Universiteit Leiden

Master Computer Science

Exhaust Gas Temperature modelling, a symbolic regression approach.

Name:	Juan Enrique de Santiago Rojo
Student ID:	s2302470
Date:	29th of June 2020
1st supervisor:	Prof.Dr. T.H.W. Baeck
2nd supervisor:	Dr. Bas van Stein

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

In this master's thesis research, a modelling technique of the exhaust gas temperature (EGT) of an aircraft, by means of symbolic regression is proposed. As an indicator of a turbofan engine's health, the clear understanding and possible behaviour of this parameter can help with reducing costs and maintenance, for engine manufactures and airlines. After analyzing the latest developments in the field of symbolic regression, a set of experiments are performed on real life continuous engine operating data (CEOD), using the free open source Matlab toolbox, GPTIPS. It is proven to be an accurate technique not only for retrieving an algebraic expression for the selected variable, but also, for extracting important parameters that are measured during an engine's operation and relate to the EGT.

Acknowledgements

I will first like to thank Marios Kefalas, without your effort and patience, nothing would have been possible. Thanks also to Prof Dr. Thomas Baeck and Dr. Bas van Stein for your thorough supervision and work. This is to my family, for giving me everything with this amazing opportunity. Gracias papá y mamá. A Borja, Nico, Ángela y Enrique. Of course, thank you Claudia, for everything, you know better.

Contents

List of Figures

List of Tables

List of Equations

1 Introduction

Characterization, modelling and prediction of data, are three fundamental problems in different fields of study such as economics, engineering or social sciences. Characterization aims to find the fundamental properties of the data. By modelling the data, an underlying mathematical description is to be found, in order to use it as the model that will fulfil some basic assumptions. Finally, the prediction's objective is to develop predictive models that help determine, with the highest accuracy, future values or other feasible solutions [41].

In order to tackle such problems, the modelling process has fundamentally been based on linear modelling such as ARIMA [5], in addition to non linear statistical models. On the other hand, other computational intelligent algorithms are based on non parametric and heuristic systems. An example are those based on neural networks or hybrid modelling systems, which provide good handling of missing data. But in many cases, these are considered "black box" models.

As a feasible alternative, symbolic regression by means of genetic programming, will be studied and analysed in this project. Symbolic regression searches in the space of mathematical equations the one which best fits the corresponding data, as a binary tree structure. In each of these trees, the mathematical operators correspond to the interior nodes and variables and parameters correspond to the leaves. The algorithmic process, is an exploration of the possible equations, by using genetic operators, such as crossover, mutation and elitism, to name a few [23].

Even though symbolic regression was originally proposed as a technique for empirically finding or determining the existing, and unknown, nonlinear relationships between the input and the output data [23]; it has been recently used for linear analysis with great success

[40].

Some of the reasons why symbolic regression had such a success are:

- Ability to discover non-linear relationships from the data, which could not be properly represented by conventional parametric techniques. The latter would be done without a-priory mathematical expressions which described the non-linearity.
- The ability to point out irrelevant variables through the execution of the algorithm.
- The ability to control the complexity of the functions found, due to the availability to set up a maximum depth of the syntactic trees that represent these functions.

A disadvantage that symbolic regression has is that the genetic algorithm might not guaranty the findings of better functions, that might have already been found by the traditional models which are commonly used e.g. for time series analysis, ARIMA models. Even though there might be a nonlinear dynamic. This can happen because the well known mathematical models which are commonly used in statistics and econometrics [24] are not taken into consideration in the search space. Also, because representing such complex models on a tree structure form derives a very complex functional structure, making it impossible for the algorithm to produce it.

Nowadays, with the growing generation of large amounts of data in the aviation industry, many applications have been developed and improved, lots of which are focused on engine health monitoring (EHM), as it is important for engine manufacturers and airlines. With the purpose of improving the availability and operation of the engines, EHM monitors the state of an engine or a wider set of engines by using past events and operational data. By optimizing maintenance operations, not only the safety is improved but also costs can be reduced, for both the manufacturers and airline companies. The aim of these data-driven solutions is to first avoid failures by preventing the possible faults on the engine, and secondly, to extend the life span of systems, making engines safer to use.

Exhaust gas temperatures has been proven to be a good indicator of the health of an aircraft engine [39]. We propose using symbolic regression in order to retrieve the physical law, or algebraic expression, that describes this parameter in an engine, from the other measured parameters present in the operation of the engine. With this, one could build a feasible model that can help with the prevention of faults in the engine.

The rest of this thesis is organized as follows. A literature study on the latest developments on symbolic regressions by means of genetic programming and an analysis of the available tools for performing it is done in Section 2. Section 3, describes the methodology used for developing the real world data experiments developed in this thesis. In section 4, we present our experimental results. Section 5, provides some concluding remarks of the research. Finally, in Section 6, a future approach for continuing this research is proposed.

2 Literature Study

Symbolic regression is a method for finding a suitable algebraic expression that describes the observed data [23]. In symbolic regression, no a-priory assumptions on a possible form of the expression is made. Meanwhile, conventional regression models, like linear regression, are based on the assumption that the relationship between the dependent variables and the independent variables is linear. Therefore, the parameters are optimized by providing a starting point to the algorithm. Nevertheless, there is a mathematical expression space provided in symbolic regression. Which contains candidate function, e.g., mathematical operators, analytic functions, state variables, constants. In order to find the most appropriate solution, symbolic regression then searches through this space of functions. One can say that not only the model structures are optimized in symbolic regression, but also, the model parameters [23].

A further assumption is made in symbolic regression. Which is that the relationship between the input and the output data can be described by an algebraic expression [23]. As the aim of finding an algebraic expression that describes these relationships is sophisticated enough, there is a need for the use of a powerful search method. Genetic programming can provide the tools for doing so. With a fast evolution on the field of genetic programming [23], new ideas and methodologies have made it a tool that could out-perform other traditional techniques when solving modelling and identification problems, such us autoregressive moving-average (ARMA) models [40].

2.1 Genetic Programming

There have been many developments for different methods based on what is called Evolutionary Computation. Throughout these methodologies, it is possible to find an analytical expression that describes the exact form of the solution to the problem.

Evolutionary computation corresponds to a stochastic optimization method in which searching rules are based on the principle of natural selection. This biologically inspired approach, was first introduced by Holland [18]. Therefore, evolutionary computation consists of a process based on the evolution of a number of possible solutions to the problem, which are modified, or evolved, through what are called genetic operators. These include crossover, mutation and replication [23].

Among some of the algorithms developed which are based on evolutionary computation theory, genetic programming was first introduced by Koza [23] as a specific implementation of genetic algorithms (GAs) [15]. The idea is to evolve the solution of a given problem following Darwin's theory of evolution and to find the fittest solution after a number of generations. Instead of using strings of binary digits to represent chromosomes as in GA, solutions in genetic programming are represented as tree-structured chromosomes, form by nodes and terminals. As an example, Figure 1 represents in a basic tree the simple expression:

$$(cos(x_1) + (x_2 * 0.5)) \tag{1}$$

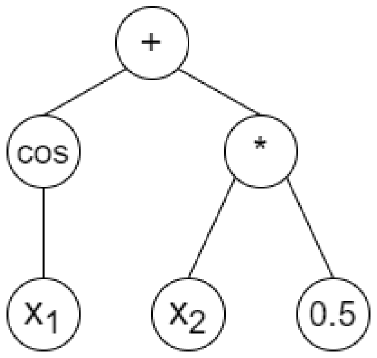


Figure 1: Basic GP tree representation

Genetic programming, therefore has two main differences from a pure genetic algorithm. The first one is that genetic programming codifies solutions with the use of tree structures, rather than a string of integers. Secondly, genetic programming allows the best solutions of a certain generation to move onward the next generation directly. This is commonly known as elitism [3]. This sort of technique has been successfully used in given problems of programming code automatic generation [30], trigonometric identities recovery [31], bots control [13] and, for our purpose, symbolic regression [23].

As mentioned, the genetic programming algorithm defined by Koza, performs in a similar way as the GA. In genetic programming, genetic operators (crossover and mutation) are applied on tree structures (individuals) that, in case of symbolic regression, represent algebraic expressions. This is in contrast to GA, where the genetic operators are applied on - usually - binary strings of 0 and 1.

Let us describe the essential components of a genetic programming algorithm. Chromosomes are represented by tree structure, and we must distinguish between terminals and operators. Terminals are variables or values, that the operator can process. These include input variables like x_n or coefficients to be used. The operators correspond to all those functions that can be applied to terminal nodes. These could be the fundamental arithmetic operators, $f = \{ +, -, *, /, \exp, \log, \sin, \cos... \}$. An individual (tree), is the hierarchic combination of operators and terminals, which is equivalent to an algebraic expression. When generating these tree structures, their computational complexity will be dependent on the used method for building them (hybrid, declarative, procedural, mathematical). A more detailed description of these tree building methodologies can be found in [35].

A genetic programming process can be defined as follows:

Let n denote the number of individuals, g the generation of the evolution, P_g the population in generation g, and $f(S_i)$the fitness function, which evaluates an individual. The following steps are to be followed in genetic programming algorithms:

- Initialization: a large number of trees will be generated, where each tree represents a possible solution. In this initial generation (g=0), an initial population (P_0) is created randomly, these initial terminals (constants and variables) and functions form the n individuals represented by tree structures of different structures and sizes. The initialization process ends once the number of user specified individuals is reached.
- Fitness Evaluation: for each individual S_i, where i=$\{1,...,n\}$, the fitness function is evaluated, $f(S_i)$. This evaluation is done by comparing the true values from the dataset with the function output. Some error metrics used are root mean square error (RMSE) and mean squared error (MSE).
- Selection: given the fitness score of an individual, some are selected for reproduction. As it is normally stated, the higher the fitness score, the larger the probability of selection. In order to achieve the optimal solution, this selection criteria follows the "survival of

the fittest" rule, which states that good features are more likely to be inherited by the next generation.

- Reproduction: the current population (P_g) is evolved by applying the genetic operators, here described, to the selected individuals. After this, a new population (P_g^*) is created.

  1. Crossover: this genetic operator takes two selected individuals as parents in order to breed their offsprings. Then, it randomly takes a sub-tree from parent (a) and substitutes another random sub-tree in parent (b) with the one from (a).
  2. Mutation: this operator just takes one parent structure and randomly substitutes a sub-tree with a randomly generated tree structure.
  3. Elitism: selects the best individuals from the population and copies them without modification into the next generation. Finally, the new offspring population (P_g^*) replaces actual population (P_g), until the population size reaches the specified number.

- Termination: the process stops when an individual with the required fitness value is found, when the maximum number of iterations (generations) is reached or when another pre-defined criterion is met, such as wall-clock time.

We illustrate in Figure 2 the process by which a solution of the symbolic regression problem is obtained by means of genetic programming.

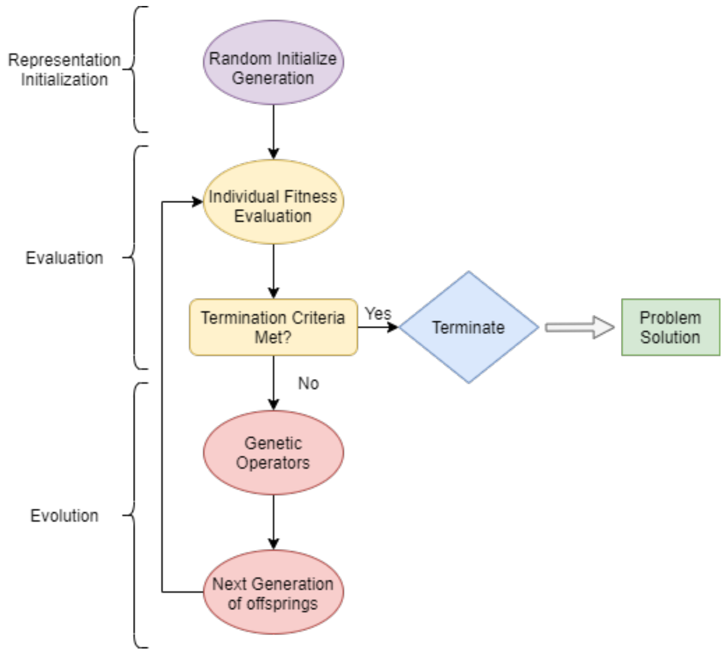

Representation Initialization
Random Initialize Generation
Evaluation
Individual Fitness Evaluation
Termination Criteria Met?
Yes
Terminate
Problem Solution
No
Evolution
Genetic Operators
Next Generation of offsprings

Figure 2: Genetic programming algorithm flowchart

2.2 Advances in Symbolic Regression

Evidently, since its introduction, there has been a great effort on improving the performance of the original genetic programming algorithm for solving symbolic regression problems. There were three major issues to be solved. First, as the performance of the next generations of individuals created during the algorithm was not guaranteed to be better than their parents, a non-deterministic optimization problem was raised. Second, it was difficult to find the proper constants due to the fact that their creation, during the initial input set or through mutations during the genetic programming algorithm, is done randomly; there was no effective way to obtain the best fitting coefficients. The third issue was that, as the fitness of an individual is evaluated based on its own complete structure, the fact of having a good feature in a sub-branch does not mean that its performance as an individual will be better. Which could be translated into loosing the good equation components in the upcoming generations. Thus, there is a limitation on preserving good components of the algebraic expression due to this fitness evaluation methodology.

Here, we describe the advances in the genetic programming algorithm that have helped solving the aforementioned issues. The evaluation of the syntax performance is conventionally used in genetic programming for solving symbolic regression problems, but the geometric semantic genetic programming approach [6], evaluates the semantic performance, i.e., the program behaviour. It represents each program in a high dimensional semantic space, so the fitness evaluation can be done by just measuring the distance between the target point in the problem and the program. It will have a better performance the closer the program is to the target point. It states that optimizing the program semantics instead of the syntax, frees the symbolic regression solutions from specific forms, making it more efficient.

In order to properly find constants or parameters, the evolutionary polynomial regression [17] hybridizes the evolutionary optimization scheme used in genetic programming with the parameter estimation technique that is being used in conventional numerical regression methods. The way of doing this is by first exploring the function space with the help of a genetic algorithm, and then performing a least squares linear regression to optimize the parameters that can be found in each mathematical building block. Even though the computational cost for this hybrid method could be higher, it improves the stochastic symbolic regression by means of genetic programming approach by making it a more deterministic approach.

