UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS num. 69

Implicit function symbolic regression system

Josip Mrđen

Zagreb, July 2020.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Kako biste uklonili ovu stranicu, obrišite naredbu `\izvornik`.

Prije svega, želio bih se zahvaliti roditeljima na svoj brizi, lekcijama, znanju, strpljenju i spoznavanju novih stvari kroz ovu životnu priču. Hvala na svim pustolovinama koje sam proživio s vama od malih nogu, sjećanje na te zgode ostat će mi zauvijek u srcu. Hvala za svaku lijepu (i ružnu) riječ koju ste mi uputili, svaka mi je koristila i bez vas ne bih bio ni upola čovjek kao što sam sada.

Zahvaljujem se mentoru prof.dr.sc Domagoju Jakobovića za svu pruženu pomoć i pristupačnost u pisanju diplomskog rada te 4 godine uspješnog mentoriranja i vođenja ostalih znanstvenih radova.

I na kraju, hvala svim mojim prijateljima, kolegama i poznanicima koji su me pratili na ovom životnom putu i ispunili moje studentske dane smijehom, srećom i vrhunskim druženjem.

CONTENTS

1. Introduction

With the development of mathematics and natural sciences throughout history, the need has arisen for the formulation of all processes in nature in order to describe their motions and actions, furthermore to predict its behavior in the future. Persistence in the formulations and notation of the process spawned the entities we call mathematical equations today. In everyday life, they serve us to come up with certain research results or to calculate an interpretable measure that can be expressed numerically. On the other hand, by observing new systems, it may happen that all values are known to us in the set of moments in which they are recorded, but at first glance it is not obvious which formula connects them. By sampling the system against a reference variable (e.g. time [Bongard i Lipson (2007), Brunton et al. (2016)]), one can obtain a set of points and drive some of the modern regression methods with which one could describe the whole system, and predict its behavior in future moments. One of such methods is symbolic regression, which in this paper deals with a specific set of mathematical functions - implicit equations.

The second chapter describes the problem of symbolic regression as well as the nature of implicit functions. The issue of functions and the distinction made in relation to explicit functions will be the main motivation for research.

The third chapter introduces approaches that can make it easier to search for solution space. Of all the methods, the one with the comparison of partial derivatives in a data set of points will be of the greatest importance. In addition, simpler but equally interesting methods will be mentioned that look at the scattering of the values of the points themselves in space. Given the expressiveness of implicit functions, refinements of the usual objective function method which works well for explicit equations should have been made for the improvement of effectiveness. In this regard, great effort has been put into the mathematical part to determine the derivations from the points experimentally. Tools suitable for working with symbolic regression are also described, emphasizing genetic programming and their variations. Their specifics and structure of individual solutions will be suitable for mathematical modeling of equations.

The fourth chapter describes the used methods, developed algorithm models and main challenges that needed to be overcome in order for the model to work properly. The implemented goal functions, their algorithms and the intuition behind them have been presented.

The fifth chapter describes the results obtained by symbolic regression for different sets of functions. The types of functions over which the implementation works satisfactorily are described, as well as the types of functions over which the implementation fails to contribute significant knowledge about the underlying correlation between points. Different variants of genetic programming were tested and the results were recorded with comments.

The sixth chapter contributes to the conclusion with possible directions for further work on the project. The advantages of the research, the progress in relation to the starting point during the whole project, but also the space for progress in the mathematical part of determining the experimental partial derivatives are pointed out.

2. Motivation for symbolic regression

Symbolic regression is a type of mathematical regression that uses a certain procedure to obtain a function that satisfactorily describes the behavior of a set of points in an N-dimensional coordinate system (Figure 2.1). The term is reminiscent of linear regression, however the important difference is that the line is used as a mold for modeling a function in linear regression, while symbolic regression does not place restrictions on operators and they can be arbitrary, whether they are linear or nonlinear. Such an approach gives the function additional degrees of freedom to model its expressiveness, but also increases the search space in which lie the mathematical functions that represent potential solutions to the problem. The problem is therefore classified as NP-heavy and therefore brute-force function searches are out of the question. An algorithm that would solve such a problem can be represented by a black box in which the input is a data set of points sampled from, say, a dynamical system [Gaucel et al. (2014), Schmidt i Lipson (2009)], and the output is a closed mathematical equation. Methods that can facilitate the process of searching for such functions will be described in detail in the next section.

2.1. Explicit functions

Functions more often researched in symbolic regression are explicit in nature, and are denoted by the following notation:

$$y = f(x_1, x_2, ..., x_n) = f(\mathbf{x}) \tag{2.1}$$

where y denotes the dependent and the vector **x** the independent variable of the function. Such functions are much easier to understand intuitively, since for a given value of the vector **x** there is a definite value of the dependent variable y. With the above notation, it is easy to use it for symbolic regression if the quadratic error function is applied:

$$L(\hat{y}, y) = \frac{1}{N} \sum_{i=1}^{n} (y_i - \hat{y_i})^2 \tag{2.2}$$

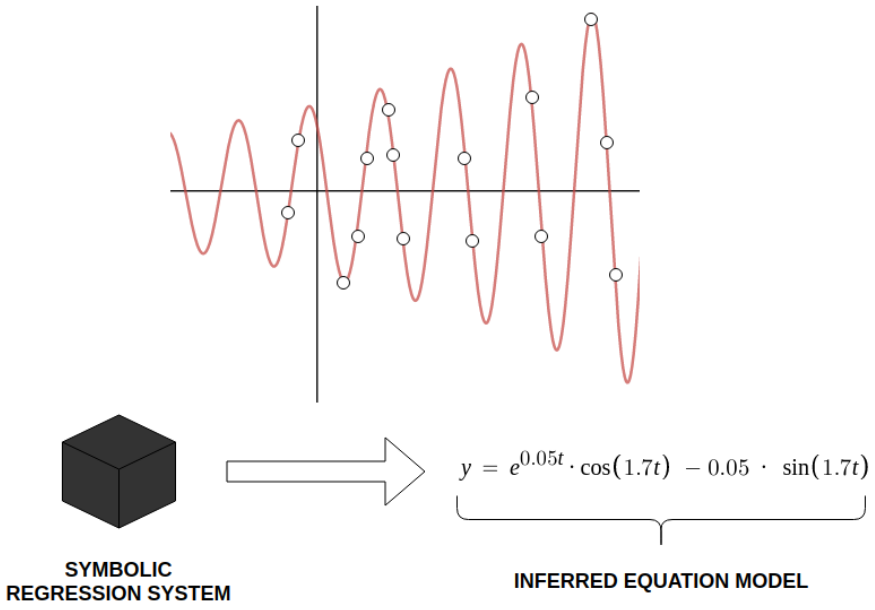

Figure 2.1: The equation itself is not known in symbolic regression, but the set of points that represent it

where y indicates the actual value of the function, $\hat{y}$ the value estimated by the current function, and N the number of examples in the training set. The error function written in this way can be easily used in various methods such as genetic programming and others to obtain a final solution for a given data set. During training, the better the function estimates the values of the dependent variable, the smaller the error will be and thus the algorithm is directed towards the best possible functions.

2.2. Implicit functions

Much more expressive functions than explicit ones are the implicit ones, which are denoted by the following notation:

$$f(x_1, x_2, ..., x_n) = 0 \rightarrow f(\mathbf{x}) = 0 \tag{2.3}$$

From the notation itself it can be seen that all variables are on one side, i.e. all variables of the vector **x** are independent. Permissible values of the function are all values of the vector that evaluate the function on the left to zero. Functions of this nature can model much more complex mathematical relationships. A simple example is a circle, which is explicitly defined with 2 formulas, while the implicit form requires only one function (Figure 2.2)

At first glance, nothing has changed in defining the equations, except that the dependent

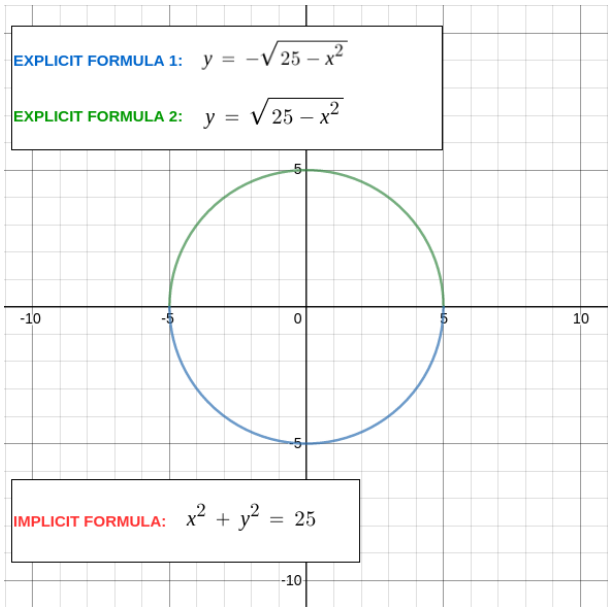

Figure 2.2: Expressiveness of the implicit equations enable the mathematical description of circle in only one equation

variable has disappeared. Therefore, the error function could be the same as the above one (2.1), with the inclusion of zero in place of the independent variable y:

$$\begin{aligned} L(\hat{y}, y) &= \frac{1}{N}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \\ &= \frac{1}{N}\sum_{i=1}^{n}(y_i - 0)^2 \\ &= \frac{1}{N}\sum_{i=1}^{n} y_i^2 \end{aligned}$$

Intuitively, the formulation seems to be correct. However, such a formulation is too general because a large number of trivial equations satisfy such an equation. Also, many trivial functions are very close to zero for all points of the set. The consequence of all this is that as a regression solution can be functions such as:

$$\begin{aligned} x_1 - x_1 &= 0 \\ \sin^2(x) + \cos^2(x) - 1 &= 0 \\ x - \frac{x \cdot y}{y} &= 0 \\ \frac{1}{1000 + x^2} &= 0 \ldots \end{aligned}$$

All of the above functions are trivial and should not be considered at all in the method for searching the function space. The focus of the paper, however, is to find nontrivial equations

that satisfy the formula (2.2). As a "quick" solution to such a problem, constraints can be introduced on functions, more precisely on their form, thus eliminating trivial solutions since they are not useful. With this approach, one of the problems that arises is that no matter what the constraints were, or how they were written, the function can be complex enough (with more operands and operators and nested operators) to eventually bypass the same constraints. In this paper, such an approach is not discussed, and in the next chapter, 2 possible methods of solving are proposed as a starting point for correct solutions.

3. Methods and tools for symbolic regression

3.1. Direct method of solving

The method presented in this subchapter uses the standard deviation as a tool for manipulating algorithm solutions. Since the evaluation of all points in the mathematical formula should be zero, the scattering of the points around the center can be represented by the standard deviation:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - \mu)} \tag{3.1}$$

Where σ represents the standard deviation, N the number of points in the data set, $\hat{y}_i$ the estimated value for each point, and μ the mean value of the estimated points. The scattering can be present around any real coefficient. However, this is good enough since the shape of the formula itself is of greater importance than the free coefficient offset from zero.

To eliminate trivial functions, the following mechanism is proposed. An arbitrary set of random points is selected from which the standard deviation as input is checked. If the standard deviation for these points is very small, there is a good chance that this function is trivial. The individual is discarded in that case. Otherwise, the individual is evaluated over all points of the set and the standard deviation is tried to be reduced, since all results must be of the same amount, regardless of the deviation from zero. An algorithm using the direct standard deviation method is proposed below.

Input: K – number of random points, N – dimensionality of the system, J – invidividual being evaluated, $threshold$ – arbitrarily small number
T := sampleRandomlyFromGaussDistribution(K, N)
E := [] (evaluations of random points)
for (point := $x_1, x_2, ...x_N$ from T) **do**
  evaluation = J.evaluate(point)
  addToSolutions(E, evaluation)
end for
stdev = calculateStandardDeviation(E)
if $stdev < threshold$ **then**
  discardSolution(J)
else
  acceptSolution(J)
end if
Algorithm 1: Method for rejecting trivial solutions using standard deviation

3.2. Solving using partial derivatives

The following method uses partial derivatives as a tool with which to more effectively evaluate a solution that estimates an error over a data set. In the paper [Schmidt i Lipson (2010)], a comparison of model derivatives with experimentally obtained derivatives at each point in the data set is proposed. The principle is based on the idea that if the model has as similar a derivation amount as possible to the actual solution, it will more faithfully describe the data set.

3.2.1. 2-dimensional system

Given the above, the two-dimensional solution formula would look like this:

$$L = \frac{1}{N}\sum_{i=1}^{N} log\left(1 + |\frac{\Delta x_i}{\Delta y_i} - \frac{\delta x_i}{\delta y_i}|\right) \tag{3.2}$$

where N is the number of points in the data set, $\frac{\Delta x}{\Delta y}$ is an experimental implicit derivation from the data at point i, and $\frac{\delta x}{\delta y}$ implicit derivation of the solution at the point i. A more efficient function would be one that, according to the formula (3.2), would correspond more to the experimentally obtained derivatives from the data.

