KE 5207:

Computational Intelligence II, 2018

CA - Genetic Algorithm

Optimised

Mathematical Equation Using

Genetic Algorithm

Team Name:

Avengers

Members:

Chin Weng Khin (A0178203J)

Goh Hui Kiang (A0178382U)

Lim Kim Chwee (A0178196M)

MASTER OF TECHNOLOGY IN

KNOWLEDGE ENGINEERING

TABLE OF CONTENTS

[1 Business Understanding 1](#_Toc527576736)

[1.1 Objective 1](#_Toc527576737)

[1.2 Project Plan 1](#_Toc527576738)

[2 Data Understanding 2](#_Toc527576739)

[2.1 Data Collection and Exploration 2](#_Toc527576740)

[3 Data Preparation 2](#_Toc527576741)

[3.1 Data Pre-processing 2](#_Toc527576742)

[4 Modelling 3](#_Toc527576743)

[4.1 Selecting Modelling Technique 3](#_Toc527576744)

[4.2 Generating Test Design 3](#_Toc527576745)

[4.3 Building Models 5](#_Toc527576746)

[5 Evaluation 8](#_Toc527576747)

[6 Conclusion 12](#_Toc527576748)

[7 Deployment 12](#_Toc527576749)

[8 Future Work 13](#_Toc527576750)

[9 References 13](#_Toc527576751)

# Business Understanding

This project will follow the CRISP-DM framework.

## Objective

There are various techniques to generate, implement and, finally, identify a “best fit” model to solve a mathematical problem/equation of a given dataset. Generally, the process includes a series of manual generation, implementation and evaluation tasks. As such, these techniques may fail when more complicated functions or models are required [1]. This project explores an alternative method that uses genetic algorithm to generate models of varying complexity and ultimately to identify a “best fit” model to solve an equation of a given dataset.

According to [2], genetic algorithm is a search algorithm that follows the principals of natural evolution. It is a combination of a competitive rule (“struggle for life”) and a selection rule (“survival of the fittest”), which is often used as a mechanism for evolutionary problem solving. For example, a population may be a set of simulated computer-coded candidate solutions to solve a complex problem. The evolutionary process consists of the set of candidate solutions competing against each other and reproducing new solutions, leading to the creation of fitter solutions that can best solve the given problem.

[2] classified general applications into three main classes and they are numerical parameter estimation, sequencing, and subset selection. Numerical parameter estimation usually involves mathematical models that defines a relationship between input data to the parameter values. In many cases, an iterative approach is conducted to generate candidate solutions to find one with the optimal parameter values to fit the input data. This iterative approach can be defined as optimisation. Sequencing and subset selection usually involve solving combinatorial problems such as scheduling, inventory or capacity planning.

In a study by [3], genetic algorithm was used as one of the techniques to perform numerical parameter estimation, in which, the objective was to optimise a simple linear equation. The optimisation method is conducted following three factors; an objective function (to be minimised or maximised), a set of variables (that affects the objective function), and a set of constraints (variables that take on a certain values).

Given a simple linear equation of a + 2b + 3c = 10, [3] used genetic algorithm to:

1. identify the best value for a, b, and c (variables),
2. between integers 0 to 10 (constraints),
3. that can obtain f(x) = 10 (objective function).

After some exploration, the optimal values are a = 1, b = 0, and c =3 using the following GA parameters:

1. Population size = 14
2. Chromosome length = 3
3. Maximum of generation = 10
4. Crossover probability = 0.25
5. Mutation probability = 0.01

## Project Plan

From the above study, it is seen that GA has the ability to obtain optimal solutions for solving linear equation with relatively simple parameter settings. In this project, we attempt to perform similar tasks. In this case, the objectives are:

1. To find an appropriate polynomial equation for any given dataset, as well as,
2. To identify an optimum solution for the given equation using GA.

However, as compare to the previous study which used a relatively simple equation to meet a single f(x) value, this project aims to increase the complexity of the equation such that it is able to optimise and create “a perfect fit” with any given dataset (multiple f(x)). To do so, numerous hypothetical equations must be generated and tested where; for every equation defined, one round of GA is applied. In this case, the same optimisation method is conducted; applying similar rules (three factors above) to obtain the least error in predicting each given objective function (f(x)).

# Data Understanding

## Data Collection and Exploration

The dataset is a sample collection of monthly shipment shared by a real company in Singapore. The sample collection contains a total of 24 data points. Figure 2.1 shows a sample plot of the observed raw dataset.

