Accelerated Genetic Programming of Polynomials

Nikolay I. Nikolaev
Dept. of Math. & Computing Sciences
Goldsmiths College
University of London
New Cross
London SE14 6NW
United Kingdom
nikolaev@mcs.gold.ac.uk

Hitoshi Iba
Dept. of Inf. and Comm. Engineering
School of Engineering
The University of Tokyo
7-3-1 Hongo, Bunkyo-ku
Tokyo 113-8656
Japan
iba@miv.t.u-tokyo.ac.jp

Abstract

An accelerated polynomial construction technique for genetic programming is proposed. This is a horizontal technique for gradual expansion of a partial polynomial during traversal of its tree-stuructured representation. The coefficients of the partial polynomial and the coefficient of the new term are calculated by a rapid recurrent least squares (RLS) fitting method. When used for genetic programming (GP) of polynomials this technique enables us not only to achieve fast estimation of the coefficients, but also leads to power series models that differ from those of traditional Koza-style GP and from those of the previous GP with polynomials STROGANOFF. We demonstrate that the accelerated GP is sucessful in that it evolves solutions with greater generalization capacity than STROGANOFF and traditional GP on symbolic regression, pattern recognition, and financial time-series prediction tasks.

1 Introduction

Genetic Programming (GP) [1, 16] has already been considered for function approximation. Instances of the function approximation problem arise in real-world applications, such as: nonlinear optimization [8], nonlinear process modeling [9], image analysis [24, 25, 27], data mining [6, 23], system identification [12], financial engineering [3, 14, 17], chaotic time-series prediction [18, 21, 31]. All these traditional GP systems evolve functions recursively constructed from primitive functions, using tree representations. The advantage of such GP over other function learners is that they automatically discover the underlying structure of the function. These GP approaches, however, are still of limited practical use because they produce solutions with uncertain mathematical properties and operate slowly with numerical values.

Our recent research has identified two dominant issues that seriously affect the results and speed of these GP systems: 1) the lack of clear reasons for the selection of a particular set of primitive functions to enter the model; 2) the slow computation of the numeric function parameters and constants. Researchers working on evolutionary induction of tree-structured models usually arbitrarily choose the set of primitive functions to accommodate in the tree nodes [1, 16]. Another drawback is the somewhat arbitrary combination of the primitive functions which leads to models with often insufficient interpolation and extrapolation potential. An alternative is provided by the GP approaches that breed tree-like series expansions, such as power series [12, 13, 23, 27], Fourier series [8], and additive series [18]. These approaches employ the GP paradigm to search for function models with known mathematical properties, and well defined primitive functions whose optimal parameters are obtained by least squares methods.

This paper addresses these issues by developing an accelerated polynomial construction technique especially conceived for learning power series function models by GP. The accelerated GP relies on polynomials composed of transfer polynomials allocated in the nodes of tree structures, as its predecessor STROGANOFF [12, 13]. Keeping the hierarchical compositions, the novelty is a horizontal technique for gradual expansion of a partial polynomial by one term at each intermediate node during tree traversal, the new term being indicated by the transfer polynomial at that node. The coefficients of the partial polynomial and the coefficient of the new term are estimated by a rapid recurrent least squares (RLS) method [5]. Thus, the coefficients are directly estimated, avoiding the need to search for their values, and, as in STROGANOFF, search is performed only in the space of polynomial structures. The benefit of RLS is in the reduction of the computational overhead due to repetitive matrix inversions in the ordinary least squares methods.

The horizontal technique for polynomial construction from trees imposes special requirements on the selection of the set of transfer polynomials, which along with these requirements should remain suitable for evolutionary

GP search. The first requirement on the transfer polynomials is that they enable successive growth of the partial polynomial by exactly one new term at each node. The second requirement is that the transfer polynomials are short enough to be manipulated efficiently even by ordinary least squares fitting when they are at the lowest, fringe tree nodes. These two requirements are met by designing a fixed set of six incomplete bivariate transfer polynomials. The set contains a small number of polynomials in order to limit the search spaces for the GP to explore.

In this paper we are concerned with the following questions: 1) does the accelerated horizontal technique speed-up the polynomial processing and, therefore, the evolutionary search process; and 2) what is the approximation quality of the results discovered by the accelerated GP equipped with the horizontal technique compared to those found by a GP with the vertical technique, and with respect to those produced by traditional Koza-style GP [16]. The accelerated system developed with the horizontal technique called *f*GP (*fast* GP) is a version of STROGANOFF, which uses the vertical technique. The *f*GP system is related to an implementation of traditional GP abbreviated as SGPC-1.1 [28]. These three GPs are prepared to use the same evolutionary computation micromechanisms in order to facilitate the comparison: they use fitness proportional selection, one-point mutation, and subtree crossover at random points. A special statistical fitness function is applied to stimulate the survival of accurate, parsimonious, and predictive polynomials. Instances of the function approximation problem from the fields of symbolic regression, pattern recognition, and financial forecasting are addressed.

The empirical results show that the horizontal technique used for genetic programming of polynomials not only achieves fast coefficients estimation, but also leads to power series models different from those of traditional Koza-style GP and from those of the previous GP with polynomials STROGANOFF. We demonstrate that the accelerated GP is successful in that it evolves solutions with higher accuracy and generalization, but similar complexity compared to those of the previous system STROGANOFF and outperform those from the traditional GP system SGPC 1.1 on the considered tasks.

Our paper is organized as follows. Section 2 outlines the polynomial models, including the ordinary least squares fitting algorithm, and the tree-like representation of polynomials in the system STROGANOFF. Section 3 explains the horizontal technique for polynomial construction from trees, the recurrent least squares fitting method, and also analyzes the theoretical and practical improvement in speed of processing. Section 4 illustrates the GP mechanisms: the set of transfer polynomials, the statistical fitness function and the genetic learning operators. Experimental results are given in Section 5. Finally, conclusions are drawn.

2 Polynomial Models

The objective of polynomial function approximation is to infer a polynomial $p(\mathbf{x})$ that is capable of mapping reliably not only the provided data, but also on average unseen data which their source may generate. The approximation problem is: given instantiated vectors of independent variables $\mathbf{x}_i = (x_{i1}, x_{i2}, ..., x_{id}) \in \mathcal{R}^d$, and dependent variable values $y_i \in \mathcal{R}$, find models $p(\mathbf{x})$ optimally close to their unknown source function $f(\mathbf{x}) = y$.

High-order multivariate polynomials are a universal format for function modeling with which any continuous mapping may be approximated up to an arbitrary precision, in average squared residual sense, if there are a sufficiently large number of terms. These polynomials are discrete analogues of the Volterra series known as *Gabor-Kolmogorov polynomials* [7], and are defined by the power series[1]:

$$p(\mathbf{x}) = a_0 + \sum_{m=1}^{M} a_m \prod_{j=1}^{d} x_j^{r_{jm}} \tag{1}$$

where a_m are the term coefficients, m ranges up to a pre-selected maximum number of terms M: $m \leq M$; x_j are the independent variable values of the input vector $\mathbf{x}$, $j \leq d$ numbers; and $r_{jm} = 0, 1, ...$ are the powers with which the j-th element x_j participates in the m-th term. It is assumed that r_{jm} is bound by a maximum polynomial order (degree) s: $\sum_{j=1}^{d} r_{jm} \leq s$ for every m.

These polynomials (1) are combinations of terms which are linear in the coefficients, and non-linear in the variables. One may envision that the cross-product and power terms are derived from a standard basis: $\boldsymbol{\phi} =(1$, x_1, x_2,..., x_d, x_1^2, x_1x_2, x_1x_3, ..., x_1^3, $x_1^2x_2$, $x_1^2x_3$, ...). Using this basis $\boldsymbol{\phi}$, a high-order multivariate polynomial can be concisely defined with the equation:

$$p(\mathbf{x}) = \sum_{m=0}^{M} a_m \phi_m(\mathbf{x}) \tag{2}$$

where a_m are the coefficients, and ϕ_m are the basis functions from $\boldsymbol{\phi}$. Instead of ϕ_m, basis functions h_m are used in what follows to emphasize that only a small subset of $\boldsymbol{\phi}$ is considered: $h_0(\mathbf{x}) = 1$, $h_1(\mathbf{x}) = x_1$, $h_2(\mathbf{x}) = x_2$, $h_3(\mathbf{x}) = x_1x_2$, $h_4(\mathbf{x}) = x_1^2$, $h_5(\mathbf{x}) = x_2^2$, $h_6(\mathbf{x}) = x_1^2x_2$, $h_7(\mathbf{x}) = x_1x_2^2$, and $h_8(\mathbf{x}) = x_2^3$. The idea behind the selection of a small set of basis functions is to enable practical construction of sparse polynomials by avoiding the exponential increase of the number of polynomial terms with the increase of the input dimension. The idea behind the selection of low order polynomials is to increase the overall polynomial degree slightly when composing the basis polynomials.

[1]Strictly speaking, a power series contains an infinite number of terms that can exactly represent a function, while for practical approximation a finite number of them is sufficient. The polynomial size M is manually fixed by a design decision.

2.1 Least Squares Fitting

Having a finite number N of data $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ from independent variable vectors $\mathbf{x}_i$ and their corresponding outcomes y_i, mean square error methods can be used for finding the polynomial coefficients a_m. These methods seek the minimum of the closeness criterion $\sum_{i=1}^{N} [y_i - p(\mathbf{x}_i)]^2$. Such is the *ordinary least squares* (OLS) fitting method defined with the normal equation:

$$\mathbf{a} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{y} \tag{3}$$

where $\mathbf{a}$ is $(m+1)\times 1$ column vector, $\mathbf{H}$ is $N\times(m+1)$ design matrix of row vectors $\mathbf{h}(\mathbf{x}_i) = (h_0(\mathbf{x}_i), h_1(\mathbf{x}_i), ..., h_m(\mathbf{x}_i))$, $m \leq M$, $i = 1..N$, $\mathbf{y}$ is the $N \times 1$ output column vector, and $\mathbf{Ha} = \mathbf{p}$. Since the covariance matrix $\mathbf{H}^T\mathbf{H}$ is symmetric, we use Cholesky decomposition to derive efficiently its inverse [26].

This OLS method (3) has a unique solution, that is a best fitting polynomial which is unique at the given data points [2].