For a two-dimensional data set of a system with corresponding coordinates (x, y), one could record the values $x(t)$ and $y(t)$ over time. The derivation would then be $\frac{\Delta x}{\Delta y} = \frac{x'}{y'}$ where

x' and y' are the derivatives of each coordinate in time, while the derivation obtained by subtracting adjacent points at appropriate coordinates. For a single function f, differentiation could be used to derive the variables of the function:

$$\frac{\delta x}{\delta y} = \frac{\frac{\delta f}{\delta y}}{\frac{\delta f}{\delta x}} \text{ or } \frac{\delta y}{\delta x} = \frac{\frac{\delta f}{\delta x}}{\frac{\delta f}{\delta y}}$$

Finally, with experimental and individual derivations obtained, the total loss of the function can be calculated with the formula (3.2).

3.2.2. N-dimensional system

The two-dimensional principle can be easily generalized to a multi-dimensional system. Let the point T_i be determined by the coordinates $(x_1, x_2, ...x_D)$ where D denotes the dimensionality of the system. By sampling each coordinate $x_i(t)$, a trajectory can be obtained for each individual coordinate of the point. In this case, each pair of different coordinates would have its own evaluation of the derivation error deviation. The final formula for a system with N points would look like this:

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{D-1} \sum_{k=j+1}^{D} log\left(1 + |\frac{\Delta x_j}{\Delta y_k} - \frac{\delta x_j}{\delta y_k}|\right) \tag{3.3}$$

where the first sum moves along all points of the set, and the second and third sums move along all pairs of coordinates. The expression can be further normalized with division by the number of combinations of different pairs of derivatives, so the final formula can also look like:

$$L = \frac{1}{\frac{D\cdot(D-1)}{2}} \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{D-1} \sum_{k=j+1}^{D} log\left(1 + |\frac{\Delta x_j}{\Delta y_k} - \frac{\delta x_j}{\delta y_k}|\right) \tag{3.4}$$

For an ordered set of points having a particular trajectory this system would theoretically work well. However, the limitation on the order of the points is too high a price to guarantee the correctness of the system. Regression systems for explicit equations do not depend on the order of the points and get such robustness “for free”. For implicit systems, it is necessary to introduce additional modifications or adjustments with which derivations could be determined experimentally with satisfactory accuracy. Moreover, the method of determining experimental derivations depends only on the data set, and pre-processing of points could already provide ready-made information on points and their derivatives. Given the conditions, the following method is proposed.

3.2.3. System with unordered data

For working with unordered data, there are multiple methods for determining the derivatives. The paper [Schmidt i Lipson (2010)] suggests determining hyperplanes in a local field of the point for which the derivative wants to be estimated. Another paper suggests a similar method [Brabanter et al. (2013)]. For 2 dimensional data, that plane would be the tangent on a point, while in a multi-dimensional system, the derivative would be calculated with an N-dimensional hyperplane (Slika 3.1).

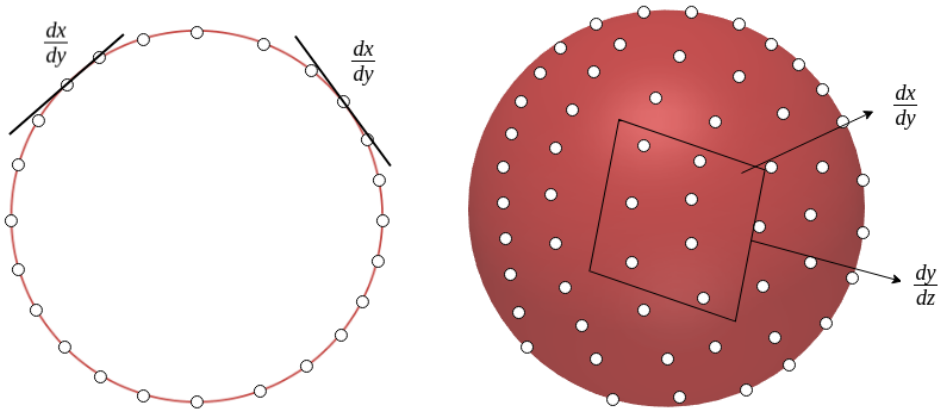

Figure 3.1: Reconstruction of the hyperplane on unordered data for cicle (2D) and sphere (3D)

In essence, a hyperplane denotes a surface of appropriate dimension that well describes locally related points. It is implicit in form, can be modeled and evaluated using a derivation definition to obtain experimental values. Since the hyperplane corresponds to only one point, their number would be equal to the set of points. Assuming that the hyperplane (denoted by $H(x_1, ...x_D)$) is found for the point, a partial derivative for the point coordinate can be obtained with

$$\frac{\delta H}{\delta x_i} = \frac{H(\mathbf{x} + \Delta h) - H(\mathbf{x})}{\Delta h}$$

where the offset $\Delta h = (0, ..., 0, \epsilon_i, 0, ..., 0)$ has an infinitesimal shift in the coordinate in which one wants to calculate the implicit derivative. With given partial derivatives for each coordinate respectively, differentiation of coordinates in 2 dimensions would look like:

$$f(x,y) = 0 \mid d$$
$$\frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy = 0$$
$$\frac{dx}{dy} = -\frac{\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}} \text{ or } \frac{dy}{dx} = -\frac{\frac{\partial f}{\partial x}}{\frac{\partial f}{\partial y}}$$

This calculation agrees with the equation (3.2.2) and can be used to evaluate the individual as an experimental derivation part in the formula.

3.2.4. Determining hyperplane using conic sections

The least squares method was used to determine the hyperplane equation. The paper [Shpitalni i Lipson (2001)] has taken the fitting of scattered points into conical shapes from the paper [Bookstein (1979)], although any shape can be used for the underlying model. The results obtained by the following model are described in the results chapter. The currently described model works well for spherical and rounded shapes while for most other types of functions it has proven to be insufficiently good.

The shape of the plane in 2 dimensional space with the coordinates of the points (x, y) is described with the formula:

$$Ax^2 + By^2 + Cxy + Dx + Ey + F = 0 \tag{3.5}$$

where the coefficients (A, B, C, D, E, F) denote the coefficients obtained by the method of least squares, and the variables (x, y) include the "center" of the hyperplane around which the derivation is determined. The equation is actually a polynomial of degree 2, but its shape resembles an ellipse, so it is said that scattered local points "fit" into elliptical and conical sections. If one denotes the cardinality of closest points (can also be referred to as local points) by N, the system of equations would look like:

$$Ax_1^2 + By_1^2 + Cx_1y_1 + Dx_1 + Ey_1 + F = 0$$
$$Ax_2^2 + By_2^2 + Cx_2y_2 + Dx_2 + Ey_2 + F = 0$$
$$\dots$$
$$Ax_N^2 + By_N^2 + Cx_Ny_N + Dx_N + Ey_N + F = 0$$

By shifting the free component F to the right and then dividing by the same member, the equations are in the form of:

$$\frac{-A}{F}x_1^2 + \frac{-B}{F}y_1^2 + \frac{-C}{F}x_1y_1 + \frac{-D}{F}x_1 + \frac{-E}{F}y_1 = 1$$
$$\dots$$
$$\frac{-A}{F}x_N^2 + \frac{-B}{F}y_N^2 + \frac{-C}{F}x_Ny_N + \frac{-D}{F}x_N + \frac{-E}{F}y_N = 1$$

where coefficients beside the variables can be denoted as a new arranged couple (A', B', C', D', E') and it can be written in matrix as:

$$\begin{vmatrix} x_1^2 & y_1^2 & x_1y_1 & x_1 & y_1 \\ x_2^2 & y_2^2 & x_2y_2 & x_2 & y_2 \\ & & \dots & & \\ x_N^2 & y_N^2 & x_Ny_N & x_N & y_N \end{vmatrix} \cdot \begin{vmatrix} A' \\ B' \\ C' \\ D' \\ E' \end{vmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Such a system is a matrix equation **Ax = b** where the column vector **x** is a vector of coefficients (A', B', C', D', E') that are sought, **A** is a matrix derived from the values of local points, while **b** is a column vector filled with ones. It should be noted that the following hyperplane equation is equivalent to the formula (3.2.4), but has one less unknown variable:

$$A'x_1^2 + B'y_1^2 + C'x_1y_1 + D'x_1 + E'y_1 - 1 = 0$$

The solution of such a system depends on whether the system is indeterminate or predetermined. With the notation N for the number of unknown hyperplane coefficients and M for the number of system equations, the following conclusions can be drawn. When the number of system equations is equal to the number of system dimensions $(M = N)$, the system has only one solution. Given the equality of the number of equations and the number of dimensions, the matrix A is quadratic and the solution can be obtained by left division:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

When the system is indeterminate, the number of system equations is less than the number of system dimensions $(M < N)$. The matrix in this case is "wide" (it has more columns

than rows) and often has more solutions. An attempt is made to take a solution **x** with a minimum norm:

$$\min_{\mathbf{x}} ||\mathbf{x}||_2^2 \tag{3.6}$$

such that **Ax = b**. Lagrange multipliers with the constraint in the form of a system equation are defined, however this sub-case is not so important for the work itself. The excerpt described in (Selecnick (2013)) gives the following system solution:

$$\mathbf{x} = \mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{b} = \mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{1} \tag{3.7}$$

When a system is predetermined, the number of equations describing it is greater than the number of variables $(M > N)$. It often happens that such a system has no solution and the matrix of such a system is "tall" (it has more rows than columns). Due to the large number of equations, such a matrix mostly has no solution, but an attempt is made to find a solution with the lowest minimum norm that satisfies the following equation:

$$J(\mathbf{x}) = ||\mathbf{b} - \mathbf{A}\mathbf{x}||_2^2 \tag{3.8}$$

In the expansion of the norm it follows:

$$\begin{aligned} J(\mathbf{x}) &= (\mathbf{b} - \mathbf{A}\mathbf{x})^T(\mathbf{b} - \mathbf{A}\mathbf{x}) \\ &= \mathbf{b}^T\mathbf{b} - \mathbf{b}^T\mathbf{A}\mathbf{x} - \mathbf{x}^T\mathbf{A}^T\mathbf{b} + \mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} \\ &= \mathbf{b}^T\mathbf{b} - 2\mathbf{b}^T\mathbf{A}\mathbf{x} + \mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} \end{aligned}$$

Deriving with respect to vector **x** and setting the derivation to zero gives:

$$\begin{aligned} \frac{\partial}{\partial \mathbf{x}} J(\mathbf{x}) &= -2\mathbf{A}^T\mathbf{b} + 2\mathbf{A}^T\mathbf{A}\mathbf{x} \\ \frac{\partial}{\partial \mathbf{x}} J(\mathbf{x}) &= 0 \rightarrow \mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b} \end{aligned}$$

Assuming that the matrix $\mathbf{A}^T\mathbf{A}$ is invertible, the hyperplane coefficients are obtained using the formula:

$$\mathbf{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{1} \tag{3.9}$$

and that is the least squares solution (Selecnick (2013)). The matrix $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$ is called a pseudoinverse and is often referred to in the literature as $\mathbf{A}^+$. Since with the obtained solution J(**x**) is the smallest, such a solution will be closest to the zero-vector in case the solution

does not exist.

The obtained expression is of special importance because in practice the cardinality of a set of points is many times higher with respect to the cardinality of the system dimension. Moreover, the neighborhood can then be chosen such that the cardinality of neighborhood for each point is greater than the dimensions of the system to determine the coefficients using pseudoinverses. However, the parameter describing the size of the neighborhood of local points is not in itself trivial to determine nor is it obtained in the paper exactly. Also, the problem with this mathematical approach to determine the hyperplanes is that the matrix $\mathbf{A}$ or $\mathbf{A}^+$ can always be singular due to the configuration of points in the environment, and in that case it will not be possible to obtain the hyperplane coefficients, i.e. no derivation point can be reached for such a neighborhood. The success results of determining the derivation of hyperplanes with respect to size of the local field are described in the chapter with results. The positive point is that the matrix $\mathbf{A}$ is mostly rarely singular.