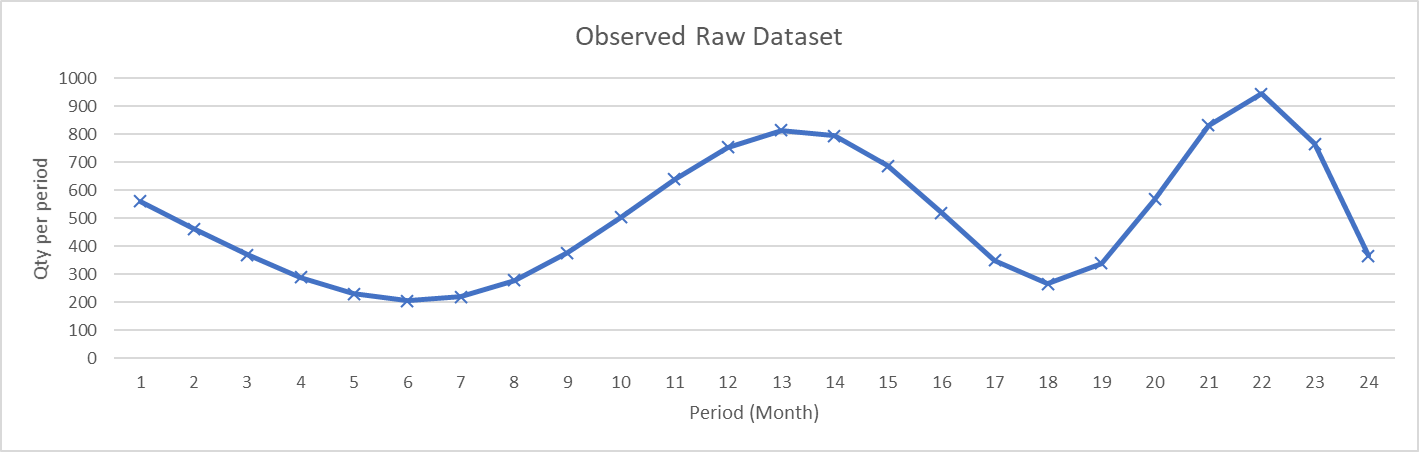


Figure 2.1. Raw dataset plot

# Data Preparation

## Data Pre-processing

In the initial stage of test runs, it was found that modelling with the raw values did not yield any satisfactory results. This is mainly due to the high standard deviation and variance in the raw dataset. Therefore, a feature scaling approach, the Min-max normalisation method, was chosen to reduce this variance before allowing GA to provide an optimum solution.

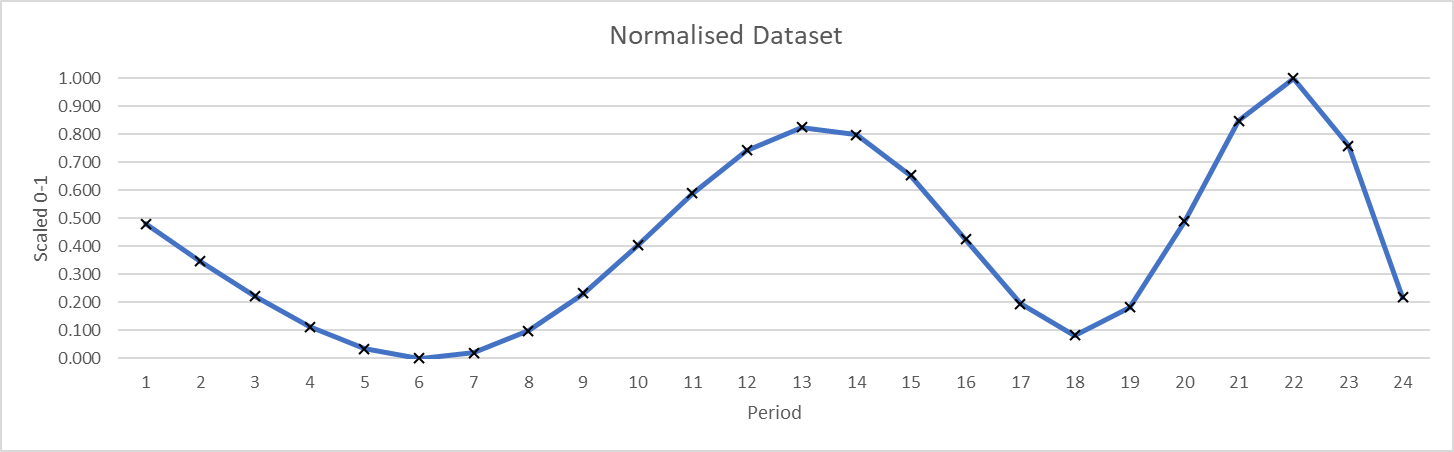


Figure 3.1. Min-Max Normalised dataset plot

# Modelling

## Selecting Modelling Technique

Python was used to generate a GA for this project.

