Creation of parametric rules to rewrite algebraic expressions in Symbolic Regression[*]

A. S. Kulunchakov
kulu-andrej@yandex.ru
Moscow Institute of Physics and Technology, 9 Institutskiy per., Dolgoprudny, Moscow, Russia

This paper investigates the problem of bloat in Symbolic Regression (SR). It develops a procedure to simplify superpositions generated by SR. The present approach borrows ideas of equivalent decision simplification and they are applied to create parametric rules of rewriting. Except from eliminating excessive parts of superpositions, these rules reduce dimensionality of parameter space of generated superpositions. Computational experiment is conducted on a dataset related to Brent Crude Oil options. There, the volatility of options prices is approximated by its strike prices and expiration date.

Keywords: *tree bloat; rule-based simplification; symbolic regression; genetic programming*

DOI: 10.21469/22233792.3.1.01

1 Introduction

This paper addresses the problem of bloat in Genetic Programming (GP) [1] and SR [2]. Symbolic Regression generates superpositions of expert-given primitive functions to fit some given dataset. The applications of SR include feature generation for time series and image processing [3, 4], time series forecasting [5–7], dimensionality reduction, and data visualization [8].

Symbolic regression constructs a superposition as syntax tree [9], which structure is evolved iteratively. It may generate overfitted and complex functions [1, 9]. For example, it tends to evolve trees with introns [9], which add significant complexity to the tree structure [1, 5]. Intron is an ineffective piece of code, which is likely to be removed from trees. In the superposition $\cos(x) + 0 \cdot \mathbf{ln}(\boldsymbol{x})$, the bold function is an intron, as it does not influence the values of the rest part of the superposition. There is a number of approaches proposed to eliminate introns [10–12]. Another kind of undesirable behavior of GP is a generation of complex superpositions. For example, the function $\sin^2(x) + \cos^2(x)$ is identically equals 1. Therefore, the calculation of its values is ineffective and this function should be simplified. The approach presented in [12] handles both removing of introns and model simplification. The authors of [12] simplify superpositions for postrun analysis. However, they do not either generate or make use of parametric rules of rewriting, which simplify parametric superpositions. A rule of rewriting is defined by a pair of syntax trees: *pattern* tree and *replacement* tree. Pattern tree should be more complex than the replacement tree. Each found occurrence of the pattern tree in superpositions is substituted by the replacement tree The details of this substitution is given below. Finally, the simplified superposition has lesser complexity.

The current paper develops an approach of generation of parametric rules of rewriting. These rules are used then during GP launches. The evolution of superpositions is analyzed in this case. This paper analyzes also the computational complexity of the simplification and compares it with the ones described in [12].

[*]The research was supported by the Russian Foundation for Basic Research grant 16-37-00486.

2 Motivation

Several papers [5, 13, 14] state that GP can benefit from preserving introns. Nevertheless, authors of [12] mention a number of problems caused by tree bloat:

- calculation of values produced by excessive pieces of trees which slows down the process of run;
- generation of sophisticated, uninterpretable models;
- excessive pieces of trees hide the true structural complexity of a superposition; models, proper in the context of application, could be eliminated from a population by complexity penalization;
- tree bloat makes it difficult to analyze the evolution of useful pieces of trees from iteration to iteration; and
- tree bloat masks the true blocks of trees which are inherent for a particular application; peeling these blocks from excessive pieces could improve convergence to a solution.

In this paper, these reasons are followed. The paper [12] considers only simplification of nonparametric superpositions. However, tree bloat influence on the generation of the parametric ones includes several other difficulties:

- evaluation of parameters presented in excessive blocks of code;
- excessive parameters slow down the convergence of parameters optimization procedures; and
- if a solution is found, the parameters in its introns are not defined uniquely; therefore, there is a need of penalty on the norm of parameter vector.

This paper solves these problems by means of rules of rewriting. Therefore, one needs experts to define these rules. However, the set of rules is different in different applications. Moreover, it is not guaranteed that expertly-given list of rules is complete and does not contain mistakes. To provide this, an automatic rule generation procedure has been developed. This procedure creates the full list of rules which are valid for a specific application. It is far better than looking over the primitive set and constructing rules manually.

3 Problem Statement

Given a finite primitive set $\mathcal{G} = \{f_1, \dots, f_K\} \cup \{x_1, \dots, x_N\}$ of primitives $\{f_1, \dots, f_K\}$ (some of them are parametric) and terminals $\{x_1, \dots x_N\}$, the set $\mathcal{F}$ of all possible superpositions of them has been considered. This set is referred to as *search space*. To simplify notation, only univariate superpositions will be considered: $N = 1$. Therefore, denote dataset $\mathfrak{D} = \{x_t\}_{t=1}^{\eta}$ where $x_t \in \mathbb{R}$ and η is the number of points.

Each superposition $s \in \mathcal{F}$ has *structural complexity* $C(s)$. It is equal to the number of elements from $\mathcal{G}$, which the superposition is composed of. For example, the structural complexity of $s = \text{plus2}(\text{linear}(x_1), x_1)$ is 4. Denote also $P(s)$ as the number of parameters $W(s)$ in superposition s. Given parameters $W(s)$, let $V(s, W(s), \mathfrak{D})$ be the values of s on the dataset $\mathcal{D}$.

Denote the values of s given x over regression points are referred to as $s(W(s), x_1)$.

Some pairs of elements from $\mathcal{F}$ form *rules of rewriting*. A rule of rewriting is a pair (s_1, s_2) such that functions s_1 and s_2 are *equal* and the inequalities $C(s_2) < C(s_1)$ and $P(s_2) \leqslant P(s_1)$ hold. The definition of the *equality* between s_1 and s_2 depends on two cases.

1. Both of them are nonparametric: $P(s_1) = P(s_2) = 0$. Then, for each $x \in \mathfrak{D}$, $s_1(x) = s_2(x)$.
2. Both of them are parametric. Then, for each $\mathfrak{D}$ and $W(s_1)$, there exist parameters $W(s_2)$ that s_1 and s_2 are equal on $\mathfrak{D} : s_1(W(s_1), x) = s_2(W(s_2), x)\ \forall x \in \mathfrak{D}$.