The formula (3.2.4) would only be useful for a 2 dimensional case, so it should be generalized for multidimensional points. This is not so difficult since for the second degree of the polynomial there is a correlation between the 2 variables only at the first power. Expanding the formula for a multidimensional ellipse/cone in N dimensions would look like:

$$\begin{gathered} A_1x_1^2 + A_2x_2^2 ... + A_Nx_N^2 \\ +B_1x_1x_2 + B_2x_1x_3 + ... + B_{\frac{N(N+1)}{2}}x_{N-1}x_N \\ +C_1x_1 + C_2x_2 + ... + C_Nx_N + D = 0 \end{gathered}$$

The equation of a multidimensional ellipse would then have N coefficients for each squared coordinate, N coefficients for each linear component, $\frac{N\cdot(N+1)}{2}$ coefficients for each pair of coordinates that can correlate these one free coefficient which would disappear in simple mathematical operations. The total number of coefficients then for each hyperplane is then $2N + \frac{N\cdot(N+1)}{2}$.

3.2.5. Other shapes for determining the hyperplane

The conical shape is just one of the models that can be included in the least squares method for the final determination of experimental derivations. In practice, the choice of shapes is infinite. The premise of the thesis initially was that the ellipsoidal model would describe any neighborhood of points well and give a sufficiently good approximation of the derivative for any implicit function of multiple variables. For spherical and spherical functions, the model

describes derivatives with great accuracy, while for others this was not true. The accuracy of the derivation approximations themselves is described in the results section.
In addition to second-degree polynomials, experiments were performed with a linear model described for the $2D$ system as:

$$Ax + By + C = 0 \tag{3.10}$$

and with a polynomial of third degree:

$$Ax^3 + By^3 + Cx^2y + Dxy^2 + Ex^2 + Fy^2 + Gxy + Hx + Iy + J = 0 \tag{3.11}$$

The results of the experiments with these 2 models only served to compare the approximations with the elliptic model for some functions and were not used as input for the regression algorithm.

3.3. Genetic programming

Of all the methods suitable for solving the problem of symbolic regression, the most commonly used is the method of genetic programming. Such a method works based on finding the best algorithm or model for a particular problem. In doing so, it uses an expression/syntactic tree as a genotype (also called a chromosome), which is extremely suitable for displaying mathematical formulas. The representation of a mathematical formula by a tree is done with the rule that the internal nodes form the operators, and leaves form the operands. Operands can be constants (values that do not change), or variables (in practice points from a data set). Operators can be mostly binary (i.e. addition, subtraction, etc.) or unary (i.e. sine, exponential, etc.)
The figure (Figure 3.2) shows a description of the function in such view. By visiting the tree in-order, the same formula is obtained in infix form, which is humanly readable. Another advantage of the tree as a genotype is the simple evaluation of the individual, i.e. the calculation of the value of the function. The value calculation, however, is not done by the in-order procedure, but mainly by pre-order reading the tree. The procedure boils down to a recursive reading of children's values at greater tree depths until all values are known. When all the values are known, the operator calculates the value between his children and propagates the result upwards. The following is a simple recursive algorithm for calculating the value of a function.

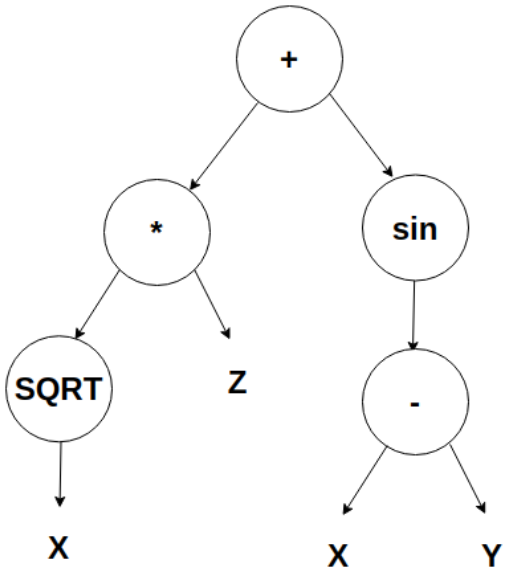

Figure 3.2: Expression/syntactic tree which represents a mathematical function

SOLVE ALGORITHM
Input: T – expression tree
if T != null **then**
  if T.type == operand **then**
    return T.value
  end if
else
  (T is an operator)
  if T.operatorType = unary **then**
    X = solve(T.child)
    return operator X
  else
    (operator is binary)
    A = solve(T.left)
    B = solve(T.right)
    return A operator B
  end if
end if

Algorithm 2: Algorithm for evaluating a mathematical function written as a expression tree

In addition to genotypes, the parts involved in genetic programming are selection, crossover, and mutation, as in any other evolutionary algorithm. The way when and how they are used is determined by the internal algorithm of genetic programming. Namely, the only fixed part of genetic programming is the tree as a genotype, while the progression algorithm can be any evolutionary algorithm (or some other). Thus, genetic algorithms (e.g. SteadyStateTournament), ClonAlg and others are often used.

The figures (3.3, 3.4) show simple crossover and mutation operations. A simple crossing between two individuals is performed by the process of determining the breaking point on both individuals, from which the subtrees of each are taken and replaced. Such a replacement may not always succeed if a maximum or minimum tree depth limit is set. A simple mutation is performed by selecting a random node in the tree, and replacing it with another operator or operand.

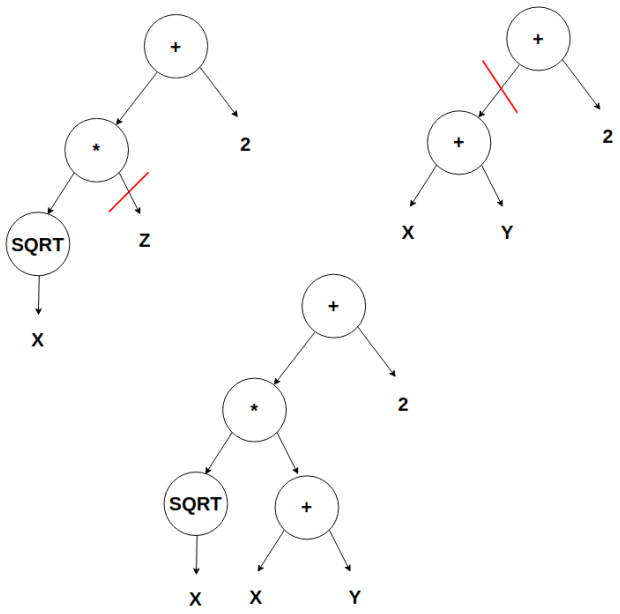

Figure 3.3: Example of a simple crossover operation for genetic programming. Point of crossover is chosen for each tree and the nodes are switched

Hyperparameters, such as the probability of mutation, the crossover probability, the method of selection, the size of the population (number of individuals) and others, play a role in the success of the algorithm. Experiments with hyperparameters were performed in the results section.

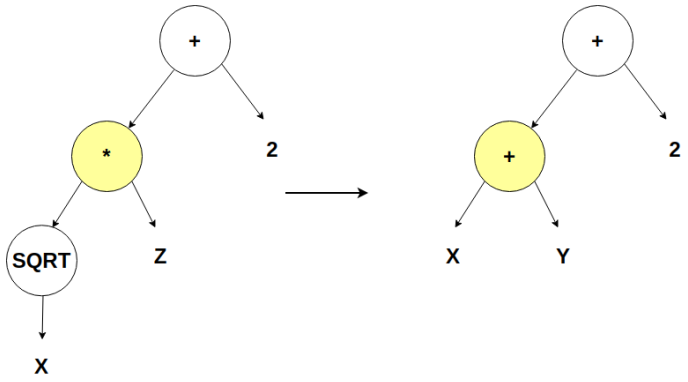

Figure 3.4: Example of a simple mutation operator for genetic programming. A node in the tree is chosen, following a random subtree generation

3.3.1. Gene expression programming (GEP)

Genetic programming bridged the genotype disadvantage of a genetic algorithm that was always fixed and represented by bitstring or decimal numbers. With its tree as a genotype that is also encoded in the same phenotype, solutions of variable length are enabled and greater expressiveness is obtained [Koza i Poli (2005)]. However, shortcomings in the use of genetic operators have also arisen. Manipulating a tree and changing it is a harder job than changing a decimal number or bitstring. For example, mutating a single node of a tree in such a way as to remove that part of the tree and construct a new random subtree has the disadvantage of constantly paying attention to the maximum size of the tree. Iterating through generations, trees are built on a large number of nodes, which is called bloating [Ferreira (2001)].

In response to these problems, the Gene Expression Programming Algorithm was born at the turn of the century. The genotype of such an algorithm is a fixed-length character string encoded as a tree in the phenotype. Since this type of programming found a pattern in the natural process of protein synthesis, the genotype thus written resembles the sequence of nucleotide bonds that make up proteins, and their arrangement and number determine the behavior of the organism, in this case the expression tree as a phenotype.

The tree is coded from the character string by building the root first, then the children of the first depth (root children) from left to right, then the children of the second depth from left to right, and so on. The algorithm stops when all the children are filled with terminal nodes because it is no longer possible to stack new nodes on them. The figure (Figure 3.5) shows the genotype in the character string and the phenotype in the expression tree of the same individual.

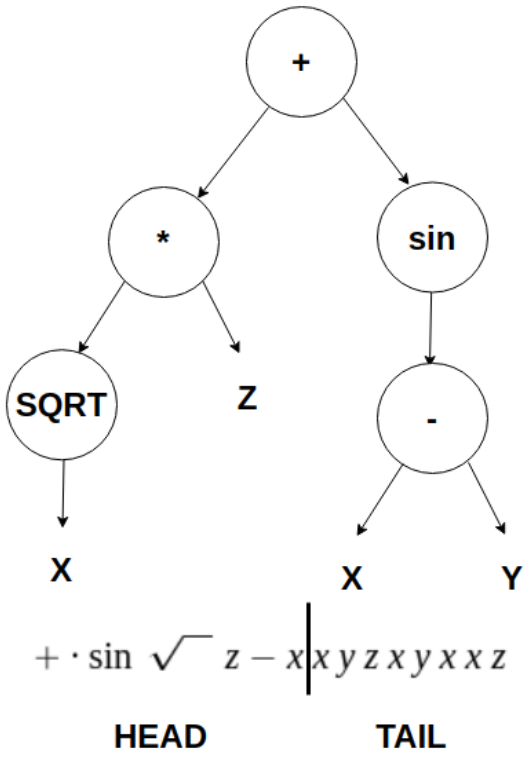

Figure 3.5: Description of an expression tree in GEP notation

Furthermore, each character string is composed of a head and a tail. The head of a string is an area with terminal and function characters, while the tail of a string is an area with only terminal characters and is always longer than the head. The goal of such a tail is to allow complete and valid tree construction in the case of a large number of operators in the head. If the number of head characters h is determined a priori, with the maximum number of children in a node as n, the formula for the tail length of the genotype looks like:

$$t = h \cdot (n - 1) + 1 \tag{3.12}$$

and the total length of the genotype is $h + t$. It can often happen that there are enough terminals in the head for the tree to be built even after only a few elements. The other elements in this case represent non-coding regions (named identically by the biological term protein structure). By such a principle, multiple genotypes can yield the same phenotype, but not the other way around, which corresponds to the correctness of the character string as the tree representation.

In addition to this presentation, minimum restrictions must be introduced. These are that in all genetic operators, if there is a change in the string, the members of the head can be

changed by terminals or function nodes, and the tail should be changed only by terminals. This is a consistency constraint, which allows the products of each genetic operator to remain valid trees. The images (3.6, 3.7) show simple crossover operations and mutations between individuals. Since changes occur at the genotype level, the need to cut the tree to preserve maximum depth has ceased. In the event that a change is made in a non-coding region, such a change is called neutral. It is still preserved, but due to the current construction of the tree it is not applied and does not affect the fitness. For the most part, such neutral changes are also encouraged in the general case as they encourage diversity and the search for solution space.