## Generating Test Design

In this work, following [3], the test approach is to develop multiple equations to solve the given dataset. The formulated equations include:

1. a + b*x*2 + c*x*
2. a + b*x*2 + c*x*3 + d*x*
3. a x cos(b*x*c)
4. a*x* + b*x* x cos(c + d*x*2)
5. (a*x*2 + b*x*) x cos(*x*)2 + d + e*x*3 + f*x*2 + g*x*
6. (a*x*3 + b*x*2 + c) x cos(*x*)2
7. (a*x*3 + b*x*2 + c) x cos(*x*)2 + d + e*x*3 + f*x*2 + g*x*
8. (a*x*3 + b*x*2 + c) x cos(*x*)2 + d + e*x*3 + f*x*2 + g*x* + (h*x*3 + i*x*2 + j*x*) x cos(*x*)2
9. (a*x*3 + b*x*2 + c) x sin(*x*)2 + d*x*3 + e*x*2 + f*x* + g
10. (a*x*2 + b*x* + c) x sin(d*x*2 + e*x* + f) + g
11. (a*x*2 + b*x*) x cos(c*x*d) + (e*x*2 + f*x*) x sin(g*x*h)
12. exp(a*x*) x sin(*x*) + b
13. a x sin(*x*) + exp(b*x*)

These equations are plotted using random generated x-values shown in Table 4.2 (To see codes for plot generation, refer to deployment section). Equations 1 and 2 seem to predict values that follows an exponential-typed linear trend. Equation 3 predicted values that follows a cosine-type trendline where variables (i.e. a, b, c) are constant. Equations 4 to 8 followed a relatively similar cosine-type but in the addition of linear trendline.

In order to generate a “best-fit” model, these were not ideal as they did not indicate the ability to vary in “curve waves” like those of a polynomial equation. As such, sin function was introduced to the equation and there were indications of varying waves. Finally, equation 10 is selected as the optimal equation as it is the least computationally complex of the latter (11 to 13 has two mathematical functions (cos + sin) while 10 only has one).

Table 4.2: Equations and their Plots

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Equation** | **Random Plot 1** | **Random Plot 2** | **Random Plot 3** | **Random Plot 4** |
| 1 and 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |
| 5 |  |  |  |  |
| 6 |  |  |  |  |
| 7 and 8 |  |  |  |  |
| 9 |  |  |  |  |
| 10 |  |  |  |  |
| 11 |  |  |  |  |
| 12 |  |  |  |  |
| 13 |  |  |  |  |

## Building Models

1. Given the selected equation, f(*x*) = (a*x*2 + b*x* + c) x sin(d*x*2 + e*x* + f) + g, the next step is to use GA to:
2. identify the best value for a, b, c, d, e, f, g (7 variables),
3. between real numbers -1 to 1 (constraints),
4. that can obtain f(x) ≈ dataset values (objective function) with an overall error rate (RSME) of less than 0.1.

### Gene/Chromosome

The element of the candidate solution is the variable or coefficient of the selected equation. As there is a total of seven coefficients, the chromosome is a combination of seven genes in real numbers constrained between -1.00 to 1.00. Figure 4.3.1-1 shows the chromosomes representation. Each gene is randomly initiated with a real number (Figure 4.3.1-2).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| a | b | c | d | e | f | g |

Figure 4.3.1-1. Chromosomes



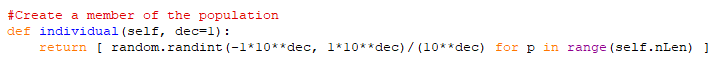


Figure 4.3.1-2. Random initialisation for each gene

Each chromosome can be represented as follows:

Chromosome[001] = [a,b,c,d,e,f,g] = [0.1,0.2,0.3,0.4,-0.57,0.6,-0.7]

Chromosome[002] = [a,b,c,d,e,f,g] = [0.5,-0.3,0.76,-0.4,0.6,0.9,0.1]

…

Chromosome[100] = [a,b,c,d,e,f,g] = [-0.3,0.6,0.5,0.2,0.96,-0.3,0.8]

### Population

The chosen population size is 100. Figure 4.3.2-1 shows the population representation. Figure 4.3.2-2 shows the code snippet for population generation.

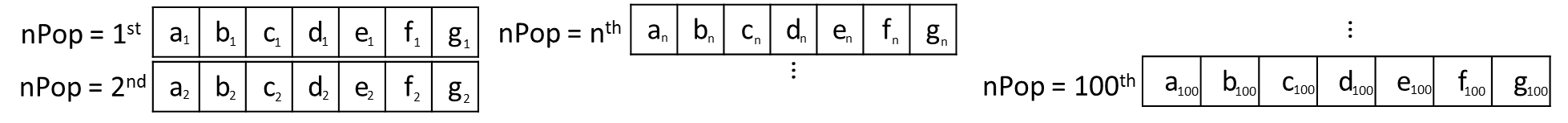


Figure 4.3.2-1. Population representation





Figure 4.3.2-2. Generate population of 100

### Fitness Function

A fitness function is calculated for every individual or chromosome [3]. Here, the overall root mean square error is computed for every chromosome. The concept is to find the optimal coefficient that can generate a predicted *y* that is closest to the actual *y*. Hence, for each chromosome, there are a total of 24 “sub-chromosomes” or function objectives (F\_obj). Figure 4.3.3 shows a snippet of the fitness function defined for this GA.

