Techno-Science

Scientific Journal of Mehmet Akif Ersoy University

https://dergipark.org.tr/pub/sjmakeu

Original Research Article

hosted by Turkish JournalPark ACADEMIC

GENETIC ALGORITHM IN SOLVING MULTI-DIMENSIONAL POLYNOMIAL FUNCTION FIT TO EXPERIMENTAL DATA

Uchenna IGBOELI[1*]

[1]*Department of Computer Science, University of Abuja, Nigeria*

ARTICLE INFO

Article History

Received : 29/03/2023
Revised : 21/06/2023
Accepted : 26/06/2023
Available online : 03/07/2023

Keywords

Polynomial, Algorithm, Chromosomes, Crossover, Mutation, Optimization

ABSTRACT

A Polynomial Genetic Algorithm (PGA) is a type of evolutionary algorithm used for optimization problems that involve finding the minimum or maximum of a polynomial function. The algorithm is based on the principles of natural selection and genetic recombination and mutation. The algorithm starts by initializing a random population of chromosomes. The fitness of each chromosome is evaluated based on the value of the polynomial function it represents. The fittest chromosomes are selected for reproduction, and their genetic material is combined through crossover and mutation to produce a new generation of chromosomes. One important consideration in using a genetic algorithm for polynomial optimization is the choice of representation for the chromosomes. Binary or integer representations can be used, with each bit or integer representing a coefficient in the polynomial. Alternatively, a floating-point representation can be used, with each chromosome representing a set of coefficients that can be used to construct the polynomial.

One advantage of using a genetic algorithm is that it can find solutions to problems that are difficult or impossible to solve using traditional methods. It can also be used to solve problems that have multiple objectives or constraints. In summary, to solve a polynomial using a genetic algorithm, we need to define a fitness function that evaluates the fitness of each chromosome based on its ability to represent a good solution to the polynomial, and then use standard genetic algorithm techniques to evolve a population of chromosomes towards a solution. The solution found in this work shows that though genetic algorithm can be used to solve polynomials, other methods like Newton-Ralpson, Secant, Regula-falsi and Bisection can easily guess the solution in a few iterations thereby saving cost and time.

1. INTRODUCTION

Genetic Algorithm (GA) is a search-based optimization technique based on the principles of genetics and natural selection. It is frequently used to find optimal or near-optimal solutions to difficult optimization problems. Nature has always been a great source of inspiration to all mankind. Genetic algorithms represent a branch of computing called evolutionary computation since they imitate the biological processes of reproduction and natural selection to solve for the 'fittest' solutions. Like in evolution, many of a genetic algorithm's processes are random, however this optimization technique allows one to set the level of randomization and the level of control. Genetic algorithms are majorly used in solving NP (Non-deterministic Polynomial Time) problems and it involves the use of optimization techniques in finding solutions to such complex problems. Optimization refers to finding the values of inputs in such a way that we get the "best" output values. The definition of "best" varies from problem to problem, but in mathematical terms, it refers to maximizing or minimizing one or more objective functions, by varying the input parameters. Using GA, the set of all possible solutions or values which the inputs can take make up the search space. In this search space, lies a point or a set of points which gives the optimal solution. This work concentrates on the use of genetic algorithm in solving polynomials which other known algorithms (Newtons Method, Secant Method, Regula Falsi) can as well solve. The adoption of the GA is to find an optimal solution within a reasonable time frame especially when complex solutions are involved.

* Corresponding Author: uchenna.igboeli@uniabuja.edu.ng

To cite this article: Igboeli U., (2023). Genetic Algorithm in Solving Multi-Dimensional Polynomial Function Fit to Experimental Data. *Scientific Journal of Mehmet Akif Ersoy University*, vol. 6, no. 1- 20-28