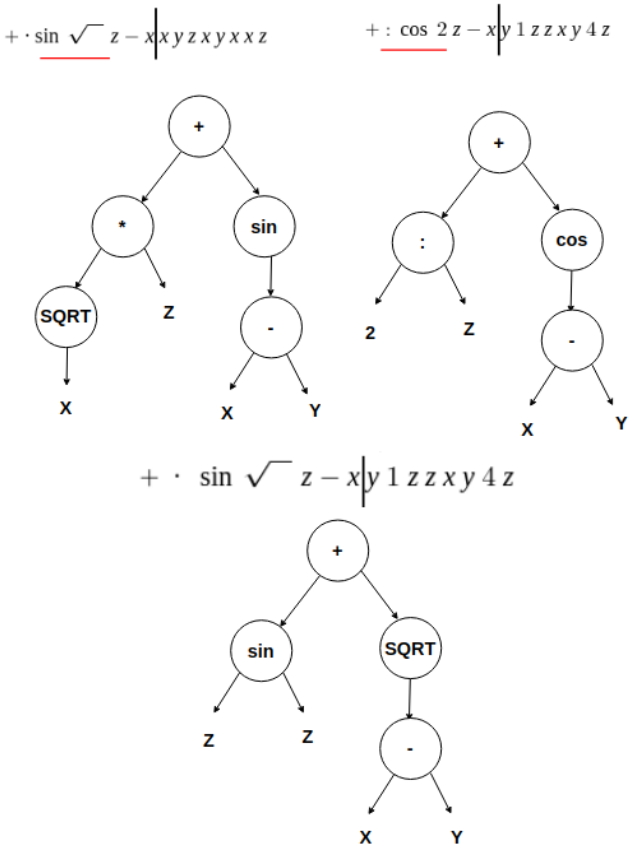

Figure 3.6: Example of a simple crossover operation for GEP. Crossover points are chosen from each tree and switched, with head and tail constraints maintained

Multiple trees can be recorded in the genotype, provided that the size of each tree representation respectively is determined in advance. In that case, the size representation of the whole genotype would be $N \cdot (h + t)$, where N denotes the number of trees, and h and t the length of the head and tail, assuming that all the trees would have the same size. The operation that connects them can be arbitrary, and when evaluating an individual, the evaluations of each tree are calculated first, following a common operator that is applied to the results.

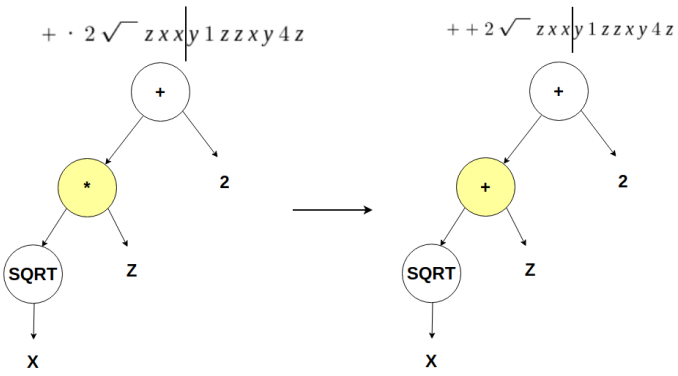

Figure 3.7: Example of a simple mutation operation for GEP. Random node in the expression string is replaced, with head and tail constraints maintained

The advantage of this approach is the possibility that if the solution contains multiple parts that are separable, each tree could learn a separate part of the solution, thus simplifying the work that would have to be done by an individual with only one tree in the genotype.

3.3.2. Analytical programming (AP)

This variation of genetic programming can also be combined with any evolutionary algorithm as a core. However, the key difference between the two is using discrete set handling (DSH) as a method for building the genotype of the solution [Kominkova Oplatkova et al. (2013)].

Before the start of algorithm, all the operators and operands must be known, as well as the number of arguments each operator takes. They are reorganized into sets, with one general function set (GFS) where all the operators and terminals are put, and additional subsets of GFS ordered by the number of arguments each operator takes (all terminals are considered to have zero arguments).

The genotype of the algorithm consists of a set of integers denoting the indices in the GFS. Since all elements in function sets can be numbered, genotype indices will denote precisely the operators and operands that are arranged in GFS. The phenotype is also a syntactic tree and the conversion from genotype occurs in two stages. The first consists of mapping each index to the required operators and operands in GFS, while the second phase is serves for security to fix invalid solutions if the genotype does not have enough terminals to complete the tree representation. The figure (Figure 3.8) shows the construction of a genotype for analytical programming. In case there are not enough arguments to complete the tree, the last indices are determined in the GFS subset with terminals by counting cyclically accord-

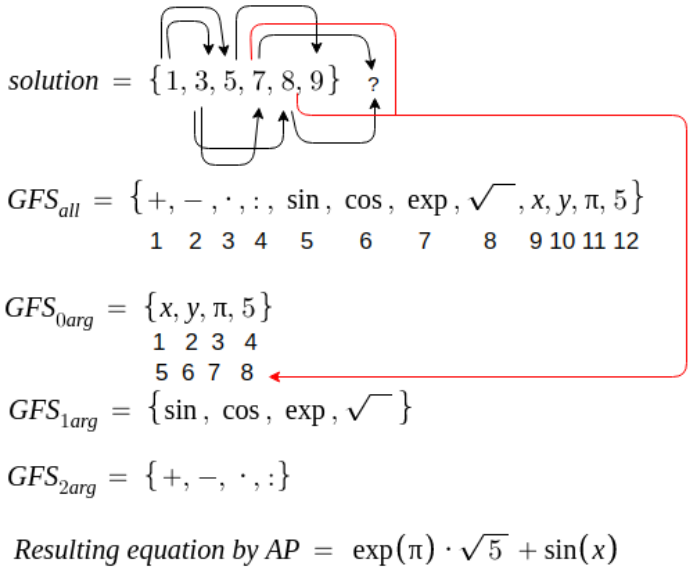

Figure 3.8: Conversion from genotype to phenotype using discrete set handling. Indices with no arguments to take are picked by terminals in cycling order

ing to the given index.

The mutation and crossover operators remain unchanged as the genotype is now a list of numbers which can be easily manipulated with any evolutionary heuristic. Together with GEP, analytical programming provides an easier algorithm manipulation of the genotype while iterating over generations, whereas maintaining the phenotype as a syntactic tree provides faster calculations on the formula itself.

4. Implementation

The problem of symbolic regression for working with implicit functions is approached in the thesis from 2 points of view, the direct method by calculating the standard deviation and the method of partial derivatives. For this purpose, 5 different methods were developed, the successes of which are described in the results chapter. The approaches themselves concern the evaluation of an individual in order to provide the best possible heuristics for the selection of the best individuals, i.e. potential solutions to problems. Since the problem of trivial solutions appeared at the beginning of the paper, the method from the algorithm (1) was used for each algorithm, and it was additionally included to check whether all variables are contained in the formula. Otherwise, the individual is punished with a large penalty and will almost always not contribute to the advancement of the algorithm. The approaches listed below represent only the goal function, driven by the evolutionary computation algorithms listed in the chapters on genetic programming and its variations.

Standard deviation method

For the standard deviation method, no data processing is needed, and the set of points alone is sufficient to evaluate the individual. The approach (goal function) is specified in the algorithm below.

Evaluate
Input: X – data set of points, T – expression tree, $PENALTY$ – arbitrarily big number for punishment
if lowStdevOnRandomPonts(T) or notAllVariablesContainedInTree(T) **then**
  (solution is trivial)
  fitness = PENALTY
  return fitness
end if
evaluations = []
for $point = (x_{i1,...x_{in}})inX$ **do**
  evaluations[i] = T.evaluate(point)
end for
stdev = getStdev(evaluations)
return stdev

Algorithm 3: Evaluating the individual directly using standard deviation

The problem with such heuristic is the lack of information from the dispersion of evaluations. The advantage is certainly speed because this function does not require any complex calculations, and in addition does not require any preprocessing of points. The success of the method is described in the results section.

Partial derivatives method

The method in which the most hope was placed in the thesis, branches into 2 subtypes. In the case of an ordered environment, no preprocessing of the points is necessary, since its derivative can be determined from each adjacent point. The method works for all dimensions since it is only necessary to take their ratios from the change in the value of the coordinates in the points. The method therefore directly agrees with the formula (3.2.2).
By calculating the derivative of an individual by coordinate as:

$$\frac{\delta J}{\delta x_i} = \frac{J(\mathbf{x} + \Delta h_i) - J(\mathbf{x})}{\Delta h_i}$$

where δh_i is an arbitrarily infinitesimal offset in the direction of a certain coordinate, the algorithm can be described in a 2 dimensional system as follows:

Evaluate
Input: D – data set of points, J – expression tree, individual, $PENALTY$ – arbitrarily big number for punishment
if lowStdevOnRandomPonts(J) or notAllVariablesContainedInTree(J) **then**
  return PENALTY
end if
fitness = 0
for currentPoint $= (x_{i1,...x_{in}})inX$ **do**
  previousPoint = D[i-1]
  DX = currentPoint.X - previousPoint.Y
  DY = currentPoint.Y - previousPoint.Y
  experimentalDerivative = DX / DY
  individualDerivative = J.derive(currentPoint)
  fitness += log(1 + |experimentalDerivative - individualDerivative|)
end for
return fitness

Algorithm 4: Evaluation of an individual with partial derivatives and ordered data

Since the evaluation of individuals takes place continuously, experimental derivatives can be stored initially in appropriate structures, to avoid frequent division operation, while here the algorithm is described unoptimized for better readability and understanding of the idea itself. For the N-dimensional system, this is especially true since the number of required ratios is $\frac{N(N+1)}{2}$. A couple of things to consider in such calculations are NaN and infinite values. For example, if the division of the form $\frac{0}{0}$ occurs, the value of the ratio will be NaN. If the division of the form $\frac{\mathbb{R}/\{0\}}{0}$ occurs, the value can be $\pm\infty$. Since such values do not contribute to progress in the solution space, they need to be sanitized. It was decided in the thesis that experimental derivatives of such amounts should be avoided. In the event that an individual produces such functional value of a derivation, both extremes should be punished with the maximum penalty.

Despite a more systematic level of heuristics than the standard deviation method, implying data ordering is not a satisfactory level of robustness for a symbolic regression system, and additional point preprocessing is required for unordered data.

Preprocessing of points and determining the hyperplane

This method is the most robust model that does not look at the order of points and determines experimental derivatives with the help of hyperplanes. Although its heuristics have superiority over the already mentioned methods, for experimental derivations it is necessary to preprocess the data first and find the hyperplanes.

The determination of the hyperplane itself is theoretically described in the previous part of the thesis. It consists of searching for a local neighborhood for each point, and determining a separate hyperplane for each point. The local neighborhood only marks N of the nearest points from the chosen one.

Partial derivations unordered method

Input: D – data set of points, J – expression tree, individual, H – set of hyperplanes $PENALTY$ – arbitrarily big number for punishemnt

if lowStdevOnRandomPonts(J) or notAllVariablesContainedInTree(J) **then**
  return PENALTY
end if
fitness = 0
for currentPoint $= (x_{i1,...x_{in}})inX$ **do**
  currentHyperplane = H[i]
  experimentalPartialDerivatives = []
  individualPartialDerivatives = []
  for $i = 1, ...DIM$ **do**
    experimentalPartialDerivatives[i] = H.derive(currentPoint, i)
    individualPartialDerivatives[i] = J.derive(currentPoint, i)
  end for
  for $i = 0, ..., DIM - 1$ **do**
    for $j = i + 1, ..., DIM$ **do**
      experimentalRatio = $\frac{experimentalPartialDerivatives[i]}{experimentalPartialDerivatives[j]}$
      individualRatio = $\frac{individualPartialDerivatives[i]}{individualPartialDerivatives[j]}$
      $fitness+ = log(1 + |experimentalRatio - individualRatio|$
    end for
  end for
end for
return fitness

Algorithm 5: Evaluation of an individual with partial derivatives and unordered data

As in the previous example, all experimental derivatives can be determined in advance. A great advantage of this model is improved heuristics, while the disadvantage is only the execution time, since despite the caching of derivations they need to be compared between each coordinate ratio.

Using multiple trees for evaluation

In the case of the separability of the formula to be discovered, it is possible by using multiple trees to allow each to learn a separate part of the formula. In this case the trees would be joined by an arbitrary operator which would ideally correspond to the same operator sharing 2 separable parts of the final formula. Since this operator is unknown in advance, the paper thesis a set of operators that can connect 2 trees, which are: addition ($+$), subtraction ($-$), multiplication ($*$) and division ($/$). Two trees were experimented, and the best evaluation of the solution with respect to the given operators was taken as a representative tree of the individual. The pseudocode evaluation of a 2-tree individual is described in the algorithm below.

Evaluate
Input: D – data set of points, $J1$ – first expression tree, $J2$ – second expression tree, $PENALTY$ – arbitrarily big number for punishment
O = [+, -, *, /] (arbitrary set of operators)
for op in O **do**
  if lowStdevOnRandomPonts(J1, J2, op) or notAllVariablesContainedInTree(J1, J2, op) **then**
    O = O/{op}
  end if
end for
if O.empty **then**
  return PENALTY
end if
fitnesses = { }
for currentPoint $= (x_{i1,...x_{in}})inX$ **do**
  evaluation1 = J1.evaluate(currentPoint)
  evaluation2 = J1.evaluate(currentPoint)
  for op in O **do**
    finalEvaluation = evaluation1 op evaluation2
    fitnesses[op] += calculateFitness(finalEvaluation)
  end for
end for
return best(fitnesses)

Algorithm 6: Evaluation of an individual with multiple trees

With this approach, it is necessary to follow trivial solutions for each selection of operators, and those who pass such elimination process are evaluated in a standard way with the forwarding of the solution with best fitness, depending on whether it is a minimization

or maximization problem. This approach can be molded into both a direct approach and a partial derivative approach. The only disadvantage of adding more trees is that it is slower, and this is most felt for partial derivatives since it is necessary to evaluate the infinitesimal shift in the direction of each coordinate.