For example,

Given nPop = 1 and Chromosome[001] = [0.1,0.2,0.3,0.4,0.5,0.6, 0.7]:

F\_Obj[01: *x=1*,*y=0.48*] = {[(0.1\*12 + 0.2\*1 + 0.3) x sin(0.4\*12 + 0.5\*1 + 0.6) + 0.7] – 0.48}

F\_Obj[02: *x=2*,*y=0.35*] = {[(0.1\*22 + 0.2\*2 + 0.3) x sin(0.4\*22 + 0.5\*2 + 0.6) + 0.7] – 0.35}

…

F\_Obj[24: *x=24*,*y=0.22*] = {[(0.1\*242 + 0.2\*24 + 0.3) x sin(0.4\*242 + 0.5\*242 + 0.6) + 0.7] – 0.22}

Get RSME for nPop[1], Chromosome[001]:

Fitness[F\_Obj(1 to 24)] =

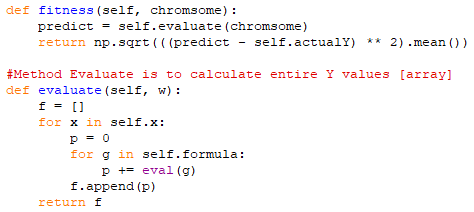


Figure 4.3.3. Fitness function

### Selection Method

The selection scheme will aid in determining the parent chromosomes for mutation and generating new individuals [3]. There are various methods to perform selection and some of the common ones include roulette wheel selection, sigma scaling techniques, tournament selection, elitist model or ranking methods.

In this work, the roulette wheel was first considered as the selection method. However, after some exploration, it was found that this technique has a local optimum issue where the selected individuals may not be the fittest. This caused a downstream impact with increased learning rate; more time is required to process the mutation to breed better children to achieve a low RSME.

The elitist model was also adopted to preserve the top n% chromosomes but the learning rate was still rather slow.

Eventually, the final selection method includes a combination of three methods and they are ranking, elitist, and tournament methods. The idea was to find an optimum method to reduce learning rate and time taken to achieve low RSME.

First, the ranking method is used via sorting the population by their fitness function (low to high RSME). Then, a modified version of the “elitist model” approach is adopted. This is done by keeping the top 20% chromosomes, and, other than preserving it for the next iteration, this 20% is used for mutation. Thereafter, the tournament selection method is used where two individuals from the top 20% are randomly selected at every iteration for mutation. This allowed the computation time to reduce significantly (at least 18 times faster) whilst achieving similarly good results at specified number of iterations. Figure 4.3.4 shows the selection method.

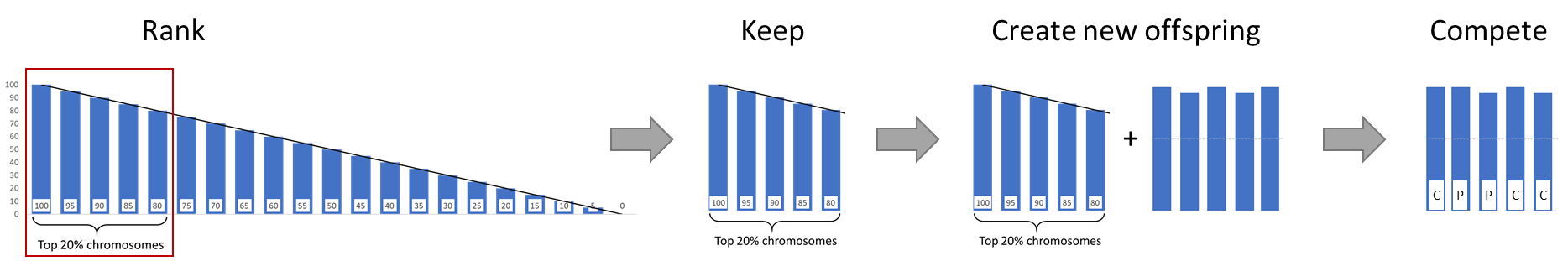


Figure 4.3.4. Selection method

### Crossover, Mutation, and Replacement

The top 20% are used as parent individuals to mutate and breed new children. There are various ways to perform the crossover and mutations [3]. For our work, the two-point crossover operator is chosen. First, two random parents are selected. Next, the first half of parent one is extracted and the second half of parent two is extracted to form a new child. Thereafter, a uniform random mutation logic used to replace genes in each child chromosome.

With the offspring generated, the tournament selection mode is implemented (survival of fittest - Parent 1, Parent 2, Child) to retain the “fittest” individual. Subsequently, new chromosomes will re-generate to replace the rest of the chromosomes to form a full population.

Figure 4.3.5-1 shows the crossover and mutation process. Figure 4.3.5-2 shows the replacement process to get the full population that is ready for the next iteration. Figure 4.3.5-3 shows the codes for selection, crossover, and mutation functions.