Syntax tree of

$$x_1 \tan x_2.$$

Functions:

$$\cdot \times \cdot, \tan.$$

Terminals:

$$x_1, x_2.$$

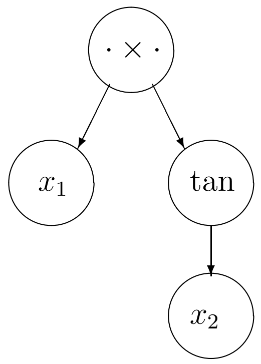

Figure 1 Syntax tree of a superposition

For example, $s_1 = \sin(w_0 x + w_1)$ and $s_2 = \cos(w_0' x + w_1')$ are equal according to this definition regardless of $\mathfrak{D}$. It holds because for each $W(s_1)$, the superposition s_2 can be fitted to the values $s(W(s_1), \mathfrak{D})$ by optimizing of $W(s_2)$. To fit the superposition, the Levenberg–Marquardt algorithm is used. As it does not provide exact values of optimal parameters, let us use a hyperparameter τ as a threshhold for accuracy. Therefore, two values $s_1(W(s_1), x)$ and $s_2(W(s_2), x)$ are called equal if

$$|s_1(W(s_1), x) - s_2(W(s_2), x)| < \tau.$$

Each superposition s is represented by a syntax tree (Fig. 1). Its *height* is equal to the length of the longest path from the root to the leaves. The height of the syntax tree shown in Fig. 1 is equal to 2.

This paper aims at extracting all possible rules of rewriting from $\mathcal{F}$ such that the height of the patterns is at most 3. Generated rules are supposed to be mathematical identities, not only approximations on a given range of independent values.

4 Automatic Procedure of Rules Generation

This paper borrows ideas presented in [12] and applies them to generate parametric rules. The authors of [12] call functions equivalent, if they have equal values on a dataset. The present authors extended this definition to parametric superpositions. In this section, the procedure of parametric rules generation is described. It is based on the following algorithm (see Algorithm 1).

First, as in [12], let us define a range for $x \in \mathfrak{D}$. It depends solely on the application. If $\mathfrak{D}$ is preprocessed and scaled to a range between 0 and 1, the range is [0,1]. The choice of the range is crucial for rules generation: functions, which are not mathematically identical, can be equal on small intervals. For example, smooth functions are well approximated by linear function in a small neighborhood of x. As this paper aims to generate pairs of mathematically identical functions, the range should be rather large. However, estimation of a function on x drawn from very large range suffers from the computational issues. The range $[-3, 3]$ was chosen as a rather good trade-off. The dataset of values of x_1 is drawn from this range.

Second, let us form a primitive set $\mathcal{G}$ of functions and terminals. Generate all possible simple superpositions of elements from the set. *Simplicity* of a superposition is user-defined. For example, this paper considers only syntax trees of height at most 3. The generated superpositions can contain parameters because parametric primitives are used. For each parametric superposition, let us set random initial parameters. If an application imposes bounds on the parameters of some primitives, the initial parameters should be drawn with respect to these requirements. If the bounds are not imposed, the parameters are drawn from standard normal distribution.

Algorithm 1 Generation of parametric rules of rewriting

```
Require: threshold $\tau$, number of regression points $\eta$, number of checking iterations $\nu$, fraction
  of misfittings $\alpha$, primitive set $\mathcal{G}$
Ensure: rules of rewriting $\{p_k, r_k\}_{k=1}$
  assign a range for values of indepenent variable $x$ and draw random points $\mathcal{D} = \{x_t\}_{t=1}$ from
  this range;
  generate a set $\mathcal{I}$ of all simple superpositions of elements from $\mathcal{G}$;
  replicate $\mathcal{I}$ and get two sets: patterns $\mathcal{I}_1$ and replacements $\mathcal{I}_1 = \mathcal{I}_2 = \mathcal{I}$;
  for each pattern $p \in \mathcal{I}_1$
    set random initial parameters in $W(p)$;
    calculate values $V(p, W(p), \mathcal{D})$;
    for each replacement $r \in \mathcal{I}_2$
      optimize parameters $W(r)$ to approximate $V$
      if $\max\limits_{t=[1,\dots N]} |p(W(p), x_t) - r(W(r), x_t)| < \tau$, then
        for iter $\in$ [1,$\nu$]
          number_of_failed_fittings := 0
          assign random initial parameters to $W(p)$;
          recalculate values $V(p, W(p), \mathcal{D})$ on new parameters $W(p)$;
          optimize parameters $W(r)$ to approximate $V(p, W(p), \mathcal{D})$
          if $\max\limits_{t=[1,\dots N]} |p(W(p), x_t) - r(W(r), x_t)| > \tau$, then
            number_of_failed_fittings++
        if number_of_failed_fittings $< \alpha \cdot \nu$, then
          save rule of rewriting $(p, r)$
    return set of saved rules.
```

Third, the set of generated superpositions is cloned. The original set acts as a set of patterns for rules and the cloned one is a set of replacements. Let us pick up the patterns consequently and try to fit replacements to them. This requires optimization of a replacement's parameters. The paper uses Levenberg–Marquardt algorithm implemented in *scipy.optimize.curve_fit* to fit parameters. To check if a replacement represents the same mathematical function with a pattern on a given range, another procedure is required.

5 Verification of Parametric Functions Equality

In this section, it is claimed that the functions are equal if they are produced. Assume that the values produced by a pattern with random initial parameters are well approximated by a replacement. Note that one successful fitting is not enough to claim that the pattern equals the replacement. For example, $\sin w_1 \cdot x + w_2$ with parameters $W = [0.01, 0.01]$ is well fitted by constant zero. However, the functions are significantly different in the values they produced in general. Therefore, the equality is verified for different random parameters of patterns. Further, let us denote patterns as p and replacements as r.