2.2 Tree-Structured Polynomials

The polynomials describing real world data can not be directly estimated with equation (3) since they usually contain a large number of terms, and this causes computational difficulties. The Group Method of Data Handling (GMDH) [15, 19] provides a remedy for this "curse of dimensionality" problem. GMDH builds a polynomial by an hierarchical composition of simple transfer polynomials whose coefficients are calculated easily by least squares methods. The GMDH technique implicitly suggests what kind of primitive functions to select: these should be very low order transfer polynomials since when composed hierarchically they should not rapidly increase the overall polynomial degree.

The GP systems from the STROGANOFF family [12, 13] considered such polynomials represented as *tree structures,* in resemblance to GMDH. Each tree contains complete bivariate second degree transfer polynomials: $p(\mathbf{x}) = a_0 + a_1x_1 + a_2x_2 + a_3x_1x_2 + a_4x_1^2 + a_5x_2^2$, in the internal functional nodes, and independent variables in the terminal leaves. A polynomial is built from a tree with a *vertical technique.* The coefficients of the transfer polynomials at each functional node are computed by OLS fitting (3). After that, the polynomials are applied to the examples to produce outcome vectors that form the design matrix $\mathbf{H}$. Next, at higher tree levels these outcomes feed the independent variables of the transfer polynomials in the nodes.

2.3 The Set of Transfer Polynomials

The GP may be expected to yield polynomials with very good approximation performance if the tree-like cascades contain different transfer polynomials in the nodes rather than one transfer polynomial only. A set of transfer polynomials increases the flexibility of GP to fit the data as it reduces its statistical bias. An important requirement for the selection of the transfer polynomials is that they are *linearly independent* otherwise when composed they will not add further details to the partial model [2]. The horizontal technique requires such a set of transfer polynomials that enables growth of the partial polynomials by exactly one term. Adopting binary trees, this means that eligible candidates are *bivariate polynomials* in which one of the independent variables appears in a separate term, or only once in a cross-product term with the other variable.

Six bivariate transfer polynomials are selected (Table 1). When an intermediate functional tree node is visited during the tree traversal process, the rightmost transfer polynomial term specifies the new term to add. The partial polynomials are still hierarchical because the tree hierarchy indicates the new term to be added, and still compositions since some of the new terms are products not only from independent variables directly, but also utilize outcomes of transfer polynomials from preceding layers in the tree.

This tree-like hierarchical composition allows us to obtain at the root a polynomial model highly non-linear in the variables by means of efficient linear approximation at each node.

3 Speeding-up the GP of Polynomials

3.1 Accelerated Polynomial Construction

A drawback of the vertical technique is that it is computationally expensive because at each tree node OLS is performed. Suggestions for avoiding the large number of multiplications and matrix inversions involved in formula (3) provide the recursive numeric techniques for coefficient estimation from linear algebra.

Inspired from the RLS method [5], we propose a *horizontal technique* for accelerated polynomial construction from tree-structures. The idea is to grow progressively a partial polynomial by adding one new term at each intermediate node, with a directly computed coefficient, while only updating the old partial polynomial coefficients passed from a subtree below. The horizontal scheme relies on *right-left-root* tree traversal, starting from the lowest rightmost functional node. The coefficients at the lowest fringe nodes, having two leaf terminal nodes, are calculated by OLS fitting (nodes p_1 and p_4 in Figure 1). At the intermediate nodes the new and old partial polynomial coefficients are estimated by RLS fitting. The rightmost term of the transfer polynomial indicates which term should be added (Table 1) as explained below.

Transfer Polynomials

$p_1(\mathbf{x}) = a_0 + a_1x_1 + a_2x_2$

$p_2(\mathbf{x}) = a_0 + a_1x_1 + a_2x_1^2x_2$

$p_3(\mathbf{x}) = a_0 + a_1x_1 + a_2x_1{x_2}^2$

$p_4(\mathbf{x}) = a_0 + a_1x_1 + a_2{x_2}^2$

$p_5(\mathbf{x}) = a_0 + a_1x_1 + a_2x_1x_2$

$p_6(\mathbf{x}) = a_0 + a_1x_1 + a_2x_2^3$

Table 1. The set of transfer polynomials

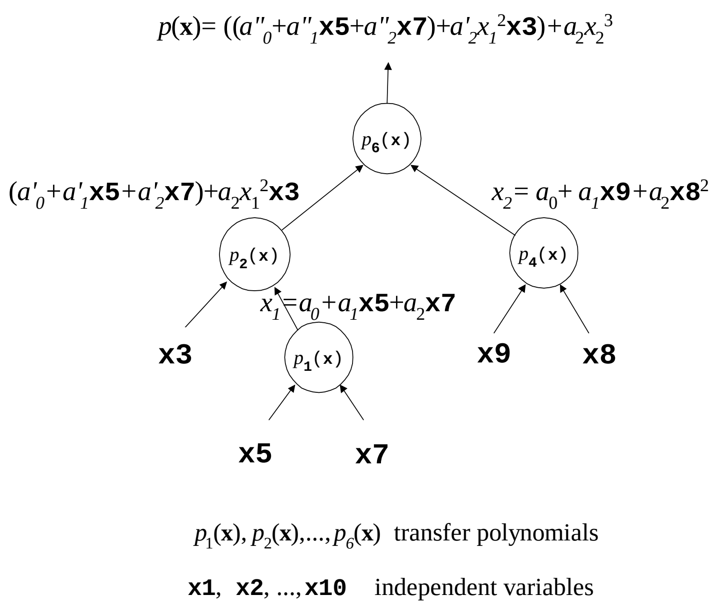
$p(\mathbf{x})= ((a''_0+a''_1\mathbf{x5}+a''_2\mathbf{x7})+a'_2x_1^2\mathbf{x3})+a_2x_2^3$
$p_6(\mathtt{x})$
$(a'_0+a'_1\mathbf{x5}+a'_2\mathbf{x7})+a_2x_1^2\mathbf{x3}$
$x_2= a_0+ a_1\mathbf{x9}+a_2\mathbf{x8}^2$
$p_2(\mathtt{x})$
$p_4(\mathtt{x})$
$x_1=a_0+a_1\mathbf{x5}+a_2\mathbf{x7}$
x3
$p_1(\mathtt{x})$
x9
x8
x5
x7
$p_1(\mathbf{x}), p_2(\mathbf{x}),\ldots, p_6(\mathbf{x})$ transfer polynomials
x1, x2, ..., x10 independent variables

Figure 1. Tree-like polynomial

The partial polynomial passed from a subtree may be expanded, but also may feed an independent variable at the parent node transfer polynomial and, thus, be suppressed from a further expansion. Depending on the tree structure there are two cases to consider: 1) at a functional tree node with one terminal leaf child, left or right, the partial polynomial is extended with the independent variable at the leaf (node p_2 in Figure 1); and 2) when a functional tree node has two children functional nodes the partial sub-polynomial from the left child is considered for further extension (root node p_6 in Figure 1). The right partial sub-polynomial outcome is used to feed the second independent variable, and it is not extended (node p_4 in Figure 1). The effect is a proportional extension of the partial polynomial by the right sub-polynomial whose coefficients are proportionally changed leading to a kind of restricted least squares regression.

Employing the horizontal technique leads to different polynomials than those from the vertical one, that is the resulting polynomials have different coefficients even if their trees of transfer polynomials are the same. In the two cases, polynomials with minimal error on the examples are derived in the intermediate tree nodes.

3.2 Recurrent Least Squares

The *recurrent least squares* (RLS) method [5] (pp.163-167) serves for implementing iterative regression by adding successively new variables to the model. The horizontal technique for accelerated GP of polynomials employs RLS to estimate the polynomial coefficients in all functional nodes except the fringe nodes. The RLS supplies formulae for direct computation of the new term coefficient, and for recurrent updating of the inverse covariance matrix $\mathbf{C}^{-1} = (\mathbf{H}^T\mathbf{H})^{-1}$. The benefit is eliminating the need to do time consuming matrix inversions, which allows to achieve fast identification of the partial polynomials at the intermediate tree nodes.

Assume that a partial polynomial $p_k(\mathbf{x})$ with k terms $(1 \leq k \leq M)$ has been generated somewhere at an intermediate functional tree node. The normal equation (3) can be simplified using the substitutions:

$$\begin{bmatrix} \mathbf{C}_k & \mathbf{u}_k \end{bmatrix} \equiv \begin{bmatrix} \mathbf{H}_k^T \mathbf{H}_k & \mathbf{H}_k^T \mathbf{y} \end{bmatrix} \tag{4}$$

where $\mathbf{C}_k \equiv \mathbf{H}_k^T \mathbf{H}_k$ and $\mathbf{u}_k \equiv \mathbf{H}_k^T \mathbf{y}$. The vector of least squares estimates of the coefficients $\mathbf{a}_k$ is then given by the concise matrix equation:

$$\mathbf{a}_k = \mathbf{C}_k^{-1} \mathbf{u}_k \tag{5}$$

where $\mathbf{C}_k^{-1}$ is the inverse of the covariance matrix $\mathbf{C}_k$.

After adding a $(k+1)$-th independent variable to the partial polynomial $p_k(\mathbf{x})$, the matrix $\mathbf{H}_{k+1}$ becomes one more column larger $\mathbf{H}_{k+1} = \begin{bmatrix} \mathbf{H}_k & \mathbf{h}_{k+1} \end{bmatrix}$. Then the *updated coefficients* $\mathbf{a}_{k+1}$ have to be derived from the augmented matrix of normal equations:

$$\mathbf{a}_{k+1} = \begin{bmatrix} \mathbf{C}_k & \mathbf{c}_{k+1} \\ \mathbf{c}_{k+1}^T & j_{k+1} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{u}_k \\ v_{k+1} \end{bmatrix} \tag{6}$$

where $\mathbf{c}_{k+1}$, j_{k+1}, and v_{k+1} are components *bordering* the old covariance matrix $\mathbf{C}_k$.

The bordering components are defined as follows:

$$\mathbf{c}_{k+1} = \mathbf{H}_k^T \mathbf{h}_{k+1} \tag{7}$$

$$j_{k+1} = \mathbf{h}_{k+1}^T \mathbf{h}_{k+1} \tag{8}$$

$$v_{k+1} = \mathbf{h}_{k+1}^T \mathbf{y} \tag{9}$$

where $\mathbf{h}_{k+1}$ is a $N \times 1$ column vector with the values of the new $(k+1)$-st independent variable[2].