e-ISSN 2651-3722

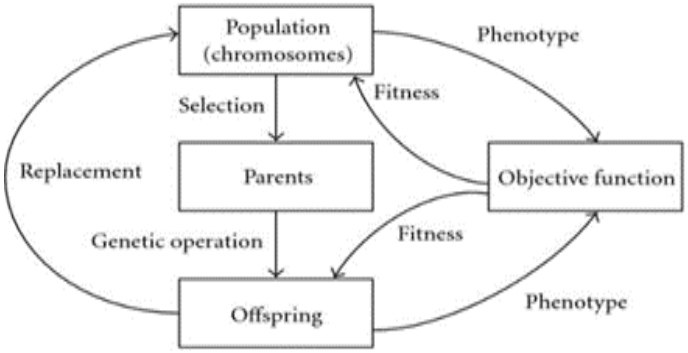

Fig 1. Genetic Algorithm Cycle Source [1]

1.1 Biological Background

A genetic algorithm is used to solve complicated problems with a greater number of variables and possible outcomes/solutions. Fig 1 above shows the evolution of the algorithm. Genetic Algorithms (GAs) and Evolution strategies (ESs) are two strata of consciously pursued attempts to imitate principles of organic evolution processes as rules for optimum seeking procedures. Both rely upon the collective learning paradigm gleaned from natural evolution [2]. GA process of natural selection starts with the selection of fittest individuals from a population of eligible individuals all meeting or satisfying the initial condition for eligibility. These selected individuals will produce offspring which inherit the characteristics of the parents and will be added to the next generation. If parents have better fitness, their offspring will be better than parents and have a better chance at surviving. This process keeps on iterating and at the end, a generation with the fittest individuals will be found. [3] developed a GA with new features in chromosome encoding, crossover, mutation, selection as well as algorithm structure that is developed to be able to ''learn'' from its experience. This is based on the fact that integration of production planning and scheduling is a class of problems commonly found in manufacturing industry and a combinatorial, NP-hard problem, in a related study, [4] examines the use of a genetic algorithm to optimize the functional form of a polynomial fit to experimental data; the aim being to locate the global optimum of the data. A genetic algorithm was also described which takes a set of data and searches for the best possible functional form of a polynomial fit to the data. GA provides search method efficiently working on population and has been applied to many problems of optimization and classification [5]. General GA processes are as follows:

1. Initial population
2. Fitness function
3. Selection
4. Crossover
5. Mutation

```
START
Generate the initial population
Compute fitness
REPEAT
Selection
Crossover
Mutation
Compute fitness
UNTIL population has converged
STOP
```

Fig. 2 Pseudocode GA Source [6]

Fig 2 above is a pseudocode of the life of a genetic algorithm implementation to problem solving. It shows the various stages and procedures that are undertaken in the course of the execution of a genetic algorithm-based solution.

1.2 Initial Population

GA processes begins with a set of individuals which is called a Population. Each individual is a solution to the problem. An individual is characterized by a set of parameters (variables) known as Genes. Genes are joined into a string to form a Chromosome (solution). The set of genes of an individual is represented using a string. Chromosome encoding can be achieved by a variety of encoding methods and the choice of method is usually dependent on the optimization problem

structure and precision requisites. Some of the popular encoding techniques include binary encoding, tree encoding, permutation encoding and value encoding. For the purpose of this research, we are adopting the binary encoding technique.

Chromosome A	01010110101001010110101011010 1
Chromosome B	01011010101110101001101101010 1

Fig 3. Example of a chromosome using binary encoding

Binary encoding consists of a system of ones and zeros, designed to represent either a "true" or a "false" value in logical operations. Binary encoding was developed by Claude Shannan in the 1930s using switches. This is shown in fig 3. [7] in their work on Initial Population for Genetic Algorithms: A Metric Approach, suggested evaluating the initial population and, depending of the problem being solved, one can choose gene-level diversity, chromosome-level diversity, population-level diversity, or a combination of those, having in mind that some metrics are more computation-ally expensive than others.