As another example of a hybrid approach, genetic programming-based relevance vector machine (RVM) [29] combines Kaizen programming [9] and an RVM algorithm to solve symbolic regression problems. This method provides the advantages of both Bayesian kernel methodologies and evolutionary computation, in where the former extracts basis functions to be built and solved within the space of the optimal solution. The latter explores the parameter space. Fundamentally, Kaizen programming is a collaborative version of a genetic programming algorithm. Here, individuals work together in order to solve the problem, so the solution for this process is nothing but a linear combination of genetic programming individual. Which means that, instead of evaluating an individual program as a complete solution, the fitness evaluation process is based on a group of individual partial solutions. An RVM algorithm extracts the important functions without prior knowledge from the given set.

With a more sophisticated design than the conventional genetic programming algorithm Cartesian genetic programming [7] represents the computer program as a directed acyclic graph. It can be visualized as a grid of nodes in two dimensions. In order to determine the mathematical function that each node performs, they own a set of genes, which as a whole, forms the program's genotype. After decoding the genotype, it expresses the phenotype. This genotype-phenotype mapping feature makes Cartesian genetic programming a closer algorithm to the real natural evolutionary process. In addition, a new crossover technique for Cartesian genetic programming is developed and introduced, which is based on directed graphs as a representation to replace the tree structures originally introduced by Koza.

With regards to the theory that allows us to understand evolutionary algorithms convergence, Schmitt et al. [32] proved asymptotic convergence to a global optima, for a set of genetic programming systems, where the population was set as a fixed number of individuals, each one of an random size. On the other hand, when it comes to the way genetic programming performs the solution search on the search space, Daida and Hills [8] identify how the tree structure has a mayor influence on how the algorithm performs this search.

The multiple regression genetic programming [2] approach helps in improving the program evaluation process. In essence, it performs multiple regressions on the solution functions' sub-expressions. Here, the mathematical expression tree is decoupled into other sub-trees so that the fitness of a possible solution is then evaluated based on the best linear combination of the decoupled sub-trees structures. This is done instead of the individual fitness evaluation. In order to solve the linear regression problem, a least angle regression algorithm is used.

While the standard representation of a genetic programming algorithm is based on the evolution of one single tree structure, multigene genetic programming [34] builds each individual by considering a number of genes, with a tree structure form, that are weighted by linear combination. There is indeed a limitation on the number of genes an individual can have, while it is not required for it to have the maximum. This limitation can be set by the user. The changes performed on the original genetic programming algorithm include the initialization process, where an individual is initially generated with a random number of genes, between one and the maximum defined by the user. More over, in order to maximize the diversity within the population at the beginning of each run, there is a supervision on the duplicated genes, so that they do not appear in the new generated individuals. When applying the genetic operators, crossover is based on the two-point high-level crossover operator [37]. Meanwhile mutation is performed in the same way as in the ordinary genetic programming algorithm. The predicted output variable is a combination of the weighted output of the multigene individual's genes plus a bias term. These weights are determined automatically by a least squares procedure, for each individual. Mathematically, a multigene regression model can be expressed as:

$$\widehat{y} = d_0 + d_1 * Tree_1 + ... + d_n * Tree_1 \tag{2}$$

where d_0 represents the bias term, n is the number of genes which constitutes a certain individual and d_1, ..., d_n are the gene weights. Figure 3 represents an example of a multigene genetic programming model that can be mathematically expressed by Equation 3

$$d_0 + d_1 * (cos(x_1) + (x_2 * 0.5)) + d_2 * (x_1/x_2 + 2) \tag{3}$$

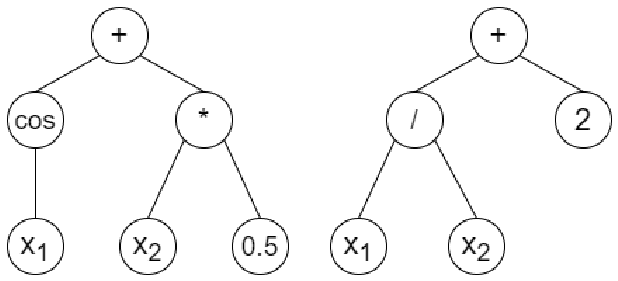


Figure 3: Multigene genetic programming model example

A different approach, not related to genetic programming is introduced in [21]. Based on the artificial bee colony algorithm that simulates the foraging behaviour of honey bee swarms, which was introduced by Karaboga in [20]. This new approach, to solving symbolic regression problems, allows to evolve expressions and constants in order to form algebraic expressions automatically. This was tested in a large set of symbolic regression benchmark problems, the author concluded a significant performance [21] after comparing it with the genetic programming approach.

2.3 Genetic Programming and Symbolic Regression Applications

There have been many fields of study in where genetic programming and symbolic regression have been used. Genetic programming was able to outperform other traditional approaches in solving identification and modelling problems. Providing solutions to classification problems [42], telecommunications problems [11] and manufacture process modelling [12]. Furthermore, [40] compares different artificial intelligence (AI) techniques to the use of genetic programming, noting that genetic programming could have a significant improvement in terms of accuracy. Genetic programming has also been used for analysing and obtaining volatility models, by means of forecasting index volatility such as S&P500 [1].

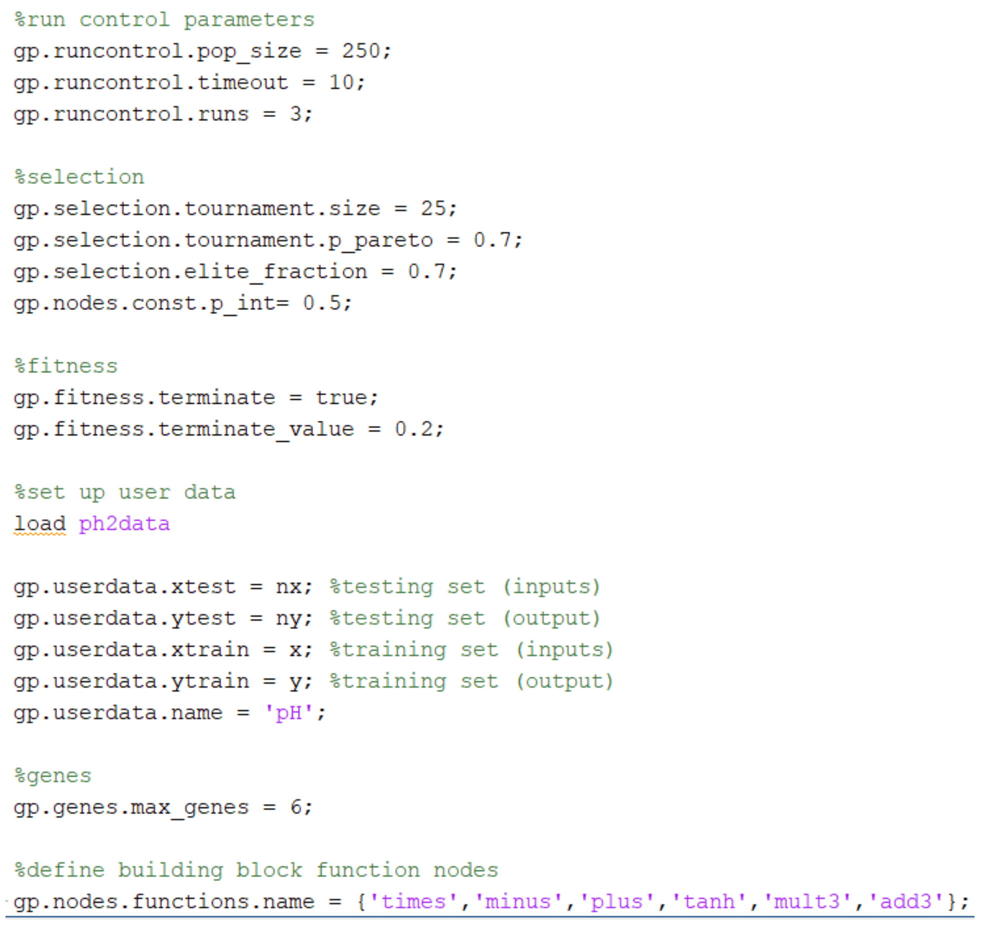
Many publications using symbolic regression based on genetic programming algorithms have been publish since the introduction to the topic was made by Koza, as introduced in section 1. One that must be mentioned, due to its industrial applications, is Bongard and Lipson [4]. They generated symbolic equations for nonlinear coupled dynamical systems in the fields of mechanics, systems biology and ecology. They also noted the differences between symbolic models and numerical in terms of complexity, making the firsts easier to interpret. Arkov et al.[25] applied symbolic regression by means of genetic programming algorithms to identify nonlinear governing equations of wind turbines. Symbolic regression has also been used when predicting economic time series [10], in multi-objective analysis [22] and in diverse data analysis applications [27]. Furthermore, this technique has also been combined with others to form hybrid approaches, in order to obtain better model. A remark is [26], in where symbolic regression is combined with neural networks.

2.4 Symbolic Regression Tools

Symbolic regression by means of genetic programming has been developed for many years. Therefore, the amount of software running genetic programming has increased through the necessity of solving not only research problems, but also business problems. We present some of the most used tools, analysing their features and characteristics.

2.4.1 GPTIPS

We will first introduce the GPTIPS tool [34], which will be later used to experiment on a real world data-set. This open-source tool is freely available for Matlab. It is based on the previously explained multi-gene genetic programming algorithm and generates explicit predictive models for symbolic regression equations, by means of analysing the relationship between the input and output data-set. By using multi-gene symbolic regression, it improves the functionality and the accuracy, and even reduces the complexity of the traditional genetic programming algorithm. The initial population are multi-tree solutions constructed by a vector of randomly generated trees. Each solution is evaluated and, based on this evaluation, part of the population is selected to be the parents of the upcoming population. This selection process is done by means of pareto tournament. Once the solutions are selected, they are evolved by crossover and mutation repeatedly, until the termination criteria is satisfied. Some of the criteria that can be used is the maximum number of generations, best model fitness or a maximum run-time. Figure 4 shows a configuration setup. Refer to the User Guide Manual [33] of the GPTIPS tool for a more detailed description of this tool.



```
%run control parameters
gp.runcontrol.pop_size = 250;
gp.runcontrol.timeout = 10;
gp.runcontrol.runs = 3;

%selection
gp.selection.tournament.size = 25;
gp.selection.tournament.p_pareto = 0.7;
gp.selection.elite_fraction = 0.7;
gp.nodes.const.p_int= 0.5;

%fitness
gp.fitness.terminate = true;
gp.fitness.terminate_value = 0.2;

%set up user data
load ph2data

gp.userdata.xtest = nx; %testing set (inputs)
gp.userdata.ytest = ny; %testing set (output)
gp.userdata.xtrain = x; %training set (inputs)
gp.userdata.ytrain = y; %training set (output)
gp.userdata.name = 'pH';

%genes
gp.genes.max_genes = 6;

%define building block function nodes
gp.nodes.functions.name = {'times','minus','plus','tanh','mult3','add3'};
```

Figure 4: GPTIPS Configuration file (Matlab)

GPTIPS model solutions are built out of the populations solutions which contain the randomly generated genes (trees). These are then combined linearly using different weights per tree. These weights are calculated by minimizing the error of fitting the model to the training data-set. In order to visualize the solutions, GPTIPS generates a model report containing the properties and the configuration parameters. In this report, the input variables used for building the best model, the model complexity and the number of trees are displayed for the user to analyse. Moreover, a symbolic form of the model is displayed and can later be saved.

Performance metrics of the best model (RMSE, MSE, MAE, R^2) are given for both the training and test data. This is one of the reasons for choosing this tool. It allows the user to explore the results in a quick way, so that not only one can check if all the necessary settings where properly arranged, but also, graphics, metrics and variables specifications of the built model are shown. Making the post-run analysis simpler.

There are, indeed, some features and parameter that have to be configured by the user before running GPTIPS, thus, it is important to be familiar not only with the tool itself, but also the programming language of Matlab. Some of the features that can be configured by the user are: population size, number of generations, depth of the tree structures and the maximum number of trees. Furthermore, the selection procedure of the algorithm can be based on solution complexity, Pareto front or goodness of fit. Complexity can be measured by the expressional complexity, determined by the sum of the number of nodes in all sub-trees of a given tree, or the total number of nodes per tree. It is important to note that by default, this tool uses root mean squared error (RMSE) in order to measure the fitness of the model, but the user can also define his own fitness function.

Some of the disadvantages that this tool shows are that Matlab is not a free software, therefore the user must purchase a license. Also when it comes to defining the initial population, the user can not set the initial seed and it might occur that the trees which form a possible solution happen to be collinear. Which means, they are not independent and can not be added up to each other.

2.4.2 Karoo GP

This genetic programming suite is written in Python. It provides CPU and GPU support by using the TensorFlow library. As a highly scalable tool, it can work with real world vectorized data. Introduced by Staats [38], it enables the user, without deep programming knowledge, to follow each of the steps of the algorithm until it gets a solution. This tool includes a desktop application and a scriptable server application.

Karoo GP generates the first population of trees following the genetic programming procedures, it then evaluates and selects the best fitting trees and evolves them into the upcoming generation. This procedure continues until the maximum number of generations is reached or, otherwise, a particular solution for the problem is obtained.

In the desktop Karoo GP solution, the user can define some of the running parameters such as tournament size, minimum number of nodes and the balance of the genetic operators. The server application is also provided with this advantage. Some of the features that characterize this tool are the user-defined minimum number of nodes, which essentially defines the least number of functions and terminals allowed in a given tree. This distinguishing feature was introduced in order to shape the evolutionary process, so it searches a particular solution space by establishing lower bounds on the evolutionary landscape. Another relevant feature is the availability of saving, modifying and reloading an evolved generation, which means that the final population is automatically saved to disk so that the user can later access it. These features allows launching the Karoo GP algorithm with a user specified seed for the population rather than with a randomly generated one. Finally, this tool provides with generic matching, regression and classification fitness functions, which are not tailored to any particular problem. If for example the user will like to perform more advanced applications using this tool, he can always customize the fitness function in order to achieve the desired outcome.

As the Karoo GP tool was specifically developed for the seek of the proposed research in [38], it was tested and proven on symbolic regression problems, these being the *Kepler's Third Law of Planetary Motion* and the *Iris data-set* classification problem. Demonstrating to having solved these real world problems efficiently and accurately, the tool provides a flexible environment for symbolic regression problem solving through genetic programming algorithms.

As stated by [19], even though the many advantages that Karoo GP offers, the tool is not equipped to introduce constants within the expression. Making it difficult to fit data that has a peak that is not concentrated at the origin. Furthermore, it is easy for Karoo GP to get stuck in a loop of never improving poorly fitting functions. Which is dependant on which functions are chosen at the beginning, and since this is generated randomly, it is a difficult problem to avoid.