Given a set of candidates $\{r_1, \dots, r_j\}$ to be equal to p, let us do the following steps to each replacement ν times where ν is user-defined. Random parameters are assigned to $W(p)$ and new dependent values $V(p, W(p), \mathfrak{D})$ are calculated. The parameters $W(r)$ are optimized to fit these values. If the fraction of misfittings is rather small with respect to a user-defined threshold α, then r is claimed to be equal to p. The number of algorithm applications and the threshold are user-defined.

6 Experimental Setup

In this section, the experiment which consists of two steps is described. First, let us launch the procedure 1 in order to generate rules of simplification. Second, having these rules, let us use simplifying system inside the GP functionality. Here, the best approximating superposition has been extracted on each iteration. Then, the evolution of its mean squared error (MSE) is plotted via the number of iteration. The resulted plot has two lines: one corresponds to the basic version of GP and the other corresponds to the one which uses the present simplification system.

At the start, let us discuss user-defined parameters mentioned in Algorithm 1. Their values are presented in Table 1.

As the paper is aimed at producing the rules of rewriting acting as mathematical identities, the range of x_1 is chosen to be $[-3, 3]$ rather than $[-1, 1]$ or $[0, 1]$ usually used in different applications. The reason is that, for example, smooth functions are well approximated by polynomials on the unit interval. The interval is not chosen either to be large because of numerical issues. Therefore, a trade-off is reached and it is set equal $[-3, 3]$. Producing proper rules of rewriting, it is necessary to check if their do a good generalization on the outer range $(-\infty, -3) \cup (3, \infty)$.

The choice of both number of regression points and number of checking iterations is based on trade-off between computational burden and correctness of algorithm. A dataset of hundred of random regression points seems to be sufficient to produce *representative* values of function. In other words, the plots of the functions are sufficiently distinct and contain necessary information to make out the function corresponding to a plot.

The other parameters, namely, threshold τ and fraction of misfittings α, are chosen empirically. Note that one τ is chosen, it does not depend on the number of regression points η. This is the main reason why there l_∞ is preferable to more frequently used MSE. The fact that the fraction of misfittings is not zero is due to imperfection of parameter optimization procedure. As one does not have any prior knowledge about parameters domain, the initial parameters should be set randomly. Therefore, for some functions and some randomly set parameters, optimization procedure, namely, Levenberg–Marquardt algorithm, does not converge or converge in local optimum, which, nevertheless, is not global. This is the reason of why multistart is used to fit a replacement r: optimization is launched from different initial guesses $W(r)$ for the parameters M_1 times. However, as most computational burden rests with optimization procedure, the use of multistart slows down the process almost in M_1 times. Therefore, M_1 is chosen rather small.

As experiment shows, M_1 is not sufficiently large for replacement parameters to be fitted exactly well, but good enough to give appropriate approximation. Therefore, as generally only a tiny fraction of replacements from $\mathcal{I}_2$ is fittable to a pattern, the following strategy is used. For 5 best approximated replacements from $\mathcal{I}_2$, $M_2 \gg M_1$ launches are performed in multistart. This strategy notably improves the generation of rules.

Table 1 User-defined parameters

Range for x_1	Number of regression points η	Number of checking iterations ν	Threshold τ	Fraction of misfittings α	Multistart iterations on fitting M_1	Multistart iterations on checking M_1
$[-3, 3]$	100	50	0.04	0.04	5	150

Table 2 Primitive set and their frequencies in patterns

Name	Function	Number of parameters	Frequency
bump(x_1)	$x_1[w_0 < x_1 \text{ and } x_1 < w_1]$	2	0.02
sinc(x_1)	$\dfrac{\sin(\pi(w_1x_1 + w_2))}{\pi(w_1x_1 + w_2)}$	2	0.04
hvs(x_1)	$x_1[w_0 < x_1]$	1	0.02
sina(x_1)	$\sin(w_1x_1 + w_2)$	2	0.04
lnl(x_1)	$\ln(w_1x_1 + w_2)$	2	0.02
expl(x_1)	$\exp(w_1x_1 + w_2)$	2	0.07
plus2(x_1, x_2)	$x_1 + x_2$	0	0.32
normal(x_1)	$\dfrac{1}{w_1}\exp\left(-\dfrac{(x_1 - w_2)^2}{w_1}\right)$	2	0.05
frac2(x_1, x_2)	$\dfrac{x_1}{x_2}$	0	0.26
neg(x_1)	$-x_1$	0	0.16
hypot(x_1, x_2)	$\sqrt{x_1^2 + x_2^2}$	0	0.29
times2(x_1, x_2)	x_1x_2	0	0.33
linear(x_1)	$w_1x_1 + w_2$	2	0.14
parabola(x_1)	$w_1x_1^2 + w_2x_1 + w_3$	3	0.10
unity()	1	0	0.36
zero()	0	0	0.18
parameter_()	w_1	1	0.41

The primitive set used for construction of superpositions is shown in Table 2. This paper assumes that x_1+x_2 and x_2+x_1 represent the same function. Therefore, for each node of syntax tree corresponding to commutative primitive, its children are ordered in some way.

7 Generation of Rules

Now, let us generate rules which have patterns with a height of their syntax trees not exceeding 3. Given the restriction on height, let us generate the set $\mathcal{I}$ of all possible patterns. The resulted set has cardinality $|\mathcal{I}(x_1)| = 22\,084$ for patterns with only one terminal (univariate case) and $|\mathcal{I}(x_1, x_2)| = 42\,499$ for two terminals (bivariate case). All of them are called *candidates* as they will form the required rules. Note that the replacements will be taken from these sets as well as patterns. Tables 3 and 4 show the distributions of generated candidates according

Table 3 Distributions of superpositions from $\mathcal{I}$ according to structural complexity $C(\cdot)$ and number of parameters $P(\cdot)$ (univariate case)

$C(\cdot)$	$P(\cdot)$								
	0	1	2	3	4	5	6	7	8
1	3	1	0	0	0	0	0	0	0
2	3	4	22	10	1	0	0	0	0
3	30	22	51	63	169	93	17	1	0
4	72	117	558	396	108	9	0	0	0
5	441	428	515	613	1502	1179	384	59	4
6	395	749	3249	3076	1325	275	20	0	0
7	2106	2349	1278	345	46	0	0	0	0