1.3 Fitness Function

The design of fitness function is very essential in genetic algorithm as the desired output depends heavily on the design of fitness function [8]. The fitness function determines how fit an individual is (the ability of an individual to compete with other individuals). It gives a fitness score to each individual. The probability that an individual will be selected for reproduction is based on its fitness score. This function takes a candidate solution to the problem as input and produces as output how “fit” or how “good” the solution is with respect to the problem in consideration. Calculation of fitness value is done repeatedly in a GA and therefore it should be sufficiently fast. A slow computation of the fitness value can adversely affect a GA and make it exceptionally slow. In most cases the fitness function and the objective function are the same as the objective is to either maximize or minimize the given objective function. [9] on similar research on the effects of fitness functions on genetic programming-based ranking discovery for web search, concluded that the design of fitness functions is instrumental in performance improvement.
There is no rule establishing that a particular function should be utilized in a particular problem. Be that as it may, certain capabilities have been taken on by researchers with respect to specific sorts of issues. For classification tasks where supervised learning is used, error measures such as Euclidean distance and Manhattan distance have been widely used as the fitness function.

1.4 Generic Requirements of a Fitness Function

The following requirements should be satisfied by any fitness function.

- The fitness function should be clearly defined. The reader should be able to clearly understand how the fitness score is calculated.
- The fitness function should be implemented efficiently. If the fitness function becomes the bottleneck of the algorithm, then the overall efficiency of the genetic algorithm will be reduced.
- The fitness function should quantitatively measure how fit a given solution is in solving the problem.
- The fitness function should generate intuitive results. The best/worst candidates should have best/worst score values.

For optimization problems, basic functions such as sum of a set of calculated parameters related to the problem domain can be used as the fitness function.

The fitness function shown in fig 4 above is used to compute the criteria for qualifying new genes into the next generation. In some cases, calculating the fitness function directly might not be possible due to the inherent complexities of the problem at hand. In such cases, we do fitness approximation to suit our needs. The basic fitness function is Rosenbrock's function, a common test function for optimizers. The function is a sum of squares as stated in equation 1:

$$f(x) = 100(x_1^2 - x_2)^2 + (1-x_1)^2 \quad (1)$$

The function has a minimum value of zero at the point [1,1]. Because the Rosenbrock function is quite steep, plot the logarithm of one plus the function.

```
public class FitnessFunction : IFitnessFunction
{
        public double Evaluate(IChromosome chromosome)
        {
           double score = 1;
           var values = (chromosome as TimeTableChromosome).Value;
           var GetoverLaps = new Func<TimeSlotChromosome,
               List<TimeSlotChromosome>>(current => values
              .Except(new []{current})
              .Where(slot=>slot.Day == current.Day)
              .Where(slot =>slot.StartAt == current.StartAt
                        || slot.StartAt <= current.EndAt&&slot.StartAt >= current.StartAt
                        || slot.EndAt >= current.StartAt&&slot.EndAt <= current.EndAt)
              .ToList());


           foreach (var value in values)
           {
              var overLaps= GetoverLaps(value);
              score -= overLaps.GroupBy(slot => slot.TeacherId).Sum(x=>x.Count()-1);
              score -= overLaps.GroupBy(slot => slot.PlaceId).Sum(x => x.Count()-1);
              score -= overLaps.GroupBy(slot => slot.CourseId).Sum(x => x.Count()-1);
              score -= overLaps.Sum(item => item.Students.Intersect(value.Students).Count());
           }

           score -= values.GroupBy(v => v.Day).Count() * 0.5;
           return Math.Pow(Math.Abs(score),-1);
        }
}
```

Fig 4. Sample code to show how the fitness function is being implemented to calculate the fitness score.

1.5 Selection

The selection pressure drives the population toward better solutions while crossover, uses genes of selected parents to produce offspring that will form the next generation. Mutation is used to avoid premature convergence and consequently escapes from the local optimal. GAs have been very successful in handling hard combinatorial optimization problems [10]. A genetic algorithm begins with a randomly chosen assortment of chromosomes, which serves as the first generation (initial population). Then each chromosome in the population is evaluated by the fitness function to test how well it solves the problem at hand. Now the selection operator chooses some of the chromosomes for reproduction based on a probability distribution defined by the user. The fitter a chromosome is, the more likely it is to be selected. For example, if *f* is a non-negative fitness function, then the probability that chromosome *i* is chosen to reproduce will be determined by equation 2:

$$P_i = \frac{f_i}{\sum_{j=1}^{n}(f_j)} \tag{2}$$

1.6 Crossover

Crossing over is a cellular process that happens during meiosis when chromosomes of the same type are aligned together. When two chromosomes from a mother and the father are lined up, parts of the chromosome can be switched. The two chromosomes contain the same genes but may have different forms of the genes [11].