Tools and technologies used

The programming languages used to implement the algorithms and run the experiments were Python and C++. Python and its numpy library, which is suitable for matrix calculations, were used to preprocess the data, determine the local field around the point, as well as the hyperplanes. Numpy library optimizations were helpful since it was necessary to determine the inverse of the matrix each time the hyperplane was calculated.

On the other hand, the implementation of the direct approach of standard deviation and the approach using partial derivatives was made in the C++ language. The ECF library from Department of Electronics, Microelectronics, Computer and Intelligent Systems (ZEMRIS) at Faculty of Electrical Engineering and Computing (FER), University of Zagreb, was used as a tool for working with evolutionary algorithms. The library provides support for setting arbitrarily written individual evaluation functions, while the selection of the evolutionary algorithm and its parameters is the functionality of the tool out of the box.

All graphs and plots presented in the results chapter were created using the programming language R which provides support for processing the results and were used for analysis over training and testing sets.

5. Experiments and results

To test the algorithms presented in the chapter on implementation, it was necessary to find a set of formulas over which each will be subjected to training. A prerequisite for this was the preprocessing of points from selected sets for methods with partial derivation, i.e., the determination of suitable hyperplanes for obtaining experimental derivative.

5.1. Verification of the least squares method

Handing the correct values to the models using partial derivatives had to be done with the verification that the method is valid and accurate, mostly for the shape of the multidimensional ellipse proposed in the paper (Schmidt i Lipson (2010)), for which the expectations were the highest. In addition, experiments were performed for the hyperplane of the shape of the third degree polynomial and the shape of the ordinary plane, in order to find similarities and differences between the different shapes.

For the first set, formulas whose shape is circular or elliptical were taken, since the results of previous works indicated a well-satisfactory behavior of the hyperplanes for points. The table (5.1) shows all the formulas used for the first part of the test.

Equation	No. Dimensions	Abbreviation	Description
$x^2 + y^2 - 25 = 0$	2	CIR	Circle
$x^2 + y^2 + z^2 - 25 = 0$	3	SPH	Sphere
$\frac{(x-1)^2}{9} + \frac{(y-2)^2}{16} - 1 = 0$	2	EL	Ellipse
$x^3 + x - y^2 - 1.5 = 0$	2	HYP	Hyperbola
$x'' - 0.1 \cdot x' + 3x = 0$	2	HM	Harmonic oscillator
$x'' - 0.1 \cdot x' + 9.8 \cdot sin(x) = 0$	2	NLHO	Non-linear harmonic oscillator

Table 5.1: Equations used for the training set

In determining the hyperplanes, the least squares calculation described in the theoretical part was ran 10 times for randomly generated 1000 points belonging to the function domain. Also, the size of the local neighborhood around the central point was taken into account, experimenting from 0.5 % to 20 % points of the whole set. The measure of success was the function (3.2.2) mentioned in the theoretical part, which would serve as a function of determining fitness during the training of the genetic programming algorithm. The function is minimizational, meaning that values closer to zero symbolized a more faithful picture of the hyperplane at a point relative to the real function. The derivative determined from the hyperplane was taken as the experimental part of the derivation, while the individual function value was precisely the solution of the function, thus serving as a proof of concept itself.

The tables („) show the results of all forms of hyperplanes for the 6 mentioned formulas from the figure (). For the elliptical form of the hyperplane, the best results were achieved for 10% of the local neighborhood with very little scattering. Such a form of the hyperplane was best able to approximate the derivation obtained from the set of points for the formulas presented, consistent with previous research results from other papers. The third-degree polynomial and the ordinary plane were able to describe the harmonic oscillator well, while in all formulas they achieved the best results for 20% of the local set of points. For other local neighborhood configurations, they failed to provide a satisfactory level of performance for all formulas.

F's	Size of local neighborhood					
	5 (0.5%)	10 (1%)	20 (2%)	50 (5%)	100 (10%)	200 (20%)
CIR	0.986±0.046	0.163±0.040	0.033±0.043	5.003±7.311 · 10−4	5.110±4.237 · 10^{-8}	3.125±4.440 · 10−8
SPH	6.577±0.045	2.622±0.011	0.079±0.091	0.008±0.099	0.003±0.020	0.002±0.000
EL	0.794±0.028	0.195±0.056	0.062±0.053	1.678±2.023 · 10^{-3}	1.804±2.649 · 10−5	2.387±4.773 · 10^{-4}
HYP	0.118±0.017	0.066±0.021	0.014±0.003	0.003±0.005	0.013±0.008	0.044±0.030
HO	0.106±0.017	0.078±0.021	0.048±0.005	0.071±0.020	0.106±0.018	0.150±0.035
NLHO	2.03±0.513	0.3918±0.015	0.399±0.101	0.405±0.112	0.962±0.199	1.208±0.533

Table 5.2: Derivatives accuracy for the training set (multidimensional ellipse)

F's	Size of local neighborhood					
	5 (0.5%)	10 (1%)	20 (2%)	50 (5%)	100 (10%)	200 (20%)
CIR	1.352±0.027	1.294±0.068	1.209±0.067	1.105±0.176	1.216±0.171	0.559±0.030
SPH	5.944±0.089	6.037±0.105	6.064±0.192	5.355±0.153	5.015±0.021	4.921±0.082
EL	1.109±0.091	1.189±0.018	1.203±0.023	1.082±0.122	0.969±0.077	0.540±0.048
HYP	0.155±0.041	0.139±0.051	0.183±0.024	0.154±0.061	0.082±0.110	0.049±0.021
HO	3.512±3.418 · 10^{-8}	7.548±2.538 · 10^{-10}	1.692±6.454 · 10^{-10}	192.7±7.827 · 10^{-12}	178.0±6.476 · 10^{-12}	164.7±5.669 · 10^{-12}
NLHO	2.653±0.310	2.498±0.157	2.599±0.111	2.426±0.095	2.140±0.072	2.275±0.101

Table 5.3: Derivatives accuracy for the training set (ordinary plane)

F's	Size of local neighborhood					
	5 (0.5%)	10 (1%)	20 (2%)	50 (5%)	100 (10%)	200 (20%)
CIR	1.294±0.133	1.266±0.084	1.281±0.143	1.129±0.022	1.044±0.026	0.976±0.025
SPH	6.954±0.123	7.338±0.076	5.552±0.236	5.437±0.198	5.276±0.178	5.193±0.165
EL	1.183±0.217	0.992±0.037	1.006±0.046	0.887±0.055	0.871±0.097	0.784±0.049
HYP	0.181±0.025	0.133±0.014	0.106±0.009	0.049±0.009	0.016±0.003	0.004±0.001
HO	0.233±0.114	0.028±0.013	$2.522 \pm 0.921 \cdot 10^{-3}$	$4.189 \pm 1.632 \cdot 10^{-3}$	$3.388 \pm 1.348 \cdot 10^{-3}$	$2.914 \pm 0.370 \cdot 10^{-3}$
NLHO	2.492±0.131	2.451±0.129	2.012±0.301	1.432±0.095	0.790±0.045	0.210±0.028

Table 5.4: Derivatives accuracy for the training set (third degree polynomial)

The next experiment was performed on equations that are at first glance not difficult in form, but generally are not spherical in shape (root function, square function and linear function). Expectations, however, differed from the results obtained. The hyperplane of the second degree polynomial failed to find satisfactory results for any of the offered formulas, while the quadratic function was a problem due to the singularity of the matrix $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$ (pseudoinverse). The third-degree polynomial had almost identical results, while the ordinary plane was able to approximate the derivatives of the quadratic function with satisfactory results. The tables are shown below.

F's	Size of local neighborhood					
	5 (0.5%)	10 (1%)	20 (2%)	50 (5%)	100 (10%)	200 (20%)
$U - mgz = 0$	1.552±0.037	1.159±0.071	1.108±0.020	1.091±0.025	1.183±0.068	1.161±0.063
$y - \sqrt{x} = 0$	1.118±0.093	0.954±0.024	0.941±0.051	0.896±0.029	0.879±0.036	0.838±0.056
$y - x^2 = 0$	/	/	/	/	/	/

Table 5.5: Derivatives accuracy for additional equations (multidimensional ellipse)

F's	Size of local neighborhood					
	5 (0.5%)	10 (1%)	20 (2%)	50 (5%)	100 (10%)	200 (20%)
$U - mgz = 0$	1.503±0.059	1.249±0.046	0.376±0.079	0.291±0.049	0.336±0.059	0.326±0.043
$y - \sqrt{x} = 0$	1.339±0.040	1.193±0.021	1.076±0.030	0.576±0.168	0.464±0.082	0.313±0.068
$y - x^2 = 0$	/	/	/	/	/	/

Table 5.6: Derivatives accuracy for additional equations (3rd degree polynomial)

F's	Size of local neighborhood					
	5 (0.5%)	10 (1%)	20 (2%)	50 (5%)	100 (10%)	200 (20%)
$U - mgz = 0$	0.444±0.166	0.256±0.037	0.376±0.079	0.237±0.027	0.255±0.030	0.287±0.039
$y - \sqrt{x} = 0$	0.379±0.015	0.445±0.044	0.440±0.059	0.597±0.071	0.370±0.040	0.381±0.123
$y - x^2 = 0$	0.199±0.115	0.013±0.001	0.014±0.001	0.013±0.002	0.011±0.1003	0.010±0.002

Table 5.7: Derivatives accuracy for additional equations (ordinary plane)

The obtained results could be used only for the first set of formulas over the hyperplane of polynomials of the second degree, because there the consistency of performance was maintained. Trying out different forms for other groups of implicit functions gives room to find other ways in which current results can be improved.

5.2. Model training

The six functions mentioned in the table (5.1) represented a set for training over the models described in the implementation. Preprocessed data and derivatives served only models that used partial derivatives, while models that used scatter evaluations obtained ready-made unprocessed data sets. Ordinary genetic programming was taken as the training algorithm. Given the nature of all the functions in the set, the operators selected for training were addition ($+$), subtraction ($-$), multiplication ($\cdot$), division ($/$), and sine (sin). The hyperparameters that were subjected to training were:

- Population size (50, 100, 200, 500, 1000)
- Mutation probability (0.1, 0.2, 0.3, 0.5, 0.7, 0.9)
- Selection method (5-tournament selection, Roulette wheel selection)

For each hyperparameter change, the model would be run 20 times for each function. Once the results for all values of one hyperparameter were obtained, it would be fixed and the next would be subjected to the same evaluation. Of the other parameters, it is important to mention:

- Evaluation count for terminating the algorithm (200 000)
- Satisfying fitness for terminating the algorithm (0)
- Maximum tree depth (10)

The following tables (Tables 5.8, 5.9, 5.10, 5.2) show the results for each model with respect to the population size parameter. The best results for each function are colored yellow, while the other best results are colored orange. The "winning" size of the population is colored green. In most models, there was not enough time to test behavior for a population size of 1,000 and therefore the results are shown only up to a population size of 500. In direct methods, the standard deviation itself was chosen as a measure of success, while for methods with partial derivatives logarithmic similarity of the difference between experimental and functional derivatives. In the tables, the methods were abbreviated in the following way:

- SSE - standard deviation evaluator
- MTE - standard deviation evaluator with 2 trees

- IEUND - partial derivation evaluator
- IEUNDMG - partial derivation evaluator with 2 trees

SSE	Veličina populacije				
	50	100	200	500	1000
CIR	$2.97\pm 1.26 \cdot 10^{-20}$	$1.445\pm 6.255 \cdot 10^{-12}$	$3.879\pm 5.192 \cdot 10^{-13}$	$1.967\pm 60.86 \cdot 10^{-12}$	$5.163\pm 4.945 \cdot 10^{-12}$
SPH	$5.051\pm 21.9 \cdot 10^{-16}$	$6.558\pm 27.78 \cdot 10^{-19}$	$6.279\pm 9.278 \cdot 10^{-20}$	$6.481\pm 1.423 \cdot 10^{-18}$	$1.453\pm 9.794 \cdot 10^{-17}$
EL	$4.28\pm 0.177 \cdot 10^{-13}$	$2.709\pm 13.22 \cdot 10^{-11}$	$9.044\pm 2.573 \cdot 10^{-12}$	$6.393\pm 6.281 \cdot 10^{-12}$	$4.159\pm 32.7 \cdot 10^{-12}$
HYP	$1.594\pm 6.950 \cdot 10^{-22}$	$1.697\pm 7.202 \cdot 10^{-26}$	$7.899\pm 5.896 \cdot 10^{-24}$	$6.331\pm 5.161 \cdot 10^{-20}$	$9.140 \pm 2.312 \cdot 10^{-18}$
HO	$2.321\pm 10.11 \cdot 10^{-8}$	$5.467\pm 2.305 \cdot 10^{-9}$	$9.307\pm 7.643 \cdot 10^{-9}$	$3.573\pm 4.778 \cdot 10^{-8}$	$8.036 \pm 3.284 \cdot 10^{-6}$
NLHO	$1.107\pm 4.82 \cdot 10^{-7}$	$3.409\pm 1.248 \cdot 10^{-8}$	$3.307\pm 1.349 \cdot 10^{-9}$	$2.167\pm 3.629 \cdot 10^{-7}$	$3.633 \pm 42.6 \cdot 10^{-6}$

