than the
   lower boundary. Additionally the upperboundary of MA1 should be smaller
   than the upperboundary of the MA2)

```

```

15 %-btttime.. backtesttime (integer value > (upperboundaryMA1+upperboundaryMA2)
    and smaller than the total amount of values in the used time serie, here
    GBPUSD with 4463 values) (The backtesttime is the number of values which
    is going to be used to optimize our Trading system, f.e. if btttime=500 the
    function uses the first 500 values to optimize the Trading System. After
    that, the function will evaluate how the optimum, based on the btttime,
    would have worked out on the rest of the timeserie (starting with the (
    btttime+1)th value).
16 %-schem... selection-/mutation rate schemes (input: 1,2 or 3), you can choose
    between 3 schemes: 1. s-/m rate stays the same over all generations, 2. s
    -/m rate decreases linearly over all generations, 3. s-/m rate decreases
    negativ logarithmically over all generations
17
18 %example input for the Matlab Comand window:
19 %gatradsys(100,0.25,0.05,100,[100 100 0.05;1 1 0.01],500,1)
20 %-----
21
22 %timeseries of GBPUSD 01/2000-02/2017 (open, high, low, close)
23 ts=xlsread('GBPUSD.xlsx');
24 minsm=1/pop; %value of selection rate and mutation rate in the last geneartion
    if the selection and mutation don't stay the same over all generations.
    The value has to be greater than 1/pop, if it's smaller the last
    generations will be the same.
25
26 %selection-/mutation rate vector dependent on the scheme
27 switch schem
28     case 1
29         %selection and mutation rate stay the same over all generations
30         sel=ones(gen-1)*sel;
31         mut=ones(gen-1)*mut;
32     case 2
33         %selection and mutation rate decrease linearly over all generations
            until they receive the value minsm
34         sel=linspace(sel,minsm,gen-1);
35         mut=linspace(mut,minsm,gen-1);
36     case 3
37         %selection and mutation rate decrease logarithmically over all
            generations until they receive the value minsm
38         sel=logspace(log(sel)/log(10),log(minsm)/log(10),gen-1);
39         mut=logspace(log(mut)/log(10),log(minsm)/log(10),gen-1);
40 end
41
42 fpar=3; %to be optimized parameters: shortMA, longMA and stoploss
43 gesm=zeros(pop,gen*fpar); %placeholder for the population of all generations
44 inpop=rand(pop,fpar);%random initial population
45
46 %bring initial population random numbers inbetween the variable boundaries
47 for k=1:fpar
48     inpop(:,k)=inpop(:,k)*(parb(1,k)-parb(2,k))+parb(2,k);
49     %the second MA has to be greater than the first MA
50     if k==2

```

```

51         for j=1:pop
52             while inpop(j,k)<=inpop(j,k-1)
53                 inpop(j,k)=rand(1)*(parb(1,k)-inpop(j,1))+inpop(j,1);
54             end
55         end
56     end
57     if k==1 || k==2
58         inpop(:,k)=ceil(inpop(:,k));
59     end
60 end
61
62 %evaluate the costs (evaluate the rate of return of the random parameters) of
    each chromosome of the initial population
63 for k=1:pop
64     c=cell(1,fpar+4);
65     for j=1:fpar
66         c{j}=inpop(k,j);
67     end
68     c{4}=bttime;
69     c{5}=0;
70     c{6}=0;
71     c{7}=ts;
72     inpop(k,fpar+1)=fMA(c{:}); %rate of return evaluated in the fMA function
73 end
74 gesm(:,1:fpar)=inpop(:,1:fpar); %initial population into totalmatrix
75 ofsnum=pop-ceil(pop*(1-sel)); %number of offspring needed, evaluated due to
    the selection rate
76 parnum=ceil(ofsnum/2); %number of parents
77
78 %generation loop
79 for i=1:gen-1 % "-1" due to the initial generation
80     %sort population matrix due to the costs ("-rate of return") of the
        chromosomes (1st chromosome.. lowest costs) and delete chromosomes
        with higher costs (selection rate)
81     [s1,s2]=sort(-inpop(:,fpar+1));
82     inpop=inpop(s2,:);
83     inpop=inpop(1:ceil(pop*(1-sel(i))),:);
84     parents=zeros(parnum(i),2); %parents
85     offs=zeros(ofsnum(i),fpar); %zeromatrix for the offspring
86
87     %random selection of parents
88     for k=1:parnum(i)
89         tosu=zeros(1,2);
90         while tosu(1,1)==tosu(1,2) %only different parents are allowed
91             tosu=sort(ceil(rand(1,3)*ceil(pop*(1-sel(i)))));
92         end
93         parents(k,:)=tosu(1:2);
94     end
95
96     %mating
97     for j=1:ofsnum(i)

```

```

98     par=ceil(j/2); %parents
99     for k=1:fpar
100         count=0; %at least once a new random number should be generated
101         if (count==0 && k==1) || (count==0 && k==3)
102             offs(j,k)=rand(1)*(inpop(parents(par,1),k)-inpop(parents(par,2),k))+inpop(parents(par,2),k);
103             if k==1
104                 offs(j,k)=ceil(offs(j,k));
105             end
106         end
107         while (count==0 && k==2) || (k==2 && offs(j,k)< offs(j,k-1))
108             if offs(j,k)==offs(j,k-1)
109                 offs(j,k)=offs(j,k)+1;
110             end
111             offs(j,k)=ceil(rand(1)*(inpop(parents(par,1),k)-inpop(parents(par,2),k))+inpop(parents(par,2),k));
112             count=1;
113         end
114     end
115 end
116 inpop((pop-ofsnum(i)+1):pop,1:fpar)=offs(:, :); %add offspring to
    population
117
118 %calculate the costs of the offspring
119 for k=pop-ofsnum(i)+1:pop
120     c=cell(1,fpar+4);
121     for j=1:fpar
122         c{j}=inpop(k,j);
123     end
124     c{4}=bttime;
125     c{5}=0;
126     c{6}=0;
127     c{7}=ts;
128     inpop(k,fpar+1)=fMA(c{:});
129 end
130
131 %sort the new population matrix
132 [s1,s2]=sort(-inpop(:,fpar+1));
133 inpop=inpop(s2,:);
134
135 %mutation
136 for j=1:ceil((pop-1)*fpar*mut(i))
137     %"-1" .. due to elitism, the "best" chromosome won't be mutated
138     rw=ceil(rand(1)*(pop-1))+1; % +1.."best" one is in the first row -
        elitism
139     cw=ceil(rand(1)*fpar);
140
141     if cw==3 inpop(rw,cw)=rand(1)*(parb(1,cw)-parb(2,cw))+parb(2,cw); end
142     if cw==1 inpop(rw,cw)=floor(rand(1)*(inpop(rw,cw+1)-parb(2,cw))+parb(2,
        cw)); end

```

```

143         if cw==2 inpop(rw,cw)=ceil(rand(1)*(parb(1,cw)-inpop(rw,cw-1))+inpop(rw
            ,cw-1));end
144
145         %actualize the costs only of the mutated chromosome
146         c=cell(1,fpar+4);
147         for k=1:fpar
148             c{k}=inpop(rw,k);
149         end
150         c{4}=bttime;
151         c{5}=0;
152         c{6}=0;
153         c{7}=ts;
154         inpop(rw,fpar+1)=fMA(c{:});
155     end
156
157     %put each population in the totalmatrix
158     gesm(:,1+i*fpar:fpar+i*fpar)=inpop(:,1:fpar);
159 end
160 %sort the population matrix
161 [s1,s2]=sort(-inpop(:,fpar+1));
162 inpop=inpop(s2,:);
163
164 optval=inpop(1,fpar+1); % optimal rate of return
165
166 %Plot the equity curve of the backtested time period and of the out of sample
    time period of the optimal evaluated parameters
167 fMA(inpop(1,1),inpop(1,2),inpop(1,3),bttime,0,1,ts);
168 fMA(inpop(1,1),inpop(1,2),inpop(1,3),bttime,1,0,ts);
169 end

```