The novel coefficient vector $\mathbf{a}_{k+1} = \mathbf{C}_{k+1}^{-1} \mathbf{u}_{k+1}$ may be obtained without the computationally expensive inversion of the covariance matrix $\mathbf{C}_{k+1} = \mathbf{H}_{k+1}^T \mathbf{H}_{k+1}$, but rather incrementally using the known quantities $\mathbf{C}_k^{-1}$, and $\mathbf{u}_k$ available from the computation of the polynomial with k variables.

The elements of the vector $\mathbf{a}_{k+1}$ are derived by solving the equation $\mathbf{C}_{k+1}\mathbf{C}_{k+1}^{-1} = \mathbf{I}$, where $\mathbf{I}$ is $(k+1) \times (k+1)$ identity matrix (see Appendix I). The solution requires several steps. First, the components $\mathbf{c}_{k+1}$ (7), j_{k+1} (8), v_{k+1} (9) are used to build the blocks of the new matrix $\mathbf{C}_{k+1}^{-1}$. Performing transformations of the resulting block matrix, one infers an element g_{k+1} for the lowest right corner of the matrix $\mathbf{C}_{k+1}^{-1}$.

The element g_{k+1} is a scalar computable as follows:

$$g_{k+1} = (j_{k+1} - \mathbf{c}_{k+1}^T \mathbf{C}_k^{-1} \mathbf{c}_{k+1})^{-1} \tag{10}$$

where the components $\mathbf{C}_k^{-1}$ and $\mathbf{c}_{k+1}$ as well as the scalar j_{k+1} in the denominator are known.

[2]According to the horizontal technique the values of the new column vector $\mathbf{h}_{k+1}$ can be not only values of independent variables given by the examples $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, but also outcomes from a partial polynomial fed by some preceding tree node.

Second, this scalar g_{k+1} is used in (6) to express the remaining blocks of the matrix $\mathbf{C}_{k+1}^{-1}$. After that, multiplication of the matrix $\mathbf{C}_{k+1}^{-1}$ with the vector $\mathbf{u}_{k+1}$ is performed, resulting at $\mathbf{a}_{k+1}$. The produced vector $\mathbf{a}_{k+1}$ is simplified taking into account that $\mathbf{a}_k = \mathbf{C}_k^{-1}\mathbf{u}_k$ (5). This yields at the bottom of the column vector $\mathbf{a}_{k+1}$ a formula for direct estimation of the new coefficient a_{k+1}:

$$a_{k+1} = -g_{k+1}(\mathbf{c}_{k+1}^T\mathbf{a}_k - v_{k+1}) \tag{11}$$

Third, the new coefficient a_{k+1} is used in the upper block of $\mathbf{a}_{k+1}$:

$$\mathbf{a}_{k+1} = \begin{bmatrix} \mathbf{a}_k - a_{k+1}\mathbf{C}_k^{-1}\mathbf{c}_{k+1} \\ a_{k+1} \end{bmatrix} \tag{12}$$

to update the old coefficients $\mathbf{a}_k$ by the amount $a_{k+1}\mathbf{C}_k^{-1}\mathbf{c}_{k+1}$.

Finally, the *new inverse matrix* $\mathbf{C}_{k+1}^{-1}$ for successive computations is obtained:

$$\mathbf{C}_{k+1}^{-1} = \begin{bmatrix} \mathbf{C}_k^{-1} + g_{k+1}\mathbf{C}_k^{-1}\mathbf{c}_{k+1}\mathbf{c}_{k+1}^T\mathbf{C}_k^{-1} & -g_{k+1}\mathbf{C}_k^{-1}\mathbf{c}_{k+1} \\ -g_{k+1}\mathbf{c}_{k+1}^T\mathbf{C}_k^{-1} & g_{k+1} \end{bmatrix} \tag{13}$$

It should be noted that the inverse matrix $\mathbf{C}_k^{-1}$ is symmetric, and therefore only half of the operations are sufficient to find the coefficients $\mathbf{a}_{k+1}$, and the new inverse matrix $\mathbf{C}_{k+1}^{-1}$.

This RLS method requires us to calculate once the column vector $\mathbf{C}_k^{-1}\mathbf{c}_{k+1}$, used three times in (10),(12),(13), after which its transpose necessary for (13) is easily derived. The scalar g_{k+1} needs to be computed also once. The least squares coefficients vector $\mathbf{a}_{k+1}$ is next estimated with these known blocks. These block components, namely the column vector $\mathbf{C}_k^{-1}\mathbf{c}_{k+1}$, its transposed row vector $\mathbf{c}_{k+1}^T\mathbf{C}_k^{-1}$, and the scalar g_{k+1} are taken ready to construct the new inverse matrix $\mathbf{C}_{k+1}^{-1}$ by formula (13), which is computationally efficient.

3.3 Time Complexity

The theoretical improvement of GP achieved with the accelerated system using the horizontal technique may be measured by the amount of time for estimation of one coefficient vector $\mathbf{a}$, which arises in each intermediate node. The analysis concerns the number of multiplications involved in the OLS (3) and RLS (12),(13) algorithms.

The running time of the computations according to formula (3) can be determined as follows: $\mathbf{H}^T\mathbf{H}$ requires $N \cdot m^2$ multiplications, the inversion $(\mathbf{H}^T\mathbf{H})^{-1}$ has a complexity m^3, the operation $\mathbf{H}^T\mathbf{y}$ takes $N \cdot m$ operations, and finally there have to be performed m^2 steps for the multiplication of the inverted matrix $(\mathbf{H}^T\mathbf{H})^{-1}$ with the vector $\mathbf{H}^T\mathbf{y}$. Hence, the time complexity of the OLS algorithm defined using equation (3) is proportional to: $\mathcal{O}_{OLS}(N, m) = N \cdot m^2 + m^3 + N \cdot m + m^2$, where N is the number of provided examples and m is the number of coefficients. Note, that this is the most efficient sequence for finding the matrix chain product, which may be checked using dynamic programming.

The running time of the computations according to formula (12) is: $N \cdot m$ steps for producing $\mathbf{c}_{k+1}$ after (7), $(N + m^3)$ multiplications for computing the scalar g_{k+1} with formula (10), the product $\mathbf{c}_{k+1}^T \mathbf{a}_k$ requires m operations, N to determine v_{k+1} using (9), and only m multiplications for evaluating $a_{k+1}\mathbf{C}_k^{-1}\mathbf{c}_{k+1}$ in (12) since the quantity $\mathbf{C}_k^{-1}\mathbf{c}_{k+1}$ is known from (10). Finally for making $\mathbf{C}_{k+1}^{-1}$ from $\mathbf{C}_k^{-1}$ by (13) one also uses $A + A^2$ multiplications. Therefore, the time complexity of RLS is: $\mathcal{O}_{RLS}(N, m) = 2 \cdot N + 2 \cdot m + m + m^2 + m^3 + N \cdot m$.

The studies here concern the use of OLS and RLS for finding the coefficients of simple transfer polynomials with $m \leq 6$ in GMDH derivative algorithms. Since in real tasks the number of the examples N is much greater than the size of coefficients in the simple transfer polynomials, $N \gg m$, the expensive multiplications needed for RLS fitting are much smaller than these necessary for OLS fitting: $\mathcal{O}_{OLS}(N, m) = [m^2(N + 1)] + m^3 + N \cdot m > \mathcal{O}_{RLS}(N, m) = [2(N + m) + m + m^2] + m^3 + N \cdot m$, for $m \geq 2$ and $N \geq 2$.

3.4 Improvement in Speed of Computation

An accelerated version of STROGANOFF [12, 13] was developed with the novel horizontal technique for building tree-like polynomial representations, called concisely fast *f*GP.

The advantage of the novel technique was investigated with the tree-structured polynomial given in Figure 1. Two polynomials were built from this tree: the first with coefficients derived with the horizontal technique used in *f*GP, and, the second with coefficients derived with the vertical technique used in STROGANOFF. These two polynomials were evaluated on subseries with an increasing number of example points. The subseries were generated by a random function defined by equation (19) below with variable values between -1 and 1, including 1000 points. The speed of computation curves are plotted in Figure 2.

Obviously there is an improvement in speed of computation of the polynomial coefficients due to the employment of the horizontal technique which is theoretically justified. We observe in Figure 2 that processing of a single polynomial acquired from even such a small size tree by the novel technique is more rapid than the vertical one, when estimated with 250 or more examples. This seriously influences the overall GP processing time. The speed of a particular GP implementation may be calculated by the difference in milliseconds times the size of the population (plus the time for performing the additional operations like selection, crossover, mutation etc.).

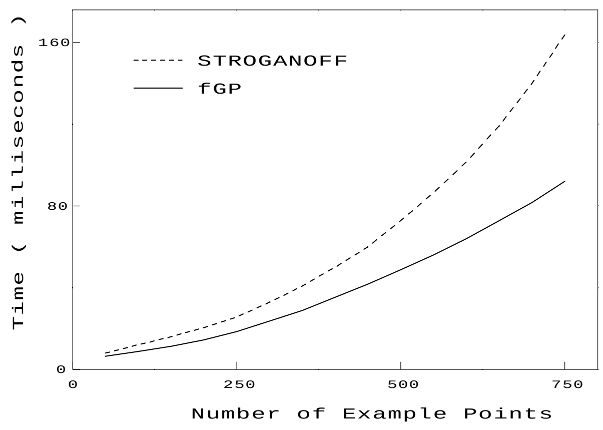

Figure 2. Speed of computing the polynomial coefficients with the horizontal technique incorporated in *f*GP related to that with the vertical technique in STROGANOFF, using the same tree-structured polynomial given in Figure 1 on a PC with Pentium processor running at 233MHz.

4 Mechanisms of the *f*GP System

The *f*GP system was implemented with *a set of transfer polynomials* (Table 1), and micromechanisms: *statistical fitness function*, *fitness proportional selection*, *steady-state reproduction*, *crossover*, and *one-point mutation*.

4.1 The Statistical Fitness Function

A statistical fitness function based on the *generalized cross-validation* (*GCV*) principle [29] is elaborated to stimulate breeding of accurate, parsimonious, and predictive polynomials. Its features are: 1) a mean-squared-error measurement that prefers fit polynomials; 2) a regularization factor that tolerates smoother polynomials with higher generalization potential; and, 3) a complexity penalty favoring short polynomials.

4.1.1 Fitting and Regularization

There are two problems to combat when using least squares methods for polynomial approximation: 1) they are numerically unstable due to ill-effects of multicollinearity, that is due to ill-conditioned covariance matrices; and 2) they often lead to overfitting polynomials with poor generalization capacities.