The eggs and sperms, which could ultimately consolidate and develop into a renewed individual, each contain just a solitary arrangement of chromosomes. In the event that it was any other way, there would be simply such a large number of chromosomes drifting around. The gametes (egg and sperm cells) occur through meiotic division of germline cells with two full chromosome sets. There is initial a duplication of the germline cell, trailed by two divisions. After the underlying duplication, sets of chromosomes might trade portions of an arm. Early examiners understood that the likelihood of a recombination is generally corresponding to the chromosome length. The distance between two focuses on a chromosome that have a 1% likelihood of being isolated by a recombination occasion is 1 centimorgan (cM), named after the early hereditary qualities pioneer Thomas Chase Morgan.

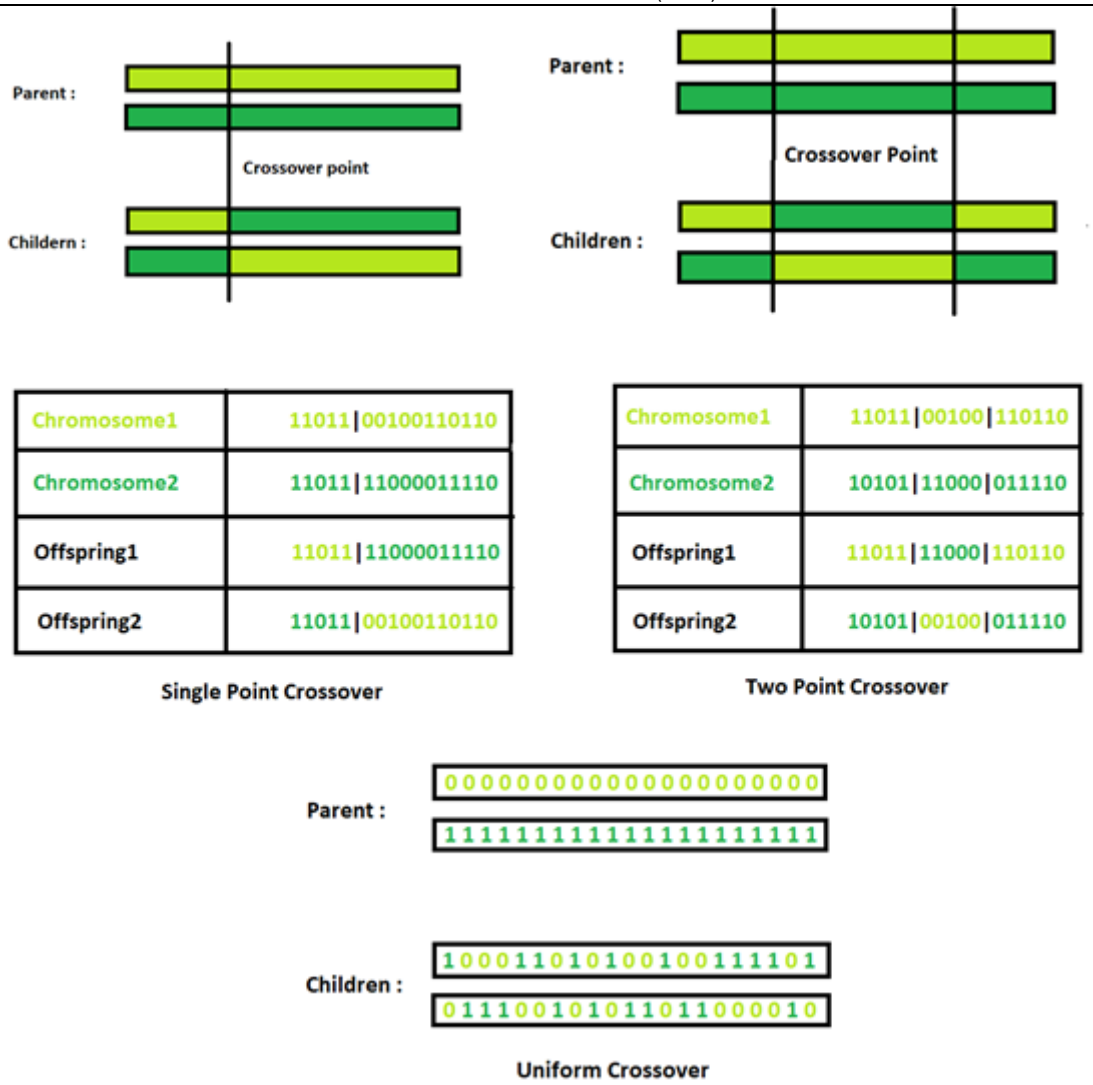

Fig 5. Crossover Techniques

A Single Point Crossover is a crossover point in which crossovers on the parent organism string is selected. All data beyond that point in the organism string is swapped between the two parent organisms. Strings are characterized by Positional Bia. In a Two-Point Crossover, a specific case of a N-point Crossover technique. Two random points are chosen on the individual chromosomes (strings) and the genetic material is exchanged at these points. In a uniform crossover however, each gene (bit) is selected randomly from one of the corresponding genes of the parent chromosomes. These crossover techniques adopted in GA based solutions are shown in fig 5 above.
In general, the crossover between two good solutions may not always yield a better or as good a solution. Since parents are good, the probability of the child being good is high. If offspring is not good (poor solution), it will be removed in the next iteration during "Selection" [12].

1.7 Mutation