```

1 function [rend]=fMA(MA1,MA2,stoploss,bttime,oosyn,isyn,ts)
2 %-----Explanation of the function:-----
3 %The function fMA() evaluates the rate of return of the Moving Average rule.
4 %In this strategy buy and sell signals are determined when the short MA
    crosses the long MA which indicates for price trends.
5 %When the short moving average exceeds the long moving average an upward trend
    is expected to be initiated and the trader takes a long position and vice
    versa.
6 %Additionally the system includes a moving stoploss, which means that stoploss
    value will be actualized permanently on the current stock value depending
    if it is a long or short position.
7
8 %input Variables:
9 %-MA1.. short MA (integer value)
10 %-MA2.. long MA (has to be longer than the shortMA) (integer value)
11 %-stoploss.. (value between 0 and 1)
12 %-bttime.. backtesttime (integer value > (upperboundaryMA1+upperboundaryMA2)
    and smaller than the total amount of values in the used time serie, here
    GBPUSD with 4463 values) (The backtesttime is the number of values which
    is going to be used to optimize our Trading system, f.e. if bttime=500 the
    function uses the first 500 values to optimize the Trading System. After

```

```

    that, the function will evaluate how the optimum, based on the btttime,
    would have worked out on the rest of the timeserie (starting with the (
    btttime+1)th value).
13 %oosyn.. validation time analysis (yes or no (1 or 0)) (If the value is 1,
    the function will evaluate the Trading system on the validation time and
    plots the equity curve.
14 %isyn.. backtesting time analysis (yes or no (1 or 0)) (If the value is 1,
    the function will evaluate the Trading system on the backtesting time and
    then plots the quity curve of it.
15 %If oosyn=isyn=0 the function will evaluate the Trading system on the
    backtesting time and returns the total rate of return.
16 %-----
17
18 %time series of daily GBPUSD, 4463 daily opening/high/low/closing prices ,
    01/2000-02/2017
19 cap=100; %start capital
20 mam=zeros(btttime-MA1+1+oosyn*(-2*btttime+length(ts)),3);% placeholder for the
    MA matrix with short MA(MA1) of the current unit in the first coloumn,
    long MA(MA2) of the current unit in the second coloumn and the current
    trading position: shortpos(-1) or longpos(1) or nopos(0). (in 3rd column)
21
22 %evaluate MA1 and MA2
23 for j=1:size(mam,1)
24     mam(j,1)=ts(j+oosyn*btttime,4);
25     for k=1:MA1-1
26         mam(j,1)=mam(j,1)+ts(j+k+oosyn*btttime,4);
27     end
28     mam(j,1)=mam(j,1)/MA1;
29     if j>MA2-MA1
30         mam(j,2)=ts(j-MA2+MA1+oosyn*btttime,4);
31         for k=1:MA2-1
32             mam(j,2)=mam(j,2)+ts(j+k-MA2+MA1+oosyn*btttime,4);
33         end
34         mam(j,2)=mam(j,2)/MA2;
35     end
36 end
37
38 %evaluate current trading position: MA<MA2=> sell(-1), MA>MA2=> buy(1), the
    first MA2-MA1 values will be zero.
39 %signal with closing prices(t), execution with opening prices(t+1)
40 mam(2:size(mam,1),3)=sign(mam(1:size(mam,1)-1,1)-mam(1:size(mam,1)-1,2));
41 mam(1:MA2-MA1,3)=0;
42
43 %tradingsystem
44 curpos=mam(MA2-MA1+1,3); %current trading position: long(1), short(-1), hold
    (0)
45 stopval=ts(MA2+1+oosyn*btttime,1)*(1-curpos*stoploss); %trailing stoploss value
46 lastpos=0; %last trading position
47 lastcurpos=0; %trading postion of the day before
48 ec=zeros(btttime-MA2+oosyn*(-2*btttime+length(ts)),1);%equity curve place holder

```

```

49 %Evaluate the equity curve values due to the generated trading system and
   trailing stoploss
50 for j=MA2-MA1+1+oosyn*bttime:bttime-MA1+1+oosyn*(length(ts)-bttime-1)
51     stophit=0;
52     %after hitting the stopval a new position will only be opened if the new
   position is different to the last one (1 <-> -1)
53     if curpos==0 && mam(j-oosyn*bttime,3)~=lastpos
54         lastpos=0;
55         curpos=mam(j-oosyn*bttime,3);
56         stopval=ts(j+MA1,1)*(1-curpos*stoploss);
57     end
58
59     %position change from 1 <-> -1
60     if mam(j-oosyn*bttime,3)~=curpos && curpos~=0
61         lastpos=curpos;
62         curpos=mam(j-oosyn*bttime,3);
63         stopval=ts(j+MA1,1)*(1-curpos*stoploss);
64     end
65
66     %if the position stays the same, just check if the value is under/over the
   stopval and if the stopval of the new value is bigger or smaller than
   the old stopval then eventually actualize it.
67     %We therefore use the closing prices.
68     if mam(j-oosyn*bttime,3)==curpos && curpos~=0
69         if (ts(j+MA1,3)<stopval && curpos==1) || (curpos==-1 && ts(j+MA1,2)>
   stopval)
70             lastpos=curpos;
71             stophit=1;
72         end
73
74         if (ts(j+MA1,4)*(1-curpos*stoploss)>stopval && curpos==1 && stophit
   ==0) || (curpos==-1 && ts(j+MA1,4)*(1-curpos*stoploss)<stopval &&
   stophit==0)
75             stopval=ts(j+MA1,4)*(1-curpos*stoploss);
76         end
77     end
78
79     if stophit==0
80         %position stays the same
81         if lastcurpos==mam(j-oosyn*bttime,3)
82             cap=cap*(sqrt((curpos-1)^2)+curpos*ts(j+MA1,4)/ts(j+MA1-1,4));
83         end
84         %position change from 0 to +1 or 1<->-1
85         if lastcurpos~=mam(j-oosyn*bttime,3)
86             if lastcurpos==0
87                 cap=cap*(sqrt((curpos-1)^2)+curpos*ts(j+MA1,4)/ts(j+MA1,1));
88             end
89             if lastcurpos~=0
90                 cap=cap*(sqrt((lastcurpos-1)^2)+lastcurpos*ts(j+MA1,1)/ts(j+MA1
   -1,4));

```