These problems may be avoided with *regularization* (*biased estimation*) techniques [22]. We apply such a technique to correct the mean squared error with a factor proportional to the magnitude of the coefficients, and define a *residual average error* (RAE) as follows:

$$RAE = \frac{1}{N}(\sum_{i=1}^{N}(y_i - p(\mathbf{x}_i))^2 + \lambda \mathbf{a}^T \mathbf{a}) \tag{14}$$

where y_i is the outcome given in the i-th example, $p(\mathbf{x}_i)$ is the polynomial outcome estimated with the same example $\mathbf{x}_i$, λ is a regularization parameter, and $\mathbf{a}$ is the vector with all M coefficients in the polynomial.

The correction in the RAE requires reformulation of the OLS, which can be derived by solving the minimization problem: $min_i \left(\sum_{i=1}^{N}(y_i - p(\mathbf{x}_i))^2 + \lambda \sum_{m=1}^{M} a_m^2\right)$. A solution to this minimization problem is the *regularized normal equation* [10]:

$$\mathbf{a} = (\mathbf{H}^T\mathbf{H} + \lambda \mathbf{I})^{-1}\mathbf{H}^T\mathbf{y} \tag{15}$$

where $\mathbf{I}$ is $(m+1) \times (m+1)$ identity matrix, and $\lambda > 0$ is the regularization parameter. This regularized normal equation (15) is adopted to perform OLS fitting in the fringe nodes instead of (3).

The RLS method is sensitive to multicollinearities in the data like OLS, and should also be regularized. When using the covariance $\mathbf{C}_k^{-1}$ in (6) one notes that the new $\mathbf{C}_{k+1}^{-1}$ will become regularized if the lower right corner element j_{k+1} is augmented. Therefore, the regularization of $\mathbf{C}_{k+1}^{-1}$ involves updating only its scalar component j_{k+1}. This can be done by adding the biasing constant λ to it: $j_{k+1} = j_{k+1} + \lambda$, after computing j_{k+1} with formula (8).

Tuning the parameter λ allows us to favour polynomials that compromise the goodness of fit with the smoothness of their curvature. Increasing λ relaxes the requirement for close fitting of the data, and so produces more "regular" generalizing approximations. Among various alternatives, the ridge trace statistical technique is adopted to find values for the regularization hyper-parameter λ [22].

The regularization techniques help to combat effects of even small deviations in the data, which may cause large deviations in the inferred solution. This is why the presented regularizations entail several advantages: 1) they increase the stability when estimating the coefficients, and diminish the sensitiveness to discrepancies in the given data; 2) they reduce the variance of polynomials, and help to improve their prediction potential; and, 3) they require only minimal processing time.

4.1.2 Complexity Penalty

Favouring of parsimonious solutions is made by synthesis of a *GCV statistical fitness function* for evaluating the high-order multivariate polynomials [29]:

$$GCV = \frac{RAE}{(1 - M/N)^2} \tag{16}$$

where RAE is the regularized error (14), M is the number of coefficients, and N is the number of examples.

The three ingredients of this statistical fitness function GCV (16) are incorporated to enforce discovery of *optimal,* in the sense of highly fit, low size solutions, that avoid overfitting with the examples. In this GCV fitness function the mean square error $\sum_{i=1}^{N}(y_i - p(\mathbf{x}_i))^2$ in the numerator measures the distance to the data, the denominator $(1 - M/N)^2$ accounts for the complexity, and $\lambda \mathbf{a}^T \mathbf{a}$ quantifies the smoothness of the hypersurface defined by the polynomial. Minimization of the fitness is the aim.

4.2 Crossover and Mutation

The *crossover* operator chooses a cut point node in each tree, and swaps the subtrees rooted in the cut-point nodes. This crossover is restricted by a predefined maximum tree size so that if an offspring tree of larger size results it is discarded and one of its parents remains taken at random.

The *mutation* operator selects a tree node, and performs one of the following elementary tree transformations: 1) insertion of a randomly chosen node before the selected one, so that the selected node becomes an immediate child of the new one, and the other child is a random terminal (leaf); 2) deletion of the selected node, and replacing it by one of its children nodes (a terminal leaf or a subtree rooted at a functional node); and 3) replacement of the selected node by another randomly chosen node. These tree transformations are implemented for binary trees only since they are taken to represent compositions of bivariate transfer polynomials.

4.3 Recombinative Guidance

Two versions of each crossover and mutation operators are implemented: with *random selection* of the cut or mutation point, and with *recombinative guidance* [13]. The recombinative guidance suggests the selection as a cut/mutation point the tree node which shows largest residual error in order to pursue better local search. We stress here the usefulness of such guidance by showing that the mean squared error component $(\mathbf{y}-\mathbf{Ha})^T(\mathbf{y}-\mathbf{Ha})$ of the residual error RAE (14) can be computed efficiently as follows:

$$MSE = \frac{1}{N}(\mathbf{y} - \mathbf{Ha})^T(\mathbf{y} - \mathbf{Ha}) = \frac{1}{N}(\mathbf{y}^T\mathbf{y} - \mathbf{a}^T\mathbf{u}) = \frac{1}{N}\mathbf{e} \tag{17}$$

where $\mathbf{y}$ is the given outcome vector, $\mathbf{Ha}$ is the estimated output using the design matrix $\mathbf{H}$, and the $M \times 1$ coefficient vector $\mathbf{a}$, and N is the number of the data (see Appendix II). For simplicity we substitute $\mathbf{y}^T\mathbf{y} - \mathbf{a}^T\mathbf{u}$ by $\mathbf{e}$. Formula (17) involves only M multiplications because the quantity $\mathbf{y}^T\mathbf{y}$ is constant for the task, and $\mathbf{u} \equiv \mathbf{H}^T\mathbf{y}$ is known from the process of estimating the coefficients.

When regularization is applied, this error (17) is corrected by the amount of regularization $\lambda\mathbf{a}^T\mathbf{a}$.

After adding the next $(k+1)$-st independent variable to the model, we may also obtain the mean squared error iteratively. Let us assume that MSE (17) is the error of a model with k independent variables, that is:

$MSE = MSE_k = \mathbf{e}_k/N$. Then, the error of the extended model can be estimated with the fast formula:

$$MSE_{k+1} = \frac{1}{N}(\mathbf{e}_k - a_{k+1}^2/g_{k+1}) \tag{18}$$

where a_{k+1} is the new coefficient, and g_{k+1} is the scalar computed according to (10). Note here in (18) that the computation of the mean squared error MSE_{k+1} takes only one multiplication, and two divisions, since the scalar g_{k+1} is known, and $\mathbf{e}_k$ is passed from the previous model.

5 Experimental Results

Experiments were conducted to find out whether the accelerated system *f*GP implemented with the horizontal technique can achieve better results than STROGANOFF with the vertical one [12], and also whether it is more successful than traditional Koza-style GP [16]. STROGANOFF used 10 transfer polynomials of up to second order [23]. The traditional GP was SGPC-1.1 [28], which is made to induce expressions from primitive functions: $+$, $-$, $*$, %, *sin*, *cos*, ln, exp, and ephemeral constant creation [16] (page 242). It should be clarified that although these GP systems produce different kinds of functional models: *f*GP and STOROGANOFF evolve functions composed of polynomials while traditional Koza-style SGPC evolves functions assembled from the above primitive functions, we attempt to find out which of them is better on concrete benchmark tasks.

The parameters in all runs with the three GP systems were: *PopulationzeSize* $= 100$, *Generations* $= 300$, $MaxTreeSize = 35$, and *fitness proportional selection*. The experiments conducted with STOROGANOFF and *f*GP use *SteadyStateReproduction* $= 50\%$, that is the elite half individuals from the population chosen by the fitness proportional selection scheme were modified by either crossover or mutation and, after that, allocated over the worst 50% individuals in the population. The initial population is generated randomly.

Benchmark instances of the following three practical problems were attacked: symbolic regression, pattern recognition, and financial time series prediction.

5.1 Symbolic Regression

The problem of *symbolic regression* [16] is to identify a function mapping from examples with numeric values of both the independent variables and the dependent variable. A non-linear multivariate function was taken and 250 examples (150 training, 100 testing) were generated with it:

$$y(x) = 3(x_1 - 0.5) + sin(x_2 * x_3) + 2\ln(1 + x_4)/x_5 \tag{19}$$

where the values for x_i were drawn randomly from the interval: $-1 \leq x_i \leq 1$, $1 \leq i \leq 5$.

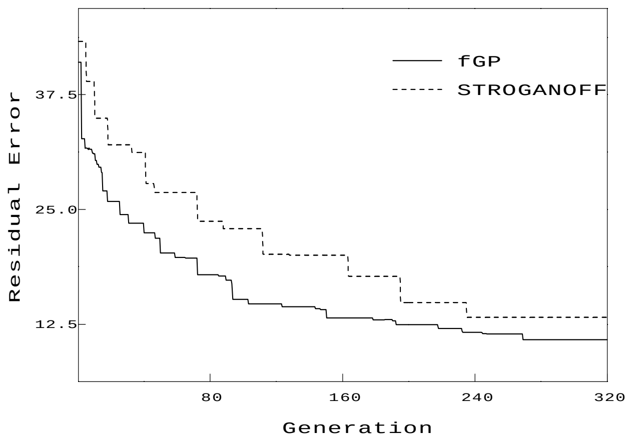

Figure 3. Learning accuracies measured by the mean squared error MSE (17), taken during typical runs with STROGANOFF and *f*GP while trained with 150 examples from the symbolic regression problem (19).

The accuracy changes in the sense of RAE (14) of the best polynomial during typical runs are plotted in Figure 3. One observes that the residual error of the best solution evolved by STROGANOFF decreases sharply, while the accelerated *f*GP system characterizes by a slower and more continuous attenuation of the best error. This is an indication that there are smaller fitness differences between the polynomials bred by *f*GP, which implies an ability to climb the fitness landscape more easily and capacity to conduct a more successful exploration of the search space.

The search abilities of the two GP systems are also investigated using the correlation characteristics of their fitness landscapes (Figure 4). The autocorrelation functions of the mutation landscapes, arising in *f*GP and STROGANOFF, were calculated during random walks with one tree-structured polynomial [20]. Starting from an arbitrary tree, random walks of length 1000 steps were made by successively modifying the tree-like polynomial with the one-point mutation operator (given in subsection 4.2), and, next, the polynomial fitnesses were recorded. The curves plotted in Figure 4 are averaged autocorrelations[3] from 10 random walks.