2.4.3 GLYPH (DEAP based)

Glyph is a Python package for symbolic regression, based on genetic programming [28]. It can be used for real world experiments with numerical solutions. It is a simple solution for those without deep programming experience, due to its generic interface design. In an effort to separate optimization methods from optimization tasks, the tool is implemented in a client-server architecture, with a minimal communication protocol which simplifies its use. Meant to be a lightweight frame work, it can build an application for finding an optimal system representation out of given data. This tool implements symbolic constant optimization and parallel execution, which can be used in complex applications. It is currently based on

DEAP [16], so all the advantages and disadvantages that DEAP presents, have a direct effect on Glyph. DEAP is a novel evolutionary computation framework for rapid prototyping and testing of ideas. Which is written in Python. This Python library can be run on Linux, Mac OS X and Microsoft Windows. It requires a Python version 2.6 or higher. By default, DEAP supports the tree-based representation, that enables tree GP, and array-like representations. The implementation of the evolutionary algorithm is done by using a pluggable genetic and selection operators. It enables the user to create genetic programming by the combination of different parts at the API level. One advantage is that many genetic operators have already been implemented. This tool also provides with all the basic data structures and genetic operators as well as some basic examples, which the user will find useful for implementing the evolutionary process. It is important to note that the fitness function is a user-defined Python function. Some drawbacks that the DEAP Python library might have are, the user must have a basic knowledge of the programming language and that it does not support graphical plotting for the experimental results. If for example the application that will use this tool has a high computation requirement, Python language is much more slower than compiled languages, such as C.

3 Real World Data Experiments

In the remainder of this master's thesis, we will show how symbolic regression is able to uncover an underlying algebraic expression for the commonly used exhaust gas temperature (EGT), measured in degrees centigrade in a turbofan engine. For this task, we have chosen to use the GPTIPS software, which was described in the previous section 2.4.1. As GPTIPS performs symbolic regression through multigene genetic programming, given a number of input variables, the aim is to find a feasible model that predicts the actual EGT. It should be point out that the models that are built out of the experiments, do not perform prediction but actually, now-casting; both terms will be used interchangeably in the rest of the thesis. As we will show, several experiments are performed in order to prove the viability of not only the tool, but also the theory of symbolic regression for finding an algebraic expression for an output variable out of measured input variables.

This section is organized as follows. We will first demonstrate the use of symbolic regression to retrieve an algebraic expression from observations with preliminary experiments in subsection 3.1. Secondly, a description of the real world data experiments developed in this thesis is provided in subsection 3.2, as well as the data, the preprocessing and the methodology used. The system setup description can be found in subsection 3.3. Finally, the measurement error metric used for analysing the accuracy of the results are described in subsection 3.4.

3.1 Preliminary Experiments

In order to familiarize ourselves with the tool, its functions and other characteristics, a set of simple experiments were performed with user generated data which followed well known physical laws, such as the Kelvin to Fahrenheit relation, the Kinetic Energy and the Gravitational Law. Although these experiments are not the main focus of this thesis, we believe that they prove the significance of the approach taken.

We will start with the most simple of the three preliminary experiments that were used to prove the use of symbolic regression. A set of randomly generated values from 0 to 50 and the output variable were given to GPTIPS. The output variable, as the target variable, resulted from manually applying the Kelvin to Fahrenheit conversion shown in equation 4. As the reader can see from equation 5, the underlying relationship between the input variable (temperature values in Kelvin), x_1, and the target variable (temperature in Fahrenheit) is successfully retrieved by GPTIPS.

$$T_{(F)} = (T_{(K)} * 9/5) - 459.67 \tag{4}$$

$$(1.8 * x_1) - 460 \tag{5}$$

A second preliminary experiment was performed to retrieve the kinetic energy of an object, which can be defined by the work needed to accelerate an object of a given mass, from the state of rest to a stated velocity. Formula 6 describes a non-rotating object with a mass *m* travelling at a speed *v*. In this simple experiment, in which the aim was to retrieve this physical formula, the mass and velocity were given to GPTIPS as input variables, along with the target variable (kinetic energy). This target variable was user generated data which was created by manually applying 6 on a set of data, which contained mass and velocity values.

The mass and velocity were randomly generated values, between 0 and 1 for the mass, and between 1 and 2 for velocity. The result of this experiment can be seen in equation 7, which closely describes the same relationship between the input and the output data, in this case, the kinetic energy. In equation 7, x_1 is *mass* and x_2 is *velocity*. The additional value stands for the bias term that multigene genetic programming adds as an error value.

$$E_k = \frac{1}{2}mv^2 \tag{6}$$

$$0.5x_1{x_2}^2 + 2.6 * 10^{-17} \tag{7}$$

As a third preliminary experiment, and with the intention of retrieving the Newton's law of universal gravitation from observed data, another user generated set of values was created. In this case, as gravitational law is described by formula 8 the different masses and the distance were given to GPTIPS as input variables, along with the target variable (gravitational law). This target variable was user generated data which was created by manually applying 8 on a data-set that contained the masses and distance values. These values were randomly generated between 0 and 1 for the masses, and between 1 and 2 for the distance. As a result, the reader can see on equation 9, how the physical law is retrieved, as well as the gravitational constant *G*, which is successfully approximated to its actual value of $6.67430 * 10^{-11}$ $(m^3 * kg^{-1} * s^{-1})$. Note that x_1 is $mass_1$, x_2 is $mass_2$ and x_3 is *distance*. The additional value stands for the bias term that multigene genetic programming adds as an error value.

$$F = G\frac{m_1 m_2}{d^2} \tag{8}$$

$$6.67 * 10^{-11}\frac{x_1 x_2}{x_3^2} + 6.6 * 10^{-28} \tag{9}$$

3.2 CEOD Data Experiments

This section is organized as follows. The CEOD data used is introduced in section 3.2.1. The preprocessing of these data is described in section 3.2.2. Finally, the methodology followed for developing the experiments is described in section 3.2.3.

3.2.1 Data

The data used for these experiments is the continuous engine operational data (CEOD) [14]. These is recorded on a recent aircraft, precisely a boeing $787-10$ (Dreamliner), the data are all from the same GEnx engine of this aircraft, and the data collection was carried out in July 2019. CEOD are a data stream made out of several hundred parameters which are measured along the entire flight duration. Due to computation limitations, the data we have worked on, has been off-loaded post-flight. These data are recorded real-time.

After examining the CEOD data, we selected the most stable phase of the flight.

This phase is assumed to be the cruising phase, due to the lack of official phase segmentation in the data. This assumption was backed up by discussions with experts. The reason for picking this phase is that it will allow for a better modelling of the underlying process, as the EGT measurements follow the same distribution. Other phases, such as take-off or landing were not investigated as they constitute a more intensive part of a flight. The possibility for analysing the entire flight duration remains an option, however an expert and deeper knowledge might be needed. Furthermore, the cruising phase allowed for a larger data size under the same phase, which is needed in order to uncover the meaningful relationships.

Three flights were selected for training and validation purposes. Which will be called flight 1, flight 2 and flight 3. A fourth flight was selected for testing purposes, this will be called flight 4. It is important to note, that the selection of the flights in order to be used as training, validation and test has been randomly done, not taking into consideration flight details or characteristics, i.e. destination or duration of the flight. The four different flights that were selected, were anonymized for confidentiality reasons. In these selected flights, a set of 696 variables were present. In our case, the selected target variable, from the CEOD data, that will be modelled is called *Selected Exhaust Gas Temperature (DEG_C)*.

3.2.2 Data Preprocessing

The preprocessing of the data is described as follows. Figure 5 shows a flowchart of this process.

1. Firstly, and as already mentioned, the cruising phase is selected for all four flights in CEOD.

2. The cruising phase for flights $1, 2$ and 3 are concatenated into a single data-frame.

3. Variables which do not provide extra information are dropped. These are:

   (a) NaN and string type variables.

   (b) Those which standard deviation was equal to zero.

   After dropping these variables, 206 remained as the input variables, in addition to EGT, the proposed output variable to be modeled.

4. A correlation analysis of the variables present on the data at this point is done. Each experiment has a different variable selection procedure as we will show in section 3.2.3.

5. The following applies on all three experiments. Here, the training set is randomly splitted into training and validation sets, with an $80 - 20$ rule. The number of data points per set is shown in table 1. As for the testing set, once the selected input variables from the training set are clear, selecting the same ones in flight 4 for the testing set is an easy process. Three testing sets, one per experiment, are computed.

	Data Points
Training Set	66585
Validation Set	16647
Testing Set	26671

Table 1: Data Points per Set

6. The last step of the data preprocessing method, is a data normalization procedure. By which, the remaining input variables and the selected output variables are standardize by removing the mean and scaling to the unit variance. This is done in all training, validation and test sets.

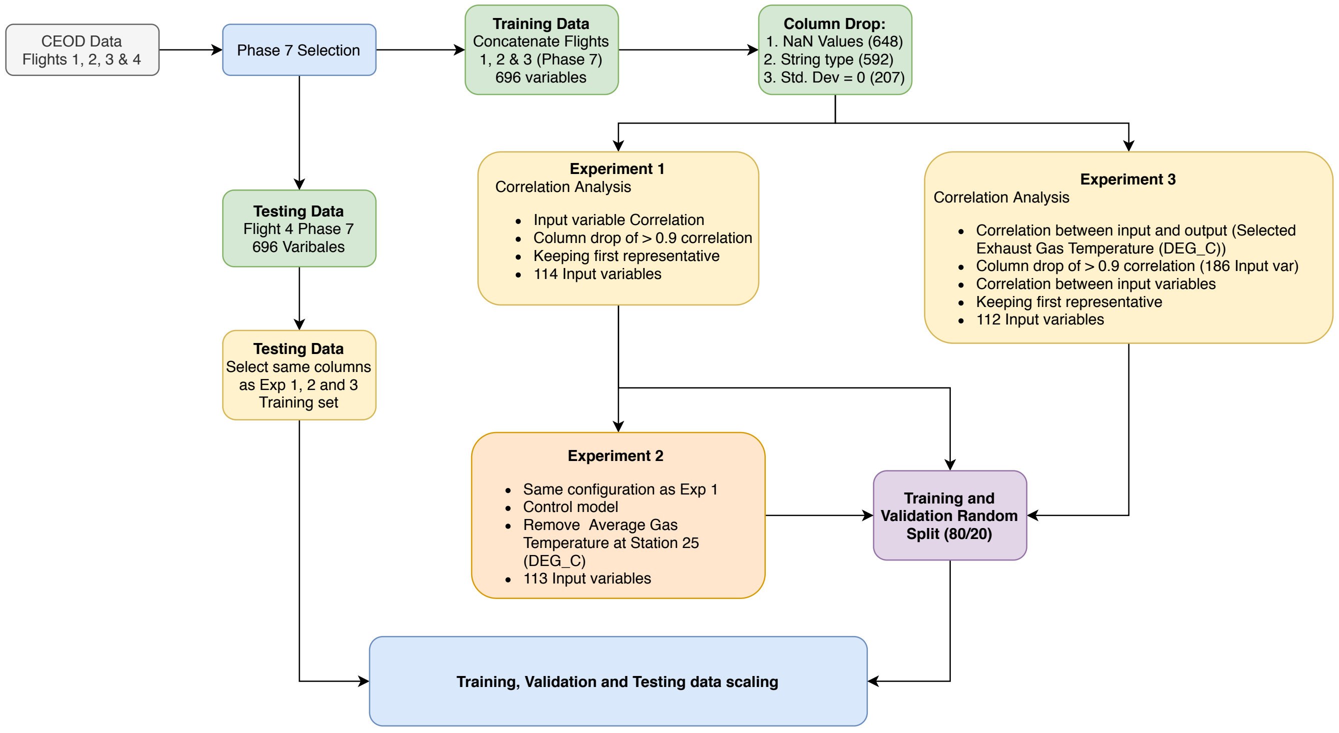

CEOD Data
Flights 1, 2, 3 & 4
Phase 7 Selection
Training Data
Concatenate Flights 1, 2 & 3 (Phase 7)
696 variables
Column Drop:
1. NaN Values (648)
2. String type (592)
3. Std. Dev = 0 (207)
Testing Data
Flight 4 Phase 7
696 Varibales
Experiment 1
Correlation Analysis
Input variable Correlation
Column drop of > 0.9 correlation
Keeping first representative
114 Input variables
Experiment 3
Correlation Analysis
Correlation between input and output (Selected Exhaust Gas Temperature (DEG_C))
Column drop of > 0.9 correlation (186 Input var)
Correlation between input variables
Keeping first representative
112 Input variables
Testing Data
Select same columns as Exp 1, 2 and 3 Training set
Experiment 2
Same configuration as Exp 1
Control model
Remove Average Gas Temperature at Station 25 (DEG_C)
113 Input variables
Training and Validation Random Split (80/20)
Training, Validation and Testing data scaling

Figure 5: Data preprocessing flowchart

3.2.3 Methodology

In this subsection, the correlation analysis done in the data preprocessing step 4 described in 3.2.2, is introduced. Each experiment is different from the others in the way this is done.

- **Experiment 1.** For this experiment, a correlation analysis is performed upon the input variables, so that those which are highly correlated are discarded. The threshold used is 0.90. Those above this threshold have been dropped, only keeping the first representative from a set of highly correlated variables. After these variables are deleted, 114 remain to be used as final input variables in addition to the EGT.

- **Experiment 2.** Once experiment 1 was completed and results were analyzed, the need for developing a control model is satisfied by this experiment. Here, one highly correlated variable, which is not above the proposed threshold, was dropped from the input variables. This variables is the *Average Temperature at Station 25 (DEG_C)*. Therefore, 113 input variables were used in GPTIPS in order to model the EGT.

- **Experiment 3.** A correlation analysis between the input variables and the output variable, EGT, is performed, dropping those variables whose correlation with EGT is higher than 0.90. After doing so, 186 input variables remained. Once this was done, and following experiment's 1 logic, a correlation analysis upon the remaining input variables is done. Again, those above the proposed threshold are deleted, only keeping the first representative from a set of highly correlated variables. Once this step was done, 112 input variables remained, plus the output variable, EGT. Table 2 shows the total number of input variables used in each one of the experiments.