Table 4 Distributions of superpositions according to their structural complexities $C(\cdot)$ and number of parameters $P(\cdot)$ (bivariate case)

$C(\cdot)$	$P(\cdot)$								
	0	1	2	3	4	5	6	7	8
1	4	1	0	0	0	0	0	0	0
2	4	5	29	11	1	0	0	0	0
3	50	29	66	79	220	107	18	1	0
4	126	186	951	555	123	9	0	0	0
5	984	741	843	971	2548	1700	482	67	4
6	910	1534	7169	5526	1910	320	20	0	0
7	6124	5344	2226	460	46	0	0	0	0

to their structural complexities $C(\cdot)$ and number of parameters $P(\cdot)$. The restriction on height provides one with candidates which are either structurally or parametrically simple. One can see from the tables that there are no many candidates, which are structurally complex and have many parameters at the same time. It alleviates the computational burden of the algorithm and allows one to use multistart while preserving reasonable operating time. Note that in this paper, the use of multistart is reasonable only for the task of generation of rules. In general, the use of multistart in the framework of GP, which builds approximating superpositions to the given data, significantly increases its computational complexity.

Then, let us generate rules which have both patterns and replacements from the set of candidates $\mathcal{I}$, see the corresponding bar plot at Fig. 2. Tables 5 and 6 show the distributions of patterns from generated rules according to their structural complexities $C(\cdot)$ and number of parameters $P(\cdot)$.

The tables show that there are again not many superpositions which are simultaneously structurally complex and have many parameters. However, one can see that in the bivariate case, there are significantly more rules which contain complex patterns. Let us explain it. Take a look at Table 2 where the frequencies of primitives are shown. Some primitives tend to appear in patterns significantly more frequently than others. The primitives of zero arity are the most frequent in patterns, but the next popular primitives are bivariate functions, not univariate. It is intuitively clear that patterns, which comprise bivariate functions, tend to be more complex as they have more degrees of freedom. Moreover, they have more terminal vertices in their syntax

Table 5 Distributions of rules patterns according to their structural complexities $C(\cdot)$ and number of parameters $P(\cdot)$ (univariate case)

$C(\cdot)$	$P(\cdot)$						
	0	1	2	3	4	5	6
2	1	3	15	9	1	0	0
3	10	8	12	2	3	2	0
4	10	10	41	22	6	0	0
5	40	18	30	6	7	1	1
6	21	16	26	26	9	0	0
7	73	78	36	0	0	0	0

Table 6 Distributions of rules patterns according to their structural complexities $C(\cdot)$ and number of parameters $P(\cdot)$ (bivariate case)

$C(\cdot)$	$P(\cdot)$						
	0	1	2	3	4	5	6
2	1	3	15	9	1	0	0
3	10	8	12	2	3	2	0
4	9	10	41	22	6	0	0
5	14	18	22	3	7	1	1
6	2	4	9	14	2	0	0
7	18	23	10	0	0	0	0

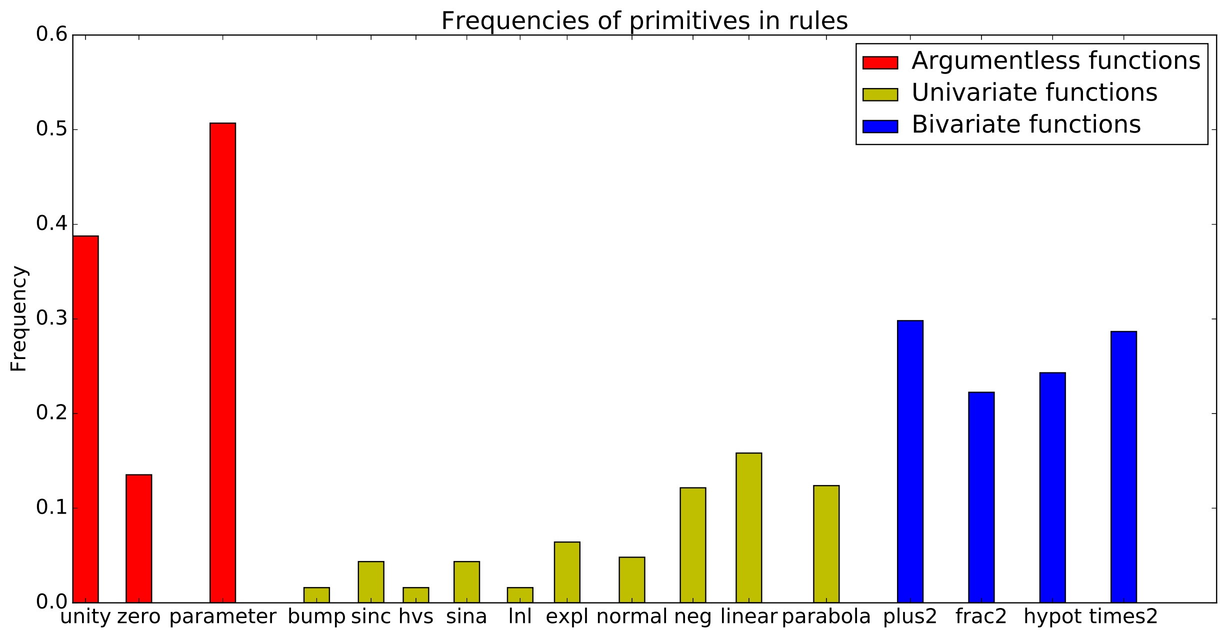

Figure 2 Frequencies of primitives in patterns

trees. Therefore, when the rules are generated in the bivariate case, there are more variants to label these vertices with variables. As a result, more possible rules can be created.

Now, add the primitive minus2$(x_1, x_2) = x_1 - x_2$ to the set. Generate parametric rules again. This produces the following bar plot (Fig. 3). Note that the frequency of minus2 is approximately equal to the one of plus2. Moreover, all complex primitives such as ln and exp occur in patterns significantly rarely. This leads to a notion that one can use these frequencies as primitives order of nonlinearity. Of course, one can see that this estimation could be barely applicable to argumentless functions: though zero is a simple primitive, it is as rare as neg in patterns.