```

91         cap=cap*(sqrt((mam(j-oosyn*bttime,3)-1)^2)+mam(j-oosyn*bttime,3)*
           ts(j+MA1,4)/ts(j+MA1,1));
92     end
93 end
94 end
95 %stopval hit-> stopval as closing price of trade
96 if stophit==1
97     cap=cap*(sqrt((lastcurpos-1)^2)+lastcurpos*ts(j+MA1,1)/ts(j+MA1-1,4));
98     cap=cap*(sqrt((curpos-1)^2)+curpos*stopval/ts(j+MA1,1));
99     curpos=0;
100 end
101 ec(j-MA2+MA1-oosyn*bttime,1)=cap/100;
102 lastcurpos=curpos;
103 end
104
105 %drawdown length and drawdown percentage evaluation and equity curve plot,
    only if isyn=1 or oosyn=1
106 if isyn==1 || oosyn==1
107     maxddl=0;%maxdrawdownlength in days
108     maxddp=1;%maxdrawdownpercent [-1,0]
109     lastk=1; %if we already have found a maxdraddownlength from i to k, then
        there won't be a bigger one in this part of the equity curve
110     maxwert=1;
111     for i=1:bttime-MA2-1+oosyn*(-2*bttime+length(ts))
112         for k=i:bttime-MA2+oosyn*(-2*bttime+length(ts))
113             if (ec(k,1)/ec(i,1)-1)<maxddp && i>=maxwert
114                 maxddp=ec(k,1)/ec(i,1)-1;
115                 if ec(k,1)>ec(i,1)
116                     maxwert=k;
117                 end
118             end
119             if (ec(i,1)<=ec(k,1) && i>=lastk) || (ec(i,1)>ec(k,1) && k==bttime-
                MA2-1+oosyn*(-2*bttime+length(ts)) && i>=lastk)
120                 if k-i>maxddl
121                     maxddl=k-i;
122                 end
123                 lastk=k;
124             end
125         end
126     end
127
128 %Plot the equity curve
129 subplot(1,2,1+oosyn)
130 plot(ec(1:bttime-MA2+oosyn*(-2*bttime+length(ts))))
131 text(size(mam,1)*0.1,2.5,{['Rendite: ', num2str((ec(bttime-MA2+oosyn*(-2*
    bttime+length(ts)))-1)*100), '%']; ['max. Drawdown Duration: ', num2str(
    maxddl)]; ['max. Drawdown Percentage: ', num2str(maxddp*100), '%']; ['MA1:
    ', num2str(MA1), ' Tage']; ['MA2: ', num2str(MA2), ' Tage']; ['Stoploss: ',
    num2str(stoploss*100), '%']; ['Number of trades(buy and sell is one
    trade): ', num2str(ceil(sum(sign(abs(diff(mam(:,3))))/2))]}))
132 legend('Equitiy Curve')

```

```

133     axis([0 length(ec) 0.5 3])
134 end
135 if isyn==1
136     title(['Backtest ',num2str(bttime),' days']);
137 end
138 if oosyn==1
139     title(['Out of sample data analyse, ',num2str(length(ts)-bttime),' days'
140           ]);
141 end
142 %if isyn=0 and oosyn=0 then return the total rate of return.
143 if isyn==0 && oosyn==0
144     rend=(ec(bttime-MA2+oosyn*(-2*bttime+length(ts)))-1)*100;
145 end
146 end

```

A.3 Grammatical Evolution - Example 1

```

1 function [oe,foe,t] = GEp2matnumeric2(pop,sel,mut,gen,schem,numb,maxf,bttime,
    crossover)
2 %-----Explanation of the function:-----
3 %Save functions 'newpop2matnumeric2.m' and 'bi2de2vec.m' in the same folder!
4
5 %The function GEp2matnumeric2() uses the Grammatical Evolution to find the
    optimal grammar in the backtest data. In this example data of 4 vectors
    with size 100 of the parameters caustic soda flow, organic acid flow,
    temperature and pH are used. The aim ist to find the effect of the first
    three parameter on pH.
6
7 %BNF (Backus-Naur form) is defined as
8 %e.. expressions
9 %b.. binary operators
10 %u.. unitary operators
11 %v.. variables (var0.. caustic soda flow, var1.. organic acid flow, var 2..
    temperature)
12 %y.. modified version of the Digit Concatenation
13
14 %N={e,b,u,v,y}
15 %T={sin,cos,exp,log,sqrt,+,-,*,/}
16 %S={e}
17 %P={e,b,u,v,y}
18
19 % b={'+', '-', '*', '/'};
20 % u={'sin', 'cos', 'exp', 'log', 'sqrt'};
21 % v={'var0', 'var1', 'var2', '<y>'};
22 % e={'<e> <b> <e>', '(<e> <b> <e>)', '<u>(<e>)', '<v>'};
23 % y={'<x<x<x<x>.<x<x<x<x>', '<x<x<x>.<x<x<x<x>', '<x>.<x<x<x<x>', '<x<x<x<x>.<x<x<x
    >', '<x<x<x>.<x<x<x>', '<x>.<x<x<x>', '<x<x<x<x>.<x>', '<x<x<x>.<x>', '<x>.<x>'};
24
25 %The GE uses random selection, 2 offspring per parents, only different parents
    allowed and elitism (the combination with the currently optimal values

```

```

    will stay in the population and is not going to mutate)
26
27 %input Variables:
28 %-pop... population size, stays the same over generations (integer value)
29 %-sel... selection rate (between 0 and 1) (the percentage of population which
    is going to be changed each generation) (the selection rate itself will be
    changed or not, due to the used scheme. If yes the last/lowest value will
    be 1/pop.)
30 %-mut... mutation rate (between 0 and 1) (the percentage of population which
    will be mutated randomly) (the mutation rate itself will be changed or
    not, due to the used scheme. If yes the last/lowest value will be 1/pop.)
31 %-gen... generations (integer value)
32 %-schem... selection-/mutation rate schemes (input: 1,2 or 3), you can choose
    between 3 schemes: 1. s-/m rate stays the same over all generations, 2. s
    -/m rate decreases linearly over all generations, 3. s-/m rate decreases
    negativ logarithmically over all generations
33 %-numb... number of binaries that will be generated for each chromosome in the
    GE, if the algorithm needs more numbers it will start with the first
    again. 8 bits per codon -> integer number between 0 and 255.
34 %-maxf... maximal number of used functions in the grammar (in matlab max 32)
35 %-btttime.. backtesttime (integer value smaller than the total amount of values
    in the used time serie, here pH with 100 values)
36 %-crossover.. which crossover method should be used.. 1: 1-point crossover, 2:
    2-point crossover
37 %-----
38
39 %100x4 matrix with values of (Caustic soda flow, Organic acid flow, Temperature
    , pH)
40 dat=xlsread('pH.xlsx');
41 v0=dat(:,1);
42 v1=dat(:,2);
43 v2=dat(:,3);
44 v3=dat(:,4);
45
46 minsm=1/pop; %minimum value of selection rate and mutation rate
47 switch schem
48     case 1
49         %selection and mutation rate stay the same over all generations
50         sel=ones(gen-1)*sel;
51         mut=ones(gen-1)*mut;
52     case 2
53         %selection and mutation rate decrease linearly over all generations
            until they receive the value minsm
54         sel=linspace(sel,minsm,gen-1);
55         mut=linspace(mut,minsm,gen-1);
56     case 3
57         %selection and mutation rate decrease logarithmically over all
            generations until they receive the value minsm
58         sel=logspace(log(sel)/log(10),log(minsm)/log(10),gen-1);
59         mut=logspace(log(mut)/log(10),log(minsm)/log(10),gen-1);
60 end

```