	Number of Input Variables
Experiment 1	114
Experiment 2	113
Experiment 3	112

Table 2: Total number of input variables used per experiment

The reason for using a validation set is that by doing so, the chances of over-fitting the model are mitigated. The tool evaluates the "best" individual in each generation against the validation set and then keeps track of the results over each run. This can help the user to identify the individuals that best performed on a data-set that is not used to build the output model.

After setting the data-sets, training, validation ans testing, for the model building, one can configure the hyper-parameters that control each run. A more specific description will be given in the upcoming Section 3.3. Refer to Figure 4 for a configuration file example used for one of the experiments provided in the GPTIPS Guide document [33]. Then, the symbolic regression by means of genetic programming process is executed. In the case of the developed experiments, and after the predictions from GPTIPS where obtained, a last step was done. Since the data used was normalized, the predictions for the output variables were also normalized. Therefore, an original re-scale method was done in order to process the model metrics accordingly. The error metrics described in section 3.4 are the most commonly used for this type of modelling [36] .

3.3 System Setup

For each one of the experiments, ten final models were created using a ten independent run process. Which means that the GPTIPS process was executed ten times per model, and the results were combined into a one single model. The reader can refer to table 3 for a parameter system setup detailed view. These values have been selected from preliminary experiments. The population size was chosen to be of 250 individuals, while the number of generations was at is maximum 150 generations. Tournament size $= 20$, Tournament Pareto which encourages less complex models was set to 0.3. Elitism $= 0.3$ % of population. Maximum tree depth was set at 5 and the maximum number of genes was selected to be 10. Finally, the function set contained these operators $= \{$times, minus, plus, rdivide, square, exp, mult3, sqrt, cube, power, negexp, neg, abs, log$\}$. In essence these operators define our alphabet. See table 3 for a quick reference of the hyperparameters used.

Hyperparameter	**Value**
Runs	10
Population size	250
Number of generations	150
Tournament size	20
Tournament Pareto	0.3
Elitism	0.3
Maximum tree depth	5
Maximum number of genes	10
Function set	times, minus, plus, rdivide, square, exp, mult3, sqrt, cube, power, negexp, neg, abs, log

Table 3: System setup parameters

By default GPTIPS, provides a multigene symbolic regression fitness function, which was used in order to minimize the root mean squared prediction error on the training data. For the genetic operators probabilities, the following configuration was used: mutation events $= 0.1$, crossover events $= 0.85$.

3.4 Error Metrics

In this section, the most common error metrics [36] used for measuring a model's accuracy are described.

1. **RMSE.** The root-mean-squared error, measures the differences between the predicted values by the model and the actual or observed values. The deviations between these two measures are called residuals. The RMSE measures the accuracy of the predictive model. It represents the root of the average of the squared errors, which more precisely, are defined as the square of the difference between the predicted value and the real value. Equation 10 shows the formula for this error metric, with y being the actual values and $\hat{y}$ the predicted values. Using n measurements.

$$RMSE(y, \hat{y}) = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}(y_i - \widehat{y_i})^2} \tag{10}$$

2. **MSE.** The mean-squared error measures the average of the squares of the errors. Which means, when talking about predictive models, the average squared difference between the predicted values and the actual value. It measures the quality of the model, making those with values closer to zero, better models. Equation 11 shows the formula for this error metric, with y being the actual values and $\hat{y}$ the predicted values. Using n measurements.

$$MSE(y, \hat{y}) = \frac{1}{n}\Sigma_{i=1}^{n}(y_i - \hat{y_i})^2 \tag{11}$$

3. **MAE.** The mean absolute error measures the average magnitude of the errors in a set of predictions, without considering their direction. It is the average over the test sample of the absolute differences between prediction and actual observation where all individual differences have equal weight. Equation 12 shows the formula for this error metric, with y being the actual values and $\hat{y}$ the predicted values. Using n measurements.

$$MAE(y, \hat{y}) = \frac{1}{m} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{12}$$

4. $\mathbf{R^2}$. As a last metric for the modelling analysis, the coefficient of determination was also used. It is a key output of regression analysis. It assesses how strong the linear relationship between two variables is, in the case of predicted values and actual values, this is the squared of the correlation between them. This coefficient is commonly known as R-squared (R^2), and is sometimes referred to as the *"goodness of fit"*, and it ranges from 0 to 1.

4 Results

In order to clearly state the results of this master's thesis, the three experiments' results will be presented. The error metrics, model expressions and figures representing the scaled values for actual EGT against the modeled, will be shown here. The correlated variables stated in Section 3.2.3, will also be shown here, for the reader to see which of the present variables on the CEOD data had a precise correlation with the exhaust gas temperature. Table 4 shows those input variables with a correlation above the 0.9 threshold used. Each model took approximately one and a half hours on a PC with 8 GB of RAM and a CPU running at 1.8 GHz. The used Pandas package version was $1.0.3$, the Python version was the $3.8.3$. The Matlab version used was the R2019b.

Input variable	Correlation with EGT	Input variable	Correlation with EGT
UVL_EGTSEL6	0.999760	Estimated Compressor Discharge Total Temperature (DEG_C)	0.982100
EGT Probe 4 (168 deg CW ALF) (DEG_C)	0.998189	Selected Compressor Delay Total Temperature (DEG_C)	0.975469
EGT Probe 1 (41 deg CW ALF) (DEG_C)	0.995394	Selected Core Speed (%)	0.938906
EGT Probe 2 (62 deg CW ALF) (DEG_C)	0.995193	Core Speed from EMU (RPM)	0.938732
EGT Probe 8 (327 deg CW ALF) (DEG_C)	0.995020	Calculated Fuel to Air Ratio	0.934373
EGT Probe 3 (147 deg CW ALF) (DEG_C)	0.994973	PHI 60 Tip Temp Protection Before Limits (PPH/PSIA)	0.920603
EGT Probe 5 (200 deg CW ALF) (DEG_C)	0.994512	N1 Accel Trajectory Target Speed (%)	0.910759
EGT Probe 6 (232 deg CW ALF) (DEG_C)	0.993013	N1 Decel Trajectory Target Speed (%)	0.910757
EGT Probe 7 (264 deg CW ALF) (DEG_C)	0.990393	Fan Speed from EMU (RPM)	0.910709
Physical N1 Command (%)	0.907051	Selected Fan Speed (%)	0.910644

Table 4: Input variable correlation towards EGT (> 0.9)

4.1 Experiment 1 Results

The testing error metrics results for experiment 1 are shown in table 5, this table displays the already mentioned metrics in Section 3.4. Each columns refers to one error metric, and every row, correspond to each one of the ten models run for the experiment. Equation 13 displays an algebraic expression for the first of the models. The reader can refer to Appendix A for the training and validation error metrics in tables 11 and 12, and the rest of the models' expressions.

Rˆ2	RMSE	MAE	MSE
0.981343	3.240743	1.357668	10.502417
0.947178	5.452943	2.940967	29.734585
0.808921	10.371179	3.410746	107.561345
0.984409	2.962510	1.224247	8.776463
0.754388	11.758352	3.642960	138.258840
0.805126	10.473665	3.333293	109.697659
0.812055	10.285767	3.238445	105.797008
0.809934	10.343668	3.325208	106.991471
0.855140	9.030162	4.563011	81.543823
0.805914	10.452461	3.109154	109.253937

Table 5: Experiment 1 testing error metrics

$$\begin{aligned} Y_1^1 &= 0.141x_4 + 0.123x_5 + 0.8x_6 + 0.0214x_{12} - 0.123x_{18} + 0.751x_{21} + 0.0261x_{60} \\ &+ 0.0405x_{74} - 0.0371|log(x_{39})| + 1.32 * 10^{-4}e^{(2x_{21})} - 0.0428|x_4| \\ &- 0.0261e^{(x_{21})} + 0.00762{x_{74}}^2 + 0.133 \end{aligned} \tag{13}$$

A scaled plot of the testing predictions (displayed in orange) against the actual values (displayed in blue) for the EGT is shown by Figure 6. The x axis represents the used datapoints. It is important to highlight that the y axis represents the scaled EGT measures. Refer to Figures 9 and 10 for all EGT predictions against actual representations of each model.

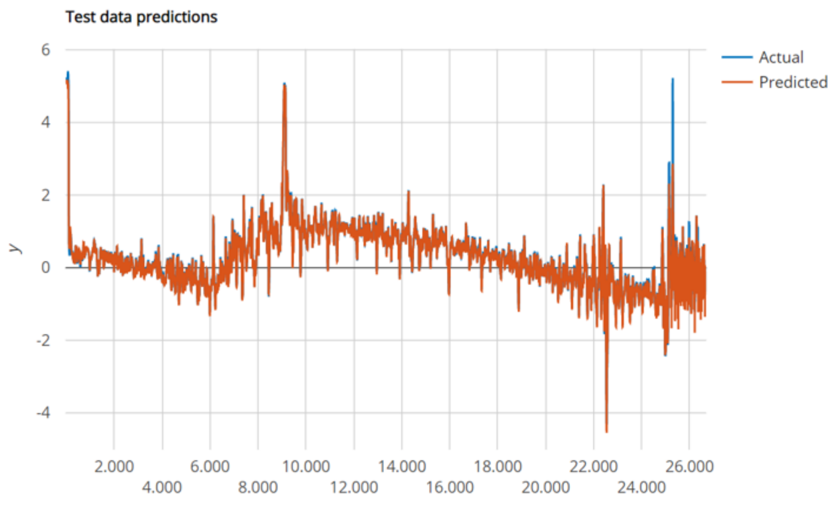


Figure 6: Scaled EGT testing predictions vs actual (Model 1 - Experiment 1)

The involved input variables percentage of appearance for the ten models is shown in table 6. Since each model provides with a different algebraic expression, the reader might find useful to know on what rate the input variables were present for this experiment over all computed models. Here, each variable is represented by an x with an index. The second columns shows the percentage of the different variables appearance.

Input variables (Index)	**% of appearance**
X4, X6, X21	10,6 %
X43	8,5 %
X12, X74, X113	5,1 %
X5, X110	4,08 %
X11, X14, X39, X59	3,06 %
X18, X23, X24, X94	2,04 %
X1, X8, X13, X25, X46, X52, X57, X60, X96, X111, X112	1,02 %

Table 6: Percentage of appearance per variable over all models (Experiment 1)

The reader might find interesting to know which of the variables have resulted from this experiment. The following list names the ones with the highest percentage of appearance and its corresponding index:

- Actual Calculated HPT Clearance (IN) – x4
- Average Gas Temperature at Station 25 (DEG_C) – x6
- Corrected Fan Speed to Station 12 (%) – x21
- FSV minimum main fuel split regulator – x43
- BPCU 1 GCU Generator Load (%) – x12
- Selected Variable Bleed Valve (VBV) Position (%) – x74
- WF/(P3*RTH25) Base (PPH/PSIA) – x113
- Altitude based on P0 (FT) – x5
- UVL_TEOSCVSEL – x110

4.2 Experiment 2 Results

After dropping the *Average Temperature at Station 25 (DEG_C)* and as a control experiment, these are the results. The testing error metrics results are shown in table 7, this table displays the already mentioned metrics in Section 3.4. Each column refers to every one of them, and every row, correspond to each one of the ten models run for this control experiment. Equation 14 displays an algebraic expression for the first of the models. Refer to Appendix B for the training and validation error metrics in tables 13 and 14 and the rest of the models' expressions.

R^2	RMSE	MAE	MSE
0.760114	11.62049	3.508928	135.0357
0.796653	10.69894	3.228156	114.4673
0.781762	11.08375	3.578363	122.8495
0.879283	8.243387	3.037525	67.95342
0.805173	10.4724	3.198549	109.6712
0.794995	10.74246	3.204305	115.4005
0.78722	10.94429	3.307758	119.7775
0.809058	10.36747	3.247515	107.4844
0.79427	10.76145	3.539824	115.8089
0.891921	7.799965	2.70033	60.83946

Table 7: Experiment 2 testing error metrics

$$\begin{aligned} Y_2^1 &= 0.0789x_4 + 0.143x_5 + 0.0207x_{10} + 0.799x_{17} - 0.524x_{19} + 0.575x_{20} \\ &- 0.0789e^{(-e^{(-x_{20})})} - 0.441e^{(-e^{(-e^{(-x_4)})})} - e^{(-e^{(-e^{(-x_{20})})})} \\ &- ((0.0186|x_{38}|)/|x_{98}|) + ((0.0092x_{38})/x_{51}) + 1.13 \end{aligned} \tag{14}$$

A scaled plot of the testing predictions (displayed in orange) against the actual values (displayed in blue) for the EGT is shown by Figure 7. The x axis represents the ordered datapoints and the y axis represents the scaled EGT measures. The reader can refer to Figures 11 and 12 for all EGT predictions against actual representations of each model.

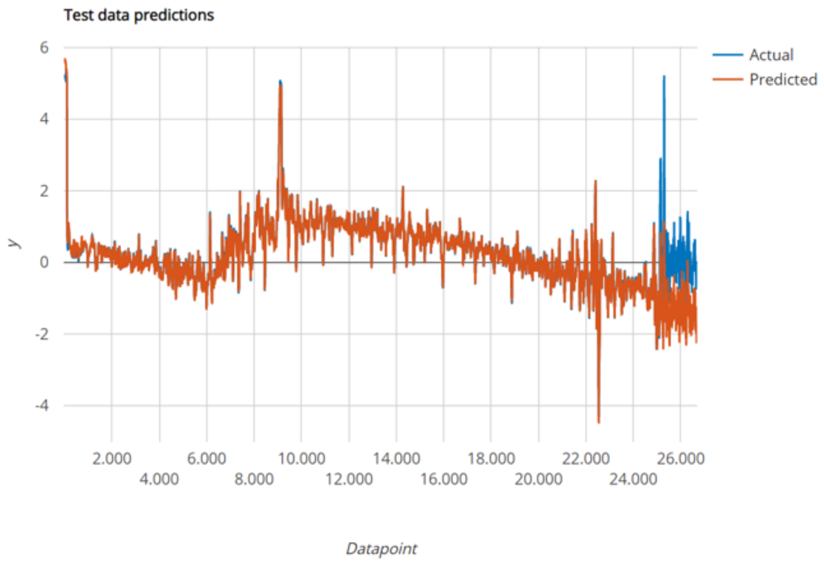


Figure 7: Scaled EGT testing predictions vs actual (Model 1 - Experiment 2)

Once more, the percentage of appearance of each one of the involved variables has been computed and displayed in table 8 for this control experiment. Here, each variable is represented by an x with an index. The second columns shows the percentage of appearance for the different variables.