The fitness evaluations for deriving the plots in Figure 4 were made with the same GCV formula (15). Only the technique for building a polynomial from the trees is different: the horizontal one is used in *f*GP, while STROGANOFF uses the vertical one. The higher autocorrelation curve of *f*GP in Figure 4 shows that there is a higher statistical correlation between the fitnesses of its mutated tree-like polynomials. Therefore, the trees sampled with the one-point mutation operator lead to horizontally constructed polynomials with closer fitnesses than the vertically built polynomials sampled by the same operator. The higher fitness correlation, that

[3]The application of the autocorrelation fomula here is made upon the assumption that the analysis is restricted only to the walked fitness landscape areas, and these walked landscapes have locally isotropic structure (Hordijk, 1996).

is closer fitnesses, indicates that *f*GP flows on a smoother landscape. A smoother fitness landscape facilitates the orientation of the GP toward the peaks, and in this since a smoother landscape is considered easier to search in most cases than a rugged one. This means that *f*GP has the ability to perform better exploitation of the search space than STROGANOFF, and should be expected to search more continuously.

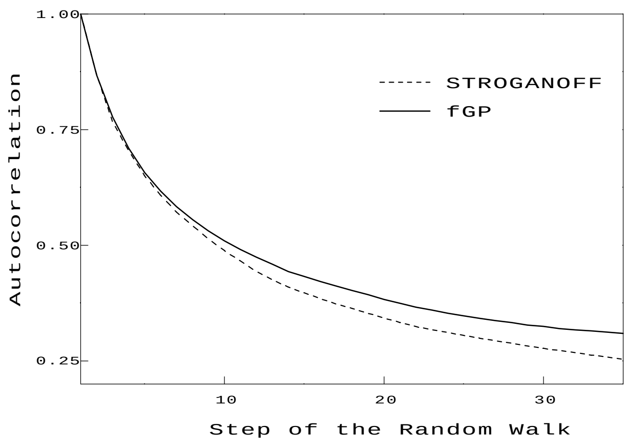

Figure 4. Landscape autocorrelations averaged from 10 runs with 150 examples from the symbolic regression problem (19), using the same fitness function but different polynomial construction techniques: the horizontal in *f*GP and the vertical in STROGANOFF.

The learning accuracy was measured with the *mean squared error* (*MSE*) (17). The smaller differences in fitness can be explained with the differences in the order of the constructed polynomials. The vertical technique in STROGANOFF may produce polynomials of higher order than *f*GP from the same tree-structure. This is due to the possibility to feed a node with two polynomial outcomes from children subtrees, leading to an increase in the order of all cross-product terms in which they participate. In contrast to this, the horizontal technique expands one partial polynomial, whose order remains the same, and only the new term may become of increased order. The new term adds less information to the partial polynomial model in *f*GP. This leads to the formation of offspring polynomials with close fitnesses to these of their parents when applying the genetic mutation operator, since the considered mutation only slightly changes the tree topology.

The results in Table 2 may be explained with this relation between the fitness and polynomial order differences. One notes that the accelerated *f*GP shows the best accuracy, and the two STROGANOFF versions outperform the traditional Koza-style GP. Having established the function points generated by equation (19) lead to a highly oscillating curve, we are inclined to think that STROGANOFF attempts to fit very closely the curve by means of very high-order polynomials. It faces, however, difficulties in composing a good solution because of its sensitivity to the particular transfer polynomials from the predefined set, and it is not so efficient as *f*GP on this task. The best *f*GP polynomial had 25 coefficients compared to 22 using STROGANOFF.

	Complexity (size)	*Accuracy* (training)	*Prediction* (testing)
system	coefficients	MSE	MSE
*f*GP	25	0.292	0.325
STROGANOFF	22	0.338	0.404
Koza-style GP	23	0.391	0.448

Table 2. Complexity, accuracy, and predictability of the best polynomials evolved in 100 runs by three GP systems on the symbolic regression task (19), with $k = 0.01$.

	Complexity (size)	*Accuracy* (training)	*Prediction* (testing)
system	coefficients	$error$	$error$
*f*GP	33	1.5%	5%
STROGANOFF	39	1.5%	7.5%
Koza-style GP	–	2.75%	12.5%

Table 3. Error percentages of the best polynomials evolved to classify the ellipse from the *Shape recognition* task, after performing 100 runs by three studied GP systems (using $k = 0.001$).

5.2 Pattern Recognition

Polynomials can be used for addressing pattern recognition tasks, many of which involve function identification. We illustrate the applicability of GP to the task of recognizing geometric shapes by identification of polynomial decision boundaries among them, that is to find boundaries between the shapes that allow correct classification of patterns into the particular shape classes. Four *Shapes* have been synthesized: ellipse, triangle, quadrilateral, and pentagon [30]. The example patterns of these shapes represent pixel values in the corresponding shape image, generated using different degree of deformation. There are 16 real-valued attributes in each example. The set includes 800 examples, divided as 400 training and 400 testing.

Experiments were made to find a polynomial that distinguishes the ellipse from the remaining shapes. After presenting an example $\mathbf{x}$, the polynomial outcome is passed through the step function: if it is $p(\mathbf{x}) > 0.5$ and the example is positive it is assumed that the example is classified correctly, if it is $p(\mathbf{x}) < 0.5$ and the example is negative it is also classified correctly, otherwise it is misclassified.

The misclassification errors made by the best ellipse shape discrimination functions evolved by the three GP systems are given in Table 3. The results in Table 3 reveal that the best *f*GP polynomial shows the same accuracy on the training data as that from STROGANOFF. The *f*GP system has found a polynomial with a higher generalization capacity when tested on the unseen examples, while STROGANOFF has discovered a polynomial that seems to overfit the data. It should be noted that a sigma-pi neural network showed 0% accuracy on the training data but it was especially designed for addressing classification tasks while our polynomials are general models.

The best polynomial evolved by accelerated *f*GP included 33 coefficients. The best polynomial evolved by STROGANOFF included 39 coefficients. The system STROGANOFF exhibits a tendency to overfit the data with overly complex polynomials, that is having too many terms. The traditional Koza-style GP is inferior to STROGANOFF and *f*GP systems on this pattern recognition task.

5.3 Financial Data Prediction

Studies into the ability of the three GP systems to learn the periodicities from real-world financial time series data are performed. The task of *stock market forecasting* is considered: given a sequence of market prices $...x_{t-1}, x_t, x_{t+1}...$, the goal is to identify nonlinear market price models f that explain them sufficiently well and may generate price forecasts: $x_{t+1} = f(\mathbf{x}) = f(x_{t-(m-1)\tau}, x_{t-(m-2)\tau}, ..., x_t)$, where: $\mathbf{x}$ are *delayed* (lagged) vectors extracted from the series, m is *embedding dimension*, and τ is *delay time*. Our concern is embedding by delay vectors with parameters $m = 10$, and $\tau = 1$.

5.3.1 Data Pre-processing

Simple techniques for preprocessing the raw financial data are applied to examine how they impact the GP learning ability and with which preprocessed series GP yields optimal results. Preprocessing is made in two steps: *data transformation* in order to isolate as much as possible significant information from the usually very noisy financial data; and *embedding* to determine how to express this information through independent variables.

Three market price series were produced from the original raw financial data as follows:

- *original series*– the given series is taken directly forming vectors $\mathbf{x} = (x_{t-m-1}, x_{t-m-2}, ..., x_t)$;

- *integral series*– each value from the original series is replaced by its moving average: $x_a = (1/l) \sum_{k=t-(l-1)/2}^{t+(l-1)/2} x_k$, computed with a smoothing period $l = 5$;

- *differential series*– each value is substituted by the difference from its origin x_t and the average of its l neighbors: $x_d = x_t - (1/l) \sum_{k=t-l-1}^{t} x_k$. A predefined interval $l = 3$ is used [4].

5.3.2 Learning of Financial Models

The data at our disposal are a *Nikkei225* training series of $3,000$ points that covers dates during the period April 1st, 1993 through April 17th, 1993. The testing series is of $30,177$ data points from the period April 18th, 1993 through September 30th, 1993. These are price averages over 225 representative stocks from the Tokyo Stock Exchange, sampled at every minute from 9:00am to 12:00 pm, and from 1:00pm to 3:00 pm.

The residual error RAE (14) in the GCV fitness function (16) is replaced by a *rational error* (RAT):

$$RAT = \frac{\sum_{i=1}^{N}(y_i - p(\mathbf{x}_i))^2}{\sum_{i=1}^{N}(y_i - y_{i-1})^2} \tag{20}$$

where y_i and y_{i-1} are two given consecutive outcomes, and $p(\mathbf{x}_i)$ is the outcome from the model produced with the vector $\mathbf{x}_i$ from the i-th example. It serves for selecting models of nonstationary time series where the mean $\bar{y}$ can not be determined in a trivial way, and it is replaced in the denominator by $y_{i-1} \approx \bar{y}$.

The employment of the rational error RAT serves to account for the expected generalization quality of the polynomials, not simply for their residual error, since it reveals objective information contained in them when apriori knowledge of the time series is lacking. More precisely, the RAT shows the residual error improvement over a random model identified from such series: when $RAT < 1$ the prediction capacity of the model is better than that of a randomly generated model, and when $RAT > 1$ the prediction is worse. This is beneficial to know for adequate tuning the control mechanism of GP so as to achieve progressive search.

The performance measures for evaluating the models are: MSE (17), hit percentage HIT, and profit gain $Profit$. The *hit percentage* (HIT) shows how the model tracks the trend directions D [14]:

$$HIT = \frac{\sum_{i=1}^{E} D_i}{N}, \text{ where } D_i = \begin{cases} 1 & x_t\hat{x}_t \geq 0 \\ 0 & otherwise \end{cases} \tag{21}$$

where the positive product $x_t\hat{x}_t \geq 0$ means the number of times when the given outcome x_t and the model outcome $\hat{x}_t$ (i.e. $\hat{x}_t = p(x_{t-10}, .., x_{t-1})$) exhibit the same tendency: both are upward raising or both are falling.

The expected $Profit$ is evaluated with a simple algorithm that generates buy/sell trade signals, with which the possible dividends from the model may be calculated. We use a previously defined profit algorithm by Iba and Sasaki (1999), starting from an initial amount of $1,000,000$ Japanese yen committed to the GP system.