8 Data

Superpositions generated by GP are fitted to approximate volatility of the European stock options. An option is a contract giving the owner the right, but not the obligation, to sell

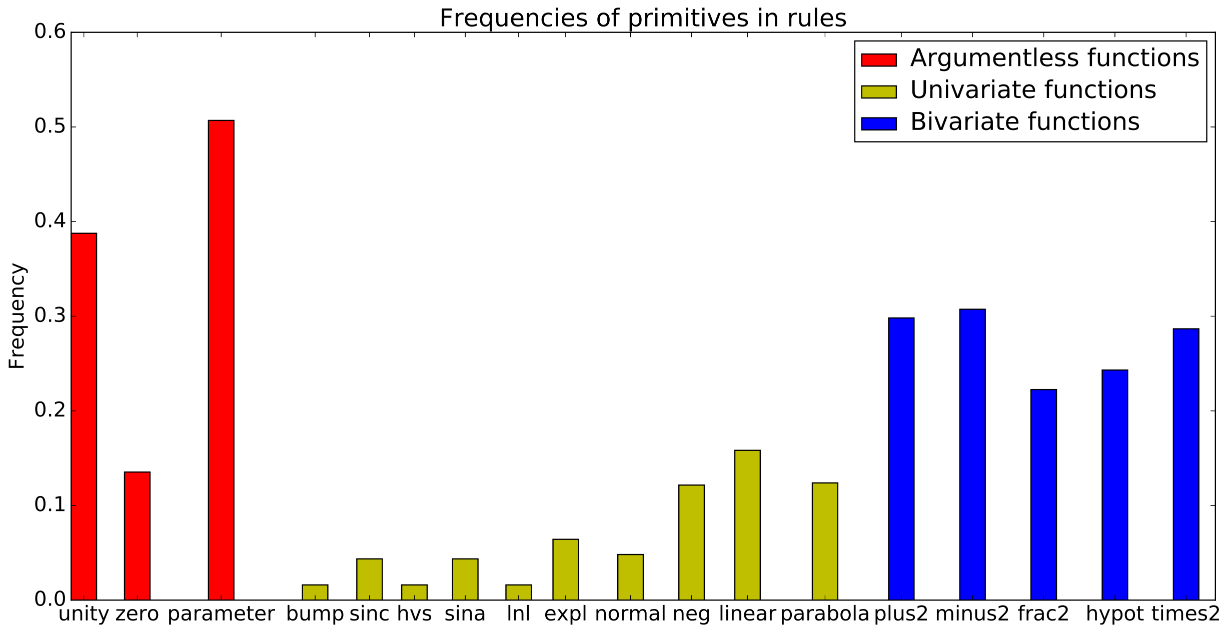

Figure 3 Frequencies of primitives in patterns (with minus2)

a specified amount of an underlying asset at a set price within a specified time called expiration date [15]. A set price of an option is called a strike price [15].

Theoretical estimation of the fair market value of European-style options is given by Black–Scholes formula [15]. It has only one parameter which cannot be observed in the market. This parameter is the average future volatility of the underlying asset. Volatility is the degree of variation of a trading price series over time as measured by the standard deviation of returns. In this paper, the volatility of a financial instrument is investigated over a specified period starting at the current time and ending at the expiration date of an option. It is estimated by the market price of an instrument in assumption that the price is relevant to expected risks.

The Black–Scholes model relies on assumptions which imply the independence of the volatility from the strike price and the expiration date. It is assumed that implied volatility value σ^{imp} is calculated as an argmin of the difference between the historical (stated on the trade) and the fair strike price in the Black–Scholes model:

$$\sigma^{\mathrm{imp}} = \arg\min\bigl(\mathcal{C}_{\mathrm{hist}} - C(\sigma, \mathrm{Pr}, B, K, t)\bigr)$$

where $\mathcal{C}_{\mathrm{hist}}$ is the historical strike price; C is the fair strike price estimated by the Black–Scholes model; Pr is the price of the instrument; B is the bank rate; K is the strike price; and t is the time left to the expiration date.

To estimate the fair strike price, one must approximate the dependence $\sigma(K, t)$ between the strike price of the instrument, its volatility, and the time left to the expiration date. The dataset is collected and described in [16]. In this paper, a superposition has been found which approximates this dependence.

9 Embedding of Simplification System into Genetic Programming

To fit the data, the superpositions of primitives have been generated using symbolic regression and genetic programming [2]. Generation algorithm iteratively creates populations of super-

Table 7 Parameters of GP

Number of crossovers	Number of mutations	Number of randomly generated	Number of best selected models	Error function $\mathcal{S}$	Maximum number of parameters	Maximum $C(f)$	Number of iterations
60	20	20	30	$\mathrm{MSE}(f) + \lambda C(f)$	20	30	25