Input variables (Index)	**% of appearance**
X4, X17, X19, X20	12,2 %
X58	8,5 %
X42	7,3 %
X112	6,1 %
X38	4,9 %
X5, X10	3,6 %
X7, X11, X22, X23, x24, X48, X51, X73, X95	1,2 % - 2,4 %

Table 8: Percentage of appearance per variable over all models (Experiment 2)

The following list, names the variables with the highest percentage of appearance and its corresponding index, resulting from this control experiment:

- Actual Calculated HPT Clearance (IN) – x4
- Center Tank Fuel Density from Aircraft (AIRPLANE UNITS) – x17
- Core Speed from EMU (RPM) – x19
- Core Speed Rate of Change (%N2/SEC) – x20
- No 1 Bearing Accel Broadband Amplitude (IPS) – x58
- FSV maximum main fuel split regulator – x42
- UVL_TMWCDATA9 – x112
- FMV Null Shift Compensation (MA) – x38

4.3 Experiment 3 Results

The last experiment, in where the correlation between the input variables with the selected output variable, EGT, was performed, with the intention to drop those with a higher correlation (refer to section 3.2.3 for a more detail description), shows these results. The testing error metrics results for experiment 3 are shown in table 9, this table displays the already mentioned metrics in Section 3.4. Each column refers to every one of the metrics, and every row, correspond to each one of the ten models run for this experiment. Equation 15 displays an algebraic expression for the first of the models. Refer to Appendix C for the training and validation table of error metrics (15 and 16) and the rest of the models' expressions.

R^2	**RMSE**	**MAE**	**MSE**
0.788536	10.91038	3.23724	119.03647
0.834811	9.64300	2.97148	92.98735
0.985225	2.88393	1.18922	8.31704
0.985034	2.90254	1.31293	8.42474
0.800509	10.59703	3.35942	112.29699
0.819613	10.07684	3.18990	101.54277
0.802258	10.55046	3.35021	111.31211
0.933994	6.09554	2.08652	37.15565
0.794766	10.74848	4.10824	115.52974
0.822608	9.99285	3.21731	99.85703

Table 9: Experiment 3 testing error metrics

$$
\begin{aligned}
Y_3^1 = {} & 0.114x_5 + 0.771x_6 + 0.0249x_{11} - 0.0763x13 + 0.395x_{19} + 1.62log(x_4 \\
& + e^{(-x_{23})} + 8.9) + 0.233e^{(-x_{23})} - 0.0534e^{(-x_{41})} - 1.36 * 10^{-5}e^{(-x_{103})} \\
& - 1.14e^{(-e^{(-e^{(-x_{19})})})} - 3.11
\end{aligned} \tag{15}
$$

An scaled plot of the testing predictions (displayed in orange) against the actual values (displayed in blue) for the EGT is shown by Figure 8. For confidentiality reasons, the scale of the *y* axis is not at the original value. The x axis represents the ordered data-points. Refer to Figures 13 and 14 for all EGT predictions against actual representations of each model.

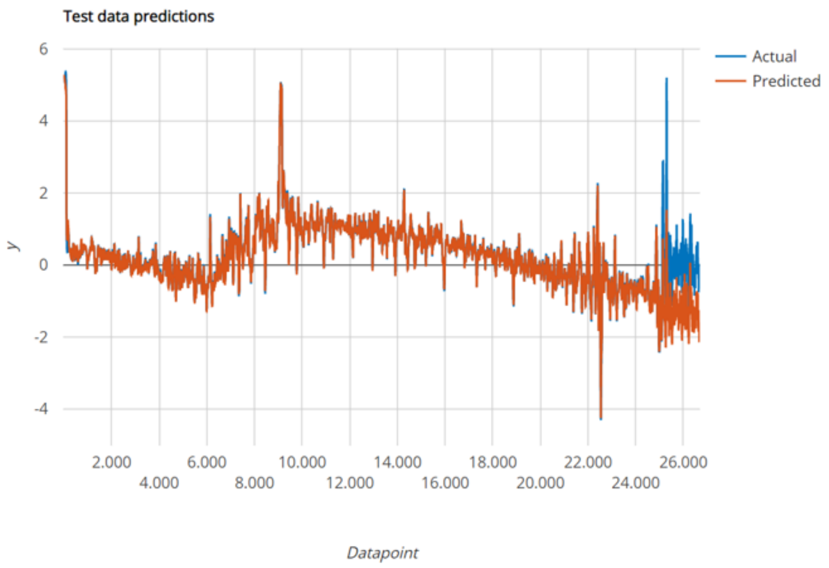


Figure 8: Scaled EGT testing predictions vs actual (Model 1 - Experiment 3)

The percentage of appearance of each one of the involved variables have been computed and displayed in table 10. Here, each variable is represented by an x with an index. The second columns shows the percentage of the different variables appearance.

Input variables (Index)	**% of appearance**
X4, X6, X9	11,3 %
X5, X41	9 %
X11	6,8 %
X72	4,5 %
X21, X23, X26, X74, X108, X111	3,4 %
X14, X37	2,3 %
X1, X12, X13, X28, X38, X40, X58, X67, X97, X103	1 %

Table 10: Percentage of appearance per variable over all models (Experiment 3)

The following list, names those variables with the highest percentage of appearance and its corresponding index, resulting from this last experiment:

- Actual Calculated HPT Clearance (IN) – x4
- Average Gas Temperature at Station 25 (DEG_C) – x6
- Corrected Fan Speed to Station 12 (%) – x19
- Altitude based on P0 (FT) – x5
- FSV minimum main fuel split regulator – x41
- BPCU 2 GCU Generator Load (%) – x11
- Selected Variable Bleed Valve (VBV) Position (%) – x72

5 Conclusion and Discussion

In this master's thesis research, a deep analysis of the latest developments and tools on symbolic regression by means of genetic programming was done. With the intention of solving the main problems present in symbolic regression, the reviewed developments show great progress in solving them. Multigene genetic programming helps on improving the evolution process. Geometric semantic genetic programming evaluates the semantic performance rather than the conventionally used syntax performance. For finding the appropriate constants or parameters, evolutionary polynomial regressions proposes a hybrid approach by adding the parameter estimation technique used in numerical regression methods to the evolutionary optimization scheme. Genetic programming-based relevance vector machines provides with the advantages of both the Bayesian kernel methodologies and evolutionary computation. The Cartesian genetic programming approach represents a more sophisticated design than the conventional genetic programming algorithm, introducing a new crossover technique. With regards to an approach which is not based on genetic programming, an artificial bee colony algorithm allows to evolve expressions and constants to form algebraic expressions automatically, providing a fresh and different point of view on the symbolic regression problem.

Three different tools were analysed in order to determine which one could be used for our purpose: funding an algebraic expression that could model exhaust gas temperature in a turbofan engine. It is important to note why EGT was chosen. This parameter helps on monitoring the health of an engine, as it transforms heat into thrust. Therefore, if this transformation is not happening EGT will increase. As a result, high measures of EGT could indicate an engine failure or damage. The GPTIPS free software toolbox for Matlab was selected. As the reader can see from the previously presented results in section 4, three different experiments were developed. The so called experiment 1 and experiment 3, show better results in terms of error of predictions and "goodness of fit", than experiment 2. As experiment 2 is a control experiment in which one highly correlated variable was dropped, it is reasonable to say that the models built out of this experiment are worse than the ones built in experiment 1 and 3. The variable that was dropped in experiment 2 was *Average Temperature in Station 25 (DEG_C)*, and the reader might find interesting to know that after checking with an expert, this dropped variable was a non-trivial and meaningful variable.

From the algebraic expressions derived from each of the models, it can be seen that they are reasonably compact. They consist of linear terms and low order non-linear transformations of the input variables. Symbolic regression not only helps on developing algebraic expressions but also helps with variable selection. Proven by the fact that, a reasonable amount - from 7 to 13 - of the available input variables (114, 113 and 112) in each experiment, were used in generating each model, as shown in section 4.

When it comes to the modelling results for exhaust gas temperature, it can be seen that the overall accuracy of the modelling has significantly good results for experiment 1 and 3. The reader can see how experiment 1 presents the best of the results, in terms of error metrics. The first model in table 5 in experiment 1 has a 0.98 coefficient of determination and just 3.28 RMSE score. Overall, and with the obtained metrics, any of the models (equations) in experiment 1 and 3, with reasonable good error metrics, could be used along with the input variables to now-cast the EGT.

As one of the purposes of this research was to find a tool that could be useful for our application, it can be seen how GPTIPS has allowed it due to the features and capabilities that provides.

6 Future Work

As the results have shown, many different models that describe EGT have been obtained, it is then open for future work to decide how such data-driven problems can be handled. We propose two different approaches for this. First, the building of a meta-model that combines all of those derived from the experiments, can be interesting to address. While also, making the process real-time and automatic. The second proposition will be making an ensemble model by, for example, taking the average or other aggregation function of the outputs provided by each of the models. The characteristics and dynamics of both approaches are open for the reader to work on. The approach that we have developed can also be applied as a model to generate simulated data that can later be used in predictive maintenance studies. Consequently, building any of these into a real-time analysis tool, will make the proposition much more valuable for a possible industrial application.

Bibliography

[1] Wafa Abdelmalek, Sana Ben Hamida, and Fathi Abid. "Selecting the best forecasting-implied volatility model using genetic programming". In: *Advances in Decision Sciences* 2009 (2009).

[2] Ignacio Arnaldo, Krzysztof Krawiec, and Una-May O'Reilly. "Multiple regression genetic programming". In: *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. 2014, pp. 879–886.

[3] Daniel Ashlock. *Evolutionary computation for modeling and optimization*. Springer Science & Business Media, 2006.

[4] Josh Bongard and Hod Lipson. "Automated reverse engineering of nonlinear dynamical systems". In: *Proceedings of the National Academy of Sciences* 104.24 (2007), pp. 9943–9948.

[5] George EP Box et al. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[6] Mauro Castelli, Sara Silva, and Leonardo Vanneschi. "A C++ framework for geometric semantic genetic programming". In: *Genetic Programming and Evolvable Machines* 16.1 (2015), pp. 73–81.

[7] Janet Clegg, James Alfred Walker, and Julian Frances Miller. "A new crossover technique for cartesian genetic programming". In: *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. 2007, pp. 1580–1587.

[8] Jason M Daida and Adam M Hilss. "Identifying structural mechanisms in standard genetic programming". In: *Genetic and Evolutionary Computation Conference*. Springer. 2003, pp. 1639–1651.

[9] Vinicius Veloso De Melo. "Kaizen programming". In: *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. 2014, pp. 895–902.

[10] Korneel Duyvesteyn and Uzay Kaymak. "Genetic programming in economic modelling". In: *2005 IEEE Congress on Evolutionary Computation*. Vol. 2. IEEE. 2005, pp. 1025–1031.

[11] Hossam Faris, Bashar Al-Shboul, and Nazeeh Ghatasheh. "A genetic programming based framework for churn prediction in telecommunication industry". In: *International Conference on Computational Collective Intelligence*. Springer. 2014, pp. 353–362.

[12] Hossam Faris, Alaa Sheta, and Ertan Öznergiz. "Modelling hot rolling manufacturing process using soft computing techniques". In: *International Journal of Computer Integrated Manufacturing* 26.8 (2013), pp. 762–771.

[13] Antonio Fernández-Ares et al. "Designing competitive bots for a real time strategy game using genetic programming." In: *CoSECivi*. Citeseer. 2014, pp. 159–172.

[14] Florent Forest et al. "A Generic and Scalable Pipeline for Large-Scale Analytics of Continuous Aircraft Engine Data". In: *2018 IEEE International Conference on Big Data (Big Data)*. IEEE. 2018, pp. 1918–1924.

[15] Stephanie Forrest. "Genetic algorithms: principles of natural selection applied to computation". In: *Science* 261.5123 (1993), pp. 872–878.

[16] Félix-Antoine Fortin et al. "DEAP: Evolutionary algorithms made easy". In: *Journal of Machine Learning Research* 13.70 (2012), pp. 2171–2175.

[17] Orazio Giustolisi and Dragan A Savic. "A symbolic data-driven technique based on evolutionary polynomial regression". In: *Journal of Hydroinformatics* 8.3 (2006), pp. 207–222.

[18] JH Holland. “Adaptation in natural and artificial systems”. In: *Ann Arbor The University of Michigan Press* (1975).

[19] Kaeli Hughes. “Genetic Programming: A Novel Method for Neutrino Analysis”. PhD thesis. The Ohio State University, 2017.

[20] Dervis Karaboga. *An idea based on honey bee swarm for numerical optimization.* Tech. rep. Technical report-tr06, Erciyes university, engineering faculty, computer . . ., 2005.

[21] Dervis Karaboga et al. “Artificial bee colony programming for symbolic regression”. In: *Information Sciences* 209 (2012), pp. 1–15.

[22] Taghi M Khoshgoftaar and Yi Liu. “A multi-objective software quality classification model using genetic programming”. In: *IEEE Transactions on Reliability* 56.2 (2007), pp. 237–245.

[23] John R Koza. *Genetic programming: on the programming of computers by means of natural selection.* Vol. 1. MIT press, 1992.

[24] Dae-Heum Kwon and David A Bessler. “Graphical methods, inductive causal inference, and econometrics: A literature review”. In: *Computational Economics* 38.1 (2011), pp. 85–106.

[25] William La Cava, Kourosh Danai, and Lee Spector. “Inference of compact nonlinear dynamic models by epigenetic local search”. In: *Engineering Applications of Artificial Intelligence* 55 (2016), pp. 292–306.

[26] Lilian M de Menezes and Nikolay Y Nikolaev. “Forecasting with genetically programmed polynomial neural networks”. In: *International Journal of Forecasting* 22.2 (2006), pp. 249–265.

[27] Una-May O'Reilly et al. *Genetic programming theory and practice II.* Vol. 8. Springer Science & Business Media, 2006.

[28] Markus Quade, Julien Gout, and Markus Abel. "Glyph: Symbolic Regression Tools". In: *arXiv preprint arXiv:1803.06226* (2018).

[29] Hossein Izadi Rad, Ji Feng, and Hitoshi Iba. "GP-RVM: Genetic Programing-based Symbolic Regression Using Relevance Vector Machine". In: *arXiv preprint arXiv:1806.02502* (2018).

[30] Conor Ryan. *Automatic re-engineering of software using genetic programming.* Vol. 2. Springer Science & Business Media, 2000.

[31] Conor Ryan, Michael O'Neill, and JJ Collins. "Grammatical evolution: Solving trigonometric identities". In: *proceedings of Mendel.* Vol. 98. 1998, 4th.