The regularization parameter k was $k = 0.005$ in all experiments. The training results in the sense of MSE (17) are estimated over the normalized training series (since training is made with the correspondingly preprocessed after normalization data series), while all testing results: HITs and $Profits$, are estimated using the original financial series (non-normalized, preprocessed).

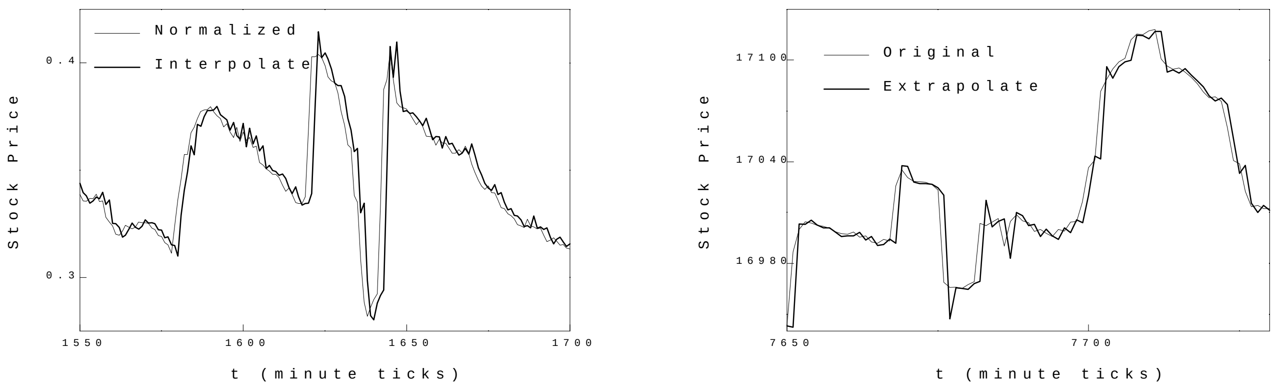

Figure 5. a) Interpolation capacity b) Extrapolation by the best (*original*) polynomial from STROGANOFF

	Complexity (size)	*Accuracy* (training)	*Prediction* (testing)	
system	coefficients	MSE	HIT	$Profit$
*f*GP	69	$3.21e-04$	57.64%	$23,097$
STROGANOFF	39	$1.39e-04$	54.02%	$14,714$
SGPC	–	$1.92e-06$	55.23%	$11,230$

Table 4. Estimates of the best polynomials learned from the *original financial series*

Training with the Original Series. The three GP systems tend to evolve overfitting solutions over the original financial series, and the predictions by these solutions were not good. The best polynomial from accelerated *f*GP outperforms the results from STROGANOFF and SGPC from an economic perspective (Table 4): it achieves highest $HIT = 57.64\%$, and highest $Profit = 23,097$ over the unseen, testing series. Related to the results on the integral (Table 5) and differential series (Table 6), however, the results in Table 4 are not the best on financial forecasting compared to these achieved over the transformed series. Most overfitting function models have been discovered by SGPC. The MSE errors of *f*GP: $3.21e-04$, and STROGANOFF: $1.39e-04$ are close in magnitude, but this is not a surprise since they both use the same fitness function which is the driving force of evolutionary search. Aiming at overfitting avoidance, these observations lead to the suggestion that preprocessing of the financial data series as a condition may help to pursue more successful learning of predictive trading models.

The number of coefficients in the best solutions are given in Table 4. One notes that the best *f*GP polynomial achieves best forecasting at the expense of having more coefficients than STROGANOFF. The interpolation and extrapolation capacity of the best *f*GP polynomial in arbitrary intervals are plotted in Figures $5a$ and $5b$. Interpolation capacity here is the degree of fitting the training data by the polynomial. Extrapolation capacity is the degree of fitting the unseen testing data by the same polynomial.

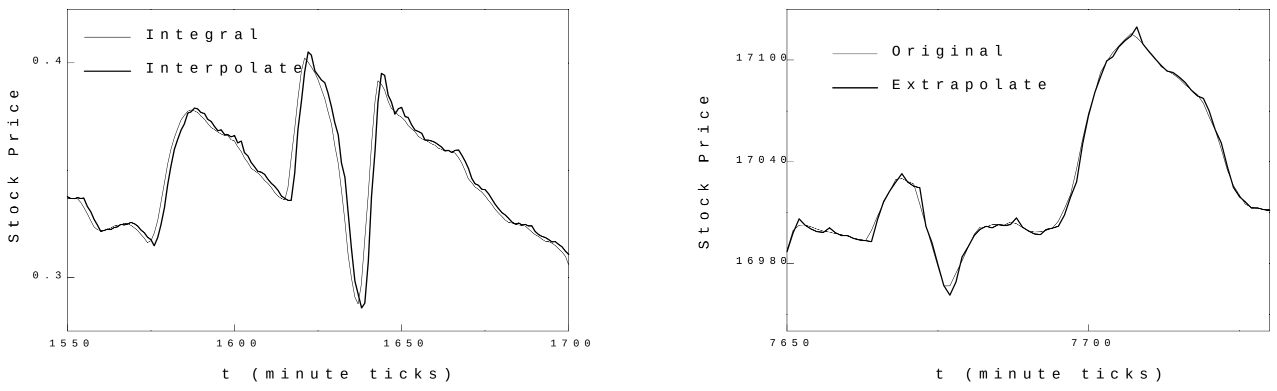

Figure 6. a) Interpolation capacity b) Extrapolation by the best (*integral*) polynomial from STROGANOFF

	Complexity (size)	*Accuracy* (training)	*Prediction* (testing)	
system	coefficients	MSE	HIT	$Profit$
*f*GP	68	$7.01e-04$	55.71%	$19,251$
STROGANOFF	62	$4.09e-05$	53.77%	$11,293$
SGPC	–	$1.90e-06$	62.18%	$28,987$

Table 5. Estimates of the best polynomials learned from the *integral financial series*

Training with the Integral Series. The results derived after moving average filtering of the original financial series were the best that were achieved by SGPC compared to all other attained by this system. Table 5 shows the accuracy of the best SGPC expression and relates it to the best polynomials found by accelerated *f*GP and STROGANOFF over the integral series. One notes that the best SGPC expression has better MSE, HITs, and $Profit$ qualities than all of its best expressions over the remaining series transformations. The observation that SGPC evolves its best solution in mean squared error sense could be attributed to the fact that the series curvature is slightly smoother than that of the original series. The interpolation and the extrapolation potential of the best *f*GP polynomial are given in Figures $6.a$ and $6.b$.

The ability to evolve a very accurate solution does not guarantee that this solution will be the best from an economic perspective. The best SGPC expression has $Profit$ higher than all other results achieved by SGPC, but not compared to the results from the *f*GP and STROGANOFF systems (see Table 6). The best *f*GP result outperforms again that from STROGANOFF. We may reason that the remaining effect of the magnitudes creates difficulties in evolving polynomial trade models that yield optimal hits on unseen future data.

Obviously there is no clear correlation between the MSE and the HITs and $Profit$, and this could be because moving average filtering cannot protect the inherent signal dynamics very well.

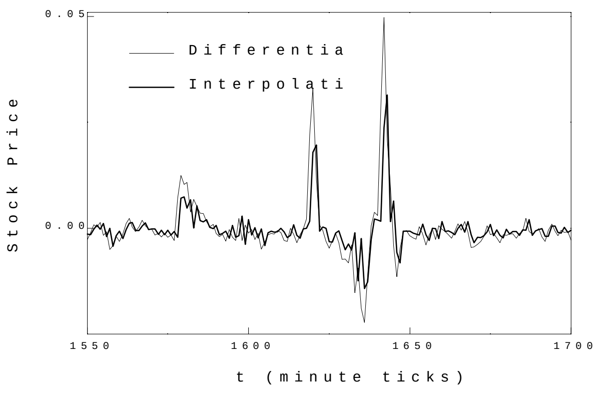

Figure 7. a) Interpolation capacity b) Extrapolation by the best (*differential*) polynomial from STROGANOFF

	Complexity (size)	*Accuracy* (training)	*Prediction* (testing)	
system	coefficients	MSE	HIT	$Profit$
*f*GP	72	$4.85e-05$	63.21%	29.862
STROGANOFF	41	$7.86e-05$	74.66%	$54,901$
SGPC	–	$1.94e-06$	49.94%	$2,078$

Table 6. Estimates of the best polynomials learned from the *differential series*

Training with the Differential Series. The highest number of HITs 74.66% achieved in all runs by the two GP systems exhibits the best polynomial from STROGANOFF using the differential series. The HITs and $Profit$s of the best evolved polynomial by STROGANOFF are related to the same qualities measured with the best expressions discovered by accelerated *f*GP and SGPC in Table 6. The *f*GP system evolved the second best solution among all those discovered with 63.21% HITs and $Profit$s= 29.862.

The usefulness of the studied differential transformation formula has been suggested by specialists in economics and it is not a surprise that good economic results are achieved with it. The way in which we define it by averaging over the three recent price data is suitable for representation of short and medium trends. If the goal is to perform long term prediction it should be adjusted to use for example 10 or more recent price data, depending on the desired prediction interval [4]. What such differential transformation provides is the possibility to filter out noise and avoid close fitting of the price magnitudes which are not essential to forecasting.

The presented results in Table 6 confirm that GP systems are also sensitive to such series transformations, and STROGANOFF-like systems that evolve polynomial models should be preferred over traditional GP systems evolving functions recursively constructed from arbitrary primitive basis functions.

An interesting observation in Table 6 is that the learning accuracy on the training data $7.86e-05$ is not the best in mean squared error sense, the best accuracy is measured with the *f*GP polynomial having

$MSE = 4.85e-05$. A highest degree of fitting the series does not imply highest level of generalization. SGPC 1.1 evolved his worst expression using the differential series. This expression failed to provide good predictions, and does not seem to be economically beneficial.

6 Discussion

The contribution of this paper is the horizontal technique for building polynomials from tree-like representations. Incorporated in GP, it was found to be successful for addressing function approximation tasks. The polynomials evolved by accelerated *f*GP seem to learn more adequately relatively smooth data curves and highly oscillating data curves with close magnitudes. This was confirmed by the good generalization performance on the symbolic regression problem (Table 2), on the pattern recognition problem (Table 3) and on the highly oscillating financial stock market data series (Tables 4 and 5). Note that the generalization performance on the financial data is evaluated by the HITs and $Profits$ on the future testing series. The system STROGANOFF equipped with the vertical technique learns very well from data whose curves exhibit oscillations with low frequencies and high sparks at irregular intervals like the differentially preprocessed financial data (Tables 6).