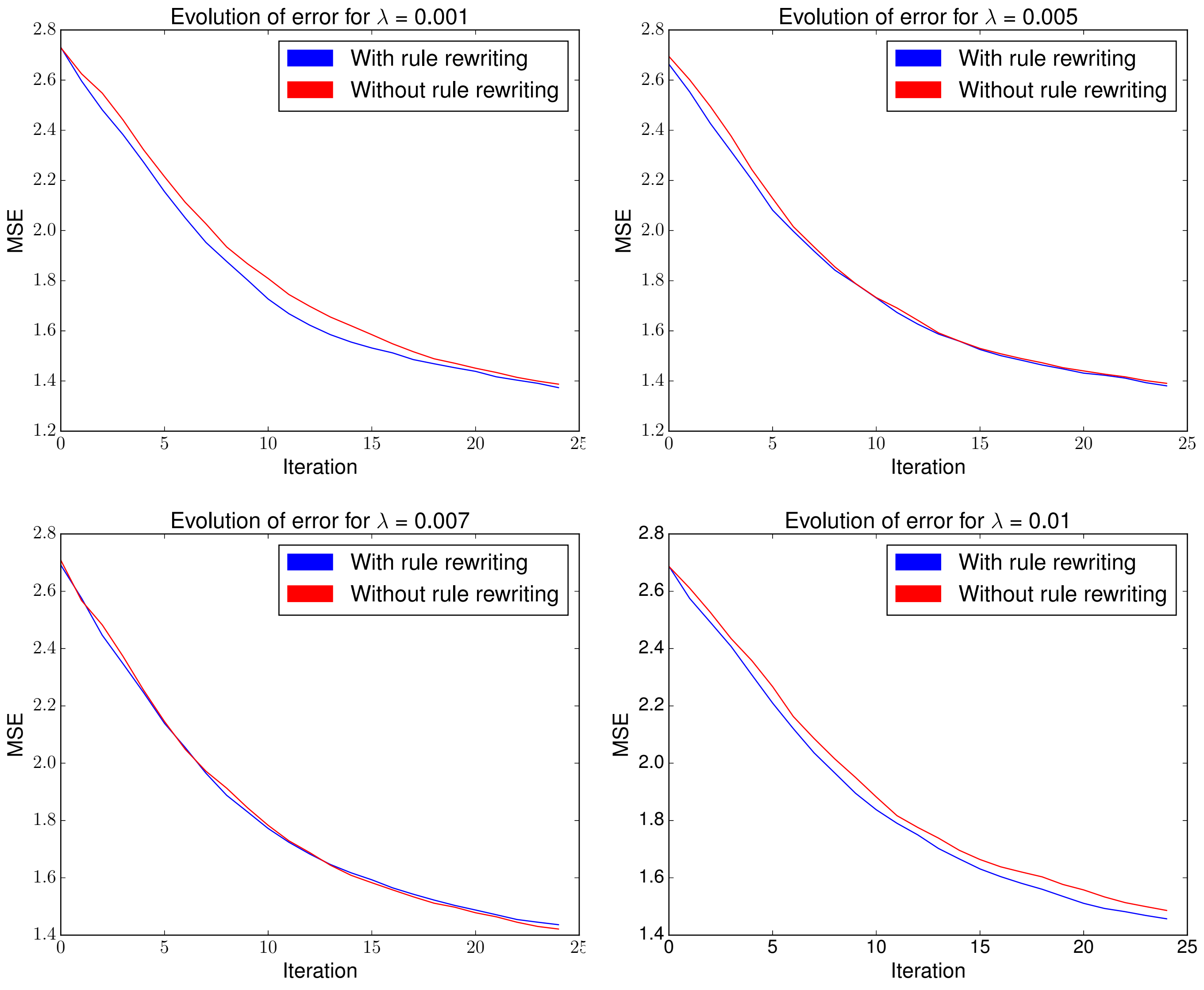

Figure 4 Evolutions of MSE for different λ

positions in the way such that MSE of the best superposition does not increase from iteration to iteration. Initial set consists of 300 randomly generated superpositions of structural complexity equal to 6. Mutation and crossover [1] are the operations used to create new superpositions. On each iteration, some randomly generated superpositions are appended to the populations in order to maintain its diversity. The parameters of the generated parametric superpositions are evaluated by Levenberg–Marquardt algorithm. Then, the best superpositions have been selected from the currently generated population according to an error function $\mathcal{S}(f)$