```

61
62 %BNF
63 b={'+', '-', '*', '/'};
64 u={'sin', 'cos', 'exp', 'log', 'sqrt'};
65 v={'var0', 'var1', 'var2', '<y>'};
66 e={'<e> <b> <e>', '(<e> <b> <e>)', '<u>(<e>)', '<v>'};
67 y={'<x><x><x>.<x><x><x>', '<x><x>.<x><x><x>', '<x>.<x><x><x>', '<x><x><x>.<x><x>',
    '<x><x>.<x><x>', '<x>.<x><x>', '<x><x><x>.<x>', '<x><x>.<x>', '<x>.<x>'};
68 s=['<e>'];
69 lb=size(b,2);
70 lu=size(u,2);
71 lv=size(v,2);
72 le=size(e,2);
73 ly=size(y,2);
74
75
76 %Initial/Actual Population
77 actmat=[]; %place holder - matrix with all decimal numbers of binaries
78 acterr=[]; %place holder - error vectors
79 actnum=[]; %place holder - number of used binaries
80 [actmat, acterr, actnum]=newpop2matnumeric2([1:pop],1,numb,actmat,acterr,actnum,
    s,maxf,e,b,u,v,y,le,lb,lu,lv,ly,v0,v1,v2,v3,bttime);
81 ofsnum=pop-ceil(pop*(1-sel)); %number of offspring needed
82 parnum=ceil(ofsnum/2); %number of parents
83
84 %sort population matrix due to the costs ("sum of errors") of the chromosomes
    (1st chromosome.. lowest costs)
85 [acterr,s2]=sort(acterr);
86 actmat=actmat(s2,:);
87 actnum=actnum(s2);
88 best(1)=acterr(1);
89
90 %generation loop
91 for i=1:gen-1 % "-1" due to the initial generation
92     parents=zeros(parnum(i),2); %parents
93
94     %random selection of parents
95     for k=1:parnum(i)
96         tosu=zeros(1,2);
97         while tosu(1,1)==tosu(1,2) %only different parents are allowed
98             tosu=sort(ceil(rand(1,3)*ceil(pop*(1-sel(i)))));
99         end
100     parents(k,:)=tosu(1:2);
101 end
102
103 %mating
104 for j=1:ofsnum(i)
105     par=ceil(j/2); %parents
106     r=mod(j,2);
107     switch crossover
108         case 1

```

```

109         %1 point crossover
110         if r==1
111             zf=min(actnum(parents(par,1)),actnum(parents(par,2)))-1;
112             z=ceil(rand(1)*zf)+1;
113         end
114         %generate new offspring with 1-point crossover of parents
115         actmat(j+pop-ofsnum(i),1:z)=actmat(parents(par,1+r*1),1:z);
116         actmat(j+pop-ofsnum(i),z+1:numb)=actmat(parents(par,2-r*1),z
            +1:numb);
117     case 2
118         %2 point crossover
119         if r==1
120             zf=min(actnum(parents(par,1)),actnum(parents(par,2)))-1;
121             if zf~=0 && zf~=1 %if zf could be 0,1 a 2 point crossover
                is senseless.
122                 z(1)=ceil(rand(1)*zf)+1;
123                 z(2)=z(1);
124                 while z(2)==z(1)
125                     z(2)=ceil(rand(1)*zf)+1;
126                 end
127                 z=sort(z);
128             end
129         end
130         %generate new offspring with 2-point crossover of parents
131         if zf~=0 && zf~=1
132             actmat(j+pop-ofsnum(i),1:z(1))=actmat(parents(par,1+r*1)
                ,1:z(1));
133             actmat(j+pop-ofsnum(i),z(1)+1:z(2))=actmat(parents(par,2-r
                *1),z(1)+1:z(2));
134             actmat(j+pop-ofsnum(i),z(2)+1:numb)=actmat(parents(par,1+r
                *1),z(2)+1:numb);
135         end
136         if zf==0 || zf==1
137             actmat(j+pop-ofsnum(i),:)=actmat(parents(par,1+r*1),:);
138         end
139     end
140 end
141
142 %calculate the costs of the offspring
143 [actmat,acterr,actnum]=newpop2matnumeric2([pop-ofsnum(i)+1:pop],0,numb,
    actmat,acterr,actnum,s,maxf,e,b,u,v,y,le,lb,lu,lv,ly,v0,v1,v2,v3,
    btttime);
144
145 %sort the new population matrix
146 [acterr,s2]=sort(acterr);
147 actmat=actmat(s2,:);
148 actnum=actnum(s2);
149
150 %mutation
151 for j=1:ceil((pop-1)*mut(i))
152     %"-1" .. due to elitism, the "best" chromosome won't be mutated

```

```

153     rw=ceil(rand(1)*(pop-1))+1; % +1.."best" one is in the first row -
        elitism.. rw is in [2,pop]
154     decpos=ceil(rand(1)*actnum(rw));
155     actmat(rw,decpo)= bi2de2vec(transpose(round(rand(8,1))));
156
157     %actualize the costs only of the mutated chromosome
158     [actmat,acterr,actnum]=newpop2matnumeric2(rw,0,numb,actmat,acterr,
        actnum,s,maxf,e,b,u,v,y,le,lb,lu,lv,ly,v0,v1,v2,v3,bttime);
159 end
160 best(i+1)=acterr(1);
161 [acterr,s2]=sort(acterr);
162 actmat=actmat(s2,:);
163 actnum=actnum(s2);
164 end
165
166 %generate optimal Grammar (same code as in newpop2matnumeric, only for row
    number 1)
167 G=s;
168 binumb=0;
169 funcnumb=0;
170 while length(strfind(G,'<'))>0 & funcnumb<maxf
171     binumb=mod(funcnumb,numb)+1;
172     switch G(findstr(G,'<')+1:findstr(G,'>')-1)
173         case 'e'
174             G=regexprep(G,'<e>',e{mod(actmat(1,binumb),le)+1},'once');
175         case 'b'
176             G=regexprep(G,'<b>',b{mod(actmat(1,binumb),lb)+1},'once');
177         case 'u'
178             G=regexprep(G,'<u>',u{mod(actmat(1,binumb),lu)+1},'once');
179         case 'v'
180             G=regexprep(G,'<v>',v{mod(actmat(1,binumb),lv)+1},'once');
181         case 'y'
182             G=regexprep(G,'<y>',y{mod(actmat(1,binumb),ly)+1},'once');
183         case 'x'
184             G=regexprep(G,'<x>',num2str(actmat(1,binumb)),'once');
185     end
186     funcnumb=funcnumb+1;
187 end
188 G=regexprep(G,'<e>','<v>');
189 while length(strfind(G,'<'))>0
190     binumb=mod(funcnumb,numb)+1;
191     switch G(findstr(G,'<')+1:findstr(G,'>')-1)
192         case 'b'
193             G=regexprep(G,'<b>',b{mod(actmat(1,binumb),lb)+1},'once');
194         case 'u'
195             G=regexprep(G,'<u>',u{mod(actmat(1,binumb),lu)+1},'once');
196         case 'v'
197             G=regexprep(G,'<v>',v{mod(actmat(1,binumb),lv)+1},'once');
198         case 'y'
199             G=regexprep(G,'<y>',y{mod(actmat(1,binumb),ly)+1},'once');
200         case 'x'

```

```

201         G=regexprep(G, '<x>', num2str(actmat(1,binumb)), 'once');
202     end
203     funcnumb=funcnumb+1;
204 end
205 optG=G;
206
207 %-----v3(1:bttime)-----
208 %compare estimated var3 and var3
209 for i=1:bttime
210     var0=v0(i);
211     var1=v1(i);
212     var2=v2(i);
213     esvar3(i)= eval(optG);
214 end
215 %sum of absolut errors of the final Grammar
216 oe=sum(abs(v3(1:bttime)-transpose(esvar3)));
217 %vector of quadratic errors of the final Grammar
218 diff=abs(v3(1:bttime)-transpose(esvar3));
219 %matrix of estimated var 3, real var3 and diff
220 DIFF=[transpose(esvar3) v3(1:bttime) diff];
221
222 %-----v3(bttime+1:100)-----
223 %compare estimated var 3 and real var3
224 for i=1:100-bttime
225     var0=v0(i+bttime);
226     var1=v1(i+bttime);
227     var2=v2(i+bttime);
228     fesvar3(i)=eval(optG);% eval(optG);
229 end
230 %sum of absolut errors of the final Grammar
231 foe=sum(abs(v3(bttime+1:100)-transpose(fesvar3)));
232 %vector of absolut errors of the final Grammar
233 fdiff=abs(v3(bttime+1:100)-transpose(fesvar3));
234 %matrix of estimated var 3, real var3 and fdiff
235 FDIFF=[transpose(fesvar3) v3(bttime+1:100) fdiff];
236
237 %plot of equity curves
238 figure
239 subplot(1,2,1)
240 hold on
241 plot(1:bttime,DIFF(1:bttime,1),'red','DisplayName','optimized grammar values')
242 plot(1:bttime,DIFF(1:bttime,2),'blue','DisplayName','objective values')
243 axis([0 bttime -inf inf])
244 xlabel('data')
245 title('backtesting data')
246 legend('show')
247 hold off
248
249 subplot(1,2,2)
250 hold on

```