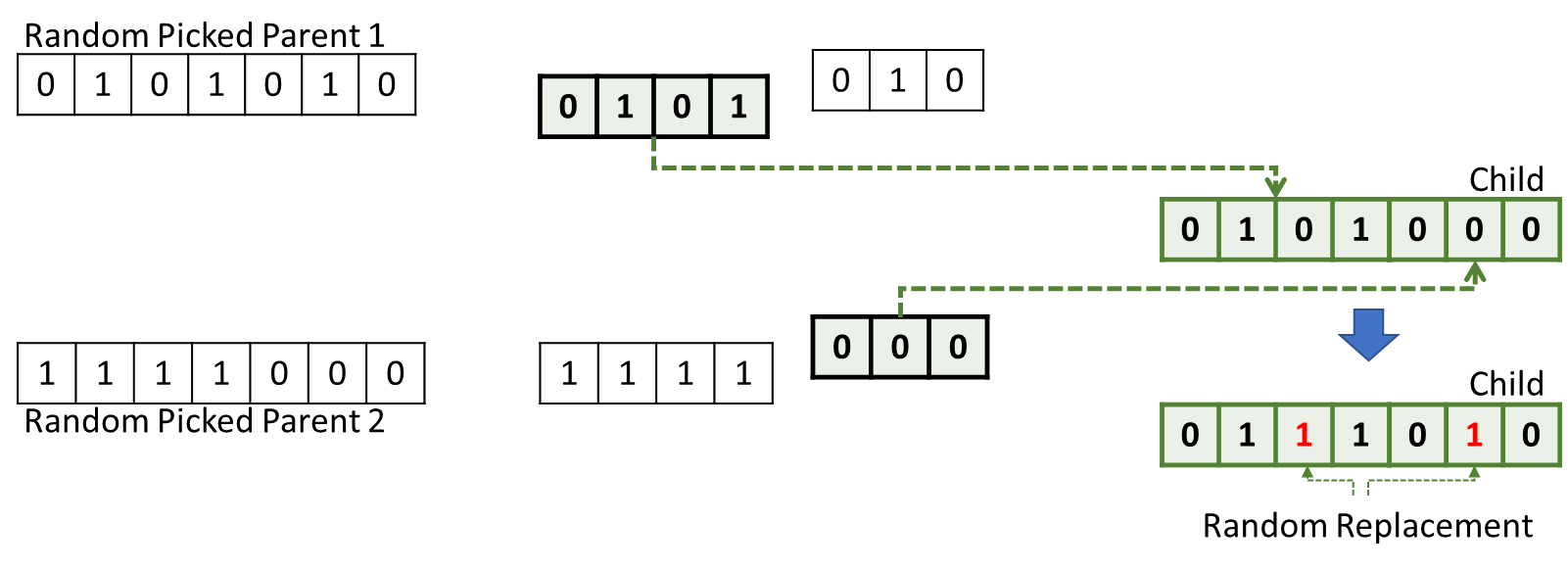


Figure 4.3.5-1. Crossover & mutation process

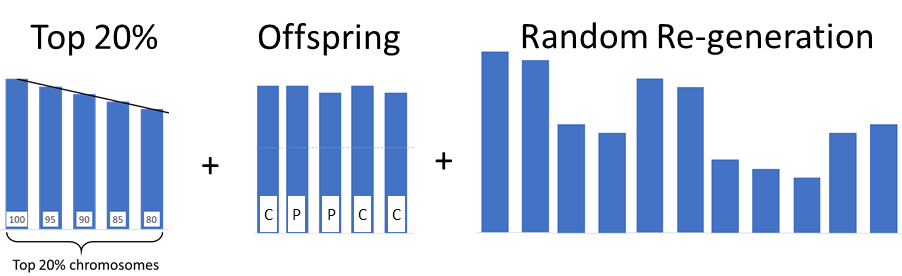
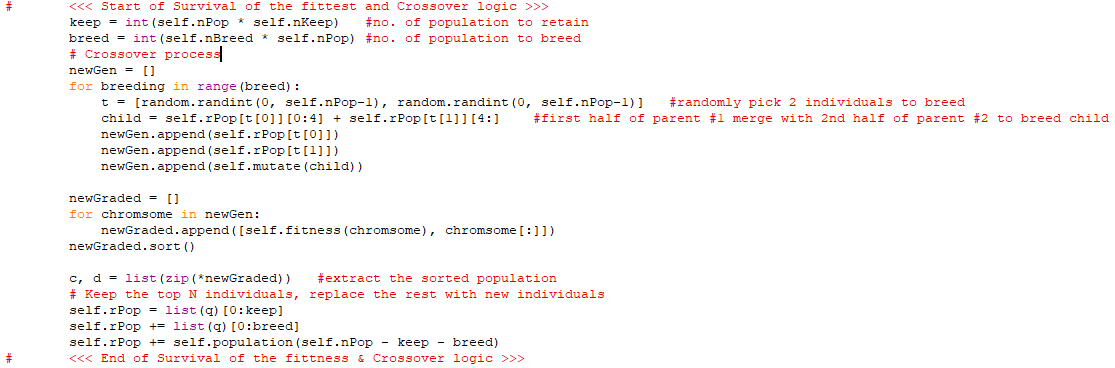