A distinguishing advantage of accelerated *f*GP, like the other systems from the STROGANOFF family, is that it handles the numeric function parameters and constants better than traditional GP. This is due to the representation of function models as hierarchical compositions of transfer polynomials allocated in the tree nodes, adopted from GMDH, and the direct estimation of their coefficients by recurrent least squares. Thus, the parameters and constants are uniquely determined, while traditional GP performs explicit search for the numeric function parameters. The use of least squares methods makes the system prone to numerical inaccuracies, and this is why we always employ regularization [23]. Other GP systems relying on least squares should also consider the presented regularization in order to stabilize the coefficients estimation [10]. Overall, the regularization improves the generalization error with respect to the unbiased model, if proper statistical methods for the selection of the regularization hyper-parameter are taken.

GP systems evolving other tree-structured function series representations can benefit from the efficient formula for estimating the mean squared error MSE (17). This formula (17) is necessary for implementing recombinative guidance. This guidance was found essential for successful navigation of the evolutionary search process in the spaces of function models [13]. Since it is used in every intermediate functional tree node, its employment makes rapid the error estimations of the partial models.

The accelerated *f*GP system was implemented with a GCV fitness function that aims at discovery of statistically optimal solutions. The GCV function [29] belongs to the class of algebraic measures of the expected prediction error, like the Final Prediction Error (FPE) and the Prediction Squared Error (PSE) with which similar results should be expected. Future research is planned to try information-based fitness evaluations, involving the logarithm of the maximal likelihood. We will study how fitness functions derived with the Schwarz's Bayesian Information Criterion (BIC), with the Akaike's Information Criterion (AIC), impact on the GP performance. Several information measures developed according to the Minimum Description Length Principle (MDL) have already proved appropriate for GP of polynomial models on classification and regression tasks [12, 23].

7 Conclusion

This paper extended the representation power and practical usefulness of the GP systems from the STROGANOFF family. This was achieved by the development of accelerated *f*GP with a special horizontal technique for polynomial construction from their tree-like representations. It was demonstrated that the horizontal technique offers two main advantages over previous GP systems: 1) improvement in speed, which was theoretically proven, and was practically measured using benchmark data; 2) ability to evolve accurate and predictive polynomial models.

The designed accelerated *f*GP, as the other members from the STROGANOFF family, generates analytical function models. Such analytical solutions are easy to comprehend by human experts, which is another reason that *f*GP is more attractive for real applications.

References

[1] W. Banzhaf, P. Nordin, R.E. Keller and F.D. Francone, Genetic Programming: An Introduction. On the Automatic Evolution of Computer Programs and Its Applications, Morgan Kaufmann: San Francisco, CA, 1998.

[2] I.S. Berezin and N.P. Zhidkov, Computing Methods, Addison-Wesley: Reading, MA, 1965.

[3] S.-H. Chen and C.-H. Yeh, "Option Pricing with Genetic Programming," in Genetic Programming 1998: Proceedings of the Second Annual Conference, Madison, Wisconsin, J.R.Koza, W.Banzhaf, K.Chellapilla, K.Deb, M.Dorigo, D.B.Fogel, M.Garzon, D.Goldberg, H.Iba and R.L.Riolo (Eds.), Morgan Kaufmann: San Francisco, CA, 1998, pp. 32-37.

[4] G.J. Deboeck and M. Cader, "Pre and Postprocessing of Financial Data," in Trading on the Edge. Neural, Genetic and Fuzzy Systems for Chaotic Financial Markets, G.J.Deboeck (Ed.), John Wiley & Sons: New York, NY, 1994, Chapter 2, pp. 27-44.

[5] D.K. Fadeev and V.N. Fadeeva, Computational Methods of Linear Algebra, W.H.Freeman: San Francisco, CA, 1963.

[6] A.A. Freitas, "Genetic Programming Framework for two Data Mining Tasks: Classification and Generalized Rule Regression," in Genetic Programming 1997: Proceedings of the Second Annual Conference, Stanford University, CA, J.R.Koza, K.Deb, M.Dorigo, D.B.Fogel, M.Garzon, H.Iba, and R.L.Riolo (Eds.), Morgan Kaufmann: San Francisco CA, 1997, pp. 96-101.

[7] D. Gabor, W. Wildes and R. Woodcock, "A Universal Nonlinear Filter, Predictor and Simulator which Optimizes Itself by a Learning Process," Proceedings of the IEE, 108B, pp. 422-438, 1961.

[8] Ch. Hafner and J. Fröhlich, "Generalized Function Analysis Using Hybrid Evolutionary Algorithms," in Proceedings 1999 Congress on Evolutionary Computation, IEEE Press: Pscataway, NJ, 1999, pp. 287-294.

[9] H. Hiden, B. McKay, M. Willis and G. Montague, " Non-Linear Partial Least Squares using Genetic Programming," in Genetic Programming 1998: Proceedings of the Second Annual Conference, Madison, Wisconsin, J.R.Koza, W.Banzhaf, K.Chellapilla, K.Deb, M.Dorigo, D.B.Fogel, M.Garzon, D.Goldberg, H.Iba and R.L.Riolo (Eds.), Morgan Kaufmann: San Francisco, CA, 1998, pp. 128-133.

[10] A.E. Hoerl and R.W. Kennard, "Ridge Regression: Biased Estimation of Nonorthogonal Problems," Technometrics, vol. 12, pp. 55-67, 1970.

[11] W. Hordijk, "A Measure of Landscapes," Evolutionary Computation, vol. 4 (4), pp. 335-360, 1996.

[12] H. Iba, H. de Garis and T. Sato, "Genetic Programming using a Minimum Description Length Principle," in: Advances in Genetic Programming, K.Kinnear Jr.(Ed.), The MIT Press: Cambridge, MA, 1994, pp. 265-284.

[13] H. Iba and H. de Garis, "Extending Genetic Programming with Recombinative Guidance," in Advances in Genetic Programming 2, P.J.Angeline and K.Kinnear (Eds.), The MIT Press: Cambridge, MA, 1996, pp. 69-88.

[14] H. Iba and T. Sasaki, "Using Genetic Programming to Predict Financial Data," in Proceedings 1999 Congress on Evolutionary Computation, IEEE Press: Pscataway, NJ, 1999, pp.244-251.

[15] A.G. Ivakhnenko, "Polynomial Theory of Complex Systems," IEEE Trans. on Systems, Man, and Cybernetics, vol. 1 (4), pp. 364-378, 1971.

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

[17] J.R. Koza, "Genetic Programming for Economic Modeling.," in Intelligent Systems for Finance and Business, S.Goonatilaje and P.Treleaven (Eds.), John Wiley Sons: London, UK, 1995, pp. 251-269.

[18] G.Y. Lee, "Genetic Recursive Regression for Modeling and Forecasting Real-World Chaotic Time Series" in Advances in Genetic Programming 3, L.Spector, W.B.Langdon, U.-M.O'Reilly and P.J.Angeline (Eds.), The MIT Press: Cambridge, MA, 1999, pp. 401-423.

[19] H.R. Madala and A.G. Ivakhnenko, Inductive Learning Algorithms for Complex Systems Modeling, CRC Press: Boca Raton, FL, 1994.

[20] B. Manderick, W. de Weger and P. Spiessens, "The Genetic Algorithm and the Structure of the Fitness Landscape," in Proceedings of the Fourth International Conference on Genetic Algorithms, R.K.Belew and L.B.Booker (Eds.), Morgan Kaufmann: San Francisco, CA, 1991, pp. 143-150.

[21] B.S. Mulloy, R.L. Riolo and R.S. Savit, "Dynamics of Genetic Programming and Chaotic Time Series Prediction," in Genetic Programming 1996: Proceedings of the First Annual Conference, Stanford University, CA, J.R.Koza, D.E.Goldberg, D.E.Fogel, and R.L.Riolo (Eds.), The MIT Press: Cambridge, MA, 1996, pp. 166-174.

[22] R.H. Myers, Classical and Modern Regression with Applications, PWS-KENT Publ., Duxbury Press: CA, 1990.

[23] N.I. Nikolaev and H. Iba, " Regularization Approach to Inductive Genetic Programmimg", IEEE Transactions on Evolutionary Computation, 2001.

[24] P. Nordin and W. Banzhaf, "Programmatic Compression of Images and Sound," in Genetic Programming 1996: Proceedings of the First Annual Conference, Stanford University, CA, J.R.Koza, D.E.Goldberg, D.E.Fogel, and R.L.Riolo (Eds.), The MIT Press: Cambridge, MA, 1996, pp. 345-350.

[25] R. Poli, "Genetic Programming for Image Analysis," in Genetic Programming 1996: Proceedings of the First Annual Conference, Stanford University, CA, J.R.Koza, D.E.Goldberg, D.E.Fogel, and R.L.Riolo (Eds.), The MIT Press: Cambridge, MA, 1996, pp. 363-368.

[26] W.H. Press, B.P. Flannery, S.A. Teukolski and W.T. Vetterling, Numerical Recipes in C: The Art of Scientific Computing, 2nd ed., Cambridge University Press: Cambridge, England, 1992.

[27] K. Rodriguez-Vazquez, C.M. Fonseca and P.J. Fleming, "An Evolutionary Approach to Non-Linear Polynomial System Identification", in Proc. 11th IFAC Symposium on System Identification, 1997, pp. 2395-2400.

[28] W.A. Tackett and A. Carmi, "The Donut Problem: Scalability and Generalization in Genetic Programming," in Advances in Genetic Programming, K.E.Kinnear Jr. (Ed.), The MIT Press: Cambridge, MA, 1994, pp. 143-176.

[29] G. Wahba, Spline Models for Observational Data, CBMS-NSF Regional Conf. Series 59, SIAM Press: Philadelphia, Pennsylvania, 1990.

[30] H.C. Yau and M.T. Manry, "Iterative Improvement of a Nearest Neighbor Classifier," Neural Networks, vol. 4 (4), pp. 517-524, 1991.

[31] B.-T. Zhang, P. Ohm, and H. Mühlenbein, "Evolutionary Induction of Sparse Neural Trees", Evolutionary Computation, vol.5 (2), pp. 213-236, 1997.