```

251 plot(1+bttime:100,FDIFF(1:100-bttime,1),'green','DisplayName','grammar values'
    )
252 plot(1+bttime:100,FDIFF(1:100-bttime,2),'blue','DisplayName','objective values'
    )
253 axis([100-bttime 100 -inf inf])
254 xlabel('data')
255 title('validation data')
256 legend('show')
257 hold off
258 end

```

```

1 function [ actmat,acterr,actnum ] = newpop2matnumeric2(kwerte,binneu,numb,
    actmat,acterr,actnum,s,maxf,e,b,u,v,y,le,lb,lu,lv,ly,v0,v1,v2,v3,bttime)
2 %-----Explanation of the function:-----
3 %The function newpop2matnumeric2() evaluates the grammar and the absolute sum
    of errors of our example with this grammar
4
5 %input Variables:
6 %all variables are the same as in the GEp2matnumeric2() function, only the
    following two are different:
7 %kwerte.. vector of rowvalues, which should be evaluated.
8 %binneu.. binary, should the algorithm create new binary values or not.
9 %-----
10
11 for k=kwerte
12     funcnumb=0; %number of used functions in the grammar
13     binumb=0; %number of used binaries, start from the first again after you'
        ve reached the last one.
14
15     if binneu==1
16         %generate random binary numbers bn and evaluate the integer numbers
17         bn=transpose(round(rand(8*numb,1)));
18         for i=1:numb
19             bi=bn(1+(i-1)*8:i*8);
20             an(i)=bi2de2vec(bi);
21         end
22         actmat(k,:)=an;
23     end
24
25     %generate the grammar G
26     G=s;
27     while length(strfind(G,'<'))>0 & funcnumb<maxf
28         binumb=mod(funcnumb,numb)+1;
29         switch G(findstr(G,'<')+1:findstr(G,'>')-1)
30             case 'e'
31                 G=regexprep(G,'<e>',e{mod(actmat(k,binumb),le)+1},'once');
32             case 'b'
33                 G=regexprep(G,'<b>',b{mod(actmat(k,binumb),lb)+1},'once');
34             case 'u'
35                 G=regexprep(G,'<u>',u{mod(actmat(k,binumb),lu)+1},'once');
36             case 'v'

```

```

37         G=regexprep(G, '<v>', v{mod(actmat(k, binumb), lv)+1}, 'once');
38     case 'y'
39         G=regexprep(G, '<y>', y{mod(actmat(k, binumb), ly)+1}, 'once');
40     case 'x'
41         G=regexprep(G, '<x>', num2str(actmat(k, binumb)), 'once');
42     end
43     funcnumb=funcnumb+1;
44 end
45
46 %after reaching maxf replace all <e> by <v> and insert values for <b>,<u>
47 and <v>
48 G=regexprep(G, '<e>', '<v>');
49 while length(strfind(G, '<'))>0
50     binumb=mod(funcnumb, numb)+1;
51     switch G(findstr(G, '<')+1:findstr(G, '>')-1)
52     case 'b'
53         G=regexprep(G, '<b>', b{mod(actmat(k, binumb), lb)+1}, 'once');
54     case 'u'
55         G=regexprep(G, '<u>', u{mod(actmat(k, binumb), lu)+1}, 'once');
56     case 'v'
57         G=regexprep(G, '<v>', v{mod(actmat(k, binumb), lv)+1}, 'once');
58     case 'y'
59         G=regexprep(G, '<y>', y{mod(actmat(k, binumb), ly)+1}, 'once');
60     case 'x'
61         G=regexprep(G, '<x>', num2str(actmat(k, binumb)), 'once');
62     end
63     funcnumb=funcnumb+1;
64 end
65 actnum(k)=funcnumb;
66 if funcnumb>=numb
67     actnum(k)=numb;
68 end
69
70 for i=1:bttime
71     var0=v0(i);
72     var1=v1(i);
73     var2=v2(i);
74     esvar3(i)=eval(G);
75 end
76
77 error=sum(abs(v3(1:bttime)-transpose(esvar3))); %sum of absolut errors
78 between the true value of var3 and the estimateted value of var3
79 acterr(k)=error;
80 if sum(imag(esvar3))~=0
81     acterr(k)=Inf;
82 end
83 end

```

```

1 function [y] = bi2de2vec(x)

```

```

2 %Converts binaries to integer values
3 laenge=size(x,2);
4 y=0;
5 for i=1:laenge
6     y=y+2.^(laenge-i)*x(i);
7 end
8 end

```

A.4 Grammatical Evolution - Example 2

```

1 function [] = GEtrsys(pop,sel,mut,gen,schem,numb,maxf,bttime,crossover,parb)
2 %-----Explanation of the function:-----
3 %save functions 'newpopGETs.m','bi2de2vec.m' and the 'GBPUSD.xlsx' in the same
  folder!
4
5 %Grammatical Evolution
6
7 %BNF (Backus-Naur form) is defined as
8 %s.. start expression (else expressions is dependent on <t>, one will be -1
  and the other one 1)
9 %e.. logical expressions to include min. one logical operator in the statement
  of the if structure
10 %tr.. trading direction (signal with closing prices(t), execution open price (
  t+1)
11 %o.. logical operators (f.e. x&&y)
12 %p.. logical operators (f.e. or(x,y))
13 %z.. relational operators
14 %y.. numerical expressions to build up combinations of mathematical functions
  with real numbers and or past values of the used time serie
15 %u.. mathematical functions
16 %v.. real numbers or past values of the used time series
17 %b.. unitary operators
18 %k.. modified version of the Digit Concatenation
19 %j.. create integer numbers from the min. to the max. used past values
20 %l.. create possible stoploss values in the range of [parb(2,2),parb(1,2)]
  with a 0.001 gap
21 %w.. matrix of all open/high/low/close prices
22
23 % s=['if (<e>) tr(i+1)=<t>; else tr(i+1)=+-1; end'];
24 % e={'<e> <o> <e>','<p>(<e>,<e>)','<y> <z> <y>'};
25 % t={' +1 ',' -1 '};
26 % o={' && ',' || '};
27 % p={' or ',' xor '};
28 % z={' kl ',' klg ',' grg ',' gr '};
29 % y={' <y<b<y> ',' (<y<b<y> ) ',' <u>(<y> ) ',' <v> '};
30 % u={' sin ',' cos ',' exp ',' log ',' sqrt '};
31 % v={' w(i<j>,4) ',' <k> '};
32 % b={' + ',' - ',' * ',' / '};
33 % k={' <d<d<d>.<d<d<d> ',' <d<d>.<d<d<d> ',' <d>.<d<d<d> ',' <d<d<d>.<d<d>
  > ',' <d<d>.<d<d> ',' <d>.<d<d> ',' <d<d<d>.<d> ',' <d<d>.<d> ',' <d>.<d> '};
34 % j={};

```