[32] Lothar M Schmitt and Stefan Droste. "Convergence to global optima for genetic programming systems with dynamically scaled operators". In: *Proceedings of the 8th annual conference on Genetic and evolutionary computation.* 2006, pp. 879–886.

[33] Dominic Searson. "Genetic Programming & Symbolic Regression for MATLAB User Guide". In: ().

[34] Dominic P Searson, David E Leahy, and Mark J Willis. "GPTIPS: an open source genetic programming toolbox for multigene symbolic regression". In: *Proceedings of the International multiconference of engineers and computer scientists.* Vol. 1. 2010, pp. 77–80.

[35] S Sette and L Boullart. "Genetic programming: principles and applications". In: *Engineering Applications of Artificial Intelligence* 14.6 (2001), pp. 727–736.

[36] Maxim Vladimirovich Shcherbakov et al. "A survey of forecast error measures". In: *World Applied Sciences Journal* 24.24 (2013), pp. 171–176.

[37] William M Spears. “Adapting crossover in evolutionary algorithms.” In: *Evolutionary programming*. 1995, pp. 367–384.

[38] Kai Staats. “Genetic programming applied to RFI mitigation in radio astronomy”. PhD thesis. University of Cape Town, 2016.

[39] Alexander Von Moll et al. “A review of exhaust gas temperature sensing techniques for modern turbine engine controls”. In: *50th AIAA/ASME/SAE/ASEE Joint Propulsion Conference*. 2014, p. 3977.

[40] Wen-Chuan Wang et al. “A comparison of performance of several artificial intelligence methods for forecasting monthly discharge time series”. In: *Journal of hydrology* 374.3-4 (2009), pp. 294–306.

[41] AS Weigend and NA Gershenfeld. “Time series prediction: forecasting the future and understanding the past.” In: *Reading, Mass, Addison-Wesley* (1993).

[42] Mengjie Zhang and Will Smart. “Multiclass object classification using genetic programming”. In: *Workshops on Applications of Evolutionary Computation*. Springer. 2004, pp. 369–378.

A Appendix. Experiment 1 Results

In this appendix, the training and validation error metrics results are displayed in tables 11 and 12.

R^2	**RMSE**	**MAE**	**MSE**
0.998102	1.239205	0.763866	1.535629
0.997968	1.281986	0.786991	1.643489
0.997507	1.420041	0.828125	2.016517
0.997964	1.283411	0.786591	1.647145
0.998020	1.265571	0.765595	1.601671
0.998023	1.264679	0.738129	1.599412
0.997745	1.350609	0.754343	1.824146
0.998036	1.260616	0.746105	1.589153
0.998123	1.232339	0.740111	1.518661
0.997676	1.371248	0.791502	1.880321

Table 11: Experiment 1 training error metrics

R^2	**RMSE**	**MAE**	**MSE**
0.99822	1.20770	0.75207	1.45853
0.99803	1.27203	0.77908	1.61805
0.99767	1.38459	0.82148	1.91708
0.99808	1.25604	0.77368	1.57764
0.99810	1.24747	0.75516	1.55618
0.99813	1.24030	0.72427	1.53834
0.99788	1.32030	0.74576	1.74320
0.99817	1.22514	0.73863	1.50097
0.99814	1.23560	0.73759	1.52672
0.99783	1.33519	0.78411	1.78273

Table 12: Experiment 1 validation error metrics

Next, each symbolic expression model is detailed here:

$Y_1^1 = 0.141x_4 + 0.123x_5 + 0.8x_6 + 0.0214x_{12} - 0.123x_{18} + 0.751x_{21} + 0.0261x_{60} + 0.0405x_{74} - 0.0371|log(x_{39})| + 1.32*10^{-4}e^{2x_{21}} - 0.0428|x_4| - 0.0261e^{x_{21}} + 0.00762{x_{74}}^2 + 0.133$

$Y_1^2 = 0.13x_4 + 0.668x_6 + 0.0264x_{12} - 0.0796x_{14} + 0.759x_{21} + 0.0484x_{43} - 0.00864x_{57} + 0.0219x_{110} - 0.0219x_{113} - 0.0264|x_4| + 0.0219|x_6| - 0.00702|x_{43} - 2.0x_{46} + 0.2{x_{21}}^3| - 0.00579{x_{21}}^3 - 6.3 * 10^{-4} * |x_{14} - 1.0x_{21}| * |x_{21}|^3 * |x_{43}| + 0.0119$

$Y_1^3 = 0.169x_4 - 0.0168x_1 + 0.125x_5 + 0.712x_6 - 0.0336x_{18} + 0.704x_{21} + 0.0376x_{43} + 0.0156x_{110} - 0.00778e^{-x_{12}} - 0.0808|x_4| - 0.0513|x_8| - 0.0156|x_{21}| + 0.0432{|x_4|}^{1/2} - 0.00102|x_{21} + abs(x_{21})|^3 + 0.0368(|x_{74}|^3)^{1/2} + 0.0736$

$Y_1^4 = 0.0758x_4 + 0.885x_6 - 0.2x_{13} - 0.156x_{14} + 0.43x_{21} - 0.0244x_{23} - 0.495e^{(-e^{(e^{(x_{43})})})} + 0.2e^{(-e^{(-x_4)})} + 0.156e^{(-e^{(-x_{14})})} + 0.00361e^{(x_{110})} - 0.787e^{(-e^{(x_{21})})} + 0.00719{x_{14}}^2 + 0.214$

$Y_1^5 = 0.0887x_4 + 0.111x_5 + 0.69x_6 + 0.0296x_{11} + 0.523x_{21} - 0.0442x_{39} - 0.121e^{(-x_{39})} - 0.499e^{(-e^{(x_{21})})} + 0.0559x_4{(e^{(x_5)})}^{1/2} - 0.00493x_{74}x_{113} + 0.353$

$Y_1^6 = 0.204x_4 + 0.592x_6 + 0.0242x_{11} + 0.485x_{21} + 0.0698x_{43} + 0.122x_{59} - 0.642e^{(-e^{(x_{21})})} - (0.00133 * |x_{43}|)/|x_{94}|^2 - 0.0205{x_4}^2 + 0.00106{x_4}^3 + 0.365$

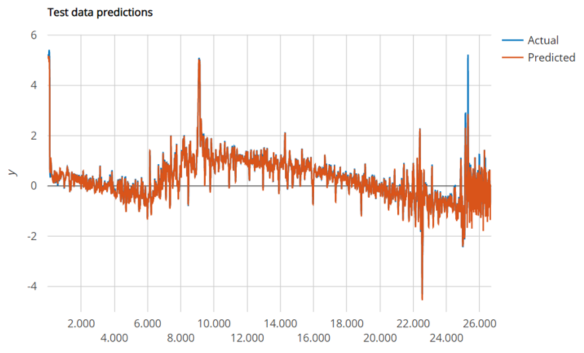
$Y_1^7 = 0.111x_4 + 0.601x_6 + 0.0264x_{11} + 0.448x_{21} - 0.0361x_{25} + 0.0459x_{43} + 0.111x_{59} - 0.0346x_{113} - 0.0553e^{(-x_4)} + 0.02e^{(x_{113})} - 0.774e^{(-e^{(x_{21})})} + 0.326$

$Y_1^8 = 0.133x_4 + 0.649x_6 + 0.00805x_{12} - 0.101x_{14} + 0.749x_{21} - 0.0178x_{23} + 0.0268x_{24} + 0.0463x_{43} + 0.00805x_{52} + 0.0155x_{74} + 0.0268x_{110} - 0.0115x_{111} - 0.0178x_{113} - 0.0178|x_4| - 0.0178|x_{74}| + 0.00455x_4x_6 - 0.0176x_{21}x_{74} + 0.0176x_{23}x_{74} - 0.0176x_{52}x_{74} + 0.0176x_{74}|x_{21}| - 0.00228{x_4}^2 - 0.00228{x_6}^2 - 0.0311{x_{21}}^2 + 0.0271$

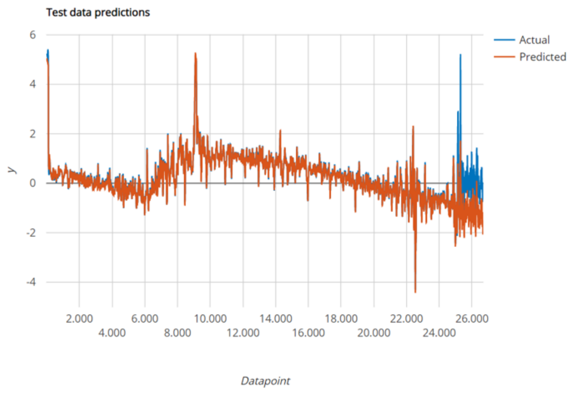
$Y_1^9 = 0.136x_4 + 0.592x_6 + 0.701x_{21} + 0.0567x_{43} + 0.127x_{59} + 0.0291x_{96} - 0.0263x_{113} - 0.00304|(x_8 - 5.11)||x_{21}|^2 + 0.00248x_4x_{59} - 0.0307x_{21}|x_{21}| - 4.08 * 10^{-4}{x_{113}}^3 + 0.0092$

$Y_1^{10} = 0.17x_4 - 0.00532x_3 + 0.121x_5 + 0.685x_6 + 0.0105x_{12} + 0.65x_{21} + 0.0375x_{24} - 0.00532x_{39} + 0.075x_{74} - 0.0079x_{112} - 0.00532x_{43}/x_{94} - 0.0108{x_4}^2 + 4.7 * 10^{-4}{x_4}^3 - 0.0202{x_{21}}^2 - 0.00258{x_{21}}^3 - 0.00258{x_{39}}^2 + 0.0246$

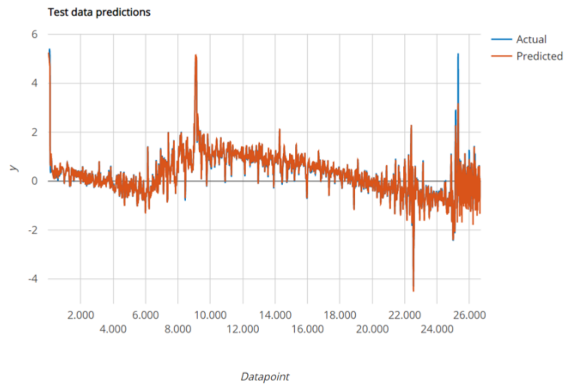
The next figures display the model predictions against the actual values of the EGT:



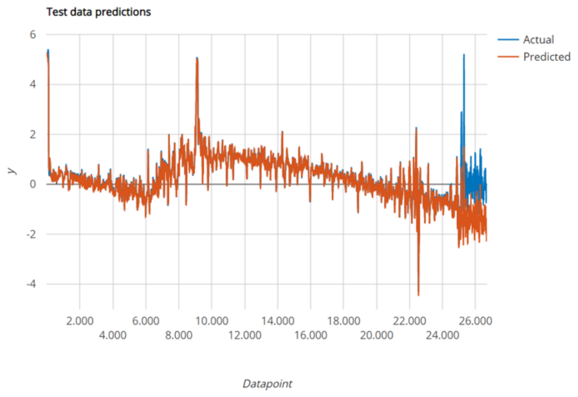
(a) Model 1



(b) Model 2

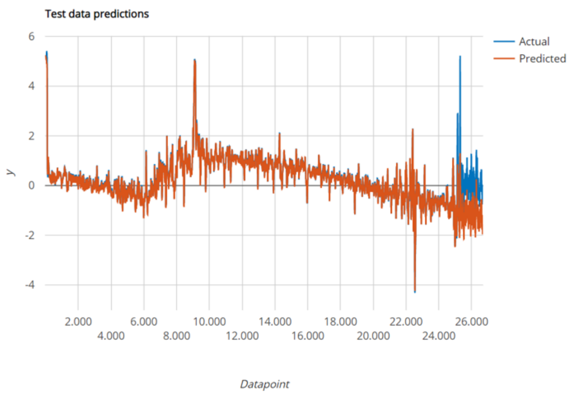


(c) Model 3

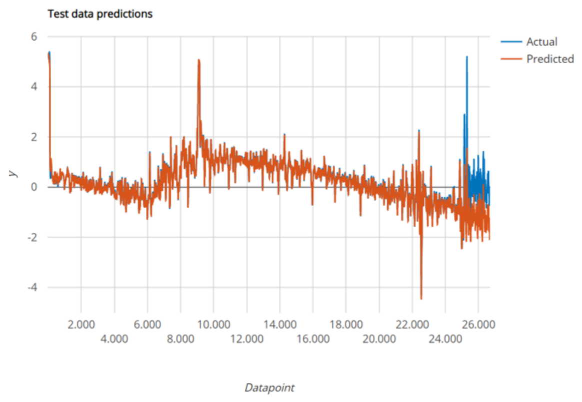


(d) Model 4

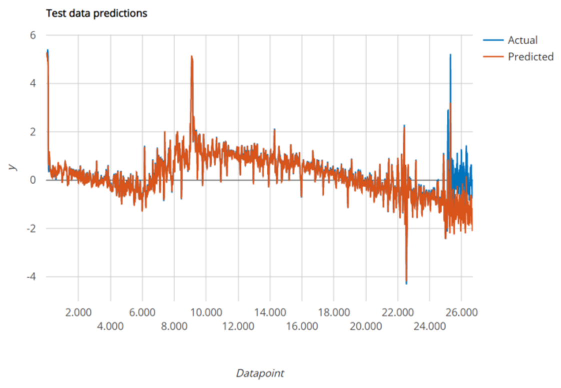
Figure 9: Scaled EGT testing predictions vs actual



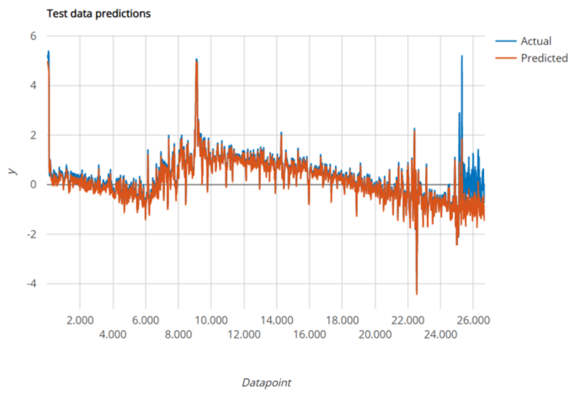
(a) Model 5



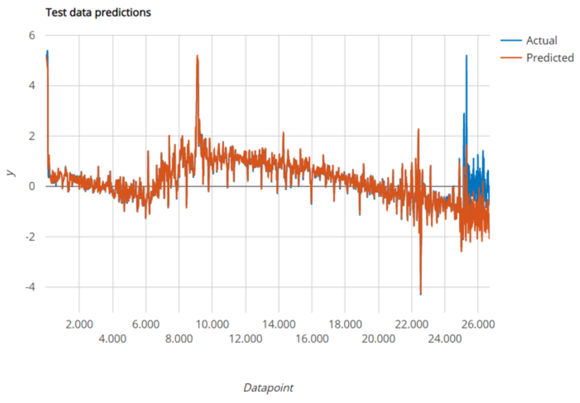
(b) Model 6



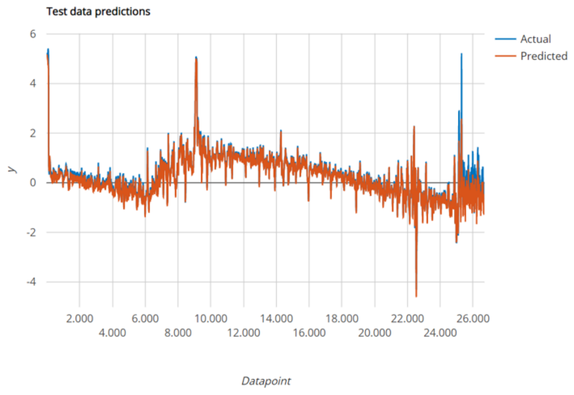
(c) Model 7



(d) Model 8



(e) Model 9



(f) Model 10

Figure 10: Scaled EGT testing predictions vs actual

B Appendix. Experiment 2 Results

In this appendix, the training and validation error metrics results are displayed in tables 13 and 14.