Appendix I: Derivation of the RLS Formulae

Let us consider the normal equation (3) for finding the coefficients $\mathbf{a}_k$ in the case of having k independent variables is (5): $\mathbf{a}_k = \mathbf{C}_k^{-1}\mathbf{u}_k$. The formulae for updating the coefficients can be derived after solving the equation:

$$\mathbf{C}_{k+1}\mathbf{C}_{k+1}^{-1} = \mathbf{I} \tag{A.1}$$

where $\mathbf{I}$ is $(k+1) \times (k+1)$ identity matrix. For clarity we will use the block matrix:

$$\mathbf{C}_{k+1}^{-1} = \mathbf{B}_{k+1} = \begin{bmatrix} \mathbf{B}_k & \mathbf{b}_{k+1} \\ \mathbf{b}_{k+1}^T & g_{k+1} \end{bmatrix} \tag{A.2}$$

The matrix blocks: $\mathbf{B}_k$, $\mathbf{b}_{k+1}$, $\mathbf{b}_{k+1}^T$, and g_{k+1}, may be expressed with some of the available quantities. Combining (A.1) and (A.2): $\mathbf{C}_{k+1}\mathbf{C}_{k+1}^{-1} = \mathbf{C}_{k+1}\mathbf{B}_{k+1}$, and applying the rules for multiplication of block matrices we get:

$$\begin{bmatrix} \mathbf{C}_k\mathbf{B}_k + \mathbf{c}_{k+1}\mathbf{b}_{k+1}^T & \mathbf{C}_k\mathbf{b}_{k+1} + g_{k+1}\mathbf{c}_{k+1} \\ \mathbf{c}_{k+1}^T\mathbf{B}_k + j_{k+1}\mathbf{b}_{k+1}^T & \mathbf{c}_{k+1}^T\mathbf{b}_{k+1} + j_{k+1}g_{k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & 0 \\ 0 & 1 \end{bmatrix} \tag{A.3}$$

This can be simplified by considering the particular block elements as follows:

$$\mathbf{C}_k\mathbf{B}_k + \mathbf{c}_{k+1}\mathbf{b}_{k+1}^T = \mathbf{I} \tag{A.4}$$

$$\mathbf{C}_k\mathbf{b}_{k+1} + g_{k+1}\mathbf{g}_{k+1} = 0 \tag{A.5}$$

$$\mathbf{c}_{k+1}^T\mathbf{B}_k + j_{k+1}\mathbf{b}_{k+1}^T = 0 \tag{A.6}$$

$$\mathbf{c}_{k+1}^T\mathbf{b}_{k+1} + j_{k+1}g_{k+1} = 1 \tag{A.7}$$

Using (A.4), the element $\mathbf{B}_k$ may be expressed by the formula:

$$\mathbf{B}_k = \mathbf{C}_k^{-1} - \mathbf{C}_k^{-1}\mathbf{c}_{k+1}\mathbf{b}_{k+1}^T \tag{A.8}$$

Analogically, using (A.5), the element $\mathbf{b}_{k+1}$ may be expressed by the formula:

$$\mathbf{b}_{k+1} = -g_{k+1}\mathbf{C}_k^{-1}\mathbf{c}_{k+1} \tag{A.9}$$

and hence:

$$\mathbf{b}_{k+1}^T = -g_{k+1}\mathbf{c}_{k+1}^T\mathbf{C}_k^{-1} \tag{A.10}$$

because the matrix $\mathbf{C}_k^{-1}$ is symmetric and: $(\mathbf{C}_k^{-1})^T = \mathbf{C}_k^{-1}$

The element $\mathbf{b}_{k+1}$ (A.9) is substituted in (A.7) to produce the remaining element g_{k+1} (10). Therefore, returning to (6) which becomes:

$$\mathbf{a}_{k+1} = \begin{bmatrix} \mathbf{B}_k & \mathbf{b}_{k+1} \\ \mathbf{b}_{k+1}^T & g_{k+1} \end{bmatrix} \begin{bmatrix} \mathbf{u}_k \\ v_{k+1} \end{bmatrix} \tag{A.11}$$

and multiplying the two block matrices, we derive:

$$\mathbf{a}_{k+1} = \begin{bmatrix} \mathbf{B}_k \mathbf{u}_k + \mathbf{b}_{k+1} v_{k+1} \\ \mathbf{b}_{k+1}^T \mathbf{u}_k + g_{k+1} v_{k+1} \end{bmatrix} \tag{A.12}$$

In this matrix we replace $\mathbf{B}_k$ with $\mathbf{C}_k^{-1}$ according to (A.8) in (A.12), expand $\mathbf{b}_{k+1}^T$ according to (A.10), and $\mathbf{b}_{k+1}$ according to (A.9), which yields:

$$\mathbf{a}_{k+1} = \begin{bmatrix} \mathbf{C}_k^{-1}\mathbf{u}_k + g_{k+1}\mathbf{C}_k^{-1}\mathbf{c}_{k+1}\mathbf{c}_{k+1}^T\mathbf{C}_k^{-1}\mathbf{u}_k - g_{k+1}\mathbf{C}_k^{-1}\mathbf{c}_{k+1}v_{k+1} \\ -g_{k+1}\mathbf{c}_{k+1}^T\mathbf{C}_k^{-1}\mathbf{u}_k + g_{k+1}v_{k+1} \end{bmatrix} \tag{A.13}$$

Taking into account equation (5): $\mathbf{a}_k = \mathbf{C}_k^{-1}\mathbf{u}_k$, it follows that:

$$\mathbf{a}_{k+1} = \begin{bmatrix} \mathbf{a}_k + g_{k+1}\mathbf{C}_k^{-1}\mathbf{c}_{k+1}(\mathbf{c}_{k+1}^T\mathbf{a}_k - v_{k+1}) \\ -g_{k+1}(\mathbf{c}_{k+1}^T\mathbf{a}_k - v_{k+1}) \end{bmatrix} \tag{A.14}$$

In the resulted formula (A.14) the coefficients $\mathbf{a}_k$ are ready from the previous iterative step, the vectors: $\mathbf{c}_{k+1}^T$, and v_{k+1}, are produced with the novel independent variable, and g_{k+1} is obtained with these ready elements according to (10). Once computing the quantity (11): $a_{k+1} = -g_{k+1}(\mathbf{c}_{k+1}^T\mathbf{a}_k - v_{k+1})$, we arrive at the efficient recurrent formula (12).

Appendix II: Derivation of the MSE Formulae

The residual mean squared error MSE (17) is obtained from the equation:

$$MSE = \frac{1}{N}(\mathbf{y} - \mathbf{Ha})^T(\mathbf{y} - \mathbf{Ha}) \tag{A.15}$$

where $\mathbf{y}$ is the given outcome vector, $\mathbf{Ha}$ is the estimated output using the design matrix $\mathbf{H}$, and the $A \times 1$ coefficient vector $\mathbf{a}$, and N is the number of the data. After performing the multiplication in (A.15), we get:

$$MSE = \frac{1}{N}(\mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{Ha} - (\mathbf{Ha})^T\mathbf{y} + (\mathbf{Ha})^{\mathbf{T}}(\mathbf{Ha})) \tag{A.16}$$

Having $(\mathbf{Ha})^T(\mathbf{Ha}) = \mathbf{a}^T\mathbf{H}^T\mathbf{Ha} = \mathbf{a}^T\mathbf{H}^T\mathbf{y}$, and eliminating $-(\mathbf{Ha})^T\mathbf{y}$ and $(\mathbf{Ha})^T\mathbf{y}$, equation (A.16) can be simplified as follows:

$$MSE = \frac{1}{N}(\mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{Ha}) = \frac{1}{N}(\mathbf{y}^T\mathbf{y} - (\mathbf{a}^T\mathbf{H}^T\mathbf{y})^T) \tag{A.17}$$

Using $\mathbf{H}^T\mathbf{y} \equiv \mathbf{u}$ in (A.17), and knowing that $(\mathbf{a}^T\mathbf{u})^T = \mathbf{a}^T\mathbf{u}$, yields finally the formula:

$$MSE = \frac{1}{N}(\mathbf{y}^T\mathbf{y} - \mathbf{a}^T\mathbf{u}) \tag{A.18}$$

Let assume that this (A.18) is the error of a model with k independent variables, and let substitute it with MSE_k, that is: $MSE \equiv MSE_k$. After adding the next $(k+1)$-st independent variable the error becomes:

$$MSE_{k+1} = \frac{1}{N}(\mathbf{y}^T\mathbf{y} - \mathbf{a}_{k+1}^T\mathbf{H}_{k+1}^T\mathbf{y}) \tag{A.19}$$

Then, we represent the matrix $\mathbf{H}_{k+1}^T$ in block format $\mathbf{H}_{k+1} = \begin{bmatrix} \mathbf{H}_k & \mathbf{h}_{k+1} \end{bmatrix}^T$, as well as the coefficient vector $\mathbf{a}_{k+1}$ by the block vector (12). Multiplying these blocks by $\mathbf{y}$ in (A.19) gives:

$$MSE_{k+1} = \frac{1}{N}(\mathbf{y}^T\mathbf{y} - (\mathbf{a}_k - a_{k+1}\mathbf{C}_k^{-1}\mathbf{c}_{k+1})^T\mathbf{u}_k - a_{k+1}v_{k+1}) \tag{A.20}$$

Next, this formula (A.20) is further simplified taking $\mathbf{a}_k = \mathbf{C}_k^{-1}\mathbf{u}_k$, from (5):

$$MSE_{k+1} = \frac{1}{N}(\mathbf{y}^T\mathbf{y} - \mathbf{a}_k^T\mathbf{u}_k + a_{k+1}({\mathbf{c}_{k+1}}^T\mathbf{a}_k - v_{k+1})) \tag{A.21}$$

Obviously here we may express $({\mathbf{c}_{k+1}}^T\mathbf{a}_k - v_{k+1}) = a_{k+1}/(-g_{k+1})$ according to (11). Then, substituting the residual amount from (A.18) by $\mathbf{e}_k$, that is: $\mathbf{e}_k = (\mathbf{y}^T\mathbf{y} - \mathbf{a}^T\mathbf{u})$, and using it in (A.21) leads to:

$$MSE_{k+1} = \frac{1}{N}(\mathbf{e}_k - a_{k+1}^2/g_{k+1}) \tag{A.22}$$