```

35 % for i=parb(2,1):parb(1,1)
36 %     j{i+1-parb(2,1)}=num2str(i);
37 % end
38 % l={};
39 % for i=1:(parb(1,2)-parb(2,2))/0.001+2
40 %     l{i}=parb(1,2)+(i-1)*0.001;
41 % end
42
43 %In this GE data of time series GBPUSD is used and the aim is to find the
    optimal trading system to maximise the return.
44 %The GE uses random selection, 2 offspring per parents, only different parents
    allowed and elitism (the combination with the currently optimal values
    will always stay in the population and is not going to mutate)
45
46 %input Variables:
47 %pop... population size, stays the same over generations (integer value)
48 %sel... selection rate (between 0 and 1) (the percentage of population which
    is going to be changed each generation) (the selection rate itself will be
    changed or not, due to the used scheme. If yes the last/lowest value will
    be 1/pop.)
49 %mut... mutation rate (between 0 and 1) (the percentage of population which
    will be mutated randomly) (the mutation rate itself will be changed or
    not, due to the used scheme. If yes the last/lowest value will be 1/pop.)
50 %gen... generations (integer value)
51 %schem... selection-/mutation rate schemes (input: 1,2 or 3), you can choose
    between 3 schemes: 1. s-/m rate stays the same over all generations, 2. s
    -/m rate decreases linearly over all generations, 3. s-/m rate decreases
    negativ logarithmically over all generations
52 %numb... number of binaries that will be generated for each chromosome in the
    GE, if the algorithm needs more numbers it will start with the first
    again. 8 bits per codon -> integer number between 0 and 255.
53 %maxf... maximal number of used functions in the grammar (in matlab max 32)
54 %btttime.. backtesttime (integer value smaller than the total amount of values
    in the used time serie)
55 %crossover.. which crossover method should be used.. 1: 1-point crossover, 2:
    2-point crossover
56 %parb... parameter boundaries [upperboundaryofpd upperboundaryofstoploss;
    lowerboundaryofpd lowerboundaryofstoploss] f.e. [100 0.05;1 0.01] (
    trailing stoploss)
57
58 %example input for the Matlab Comand window:
59 %GETrsys(50,0.5,0.3,40,3,50,30,400,1,[100 0.05;1 0.1])
60 %-----
61
62 %timeseries of GBPUSD 01/2000-02/2017
63 dat=xlsread('GBPUSD.xlsx');
64
65 minsm=1/pop; %minimum value of selection rate and mutation rate
66 switch schem
67     case 1
68         %selection and mutation rate stays the same over all generations

```

```

69         sel=ones(gen-1)*sel;
70         mut=ones(gen-1)*mut;
71     case 2
72         %selection and mutation rate decreases linearly over all generations
73         %until they receive the value minsm
74         sel=linspace(sel,minsm,gen-1);
75         mut=linspace(mut,minsm,gen-1);
76     case 3
77         %selection and mutation rate decreases logarithmically over all
78         %generations until they receive the value minsm
79         sel=logspace(log(sel)/log(10),log(minsm)/log(10),gen-1);
80         mut=logspace(log(mut)/log(10),log(minsm)/log(10),gen-1);
81 end
82 %BNF
83 s=[ 'if (<e> tr(i+1)=<t>; else tr(i+1)=+-1; end' ];
84 e={ '<e> <o> <e>', '<p>(<e>,<e>)', '<y> <z> <y>' };
85 t={ '+1', '-1' };
86 o={ '&&', '||' };
87 p={ 'or', 'xor' };
88 z={ 'kl', 'kg', 'gr', 'gg' };
89 y={ '<y><b><y>', '(<y><b><y>)', '<u>(<y>)', '<v>' };
90 u={ 'sin', 'cos', 'exp', 'log', 'sqrt' };
91 v={ 'w(i-<j>,4)', '<k>' };
92 b={ '+', '-', '*', '/' };
93 k={ '<d><d><d>.<d><d><d>', '<d><d>.<d><d><d>', '<d>.<d><d><d>', '<d><d><d>.<d><d>',
94     '<d><d>.<d><d>', '<d>.<d><d>', '<d><d><d>.<d>', '<d><d>.<d>', '<d>.<d>' };
95 j={ };
96 for i=parb(2,1):parb(1,1)
97     j{i+1-parb(2,1)}=num2str(i);
98 end
99 l={ };
100 for i=1:(parb(1,2)-parb(2,2))/0.001+1
101     l{i}=parb(2,2)+(i-1)*0.001;
102 end
103 %Initial Population
104 actmat=[]; %place holder - matrix with all decimal numbers of binaries
105 acterr=[]; %place holder - error vectors
106 actnum=[]; %place holder - number of used binaries
107 [actmat,actret,actnum]=newpopGETs([1:pop],1,numb,actmat,actret,actnum,s,maxf,e
108     ,t,o,p,z,y,u,v,b,k,j,l,bttime,dat(1:bttime,1:4),0);
109 ofsnum=pop-ceil(pop*(1-sel)); %number of offspring needed
110 parnum=ceil(ofsnum/2); %number of parents
111 %sort population matrix due to the return of the chromosomes (1st chromosome..
112 %highest return)
113 [actret,s2]=sort(actret,'descend');
114 actmat=actmat(s2,:);
115 actnum=actnum(s2);

```

```

115 %generation loop
116 for i=1:gen-1 % "-1" due to the initial generation
117     parents=zeros(parnum(i),2); %parents
118
119     %random selection of parents
120     for kk=1:parnum(i)
121         tosu=zeros(1,2);
122         while tosu(1,1)==tosu(1,2) %only different parents are allowed
123             tosu=sort(ceil(rand(1,3)*ceil(pop*(1-sel(i)))));
124         end
125         parents(kk,:)=tosu(1:2);
126     end
127
128     %mating
129     for jj=1:ofsnum(i)
130         par=ceil(jj/2); %parents
131         r=mod(jj,2);
132         switch crossover
133             case 1
134                 %1 point crossover
135                 if r==1
136                     zf=min(actnum(parents(par,1)),actnum(parents(par,2)))-1;
137                     zz=ceil(rand(1)*zf)+1;
138                 end
139                 %generate new offspring with 1-point crossover of parents
140                 actmat(jj+pop-ofsnum(i),1:zz)=actmat(parents(par,1+r*1),1:zz);
141                 actmat(jj+pop-ofsnum(i),zz+1:numb)=actmat(parents(par,2-r*1),
142                     zz+1:numb);
143             case 2
144                 %2 point crossover
145                 if r==1
146                     zf=min(actnum(parents(par,1)),actnum(parents(par,2),3))-1;
147                     zz(1)=ceil(rand(1)*zf)+1;
148                     zz(2)=0;
149                     while zz(2)==zz(1) || zz(2)==0
150                         zz(2)=ceil(rand(1)*zf)+1;
151                     end
152                     zz=sort(zz);
153                 end
154                 %generate new offspring with 2-point crossover of parents
155                 actnum(jj+pop-ofsnum(i),1:zz(1))=actnum(parents(par,1+r*1),1:
156                     zz(1));
157                 actnum(jj+pop-ofsnum(i),zz(1)+1:zz(2))=actnum(parents(par,2-r
158                     *1),zz(1)+1:zz(2));
159                 actnum(jj+pop-ofsnum(i),zz(2)+1:numb)=actnum(parents(par,1+r
160                     *1),zz(2)+1:numb);
161             end
162         end
163     end
164
165     %calculate the return of the offspring

```