R^2	RMSE	MAE	MSE
0.99768	1.36893	0.85751	1.87396
0.99756	1.40389	0.88441	1.97090
0.99751	1.41842	0.87606	2.01192
0.99696	1.56913	0.95831	2.46217
0.99743	1.44187	0.87915	2.07899
0.99765	1.37774	0.85544	1.89818
0.99741	1.44669	0.86240	2.09291
0.99756	1.40450	0.87028	1.97263
0.99746	1.43365	0.86886	2.05535
0.99762	1.38674	0.85581	1.92304

Table 13: Experiment 2 training error metrics

R^2	RMSE	MAE	MSE
0.99781	1.34160	0.84267	1.79988
0.99766	1.38506	0.87861	1.91840
0.99766	1.38600	0.86851	1.92098
0.99701	1.56784	0.95604	2.45812
0.99751	1.42995	0.87120	2.04477
0.99784	1.33301	0.84164	1.77691
0.99748	1.43730	0.85617	2.06582
0.99767	1.38220	0.86608	1.91047
0.99755	1.41698	0.86252	2.00783
0.99777	1.35305	0.84415	1.83074

Table 14: Experiment 2 validation error metrics

Next, each symbolic expression model is detailed here:

$Y_2^1 = 0.0789x_4 + 0.143x_5 + 0.0207x_{10} + 0.799x_{17} - 0.524x_{19} + 0.575x_{20} - 0.0789e^{(-e^{(-x_{20})})} -$
$0.441e^{(-e^{(-e^{(-x_4)})})} - e^{(-e^{(-e^{(-x_{20})})})} - ((0.0186|x_{38}|)/|x_{98}|) + ((0.0092x_{38})/x_{51}) + 1.13$

$Y_2^2 = 0.13x_4 + 0.0229x_{11} + 0.694x_{17} - 0.394x_{19} + 0.723x_{20} + 0.0604x_{42} + 0.107x_{58} -$
$0.0261|x_{19}||x_{20}| + 0.0168x_{42}x_{112} - 0.033x_{20}|x_{20}| - 0.0162x_{42}|x_{20}| - 0.013x_{112}|x_{19}| + 0.0126$

$Y_2^3 = 0.123x_4 + 0.0305x_{10} + 0.713x_{17} - 0.327x_{19} + 0.628x_{20} + 0.0582x_{42} + 0.0949x_{58} +$
$0.0263x_{73} - 0.0303x_{112} - 0.0119{x_{20}}^2x_{42} + 0.00167$

$Y_2^4 = 0.1x_4 + 0.671x_{17} - 0.389x_{19} + 0.506x_{20} - 0.0408x_{38} + 0.138x_{58} - 0.108e^{(-x_{38})} +$
$0.929e^{(-e^{(-e^{(-e^{(-x_4)})})})} - 0.0472|x_7| - 0.633e^{(-e^{(x_{20})})} - 0.0392$

$Y_2^5 = 0.11x_4 + 0.662x_{17} - 0.39x_{19} + 0.526x_{20} + 0.0537x_{42} + 0.141x_{58} + 0.0269x_{95} -$
$0.0609e^{(-x_4)} - 0.00223e^{(x_{20})} - 0.587e^{(-1.1e^{(x_{20})})} - 0.00143x_{23}{x_{112}}^2 - 0.00143{x_{23}}^2x_{112} - 4.76 *$
$10^4{x_{23}}^3 - 4.76 * 10^4{x_{112}}^3 + 0.276$

$Y_2^6 = 0.647x_{17} - 0.412x_{19} + 0.549x_{20} + 0.0505x_{42} + 0.157x_{58} + 0.0243x_{95} - 0.826e^{(-e^{(-e^{(-x_{20})})})} -$
$1.59e^{(-e^{(-e^{(-x_4)})^{1/2}})} + 0.00578{x_4}^2 - 7.58 * 10^5{x_{51}}^4 + 1.69$

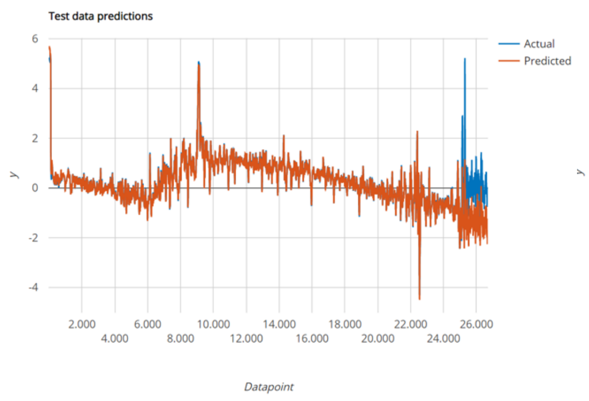
$Y_2^7 = 0.0925x_4 + 0.113x_5 + 0.0195x_{11} + 0.797x_{17} - 0.427x_{19} + 0.518x_{20} - 0.0181x_{112} -$
$0.452e^{(-x_{38})} - 1.13e^{(-e^{(-x_{38})})} - 0.45e^{(-e^{(-e^{(-x_4)})})} - 0.847e^{(-e^{(-e^{(-x_{20})})})} + 1.89$

$Y_2^8 = 0.158x_4 + 0.131x_5 + 0.8x_{17} - 0.46x_{19} + 0.53x_{20} - 0.0231x_{22} - 0.0231e^{(-x_4)} -$
$0.0826e^{(-x_{38})} - 0.0826e^{(-2e^{(-x_{20})})} - 0.928e^{(-e^{(-e^{(-x_{20})})})} - 0.00491{x_4}^2 - 0.00827{x_{38}}^2 + 0.811$

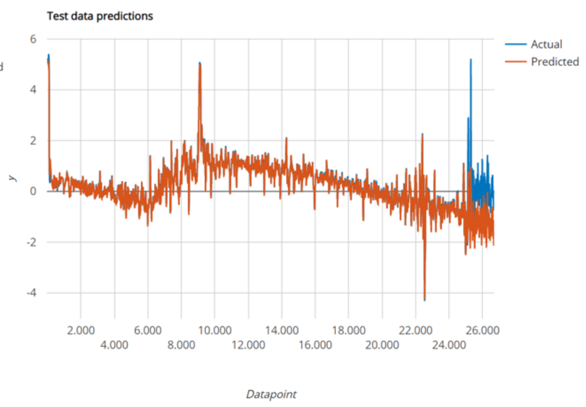
$Y_2^9 = 0.172x_4 + 0.0239x_{10} + 0.667x_{17} - 0.403x_{19} + 0.593x_{20} + 0.0475x_{42} + 0.133x_{58} +$
$0.0481x_{73} - 0.0481e^{(0.567x_{20})} - 0.623e^{(-e^{0.73x_{20}})} - 0.0647|x_4| + 0.316$

$Y_2^{10} = 0.129x_4 + 0.69x_{17} - 0.373x_{19} + 0.455x_{20} - 0.0382x_{24} + 0.121x_{58} - 0.0303x_{112} + 0.509e^{(-e^{(-e^{(-e^{(-x_{42})})})})} + 0.0771e^{(-2x_7{}^2)} - 0.707e^{(-e^{x_{20}})} - 0.0137$

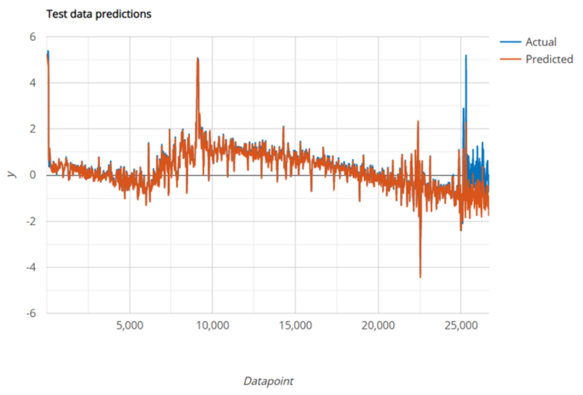
The figures in where the model predictions against the actual values of the EGT per model are displayed below:



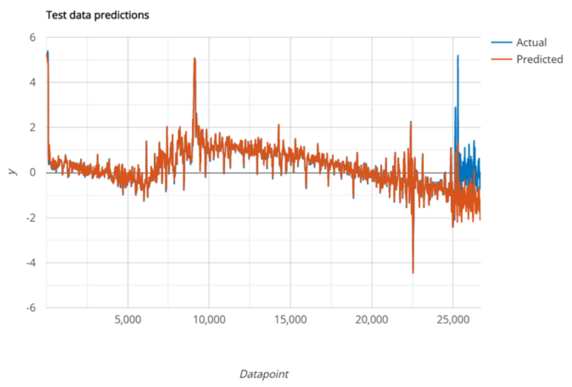
(a) Model 1



(b) Model 2

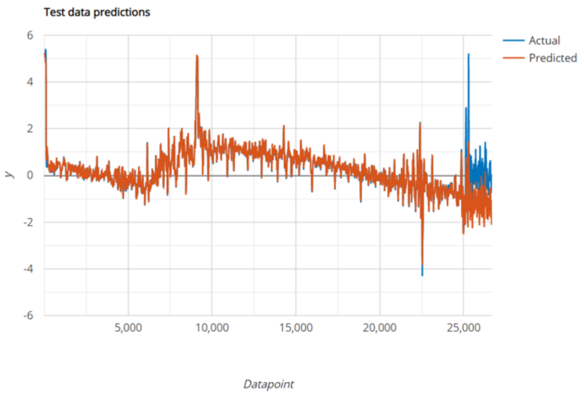


(c) Model 3

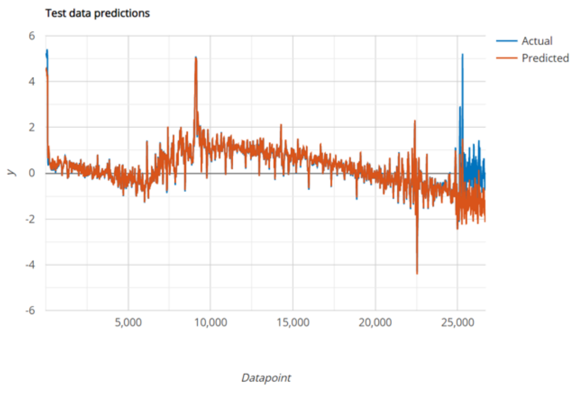


(d) Model 4

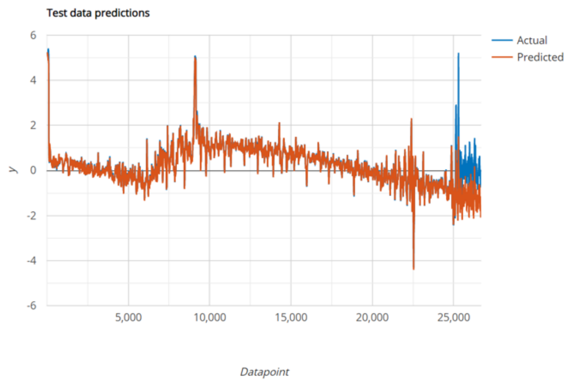
Figure 11: Scaled EGT testing predictions vs actual



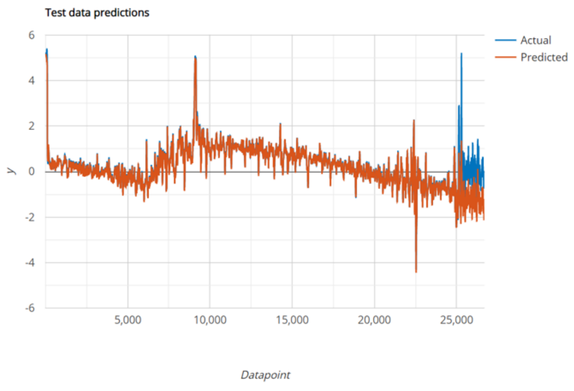
(a) Model 5



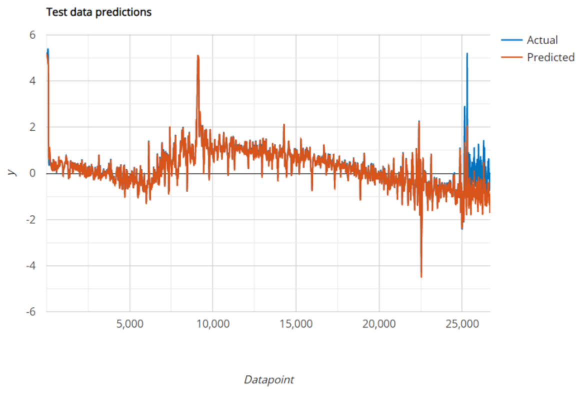
(b) Model 6



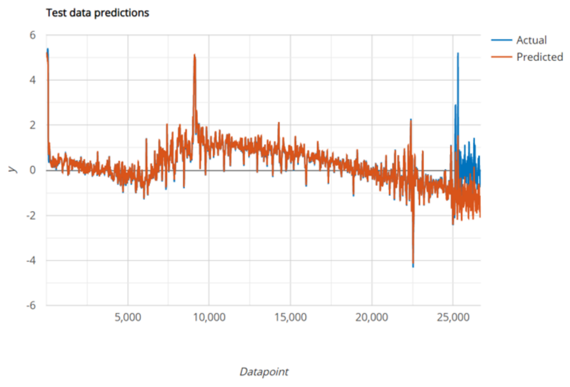
(c) Model 7



(d) Model 8



(e) Model 9



(f) Model 10

Figure 12: Scaled EGT testing predictions vs actual

C Appendix. Experiment 3 Results

In this appendix, the training and validation error metrics results are displayed in tables 15 and 16.