Figure 4.3.5-2. Replacement process



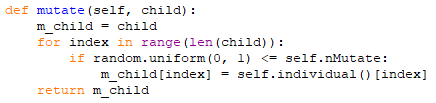


Figure 4.3.5-3. Selection, crossover, and mutation functions

### Parameters Tuning

The current parameters settings designed for this GA includes:

* Population size = 100
* Chromosome length = 7 (seven co-efficient to optimise)
* Survival Rate = 0.2 (percentage of population to keep)
* Mutation Rate = 0.4 (Mutation probability)
* Breeding Rate = 0.2 (percentage of population to generate offspring)
* Random seed = 5 (to allow replication of results)

# Evaluation

As mentioned in the fitness function, the RSME is used to optimise the equation and maximise the objective functions. A learning curve is generated to observe the improvement in chromosome generation over 5,000 iterations/epoch.

Figure 5.1 to 5.4 shows the equation, changes and improvement over time and iterations. At the 40th iteration, the fitted plot shows a huge difference between actual and predicted y or f(x) value with an error of 0.793. The learning curve shows a significant drop from starting error of approximately 3.4 to less than 1.0 at the 7th iteration. Subsequently, the model begins to stabilise with a consistent error of less than 1.0. At the 1100th iteration, the fitted plot shows a good improvement, with the predicted y hovering near the actual y. The error has also reduced from 0.8 to less than 0.5 at 0.191. At the 3160th iteration, the model continues to improve, with the predicted y fitting even closer to the actual y. At this point, the error rate is retained within the 0.1 to 0.19 range.

In order to understand if GA can continue to improve the model, the system was allowed to process further and at the 5320th iteration, the error rate drop to less than 0.1 at 0.094. At this point, a decision was made to stop the system as the model has managed to achieve the objective result of RSME of less than 0.1.

Apart from the improvements in predicted values and error rate, it can be observed that there are significant changes to the coefficients of the equations from the 40th to 1100th iteration. Subsequently, coefficient *a* and *b* stopped changing, followed by *c*, *d*, and *g* at 3160th iteration. At 5320th iteration, only coefficients *e* and *f* required further mutation.

Table 5.1 shows the predicted values generated at the four different iterations. Although many of the predicted values has a small difference as compared to the actual value, there are still some datapoints with huge error (i.e. datapoint 8 and 18). This is mainly due to the huge variance for such datapoints. In real case scenarios in demand and forecasting issues, this is normally known as irregular fluctuation that happens due to unexpected events such as disasters or terror attacks, causing a huge change in demand for the affected period. This is usually a key limitation for any forecasting problem.

|  |  |
| --- | --- |
| **At 40th Epoch** | |
| f(x) = (0.1*x*2 + -0.8*x* + 0.9) x sin(-0.1*x*2 + 0.6*x* + 0.2*x*) + 1.0 | |
| Fitted predicted Y against actual Y | RSME = 0.7929027248390265 |
|  |  |

Figure 5-1. Equation, Plots (Actual vs Predicted Y – Main, RSME – learning curve) at 40th epoch

|  |  |
| --- | --- |
| **At 1100th Epoch** | |
| f(x) = (0.0*x*2 + 0.0*x* + 0.3) x sin(-0.1*x*2 + 0.3*x* + 0.4*x*) + 0.4 | |
| Fitted predicted Y against actual Y | RSME = 0.19148997043929758 |
|  |  |

Figure 5-2. Equation, Plots (Actual vs Predicted Y – Main, RSME – learning curve) at 1100th epoch

|  |  |
| --- | --- |
| **At 3160th Epoch** | |
| f(x) = (0.0*x*2 + 0.0*x* + 0.4) x sin(-0.1*x*2 + -0.4*x* + 0.4*x*) + 0.4 | |
| Fitted predicted Y against actual Y | RSME = 0.1403681840670127 |
|  |  |

Figure 5-3. Equation, Plots (Actual vs Predicted Y – Main, RSME – learning curve) at 3160th epoch

|  |  |
| --- | --- |
| **At 5320th Epoch** | |
| f(x) = (0.0*x*2 + 0.0*x* + 0.4) x sin(-0.1*x*2 + -0.3*x* + 0.2*x*) + 0.4 | |
| Fitted predicted Y against actual Y | RSME = 0.09366632655816118 |
|  |  |

Figure 5-4. Equation, Plots (Actual vs Predicted Y – Main, RSME – learning curve) at 5320th epoch