Table 5.8: Training results for SSE method using GP with varying population size

MTE	Veličina populacije			
	50	100	200	500
CIR	$8.232 \pm 35.7 \cdot 10^{-36}$	$5.582 \pm 2.429 \cdot 10^{-36}$	$3.718 \pm 3.484 \cdot 10^{-36}$	$5.224 \pm 6.624 \cdot 10^{-37}$
SPH	$4.983 \pm 2.054 \cdot 10^{-13}$	$9.917 \pm 17.83 \cdot 10^{-14}$	$9.195 \pm 3.659 \cdot 10^{-15}$	$4.434 \pm 9.514 \cdot 10^{-15}$
EL	$2.226\pm 9.513 \cdot 10^{-8}$	$2.492 \pm 1.851 \cdot 10^{-9}$	$6.069 \pm 7.849 \cdot 10^{-9}$	$6.580 \pm 2.617 \cdot 10^{-9}$
HYP	$1.442\pm 6.058 \cdot 10^{-25}$	$8.915 \pm 33.30 \cdot 10^{-26}$	$9.932 \pm 2.674 \cdot 10^{-26}$	$3.534 \pm 35.7 \cdot 10^{-26}$
HO	$2.912\pm 8.897 \cdot 10^{-16}$	$4.991 \pm 6.012 \cdot 10^{-16}$	$4.023 \pm 35.7 \cdot 10^{-17}$	$2.179 \pm 7.984 \cdot 10^{-17}$
NLHO	$2.102\pm 9.161 \cdot 10^{-28}$	$3.570 \pm 5.751 \cdot 10^{-28}$	$9.124 \pm 9.821 \cdot 10^{-29}$	$1.153 \pm 5.699 \cdot 10^{-28}$

Table 5.9: Training results for MTE method using GP with varying population size

IEUND	Veličina populacije			
	50	100	200	500
CIR	0.192 ± 0.264$	0.259 ± 0.272$	0.292 ± 0.018$	0.288 ± 0.163$
SPH	2.151 ± 1.172$	1.730 ± 1.238$	1.557 ± 0.189$	1.603 ± 0.192$
EL	0.525 ± 0.128$	0.499 ± 0.145$	0.450 ± 0.137$	0.467 ± 0.198$
HYP	0.006 ± 0.006$	0.003 ± 0.001$	0.004 ± 0.002$	0.007 ± 0.002$
HO	0.379 ± 0.473$	0.250 ± 0.174$	0.281 ± 0.129$	0.301 ± 0.101$
NLHO	0.565 ± 0.154$	0.451 ± 0.109$	0.478 ± 0.193$	0.500 ± 0.154$

Table 5.10: Training results for IEUND method using GP with varying population size

IEUNDMG	Veličina populacije			
	50	100	200	500
CIR	0.202 ± 0.246$	0.181 ± 0.521$	0.166 ± 0.308$	0.170 ± 0.298$
SPH	2.391 ± 0.430$	1.954 ± 1.299$	1.768 ± 0.129$	1.652 ± 0.138$
EL	0.491 ± 0.204$	0.395 ± 0.101$	0.400 ± 0.121$	0.405 ± 0.173$
HYP	0.006 ± 0.004$	0.004 ± 0.001$	0.005 ± 0.003$	0.007 ± 0.001$
HO	0.256 ± 0.447$	0.244 ± 0.115$	0.270 ± 0.138$	0.269 ± 0.111$
NLHO	0.487 ± 0.267$	0.480 ± 0.101$	0.475 ± 0.124$	0.455 ± 0.167$

Table 5.11: Training results for IEUNDMG method using GP with varying population size

The following tables (Tables 5.12, 5.13, 5.14, 5.15) are shown for various mutation probabilities, with fixed population size:

SSE	Mutation probability (Population size = 200)					
	0.1	0.2	0.3	0.5	0.7	0.9
CIR	$3.879 \pm 5.192 \cdot 10^{-13}$	2.451 ±1.068\cdot 10^{-25}$	0	$3.082 \pm 13.43 \cdot 10^{-25}$	$8.942 \pm 38.97 \cdot 10^{-37}$	0
SPH	$6.279 \pm 9.278 \cdot 10^{-20}$	$3.127 \pm 13.63 \cdot 10^{-14}$	$8.216 \pm 35.68 \cdot 10^{-20}$	$3.399 \pm 14.73 \cdot 10^{-16}$	$1.799 \pm 7.241 \cdot 10^{-21}$	$1.569 \pm 6.805 \cdot 10^{-23}$
EL	$9.044 \pm 2.573 \cdot 10^{-12}$	$2.955 \pm 10.12 \cdot 10^{-12}$	$2.144 \pm 9.306 \cdot 10^{-12}$	$1.779 \pm 6.707 \cdot 10^{-20}$	$5.279 \pm 21.94 \cdot 10^{-16}$	$3.129 \pm 10.17 \cdot 10^{-14}$
HYP	$7.899 \pm 5.896 \cdot 10^{-24}$	$1.526 \pm 6.651 \cdot 10^{-23}$	$7.236 \pm 31.54 \cdot 10^{-24}$	0	0	0
HO	$9.307 \pm 7.643 \cdot 10^{-9}$	$3.267 \pm 14.23 \cdot 10^{-20}$	$9.891 \pm 49.10 \cdot 10^{-19}$	$1.678 \pm 7.295 \cdot 10^{-15}$	$3.157 \pm 13.76 \cdot 10^{-14}$	$3.157 \pm 13.76 \cdot 10^{-14}$
NLHO	$3.307 \pm 1.349 \cdot 10^{-9}$	$1.943 \pm 8.469 \cdot 10^{-21}$	$8.825 \pm 38.47 \cdot 10^{-30}$	$2.293 \pm 9.994 \cdot 10^{-27}$	$2.784 \pm 12.14 \cdot 10^{-36}$	$9.358 \pm 40.79 \cdot 10^{-16}$

Table 5.12: Training results for SSE method using GP with fixed population size and varying mutation probability

MTE	Mutation probability (Population size = 500)					
	0.1	0.2	0.3	0.5	0.7	0.9
CIR	$5.224 \pm 6.624 \cdot 10^{-37}$	2.392 ±7.178\cdot 10^{-35}$	4.304 ±5.012\cdot 10^{-33}$	$1.021 \pm 99.91 \cdot 10^{-31}$	$6.379 \pm 9.519 \cdot 10^{-31}$	$4.054 \pm 2.805 \cdot 10^{-30}$
SPH	$4.434 \pm 9.514 \cdot 10^{-15}$	$1.900 \pm 3.802 \cdot 10^{-27}$	$5.861 \pm 9.736 \cdot 10^{-30}$	$8.634 \pm 8.991 \cdot 10^{-16}$	$1.147 \pm 4.189 \cdot 10^{-17}$	$8.940 \pm 2.390 \cdot 10^{-19}$
EL	$6.580 \pm 2.617 \cdot 10^{-9}$	$1.614 \pm 2.165 \cdot 10^{-17}$	$2.318 \pm 42.42 \cdot 10^{-18}$	$4.546 \pm 5.513 \cdot 10^{-20}$	$8.696 \pm 4.637 \cdot 10^{-21}$	$5.855 \pm 84.89 \cdot 10^{-20}$
HYP	$3.534 \pm 35.7 \cdot 10^{-26}$	0	$4.412 \pm 6.811 \cdot 10^{-30}$	0	$2.126 \pm 3.556 \cdot 10^{-25}$	0
HO	$2.179 \pm 7.984 \cdot 10^{-17}$	$1.084 \pm 3.250 \cdot 10^{-19}$	$5.033 \pm 1.937 \cdot 10^{-18}$	$3.992 \pm 56.10 \cdot 10^{-15}$	$3.580 \pm 3.026 \cdot 10^{-19}$	$2.536 \pm 6.899 \cdot 10^{-18}$
NLHO	$1.153 \pm 5.699 \cdot 10^{-28}$	$7.956 \pm 18.13 \cdot 10^{-28}$	$7.279 \pm 2.969 \cdot 10^{-29}$	$9.225 \pm 7.878 \cdot 10^{-27}$	$1.112 \pm 8.784 \cdot 10^{-27}$	$2.073 \pm 3.992 \cdot 10^{-28}$

Table 5.13: Training results for MTE method using GP with fixed population size and varying mutation probability

Final tables (Tables 5.16, 5.17, 5.18, 5.19) are shown for changing the selection of the algorithm, with fixed population size and mutation probability picked from the previous training iterations.

The Roulette wheel selection method was not so effective against tournament selection in any of the shown models. With the hyperparameters determined on the training methods, the models were to be tested against an unseen set of functions.

IEUND	Mutation probability (Population size = 100)					
	0.1	0.2	0.3	0.5	0.7	0.9
CIR	0.259 ± 0.272	0.189 ± 0.218	0.100 ± 0.151	0.066 ± 0.162	0.063 ± 0.120	0.032 ± 0.053
SPH	1.730 ± 1.238	2.323 ± 0.750	2.050 ± 0.928	1.523 ± 1.312	1.526 ± 1.219	1.349 ± 1.156
EL	0.499 ± 0.145	0.483 ± 0.132	0.487 ± 0.121	0.483 ± 0.117	0.388 ± 0.176	0.377 ± 0.142
HYP	0.003 ± 0.001	0.004 ± 0.003	0.005 ± 0.005	0.003 ± 0.002	0.004 ± 0.008	0.003 ± 0.004
HO	0.250 ± 0.174	0.445 ± 0.696	0.269 ± 0.413	0.108 ± 0.147	0.208 ± 0.297	0.284 ± 0.591
NLHO	0.451 ± 0.109	0.373 ± 0.104	0.421 ± 0.178	0.419 ± 0.219	0.346 ± 0.096	0.372 ± 0.139

Table 5.14: Training results for IEUND method using GP with fixed population size and varying mutation probability

IEUNDMG	Mutation probability (Population size = 100)					
	0.1	0.2	0.3	0.5	0.7	0.9
CIR	0.181 ± 0.521	0.116 ± 0.190	0.138 ± 0.219	0.171 ± 0.218	0.116 ± 0.216	0.987 ± 0.182
SPH	1.954 ± 1.299	1.828 ± 1.122	2.052 ± 1.011	1.639 ± 1.137	1.095 ± 1.080	1.109 ± 1.117
EL	0.395 ± 0.101	0.435 ± 0.196	0.391 ± 0.199	0.409 ± 0.201	0.407 ± 0.201	0.382 ± 0.163
HYP	0.004 ± 0.001	0.004 ± 0.004	0.004 ± 0.003	0.003 ± 0.003	0.004 ± 0.005	0.003 ± 0.004
HO	0.244 ± 0.115	0.180 ± 0.251	0.198 ± 0.247	0.176 ± 0.255	0.132 ± 0.210	0.231 ± 0.393
NLHO	0.480 ± 0.101	0.448 ± 0.268	0.315 ± 0.134	0.314 ± 0.139	0.341 ± 0.149	0.277 ± 0.058

Table 5.15: Training results for IEUNDMG method using GP with fixed population size and varying mutation probability

SSE	Mutation probability (Population size = 200, mutation probability = 0.9)	
	Tournament (size 5)	Roulette wheel
CIR	0	$7.868 \pm 16.13 \cdot 10{-}12$
SPH	$1.569 \pm 6.805 \cdot 10{-}23$	$4.391 \pm 6.111 \cdot 10{-}7$
EL	$3.129 \pm 10.17 \cdot 10{-}14$	$2.184 \pm 4.271 \cdot 10{-}12$
HYP	0	$5.504 \pm 11.04 \cdot 10{-}8$
HO	$3.157 \pm 13.76 \cdot 10{-}14$	$4.067 \pm 10.18 \cdot 10{-}6$
NLHO	$9.358 \pm 40.79 \cdot 10{-}16$	$2.086 \pm 5.362 \cdot 10{-}9$

Table 5.16: Training results for SSE method using GP with fixed population size, fixed mutation probability and varying selection method