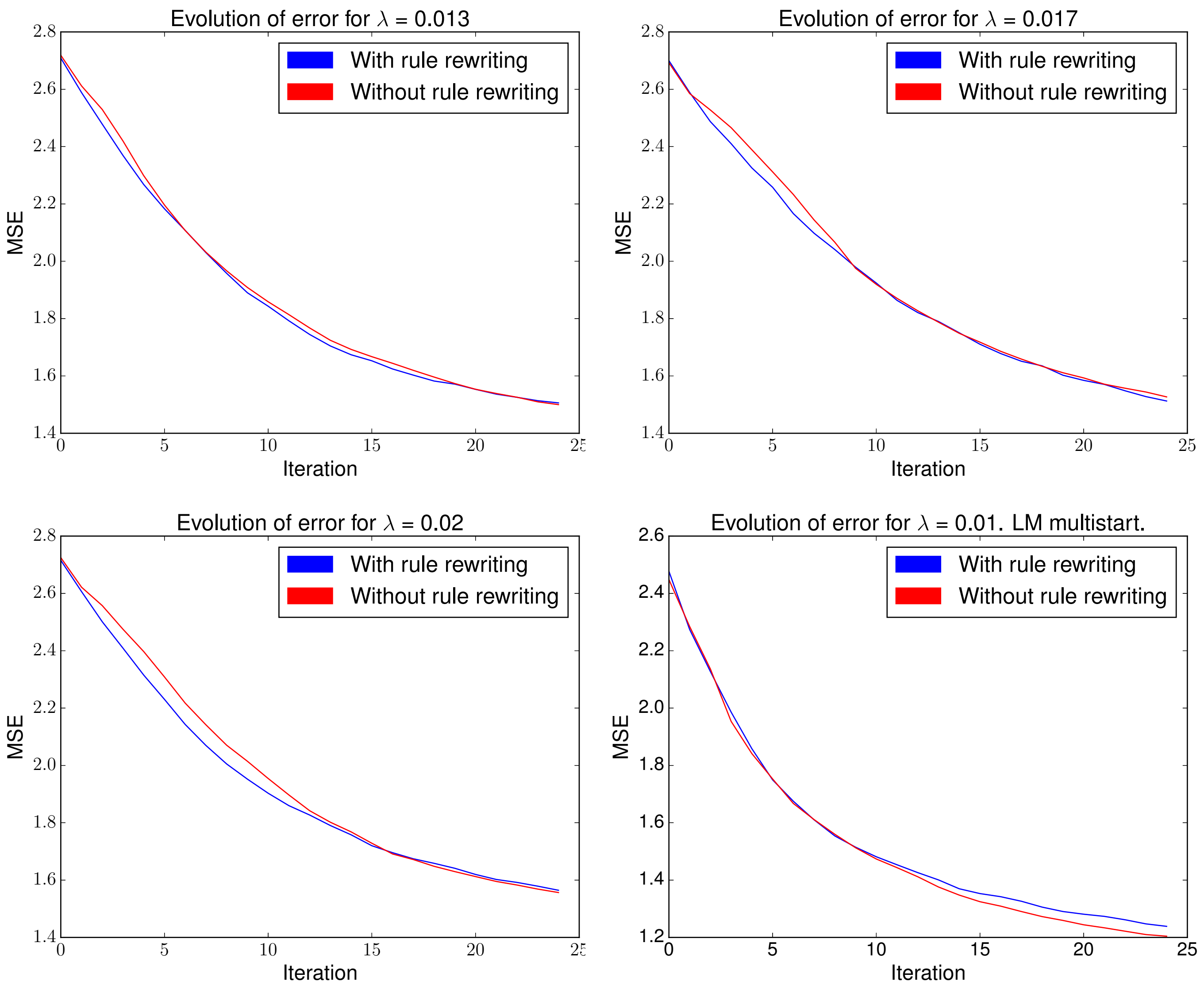

Figure 5 Evolutions of MSE for different λ

$= \mathrm{MSE} + \lambda C(f)$. On each iteration, the best 30 superpositions according to $\mathcal{S}$ form the population for the next iteration. All parameters of GP are shown in Table 7.

The parameter λ in the expression of $\mathcal{S}$ varies for different experiments. Its values are taken from the set $\lambda \in [0.001, 0.005, 0.007, 0.01, 0.013, 0.017, 0.02]$. If it is either smaller than 0.001 or larger than 0.02, the generated superpositions either complicated or inaccurate, respectively. For each value, GP was lauched 400 times, evolutions of MSE of the best approximating superposition were stored, and they were averaged over the launches. The averaged evolutions of MSE were analyzed for two approaches: the basic version of GP and the modified one, which uses simplification system. This analysis in conducted with respect to the value of λ.

The results are presented on the first 7 plots of Figs. 4 and 5. One can see that the results are quite unstable: the gap between these two lines keeps appearing and disappearing. At the same time, it is notable that the blue line, corresponding to the rule rewriting case, is always below the red line. It means that application of the simplification system slightly improves GP. However, the relative difference between the lines becomes negligible by the 25th iteration for all values of λ. Therefore, the actual improvement of GP is not truly significant.

Finally, take a look at the last plot in Fig. 5. In this particular experiment, the same functionality of GP has been used except for the procedure, which estimates parameters of superpositions. In the previous experiments, only one launch of Levenberg–Marquardt algorithm has been used for this purpose. Now, we make three launches with different initial random guesses: we perform the so-called *multistart* procedure. The evolutions of MSE in this case are plotted on the last plot of Fig. 5. One can see that here, we have growing gap between the lines and the basic version of GP is better. Therefore, the slight advantage of the modified GP version is vanished by using of multistart. However, the use of multistart significantly increases the CPU required by GP. Therefore, if one cannot increase computational complexity of GP, the modified version of GP remains a little bit preferable to the basic one.

10 Concluding Remarks

In this paper, an exhaustive algorithm of rules creation has been proposed. These rules are used to simplify superpositions generated by GP. It allows to improve their structural complexity and reduce dimensionality of the parameters space. The distributions of primitives have been analyzed in the created rules. It has been found that the frequency of a primitive could be treated as a crude estimation of its nonlinearity. Averaged evolutions of MSE of the best approximating superposition have been analyzed for basic and modified versions of GP. It has been shown a slight improvement of the modified version over the basic one, though asymptotically they are equivalent. This improvement is vanished when one uses multistart to tune parameters of superpositions by Levenberg–Marquardt algorithm. However, multistart significantly increases the computational complexity of GP.

References

[1] Koza, J. R. 1992. *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA: MIT Press. 836 p.

[2] Zelinka, I., Z. Oplatkova, and L. Nolle. 2005. Analytic programming — symbolic regression by means of arbitrary evolutionary algorithms. *Int. J. Simul. Syst. Sci. Technol.* 6(9):44–56.

[3] Eads, D., D. Hill, S. Davis, *et al.* 2002. Genetic algorithms and support vector machines for time series classification. *Applications and science of neural networks, fuzzy systems, and evolutionary computation V*. Eds. B. Bosacchi, D. B. Fogel, and J. C. Bezdek. Proc. SPIE. Vol. 4787. doi: 10.1117/12.453526.

[4] Eads, D., K. Glocer, S. Perkins, and J. Theiler. 2005. Grammar-guided feature extraction for time series classification. *9th Annual Conference on Neural Information Processing Systems Proceedings*. 8 p.

[5] Wagner, N., Z. Michalewicz, M. Khouja, and R. R. McGregor. 2007. Time series forecasting for dynamic environments: The dyfor genetic program model. *IEEE Trans. Evolut. Comput.* 11(4):433–452. doi: 10.1109/TEVC.2006.882430.

[6] Pennachin, C., M. Looks, and J. A. de Vasconcelos. 2011. Improved time series prediction and symbolic regression with affine arithmetic. *Genetic programming theory and practice IX*. Eds. R. Riolo, E. Vladislavleva, and J.-H. Moore. Genetic and evolutionary computation ser. New York, NY: Springer-Verlag. 97–112.

[7] Sheta, A. F., S. E. M. Ahmed, and H. Faris. 2015. Evolving stock market prediction models using multigene symbolic regression Genetic Programming. *AIML* 15(1).

[8] Icke, I., and A. Rosenberg. 2010. Dimensionality reduction using symbolic regression. *12th Annual Conference Companion on Genetic and Evolutionary Computation Proceedings*. New York, NY: ACM. 2085–2086. doi: 10.1145/1830761.1830874.

[9] Poli, R., W. B. Langdon, and N. F. McPhee. 2008. *A field guide to Genetic Programming*. Lulu Enterprises, UK Ltd. 250 p.

[10] Soule, T., J. A. Foster, and J. Dickinson. 1996. Code growth in genetic programming. *1st Annual Conference on Genetic Programming Proceedings*. Cambridge, MA: MIT Press. 215–223. Available at: `http://dl.acm.org/citation.cfm?id=1595536.1595563` (accessed May 31, 2017).