In certain new offspring formed, some of their genes can be subjected to a mutation with a low random probability. Normally, mutation takes place after crossover is done. This operator applies the changes randomly to one or more "genes" to produce a new offspring, so it creates new adaptive solutions good avoid local optima. For example, in binary encoding, one or more randomly chosen bits can be switched from 0 to 1 or from 1 to 0 [13].

Before Mutation

A5	1	1	1	0	0	0

After Mutation

A5	1	1	0	1	1	0

Mutation occurs to maintain diversity within the population and prevent premature convergence [14].

In optimization, we start with some kind of initial values for the variables used in the experiment. Since these values and qualities may not be the best ones to utilize, we ought to transform them until getting the best ones. At times, these values are produced by complex capabilities that can't be tackled physically without any problem. However, it is vital to do improvement on the grounds that a classifier might create a bad classification accuracy. For instance, the data is noisy, or the used learning algorithm is weak but due to the bad selection of the learning parameters initial values. Subsequently, there are different optimization strategies proposed and suggested by operation research (OR) researchers to do such work of optimization [15]. The genetic algorithm solves optimization problems by mimicking the principles of biological evolution by repeatedly modifying a population of individual points using rules modelled on gene combinations in biological reproduction. Due to its random nature, the genetic algorithm improves the chances of finding a global solution. Thus, they prove to be very efficient and stable in searching for global optimum solutions. It helps to solve unconstrained, bound-constrained, and general optimization problems, and it does not require the functions to be differentiable or continuous [16].

Table 1. The built-in standard genetic operators in the GA package in R

Selection genetic operator	Crossover genetic operator	Mutation genetic operator
linear-rank selection	single-point crossover	uniform random mutation
tournament selection	whole arithmetic crossover	nonuniform random mutation
truncation selection	local arithmetic crossover	random mutation around the solution