MTE	Mutation probability (Population size = 500, mutation probability = 0.2)	
	Tournament (size 5)	Roulette wheel
CIR	$2.392 \pm 7.178 \cdot 10^{-35}$	$2.077 \pm 4.869 \cdot 10^{-8}$
SPH	$1.900 \pm 3.802 \cdot 10^{-27}$	$5.390 \pm 0.01 \cdot 10^{-5}$
EL	$1.614 \pm 2.165 \cdot 10^{-17}$	$4.720 \pm 8.730 \cdot 10^{-5}$
HYP	0	$2.340 \pm 3.973 \cdot 10^{-6}$
HO	$1.084 \pm 3.250 \cdot 10^{-19}$	$2.103 \pm 5.167 \cdot 10^{-5}$
NLHO	$7.956 \pm 18.13 \cdot 10^{-28}$	$2.049 \pm 6.070 \cdot 10^{-7}$

Table 5.17: Training results for MTE method using GP with fixed population size, fixed mutation probability and varying selection method

IEUND	Mutation probability (Population size = 100, mutation probability = 0.9)	
	Tournament (size 5)	Roulette wheel
CIR	0.032 ± 0.053	0.001 ± 0.003
SPH	1.349 ± 1.156	2.309 ± 1.186
EL	0.377 ± 0.142	0.510 ± 0.135
HYP	0.003 ± 0.004	0.009 ± 0.005
HO	0.284 ± 0.591	0.143 ± 0.170
NLHO	0.372 ± 0.139	0.347 ± 0.075

Table 5.18: Training results for IEUND method using GP with fixed population size, fixed mutation probability and varying selection method

IEUNDMG	Mutation probability (Population size = 100, mutation probability = 0.9)	
	Tournament (size 5)	Roulette wheel
CIR	0.987 ± 0.182	$6.744 \pm 3.305 \cdot 10^{-7}$
SPH	1.109 ± 1.117	1.967 ± 1.313
EL	0.382 ± 0.163	0.553 ± 0.157
HYP	0.003 ± 0.004	0.009 ± 0.004
HO	0.231 ± 0.393	0.351 ± 0.600
NLHO	0.277 ± 0.058	0.312 ± 0.049

Table 5.19: Training results for IEUNDMG method using GP with fixed population size, fixed mutation probability and varying selection method

5.3. Model testing

For model testing, 3 functions were chosen based on the results of the multidimensional ellipse as a hyperplane model, with the intention that all models be tested over the same test functions and that in these functions the hyperplane gave satisfactory corresponding derivatives. The table below (5.20) shows the selected test functions.

Equation	No. Dimensions	Abbreviation	Description
$\frac{(x-3)^2}{2.5^2} + \frac{(y-4.5)^2}{3^2} - 1 = 0$	2	CIR-T	Circle with offset from centre
$(x-1)^2 + (y-2)^2 - 36 = 0$	2	EL-T	Ellipse with offset from centre
$y - e^{\frac{-\frac{x^2}{2}}{\sqrt{2 \cdot \pi}}} = 0$	2	G-T	Gauss function

Table 5.20: Equations used as a testing set

Standard deviation methods would always find a satisfactory solution given their optimization function, however for a real insight into the results over the test data it is necessary to manually check which solutions were obtained. The analysis of the solution is described in the next chapter on Pareto fronts. The following table with boxplots shows the results of methods with partial derivations over test data. Since the equation of a circle and an ellipse are those functions that could be linearly separated from multiple trees, counterintuitively the single-tree method showed better results. For a Gaussian function that is not trivially separable, the single-tree method also outperformed the multi-tree method.

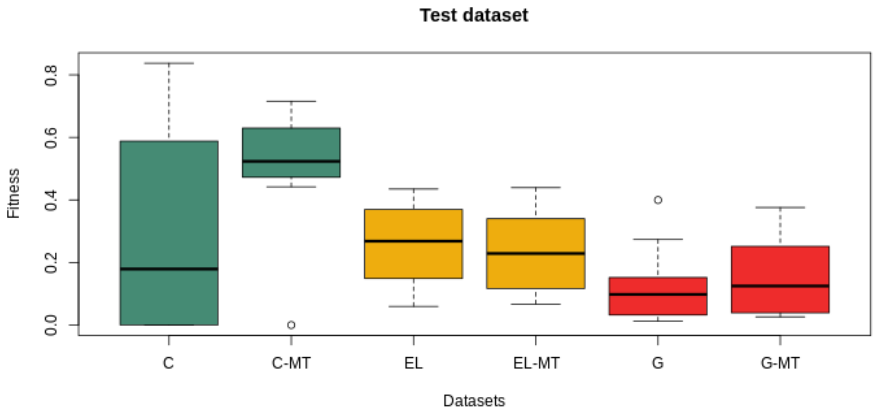

Figure 5.1: Boxplot of testing results for each formula on partial derivation methods using GP. Multiple tree method is denoted with suffix "MT". The formulas are denoted with: C-circle, EL-ellipse and G-Gauss

5.4. Variation of the algorithm

Experiments on equations from the testing data set were also performed with gene expression programming (Figure 5.2) and analytical programming (Figure 5.3). For GEP, a genotype with 2 genes and head sizes of 10, 15 and 20 were conducted with hyperparameters staying the same as the partial derivation model with a single genotype, and keeping the number of evaluations and number of runs identical to the parameters in the testing phase. Results remained stable for each of the head lengths throughout the equations. However, performance against ordinary genetic programming algorithm did not improve.

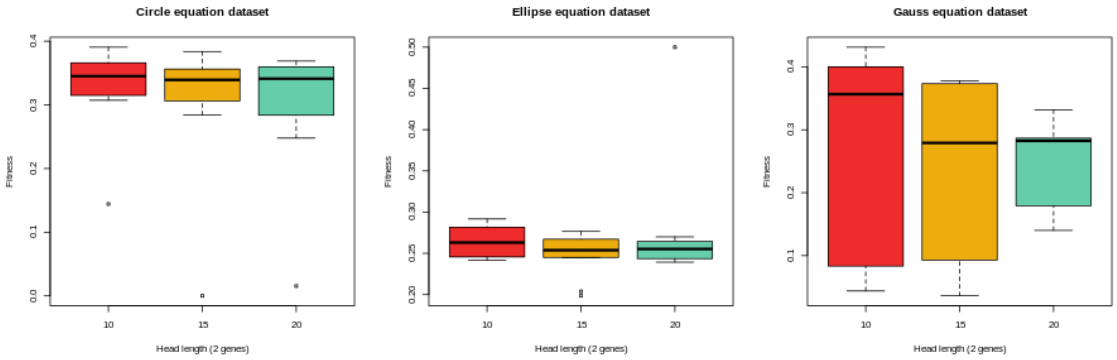

Figure 5.2: Results on testing equations for GEP with head lengths of 10, 15 and 20

Analytical programming method was conducted as well with same parameters. The method showed great improvement in every equation, with the lowest fitness obtained in the Gauss equation which was constant throughout each iteration.

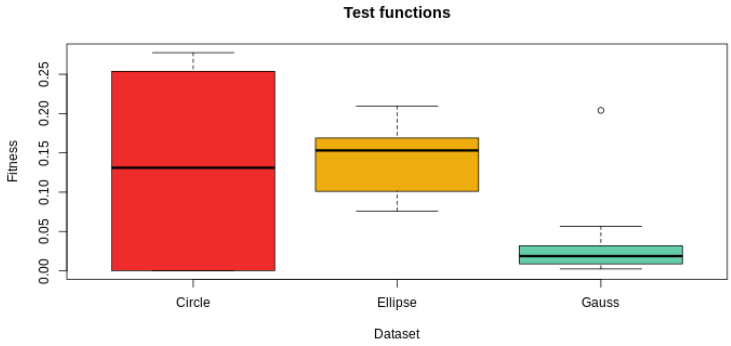

Figure 5.3: Results on testing equations for AP

For better insight, a table with hit rates depending on fitness values were recorded (Table 5.21). Best results once again came from AP, following up by GP and the lowest performance for GEP.

Hit rate (20 runs)	GP			GEP			AP		
	C	EL	G	C	EL	G	C	EL	G
< 0.3	13	12	17	7	20	16	20	20	20
< 0.1	10	5	11	2	0	1	11	7	19
< 0.01	5	1	0	0	0	0	8	0	11

Table 5.21: Hit rates of testing equations for GP, GEP with head length of 20 and AP, counted final solutions below the thresholds for each of the runs

5.5. Tracking of pareto fronts

As part of the implementation, tracking of best results for every equation throughout all of the runs was recorded. For this purpose, pareto fronts with respect to tree sizes and fitness were generated. The shape of pareto front is oftenly parabolic with biggest tree sizes obtaining the lowest fitness and vice versa.

The idea of pareto fronts is getting insight into how the fitness of the solutions decreases if one adds more capacity in the expression tree of the equation. Bigger trees commonly yield a lot of noise to the equation although having best performances. In spite of that, the algorithm can generate a more elegant tree with fewer nodes and a satisfactory performance on the data set.

Pareto fronts on figures (Figure 5.4, 5.5) represent tracked results for ellipse test data set for ordinary genetic programming algorithm and analytical programming algorithm with varying maximum tree depths. Since adding depth increases the tree sizes proportionally, range of depths were between 5 and 10 to analyze performance on each depth.

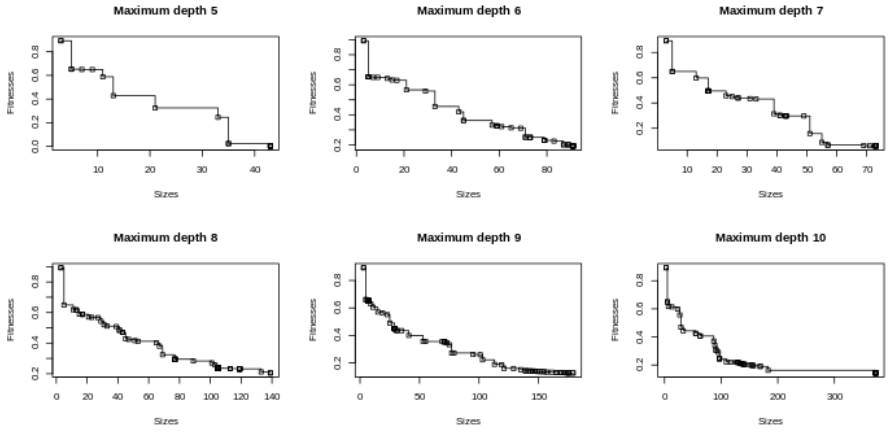

Figure 5.4: Graphs of pareto fronts for genetic programming algorithm for maximum tree depths of 5 through 10

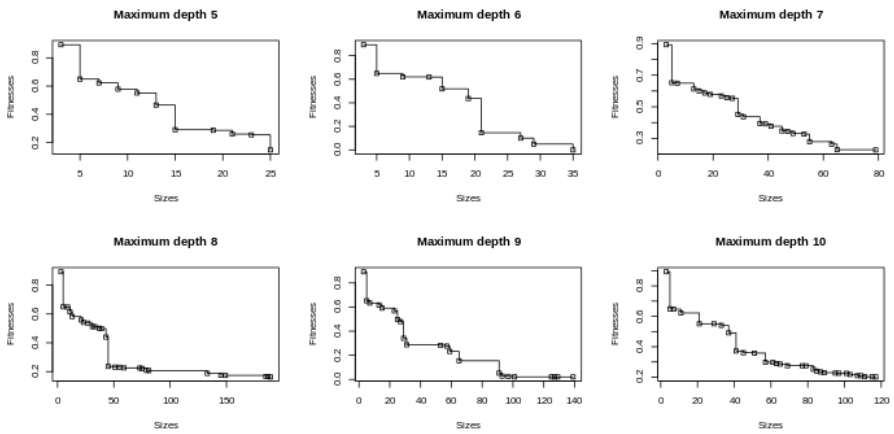

Figure 5.5: Graphs of pareto fronts for analytical programming algorithm for maximum tree depths of 5 through 10

5.6. Visualizations

Although the fitnesses can be low, one can not assume w.h.p. that the final solutions from the algorithm are actually correct on the testing data sets. For that purpose, visualizations were provided on a few of best solutions produced by the algorithms throughout the whole testing process.