```

161     [actmat, actret, actnum]=newpopGEts([pop-ofsnum(i)+1:pop],0,numb,actmat,
        actret,actnum,s,maxf,e,t,o,p,z,y,u,v,b,k,j,l,bttime,dat(1:bttime,1:4)
        ,0);
162
163     %sort the new population matrix
164     [actret,s2]=sort(actret,'descend');
165     actmat=actmat(s2,:);
166     actnum=actnum(s2);
167
168     %mutation
169     for jj=1:ceil((pop-1)*mut(i))
170         %"-1" .. due to elitism, the "best" chromosome won't be mutated
171         rw=ceil(rand(1)*(pop-1))+1; % +1.."best" one is in the first row -
            elitism.. rw is in [2,pop]
172         decpos=ceil(rand(1)*actnum(rw));
173         actmat(rw,depos)= bi2de2vec(transpose(round(rand(8,1)))));
174
175         %actualize the costs only of the mutated chromosome
176         [actmat,actret,actnum]=newpopGEts(rw,0,numb,actmat,actret,actnum,s,maxf
            ,e,t,o,p,z,y,u,v,b,k,j,l,bttime,dat(1:bttime,1:4),0);
177     end
178     [actret,s2]=sort(actret,'descend');
179     actmat=actmat(s2,:);
180     actnum=actnum(s2);
181 end
182
183 %Plot of equity curves
184 [actmat1,actret1,actnum1]=newpopGEts(1,0,numb,actmat,actret,actnum,s,maxf,e,t,
    o,p,z,y,u,v,b,k,j,l,bttime,dat(1:bttime,1:4),1);
185 [actmat2,actret2,actnum2]=newpopGEts(1,0,numb,actmat,actret,actnum,s,maxf,e,t,
    o,p,z,y,u,v,b,k,j,l,size(dat,1)-bttime,dat(bttime+1:size(dat,1),1:4),1);
186 pastvalues=actnum1{1}(4,2);
187 xlbt=pastvalues+2;
188 xrbt=bttime-xlbt;
189 xrrt=size(dat,1)-bttime;
190 yu=max(max(actnum1{1}(xlbt:bttime-1,1)),max(actnum2{1}(xlbt:size(dat,1)-bttime
    -1,1)));
191 yl=min(min(actnum1{1}(xlbt:bttime-1,1)),min(actnum2{1}(xlbt:size(dat,1)-bttime
    -1,1)));
192 yu=yu+(yu-yl)*0.1;
193 yl=yl-(yu-yl)*0.1;
194
195 figure
196 subplot(1,2,1)
197 hold on
198 plot(actnum1{1}(pastvalues+2:bttime-1,1))
199 axis([1 xrbt yl yu])
200 xlabel('backtesting time')
201 legend('show')
202 hold off
203

```

```

204 subplot(1,2,2)
205 hold on
206 plot(actnum2{1}(pastvalues+2:size(dat,1)-btttime-1,1))
207 axis([1 xrrt yl yu])
208 xlabel('validation time')
209 legend('show')
210 hold off
211 end

1 function [actmat,actret,actnum] = newpopGETs(kwerte,binneu,numb,actmat,actret,
    actnum,s,maxf,e,t,o,p,z,y,u,v,b,k,j,l,btttime,dat,yn)
2 %-----Explanation of the function:-----
3
4 %In general the function will be used in combination with the 'GETsys.m'
5 %function. It evaluates the grammatical evolution, the
6 %algorithm(chromosome), the trading system and if needed the maximal
7 %drawdown length and percentage.
8
9 %input Variables:
10
11 %all variables are the same as in the 'GETsys.m function' with some
12 %exceptions:
13
14 %kwerte... vector of rowvalues, which should be actualised due to some
15 %change in the chromosome values in the given actmat matrix
16 %binneu... should the algorithm randomly create new binary values (1) or not
    (0).
17 %yn... should the algorithm evaluate drawdown percentage and length, used
18 %for the plot and some analysis of the results.
19
20 %example input for the Matlab Comand window:
21 %newpopGETs([1:10],1,50,[],[],[],s,30,e,t,o,p,z,y,u,v,b,k,j,l,300,dat,0)
22 %newpopGETs([1,3,4],0,50,actmat,actret,actnum,s,30,e,t,o,p,z,y,u,v,b,k,j,l
    ,300,dat,0)
23 %newpopGETs([1],0,50,actmat,actret,actnum,s,30,e,t,o,p,z,y,u,v,b,k,j,l,300,dat
    ,1)
24
25 %-----
26
27 for kk=kwerte
28     funcnumb=0; %number of used functions in the grammar
29     binumb=0; %number of used binaries, start from the first again after you'
        ve reached the last one.
30     bnumused=0; %number of used binaries
31
32     if binneu==1
33         %generate random binary numbers bn and evaluate the integer numbers
34         bn=transpose(round(rand(8*numb,1)));
35         for i=1:numb
36             bi=bn(1+(i-1)*8:i*8);
37             an(i)=bi2de2vec(bi);

```

```

38         end
39         actmat(kk,:) = an;
40     end
41
42     %generate the grammar G
43     G = s;
44     while length(strfind(G, '<')) > 0 && funcnumb < maxf && bnumused < numb*3
45         binumb = mod(bnumused, numb) + 1;
46         switch G(findstr(G, '<') + 1 : findstr(G, '>') - 1)
47             case 'e'
48                 G = regexprep(G, '<e>', e{mod(actmat(kk, binumb), length(e)) + 1}, 'once');
49                 if mod(actmat(kk, binumb), length(e)) + 1 == 2
50                     funcnumb = funcnumb + 1;
51                 end
52             case 't'
53                 G = regexprep(G, '<t>', t{mod(actmat(kk, binumb), length(t)) + 1}, 'once');
54             case 'o'
55                 G = regexprep(G, '<o>', o{mod(actmat(kk, binumb), length(o)) + 1}, 'once');
56             case 'p'
57                 G = regexprep(G, '<p>', p{mod(actmat(kk, binumb), length(p)) + 1}, 'once');
58             case 'z'
59                 G = regexprep(G, '<z>', z{mod(actmat(kk, binumb), length(z)) + 1}, 'once');
60             case 'y'
61                 G = regexprep(G, '<y>', y{mod(actmat(kk, binumb), length(y)) + 1}, 'once');
62                 if mod(actmat(kk, binumb), length(y)) + 1 == 2
63                     funcnumb = funcnumb + 1;
64                 end
65             case 'u'
66                 G = regexprep(G, '<u>', u{mod(actmat(kk, binumb), length(u)) + 1}, 'once');
67                 funcnumb = funcnumb + 1;
68             case 'v'
69                 G = regexprep(G, '<v>', v{mod(actmat(kk, binumb), length(v)) + 1}, 'once');
70             case 'b'
71                 G = regexprep(G, '<b>', b{mod(actmat(kk, binumb), length(b)) + 1}, 'once');
72             case 'k'
73                 G = regexprep(G, '<k>', k{mod(actmat(kk, binumb), length(k)) + 1}, 'once');
74             case 'd'
75                 G = regexprep(G, '<d>', num2str(actmat(kk, binumb)), 'once');
76             case 'j'
77                 G = regexprep(G, '<j>', j{mod(actmat(kk, binumb), length(j)) + 1}, 'once');

```