Table 5-1: actual versus f(x) values at different iteration/epoch

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **SNo.** | **Actual Y** | **Predicted Y [f(x)]** | | | |
| **Epoch:40** | **Epoch: 1100** | **Epoch: 3160** | **Epoch: 5320** |
| 1 | 560 | 1,178.80 | 516.83 | 555.77 | 320.53 |
| 2 | 462 | 1,244.26 | 474.41 | 482.28 | 263.09 |
| 3 | 369 | 1,168.56 | 419.04 | 390.62 | 195.55 |
| 4 | 289 | 1,019.35 | 351.78 | 284.90 | 124.02 |
| 5 | 231 | 847.38 | 276.52 | 174.77 | 58.28 |
| 6 | 206 | 685.43 | 201.11 | 76.47 | 12.05 |
| 7 | 220 | 552.24 | 137.83 | 12.12 | 1.70 |
| 8 | 278 | 457.75 | 102.59 | 5.96 | 43.11 |
| 9 | 377 | 407.55 | 111.89 | 76.87 | 145.91 |
| 10 | 504 | 405.40 | 177.31 | 227.75 | 306.28 |
| 11 | 640 | 453.40 | 298.01 | 435.11 | 500.17 |
| 12 | 754 | 549.77 | 453.86 | 644.86 | 681.46 |
| 13 | 814 | 684.66 | 603.56 | 781.74 | 789.91 |
| 14 | 795 | 835.05 | 692.64 | 776.67 | 771.77 |
| 15 | 688 | 961.25 | 674.07 | 606.79 | 609.75 |
| 16 | 519 | 1,008.96 | 536.76 | 328.85 | 349.48 |
| 17 | 350 | 921.87 | 327.79 | 77.27 | 101.32 |
| 18 | 267 | 668.36 | 147.99 | 5.48 | 0.14 |
| 19 | 340 | 280.27 | 107.14 | 184.29 | 126.70 |
| 20 | 567 | -110.27 | 250.33 | 519.65 | 431.04 |
| 21 | 832 | -264.34 | 501.56 | 774.94 | 724.28 |
| 22 | 944 | 102.01 | 684.77 | 731.69 | 784.42 |
| 23 | 765 | 1,164.35 | 643.58 | 396.38 | 536.21 |
| 24 | 367 | 2,774.39 | 389.91 | 56.34 | 163.17 |

# Conclusion

Genetic algorithm has shown its ability to improve and optimise equations while minimising error rate. It was able to continuously improve the objective functions through the concept of “survival of the fittest”, searching and keeping good chromosomes, breeding stronger offspring, while eliminating weaker individuals.

However, akin to the study by [3], to achieve the desired objective results/goals (i.e. our work of less than .1 error rate), much time is required to explore and select an appropriate mathematical equation to fit a given dataset. Careful planning and scaling of variables and constraints is highly critical for both time and accuracy performance.

Overall, the algorithm is easy to understand as it uses an intuitive approach to search for optimum solution. The algorithm allows a good level of customisation which allowed this work to reach its objective goal.

# Deployment

There is a total of two executable python files:

1. Equation\_test-plots.py: This program allows the user to form new equations and evaluate their performance on randomly generated datasets.
2. GA\_onSelectedEquation.py: This program allows the user to run genetic algorithm using the selected equation from earlier.

Figure 7-1 shows a sample output of Equation\_test-plots.py. The selected equation will be plotted with changing randomly generated dataset at every frame at a specified interval. The rate of change per frame can be adjusted by amending the interval (in miliseconds) of this line of code:

anim = animation.FuncAnimation(fig, animate, frames=2000, interval=***500***)

Figure 7-2 shows a sample output of GA\_onSelectedEquation.py. User can use any dataset, adjust the parameters, execute and observe the changes to the equation and performance over iterations. The user will need to perform a manual termination of program once the desired result/goal is achieved.