Standard deviation model was tested on an additional equation which did not have an appropriate hyperplane derivation estimation. The formula itself represents calculating center of mass coordinates against 2 objects with certain masses:

$$R - \frac{m_1 \cdot r_1 + m_2 \cdot r_2}{m_1 + m_2} = 0$$

Since the formula is 5-dimensional, visualization could not be provided, but some of the best solutions for the problem were:

$$R = (m_2 + m_1) \cdot (r_2 + r_1)$$
$$R = (m_1 + R) \cdot (m_2 + R) \cdot (r_1 + r_2)$$
$$R + \frac{m_1 \cdot m_2}{m_1 \cdot r_1 + m_2 \cdot r_2} = 0$$

Visualizations on 2-dimensional models were provided for some of the partial derivative methods. On figures (Figures 5.6, 5.7) one can see how both genetic and analytical programming solutions maintained the consistency in the derivatives for their best solutions. The final solutions are not exact as the ground truth, however the form and shape of the underlying equation was preserved. Since the heuristic itself was based on keeping the ratio of derivatives as similar as possible, the results obtained were as expected.

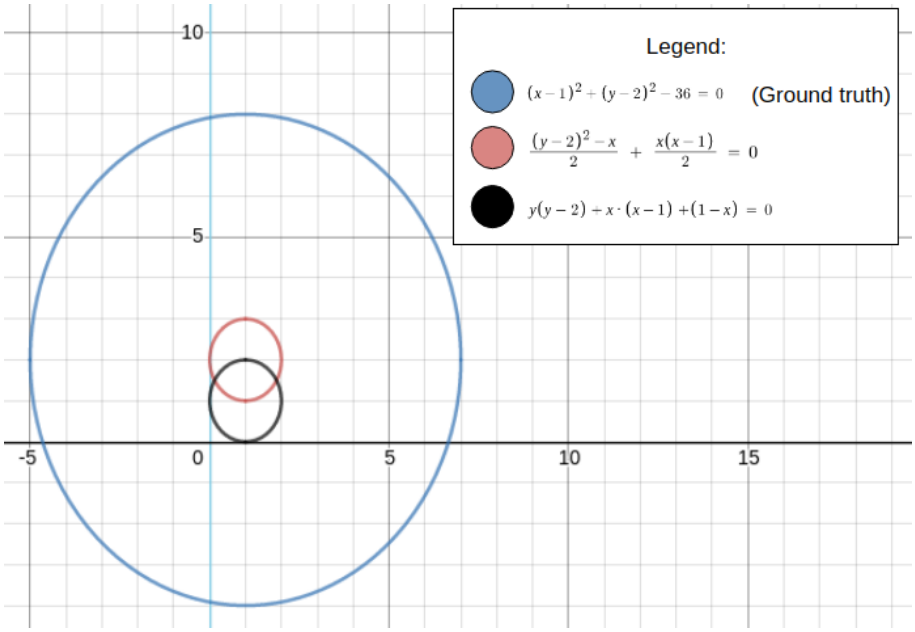

Figure 5.6: Best solutions for circle testing equations with GP plotted against the ground truth

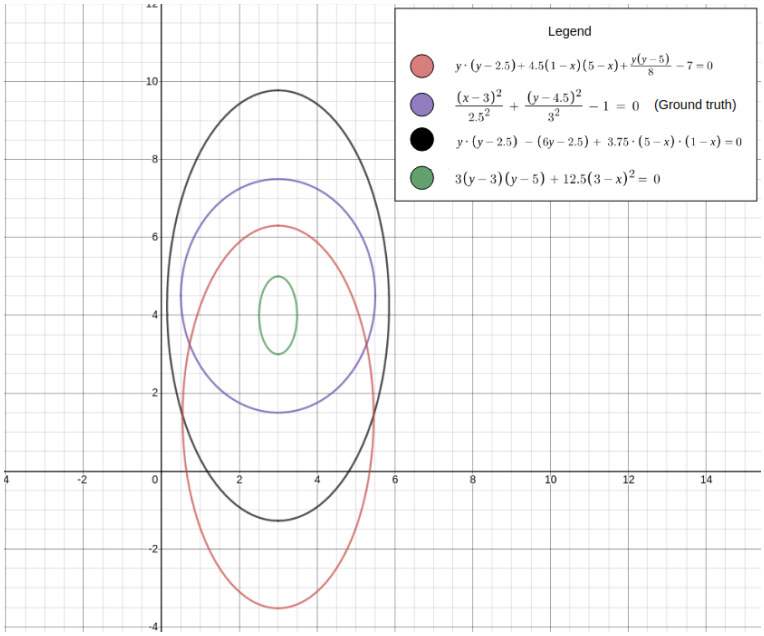

Figure 5.7: Best solutions for ellipse testing equations with AP plotted against the ground truth

On the other figure (Figure 5.8), Gauss function results with genetic programming were visualized. This time the results were not as great, with derivatives getting chaotic in the right quadrants of the coordinate system. Possible explanation of this is having a lot of freedom with sine and exponential functions in modeling of the solutions.

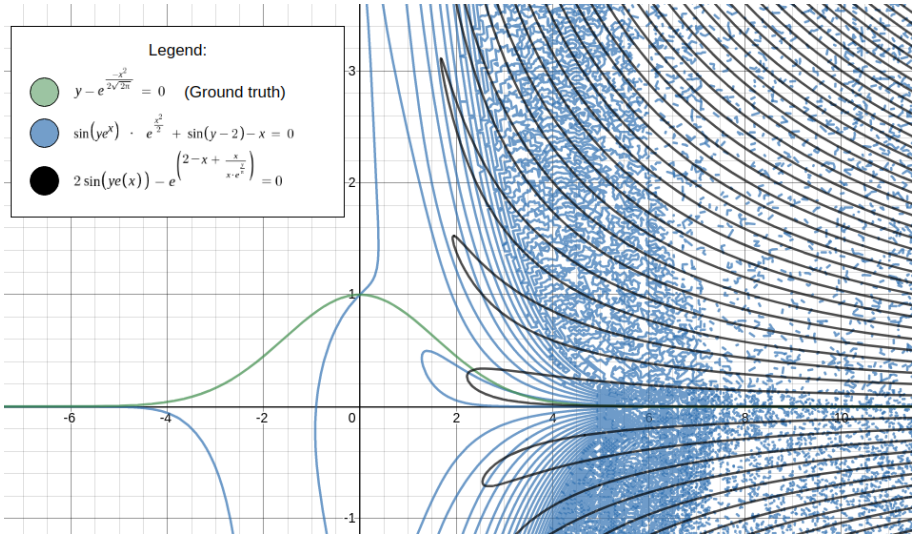

Figure 5.8: Best solutions for Gauss testing equations with GP plotted against the ground truth

6. Conclusion

All things considered, the thesis managed to reproduce at least similar results as the papers who addressed the problem with same principles. As for the implemented goal functions for the genetic programming algorithm, they were all successfully implemented and showed a degree of search for better solutions at a satisfactory level. It should be emphasized that despite the advancement of the algorithm, the obtained solutions do not necessarily correspond to the functions that describe the underlying system itself. It is a legitimate case that by increasing the depth of the tree, unnecessary noise can be introduced, and by using the Pareto fronts of the final solutions, this problem was somewhat addressed. Final equation plots in the results chapter show that although majority of the functions do not correspond to the actual solutions, the form and shape which was implied from the calculated derivatives was preserved.

Since most of the research was focused on the determination of derivations in the method of partial derivatives, expectations were partially met. For conical and spherical functions, the determination of hyperplanes using multidimensional conical shapes worked with negligible error, while for all other sets of functions, even the simplest ones, the method showed weaknesses. In the hope that the error between the derivatives of the real functions and the hyperplanes will be smaller, models for the 3rd degree polynomial and the ordinary plane were made, but even these methods did not contribute to significant results. Given this, the implementation of the research part is made in a modular way for free exploration of possible shapes that could be favorable for other groups of functions of non-spherical shape.

In addition to the least squares method, for further work on the thesis, it is possible to use some other mathematical method that could potentially obtain more accurate derivations to run evolutionary algorithms over them. Since the algorithms themselves have shown that they cope well with spherical functions that are faithfully described by their derivatives, there is no reason to think why they would not behave in the same way with functions whose shape is not spherical.

BIBLIOGRAPHY

Josh Bongard i Hod Lipson. Automated reverse engineering of nonlinear dynamical systems. 104(24):9943–9948, 2007.

Fred L. Bookstein. Fitting conic sections to scattered data. *Computer Graphics and Image Processing*, 9(1):56 – 71, 1979.

Kris Brabanter, Jos De Brabanter, Bart De Moor, i I. Gijbels. Derivative estimation with local polynomial fitting. *The Journal of Machine Learning Research*, 14:281–301, 2013.

Steven L. Brunton, Joshua L. Proctor, i J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. 113(15):3932–3937, 2016.

Candida Ferreira. Gene expression programming: A new adaptive algorithm for solving problems. 2001.

Sébastien Gaucel, Maarten Keijzer, Evelyne Lutton, i Alberto Tonda. Learning dynamical systems using standard symbolic regression. U Miguel Nicolau, Krzysztof Krawiec, Malcolm I. Heywood, Mauro Castelli, Pablo García-Sánchez, Juan J. Merelo, Victor M. Rivas Santos, i Kevin Sim, urednici, *Genetic Programming*, stranice 25–36. Springer Berlin Heidelberg, 2014.

Zuzana Kominkova Oplatkova, Roman Senkerik, Ivan Zelinka, i Michal Pluhacek. Analytic programming in the task of evolutionary synthesis of a controller for high order oscillations stabilization of discrete chaotic systems. *Computers Mathematics with Applications*, 66:177–189, 2013.

John R. Koza i Riccardo Poli. *Genetic Programming*, stranice 127–164. Springer US, 2005.

Michael Schmidt i Hod Lipson. Distilling free-form natural laws from experimental data. 324(5923):81–85, 2009.

Michael Schmidt i Hod Lipson. *Symbolic Regression of Implicit Equations*, stranice 73–85. Springer US, 2010.

Ivan Selecnick. Least squares with examples in signal processing. 2013.

Moshe Shpitalni i H. Lipson. Classification of sketch strokes and corner detection using conic sections and adaptive clustering. *ASME Journal of Mechanical Design*, 119, 2001.

Sažetak

Simbolička regresija je metoda kojom se iz skupa točaka u prostoru pokušava inferirati jednadžba sustava koji ga opisuje. S obzirom da su eksplicitne formule uglavnom bile predmet istraživanja tom metodom tokom godina, ova teza se odlučila baviti simboličkom regresijom implicitnih funkcija. Implicitne funkcije su zbog svojih određenih svojstava puno ekspresivnije te su teže za inferirati te su napravljene različite heuristike za dolazak do zadovoljavajućih rješenja. Jednostavnija metoda obuhvaćala je standardnu devijaciju kandidata rješenja problema, te se minimiziranjem raspršenosti proizvoljni algoritam navodio na konačno rješenje regresije. Druga metoda iskoristila je usporedbu parcijalnih derivacija između kandidata rješenja te eksperimentalno dobivenih derivacija. U dvodimenzionalnim sustavima računanje derivacije je jednostavno dok je za višedimenzionalne sustave potrebno odrediti ravninu u lokalnoj blizini neke središnje točke za dobivanje ispravne derivacije. Za određivanje ravnina koristila se metoda najmanjih kvadrata, te je u konačnici za određeni podskup funkcija radio vrlo dobro, dok je nad drugima potrebno ipak više istraživanja i pokušaja za dobivanje ispravnijih podataka. Heuristike su upogonjene algoritmima genetskog programiranja, genskog ekspresijskog programiranja (GEP), analitičkog programiranja (AP) su za izvjesni podskup funkcija pokazale iznimno dobre rezultate.

Ključne riječi: Simbolička regresija, implicitne funkcije, genetsko programiranje, gensko ekspresijsko programiranje, metoda najmanjih kvadrata

Implicit function symbolic regression system

Abstract

Symbolic regression is a method by which an equation of the system describing it is inferred from a set of points in space. Since explicit formulas have been the subject of research by this method over the years, this thesis decided to deal with the symbolic regression of implicit functions. Implicit functions are much more expressive due to their certain properties and are harder to infer than explicit ones. Therefore, different heuristics have been made to arrive at satisfactory solutions. A simpler method involved minimizing the standard deviation of evaluations for candidate solutions (equations) to the problem. By lowering the scattering for all the points in the set, the arbitrary algorithm was guided to the final solution of the regression. The second method used the comparison of partial derivations between solution equations and experimentally obtained derivatives. In two-dimensional systems, the derivation calculation is simple, while for multidimensional systems it is necessary to determine the plane in the local vicinity of some central point to obtain the correct derivation. The least squares method was used to determine the planes, and in the end it worked very well for a certain subset of functions which were conic or spherical, while more research and attempts are still needed on others to obtain more accurate data. Heuristics were driven by genetic programming algorithms, gene expression programming (GEP) and analytical programming (AP) which have shown extremely good results for a certain subset of functions.

Keywords: Symbolic regression, implicit functions, genetic programming, gene expression programming, least squares method