R^2	RMSE	MAE	MSE
0.99800	1.27580	0.74480	1.62768
0.99812	1.23542	0.75659	1.52627
0.99807	1.25291	0.76230	1.56978
0.99822	1.20218	0.75237	1.44524
0.99809	1.24816	0.73935	1.55789
0.99832	1.16893	0.75271	1.36640
0.99801	1.27283	0.78877	1.62009
0.99818	1.21660	0.73428	1.48011
0.99820	1.21099	0.72872	1.46649
0.99835	1.16035	0.71518	1.34641

Table 15: Experiment 3 training error metrics

R^2	RMSE	MAE	MSE
0.99792	1.29194	0.74879	1.66910
0.99799	1.26866	0.76180	1.60950
0.99801	1.26294	0.75916	1.59503
0.99810	1.23463	0.75536	1.52432
0.99802	1.26004	0.74229	1.58769
0.99823	1.19080	0.75385	1.41801
0.99793	1.28963	0.78988	1.66315
0.99812	1.22731	0.73710	1.50629
0.99813	1.22366	0.73682	1.49735
0.998197	1.202644	0.720882	1.446352

Table 16: Experiment 3 validation error metrics

Next, each symbolic expression model is detailed here:

$Y_3^1 = 0.114x_5 + 0.771x_6 + 0.0249x_{11} - 0.0763x13 + 0.395x_{19} + 1.62log(x_4 + e^{(-x_{23})} + 8.9) + 0.233e^{(-x_{23})} - 0.0534e^{(-x_{41})} - 1.36 * 10^{-5}e^{(-x_{103})} - 1.14e^{(-e^{(-e^{(-x_{19})})})} - 3.11$

$Y_3^2 = 0.0789x_4 + 0.123x_5 + 0.696x_6 + 0.0219x_{11} + 0.68x_{19} + 0.156x_{72} + 0.357e^{(-e^{(-x_4)})} + 0.29e^{(-e^{(-x_{37})})} + 0.54e^{(-{x_{19}}^4 * {x_{72}}^2)} + 0.393e^{(-{x_{37}}^2)} - 0.996$

$Y_3^3 = 0.0765x_4 + 0.683x_6 - 0.114x_{14} + 0.683x_{19} - 0.0189x_{21} - 0.0228x_{40} + 0.0371x_{41} + 0.159x_{72} - 0.585e^{(-e^{(-e^{(-x_4)})})} + 0.0371e^{(-e^{(-e^{(-x_{26})})})} - 0.136|x_{19} - 1.23| + 0.114e^{(-real(e^{(-x_{26})}))} - 0.699e^{(-real(e^{(-x_{72})}))} + 0.766$

$Y_3^4 = 0.139x_4 + 0.107x_5 + 0.698x_6 + 0.438x_{19} - 0.039x_{23} + 0.0508x_{41} - 0.0265x_{111} - 0.145e^{(-e^{(-e^{(-x_{11})})})} - 0.825e^{(-e^{(x_{19})})} - 2.66 * 10 - 4{x_4}^3 + 0.411$

$Y_3^5 = 0.0906x_4 + 0.642x_6 + 0.028x_{11} + 0.562x_{19} + 0.0752x_{37} + 0.0171x_{67} + 0.0338x_{74} - 0.791e^{(-real(e^{(-e^{(-x_{19})})}))} - 0.115|x_{37}| + (0.208 * 0.657^{x_{14}} * 0.657^{x_{74}})/0.657^{x_{108}} + 0.417$

$Y_3^6 = 0.101x_4 + 0.136x_5 + 0.654x_6 + 0.448x_{19} - 0.0226x_{21} + 0.0426x_{26} + 0.0484x_{41} - 0.0602x_{74} + 0.082x_{108} - 0.605e^{(-e^{(x_{19})})} + 0.224$

$Y_3^7 = 0.0831x_4 + 0.117x_5 + 0.695x_6 + 0.0233x_{11} + 0.461x_{19} - 0.0436x_{23} + 0.0436x_{41} + 0.0544x_{72} - 0.729e^{(e^{(-e^{(x_{19})})})} + 0.81e^{(-e^{(e^{(x_{19})})})} + 0.368e^{(-e^{(-x_4)})} + 0.883$

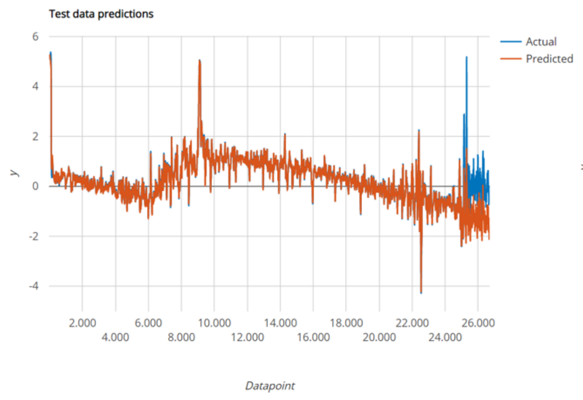
$Y_3^8 = 0.0596x_4 + 0.118x_5 + 0.671x_6 + 0.474x_{19} + 0.0446x_{41} + 0.0298x_{108} - 0.014x_{111} - 0.735e^{-e^{(x_{19})}} + 0.753e^{{(-{e^{(-x_4)}}^{1/2})}^{1/2}} - 3.52 * 10^{-6}{x_{111}}^3 x_4 + x_5 + {x_{21}}^3 - 0.175e^{{(-e^{(-x_{21})})}^{1/2}} - 0.0498$

$Y_3^9 = 0.106x_4 + 0.125x_5 + 0.689x_6 + 0.0261x_{11} + 0.45x_{19} + 0.0515x_{41} + 0.00826x_{58} -$

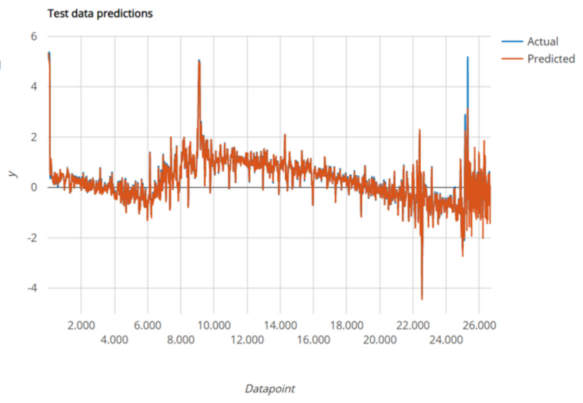
$0.744e^{(-e^{(x_{19})})} + 2.08 * 10^{-7}e^{(2x_{97})}|x_{111}|^2 + 0.00826x_5x_6 - 0.00826x_5x_{28} - 0.00511x_4x_{74} + 0.00826x_{38}x_{74} + 0.00511{x_4}^2e^{(-x_4)} - 0.176e^{(-x_4)^{1/2}} + 0.467$

$Y_3^{10} = 0.0825x_4 + 0.134x_5 + 0.688x_6 + 0.0174x_{12} + 0.47x_{19} + 0.0304x_{26} + 0.0409x_{41} - 0.523e^{(-e^{(-e^{(-x_4)})})} - 0.703e^{(-e^{(x_{19})})} + 2.93 * 10^{-4}{x_{72}}^3e^{(-x_1)} + 0.635$

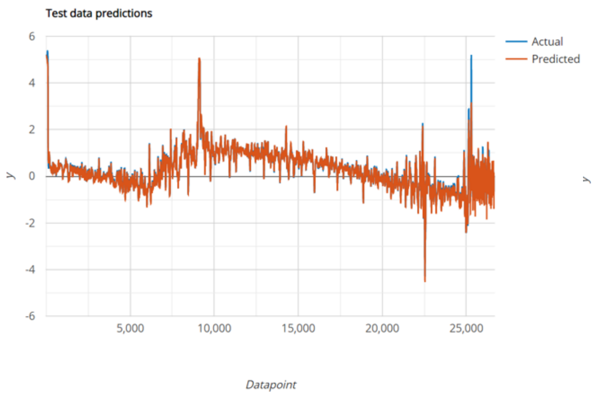
The figures in where the model predictions against the actual values of the EGT per model are displayed below:



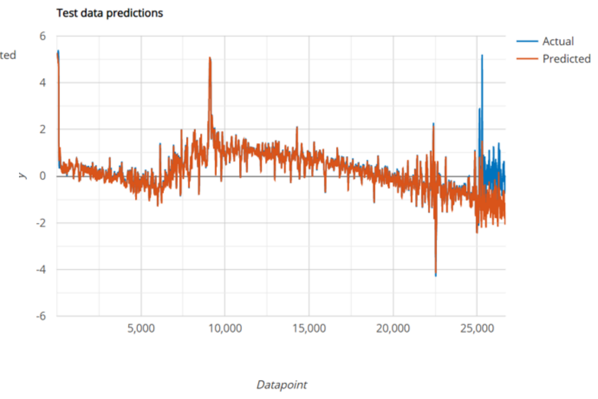
(a) Model 1



(b) Model 2

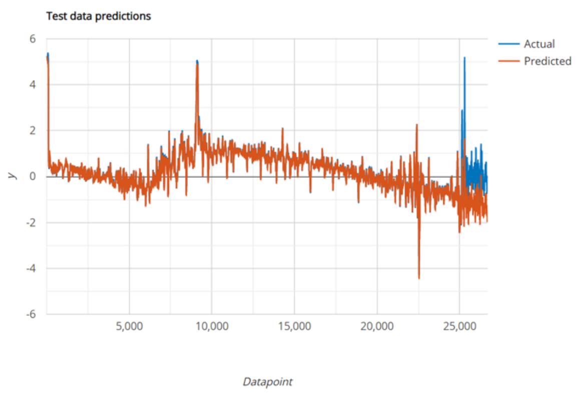


(c) Model 3

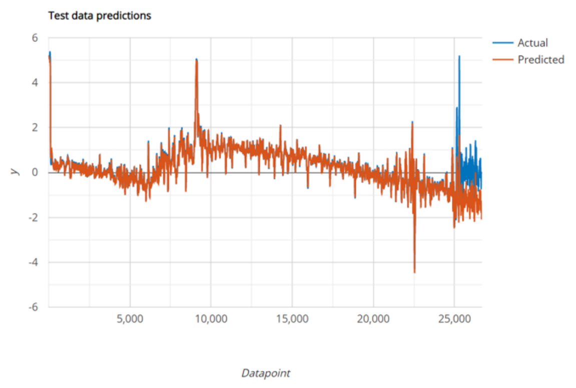


(d) Model 4

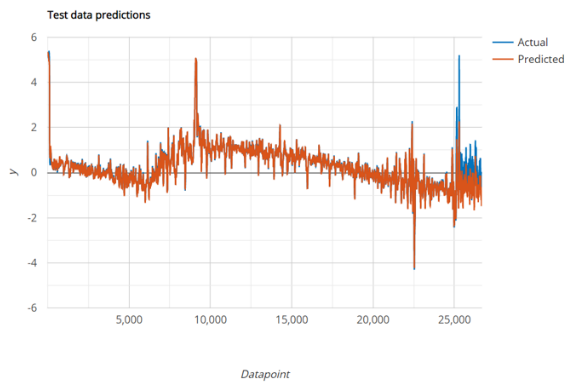
Figure 13: Scaled EGT testing predictions vs actual



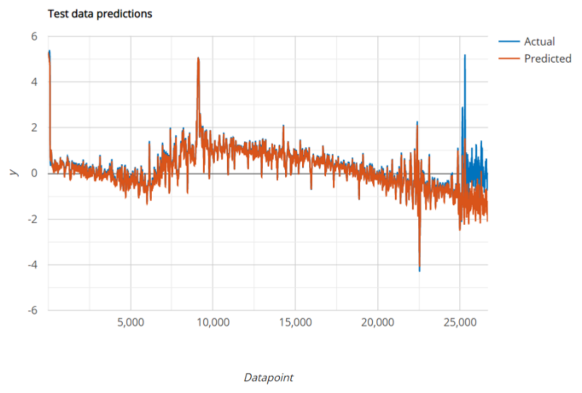
(a) Model 5



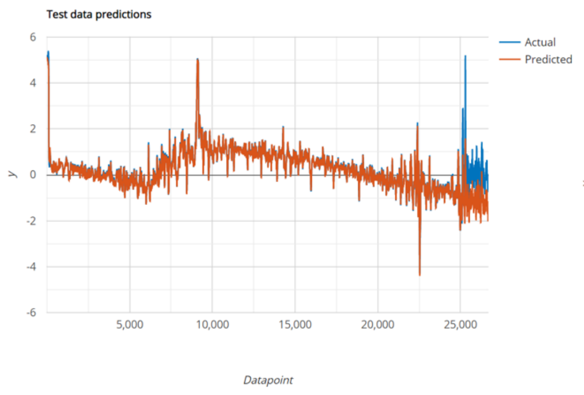
(b) Model 6



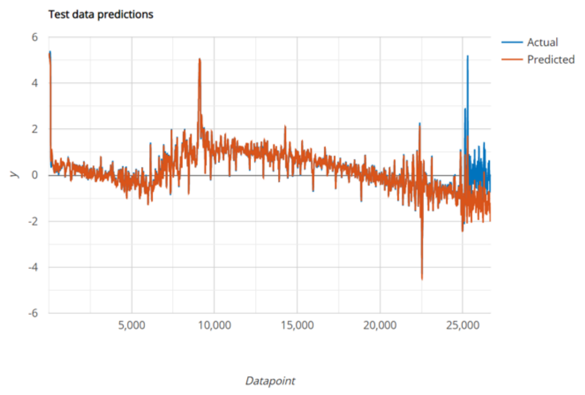
(c) Model 7



(d) Model 8



(e) Model 9



(f) Model 10

Figure 14: Scaled EGT testing predictions vs actual