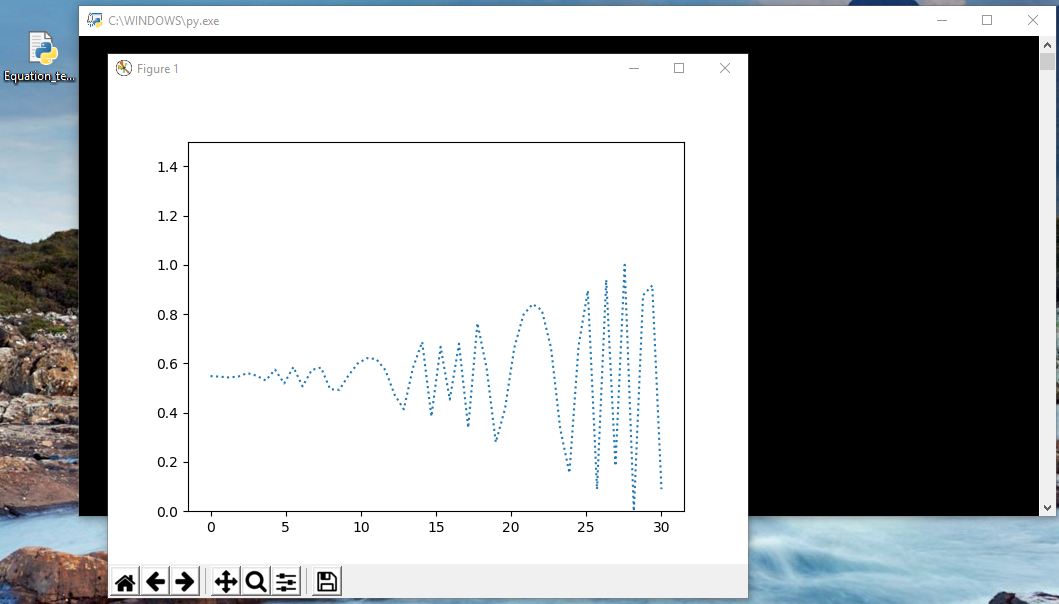


Figure 7-1. Equation\_test-plots.py

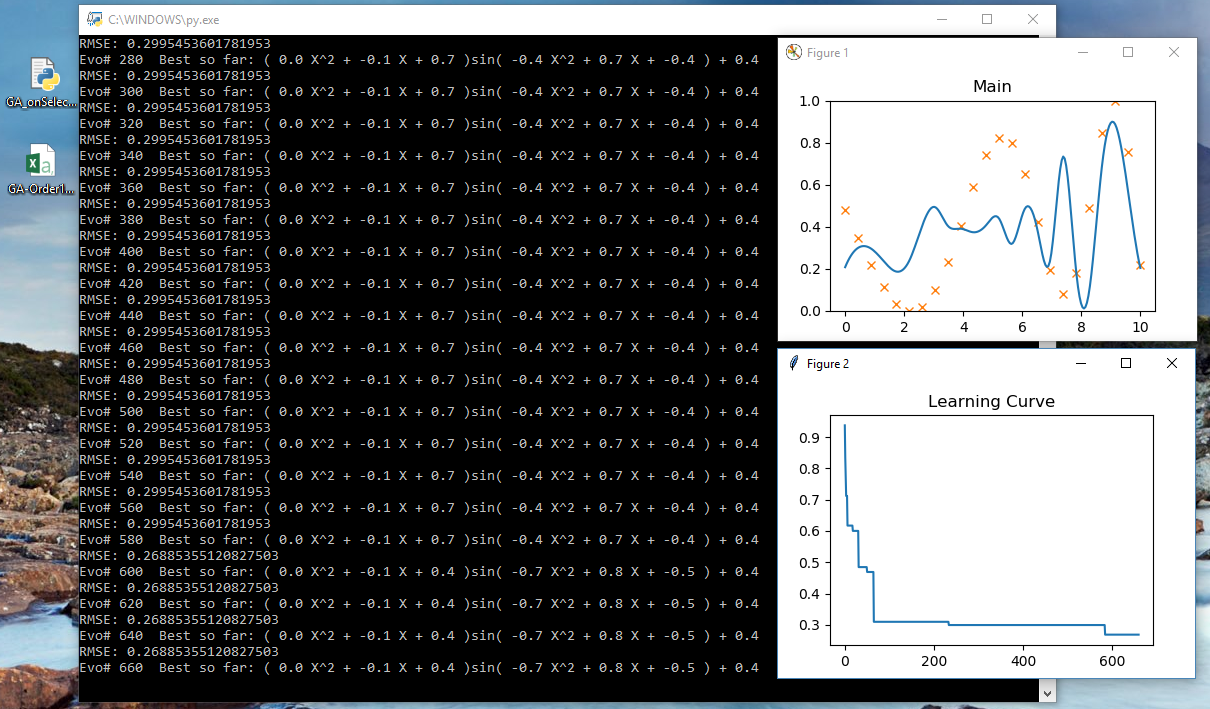


Figure 7-2. GA\_onSelectedEquation.py

# Future Work

The derived equation is suitable for implementation in optimisation problems such as forecasting demand of goods or products. Hence, the next step is to obtain a dataset of real goods or products demand, formulate an appropriate equation, and use the current GA system to generate the optimum function objective and variables (coefficients) that can best forecast the demand. For the forecasting task, it will be appropriate to use a 70/30 train-test approach to evaluate and validate the performance of the model on unseen dataset.

Further enhancement can also be done to create a hybrid system that can use GA to automate the selection of “best fit” equation for a given dataset before the next step of finding optimum coefficients to the “best fit” equation.

# References

|  |  |
| --- | --- |
| [1] | M. L. K Messa, “Curve Fitting Using Genetic Algorithms,” NAVAL OCEANOGRAPHIC AND ATMOSPHERIC RESEARCH LAB STENNIS SPACE CENTER MS, 1991. |
| [2] | G. K. C.B. Lucasius, “Understanding and using genetic algorithms,” *Chemometrics and intelligent laboratory systems,* vol. 19, no. 1, pp. 1-33, 1993. |
| [3] | L. Z. Bashir, “Solve Simple Linear Equation using Evolutionary,” *World Scientific News 19,* pp. 148-167, 2015. |