```

78         end
79         bnumused=bnumused+1;
80     end
81
82     %generate the stoploss
83     binumb=mod(bnumused,numb)+1;
84     stoploss=1{mod(actmat(kk,binumb),length(1))+1};
85
86     %after reaching maxf replace all <e> by <v> and insert values
87     %for <b>,<u> and <v>
88     G=regexprep(G,'<e>','<v> <z> <v>');
89     G=regexprep(G,'<y>','<v>');
90     while length(strfind(G,'<'))>0 && bnumused<numb*20
91         binumb=mod(bnumused,numb)+1;
92         switch G(findstr(G,'<')+1:findstr(G,'>')-1)
93             case 't'
94                 G=regexprep(G,'<t>',t{mod(actmat(kk,binumb),length(t))+1},'once
95                     ');
96             case 'o'
97                 G=regexprep(G,'<o>',o{mod(actmat(kk,binumb),length(o))+1},'once
98                     ');
99             case 'p'
100                 G=regexprep(G,'<p>',p{mod(actmat(kk,binumb),length(p))+1},'once
101                     ');
102             case 'z'
103                 G=regexprep(G,'<z>',z{mod(actmat(kk,binumb),length(z))+1},'once
104                     ');
105             case 'u'
106                 G=regexprep(G,'<u>',u{mod(actmat(kk,binumb),length(u))+1},'once
107                     ');
108             funcnumb=funcnumb+1;
109             case 'v'
110                 G=regexprep(G,'<v>',v{mod(actmat(kk,binumb),length(v))+1},'once
111                     ');
112             case 'b'
113                 G=regexprep(G,'<b>',b{mod(actmat(kk,binumb),length(b))+1},'once
114                     ');
115             case 'k'
116                 G=regexprep(G,'<k>',k{mod(actmat(kk,binumb),length(k))+1},'once
117                     ');
118             case 'd'
119                 G=regexprep(G,'<d>',num2str(actmat(kk,binumb)), 'once ');
120             case 'j'
121                 G=regexprep(G,'<j>',j{mod(actmat(kk,binumb),length(j))+1},'once
122                     ');
123
124         end
125         bnumused=bnumused+1;
126     end
127
128     %replace kl,gr,kg,gg with <,>,<=,>=
129     while length(strfind(G,'kl'))>0

```

```

120         bb=strfind(G,'kl');
121         G(bb(1):bb(1)+1)='< ';
122     end
123     while length(strfind(G,'gr'))>0
124         bb=strfind(G,'gr');
125         G(bb(1):bb(1)+1)='> ';
126     end
127     while length(strfind(G,'kg'))>0
128         bb=strfind(G,'kg');
129         G(bb(1):bb(1)+1)='<=';
130     end
131     while length(strfind(G,'gg'))>0
132         bb=strfind(G,'gg');
133         G(bb(1):bb(1)+1)='>=';
134     end
135
136     %find the maximal used past value in the algorithm
137     pos=strfind(G,'w');
138     pasttrd=0;
139     for i=1:length(pos)
140         pasttrd(i)=str2num(G(pos(i)+4:pos(i)+strfind(G(pos(i):pos(i)+10),')')
141             -4));
142     end
143     statrd=max(pasttrd);
144
145     actnum(kk)=bnumused;
146     if bnumused>=numb
147         actnum(kk)=numb;
148     end
149
150     len=length(G);
151     if (G(len-24)=='-') G(len-6)=''; end
152     if (G(len-24)=='+') G(len-7)=''; end
153
154     cap=100; %start capital
155     w=dat; %open/high/low/close
156     tr=zeros(bttime,1);
157     %evaluate trading positions
158     for i=statrd+1:bttime-1
159         eval(G);
160     end
161     M=[w, tr];
162
163     %trading system
164     curpos=M(statrd+2,5); %current trading position: long(1), short(-1), hold
165         (0)
166     stopval=M(statrd+2,1)*(1-curpos*stoploss); %trailing stoploss value
167     lastpos=0; %last trading position
168     lastcurpos=0; %trading position of the day before
169     ec=zeros(bttime-1,1); %equity curve place holder

```

```

168 %Evaluate the equity curve values due to the generated trading system and
    trailing stoploss
169 for jj=statrd+2:bttime-1
170     stophit=0;
171     %after hitting the stopval a new position will only be opened if the
        new position is different to the last one (1 <-> -1)
172     if curpos==0 && M(jj,5)~=lastpos
173         lastpos=0;
174         curpos=M(jj,5);
175         stopval=M(jj,1)*(1-curpos*stoploss);
176     end
177
178     %position change from 1 <-> -1
179     if M(jj,5)~=curpos && curpos~=0
180         lastpos=curpos;
181         curpos=M(jj,5);
182         stopval=M(jj,1)*(1-curpos*stoploss);
183     end
184
185     %if the position stays the same, just check if the value is under/over
        the stopval and if the stopval of the new value is bigger or
        smaller than the old stopval then eventually actualize it.
186     %We therefore use the closing prices.
187     if M(jj,5)==curpos && curpos~=0
188         if (M(jj,3)<stopval && curpos==1) || (curpos==-1 && M(jj,2)>
            stopval)
189             lastpos=curpos;
190             stophit=1;
191         end
192         if (M(jj,4)*(1-curpos*stoploss)>stopval && curpos==1 && stophit
            ==0) || (curpos==-1 && M(jj,4)*(1-curpos*stoploss)<stopval &&
            stophit==0)
193             stopval=M(jj,4)*(1-curpos*stoploss);
194         end
195     end
196
197     if stophit==0
198         %position stays the same
199         if lastcurpos==M(jj,5)
200             cap=cap*(sqrt((curpos-1)^2)+curpos*M(jj,4)/M(jj-1,4));
201         end
202         %position change from 0 to +1 or 1<->-1
203         if lastcurpos~=M(jj,5)
204             if lastcurpos==0
205                 cap=cap*(sqrt((curpos-1)^2)+curpos*M(jj,4)/M(jj,1));
206             end
207             if lastcurpos~=0
208                 cap=cap*(sqrt((lastcurpos-1)^2)+lastcurpos*M(jj,1)/M(jj
                    -1,4));
209                 cap=cap*(sqrt((M(jj,5)-1)^2)+M(jj,5)*M(jj,4)/M(jj,1));
210             end

```

```

211         end
212     end
213     %stopval hit-> stopval is closing price of trade
214     if stophit==1
215         cap=cap*(sqrt((lastcurpos-1)^2)+lastcurpos*M(jj,1)/M(jj-1,4));
216         cap=cap*(sqrt((curpos-1)^2)+curpos*stopval/M(jj,1));
217         curpos=0;
218     end
219     ec(jj)=cap/100;
220     lastcurpos=curpos;
221 end
222
223 %evaluate length and drawdown percentage, only if yn=1
224 if yn==1
225     maxddl=0;%maxdrawdownlength in days
226     maxddp=1;%maxdrawdownpercentage [-1,0]
227     lastk=statrd+2; %if we already have found a maxdraddownlength from i
        to k, then there won't be a bigger one in this part of the equity
        curve
228     maxwert=statrd+2;
229
230     for i=statrd+2:btime-2
231         for k=i+1:btime-1
232             if (ec(k,1)/ec(i,1)-1)<maxddp && i>=maxwert
233                 maxddp=(ec(k,1)/ec(i,1)-1);
234                 if ec(k,1)>ec(i,1)
235                     maxwert=k;
236                 end
237             end
238             if (ec(i,1)<=ec(k,1) && i>=lastk) || (ec(i,1)>ec(k,1) && k==
                btime-1 && i>=lastk)
239                 if k-i>maxddl
240                     maxddl=k-i;
241                 end
242                 lastk=k;
243             end
244         end
245     end
246
247     actnum={ec;G};
248     actnum{1}(1,2)=maxddl;
249     actnum{1}(2,2)=maxddp;
250     actnum{1}(3,2)=stoploss;
251     actnum{1}(4,2)=statrd;
252 end
253 actret(kk)=(ec(btime-1)-1)*100; %final return (percentage)
254 end

```

A.4.1 Optimal Trading Strategy

```
1 if (w(i-97,4)<=sin(w(i-0,4)*log(log((w(i-39,4)/cos((exp(57.18312681)
    *(23819.14312884/(w(i-74,4)+sqrt(((w(i-44,4)-w(i-89,4)-w(i-32,4)))/(w(i
    -55,4)/log(w(i-91,4)-w(i-95,4))))*(exp(w(i-40,4))+cos((w(i-84,4)*w(i-42,4))
    )))))))))))
2     tr(i+1)=-1;
3 else
4     tr(i+1)=+1;
5 end
6 stoploss=0.01;
```