[11] Soule, T., and J. A. Foster. 1997. Support for multiple causes of code growth in GP. *Workshop on Evolutionary Computation with Variable Size Representation at ICGA-97*. East Lansing, MI.

[12] Mori, N., R. McKay, X. H. Nguyen, and D. Essam. 2007. Equivalent decision simplification: A new method for simplifying algebraic expressions in Genetic Programming. *11th Asia-Pacific Symposium on Intelligent and Evolutionary Systems Proceedings*. Yokosuka, Japan.

[13] Iba, H., H. de Garis, and T. Sato. 1994. Genetic Programming using a minimum description length principle. *Advances in Genetic Programming*. Ed. K. E. Kinnear, Jr. Cambridge, MA: MIT Press. 265–284.

[14] Levenick, J. 1999. Swappers: Introns promote flexibility, diversity and invention. *1st Annual Conference on Genetic and Evolutionary Computation Proceedings*. San Francisco, CA: Morgan Kaufmann Publishers Inc. 1:361–368.

[15] Hull, J. C. 2008. *Options, futures, and other derivatives*. 7th ed. Prentice Hall. 848 p.

[16] Strijov, V. 2009. The inductive generation of the volatility smile models. Ph.D. Thesis.

Received December 15, 2016

Порождение параметрических правил упрощения алгебраических выражений в задаче символьной регрессии[*]

А. С. Кулунчаков

kulu-andrej@yandex.ru

Московский физико-технический институт, Россия, г. Долгопрудный, Институтский пер., д. 9

Исследуется проблема раздувания кода в символьной регрессии. Предлагается процедура упрощения суперпозиций, порождаемых символьной регрессией. Предлагаемый подход основан на идее эквивалентных преобразований суперпозиций, которая применяется к порождению параметрических правил упрощения. Помимо удаления неэффективного кода суперпозиций эти правила сокращают размерность их пространства параметров. Вычислительный эксперимент проводится на выборке по опционам Brent Crude Oil. Их волатильность аппроксимируется через цену исполнения опциона и дату окончания его срока действия.

Ключевые слова: *раздутие кода; упрощение по правилам; символьная регрессия; генетическое программирование*

DOI: 10.21469/22233792.3.1.01

Литература

[1] *Koza J. R.* Genetic Programming: On the programming of computers by means of natural selection. — Cambridge, MA, USA: MIT Press, 1992. 836 p.

[*]Работа была поддержана грантом РФФИ № 16-37-00486.

[2] *Zelinka I., Oplatkova Z., Nolle L.* Analytic programming — symbolic regression by means of arbitrary evolutionary algorithms // Int. J. Simul. Syst. Sci. Technol., 2005. Vol. 6. No. 9. P. 44–56.

[3] *Eads D., Hill D., Davis S., et al.* Genetic algorithms and support vector machines for time series classification // Applications and science of neural networks, fuzzy systems, and evolutionary computation V / Eds. B. Bosacchi, D. B. Fogel, J. C. Bezdek. — Proc. SPIE, 2002. Vol. 4787. doi: 10.1117/12.4535526.

[4] *Eads D., Glocer K., Perkins S., Theiler J.* Grammar-guided feature extraction for time series classification // 9th Annual Conference on Neural Information Processing Systems Proceedings, 2005. 8 p.

[5] *Wagner N., Michalewicz Z., Khouja M., McGregor R. R.* Time series forecasting for dynamic environments: The dyfor genetic program model // IEEE Trans. Evolut. Comput., 2007. Vol. 11. No. 4. P. 433–452. doi: 10.1109/TEVC.2006.882430.

[6] *Pennachin C., Looks M., de Vasconcelos J. A.* Improved time series prediction and symbolic regression with affine arithmetic // Genetic programming theory and practice IX / Eds. R. Riolo, E. Vladislavleva, J.-H. Moore. — Genetic and evolutionary computation ser. — New York, NY, USA: Springer-Verlag, 2011. P. 97–112.

[7] *Sheta A. F., Ahmed S/E. M., Faris H.* Evolving stock market prediction models using multigene symbolic regression Genetic Programming // AIML, 2015. Vol. 15. Iss. 1.

[8] *Icke I., Rosenberg A.* Dimensionality reduction using symbolic regression // 12th Annual Conference Companion on Genetic and Evolutionary Computation Proceedings. — New York, NY, USA: ACM, 2010. P. 2085–2086. doi: 10.1145/1830761.1830874.

[9] *Poli R., Langdon W. B., McPhee N. F.* A field guide to Genetic Programming. — Lulu Enterprises, UK Ltd., 2008. 250 p.

[10] *Soule T., Foster J. A., Dickinson J.* Code growth in genetic programming // 1st Annual Conference on Genetic Programming Proceedings. — Cambridge, MA, USA: MIT Press, 1996. P. 215–223. http://dl.acm.org/citation.cfm?id=1595536.1595563.

[11] *Soule T., Foster J. A.* Support for multiple causes of code growth in GP // Workshop on Evolutionary Computation with Variable Size Representation at ICGA-97. — East Lansing, MI, USA, 1997.

[12] *Mori N., McKay R., Nguyen X. H., Essam D.* Equivalent decision simplification: A new method for simplifying algebraic expressions in Genetic Programming // 11th Asia-Pacific Symposium on Intelligent and EvolutionarySistems Proceedings. — Yokosuka, Japan, 2007.

[13] *Iba H., de Garis H., Sato T.* Genetic Programming using a minimum description length principle // Advances in Genetic Programming / Ed. K. E. Kinnear, Jr. — Cambridge, MA, USA: MIT Press, 1994. P. 265–284.

[14] *Levenick J.* Swappers: Introns promote flexibility, diversity and invention // 1st Annual Conference on Genetic and Evolutionary Computation Proceedings. — San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999. Vol. 1. P. 361–368.

[15] *Hull J. C.* Options, futures, and other derivatives. — 7th ed. — Prentice Hall, 2008. 848 p.

[16] *Strijov V.* The inductive generation of the volatility smile models. Ph. D. Thesis, 2009.

Поступила в редакцию 15.12.2016