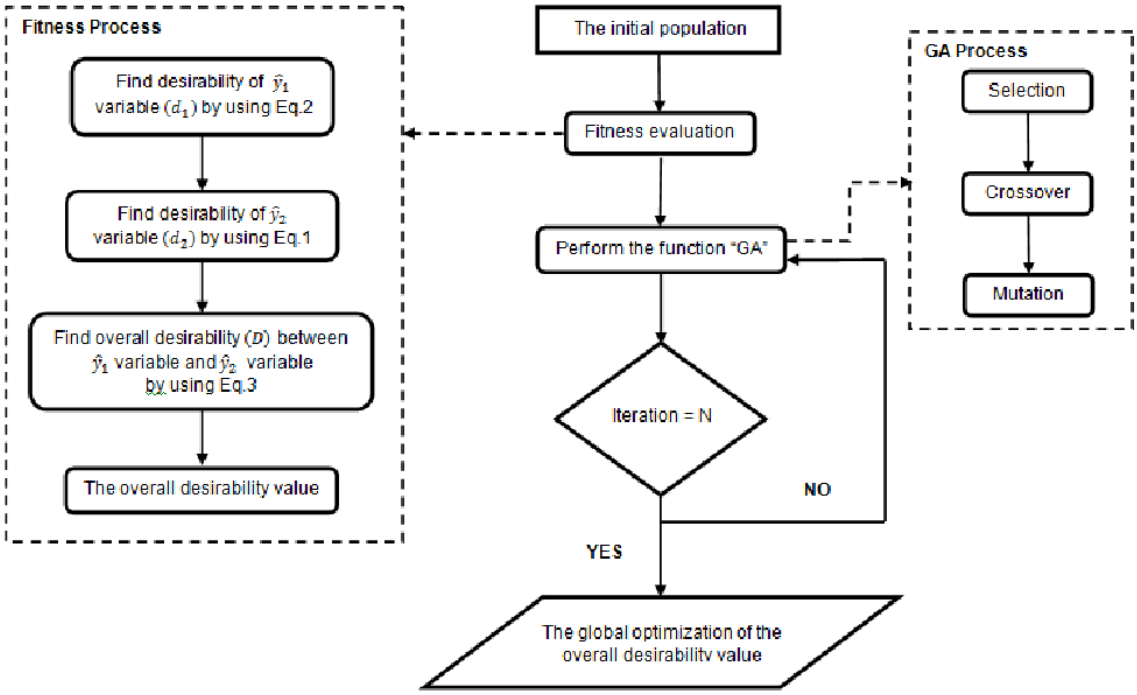

Fig 6. The Process of GA package in R

2. MATERIALS AND METHODS

In this section, a GA is described that is used to find the best polynomial fit to a set of data. The entire process as implemented in R programming language is shown in fig 6. The population in this particular GA consists of strings of integers, each string of integers representing one particular polynomial. A maximum power for each term in the polynomial and a maximum number of terms in the polynomial are pre-set before the GA begins. Each term in a general polynomial consists of a constant multiplying a term as shown in equation 3.

$$X_1^{p_1} X_2^{p_2} X_3^{p_3} X_4^{p_4} X_5^{p_5} \ldots\ldots X_n^{p_n} \quad (3)$$

Where n is the number of variables and $\sum_{i=1} p\,i$ must be less than or equal to the maximum power. Note that a constant term in the polynomial will have $p_i=0$ for all i. An example of how a particular polynomial is represented by a string of integers is given by the polynomial:

$$\alpha_1 x^5 x_2^2 x_3^1 + \alpha_2 x_1^0 x_2^2 x_3^3 + \alpha_3 x_1^1 x_2^7 x_3^0 + \alpha_4 x_1^0 x_2^0 x_3^0 \quad (4)$$

which has four terms and three variables. This polynomial is represented within the GA by the string of integers:

{(5,2,1), (0,2,3), (1,7,0), (0,0,0)}

The cost function (or fitness function) in the GA is defined as the value after a linear least square fit has been performed to find the optimal values of the coefficients, $\alpha_1, \alpha_2, \alpha_3 \ldots \alpha_4$.

The initial population is a set of randomly chosen polynomials. A random number is chosen to represent the number of terms in the polynomial (which must be less than the maximum number of terms) and random sequences of integers are chosen as the powers (with the restriction that their sum for each term must be less than the maximum power). To describe the crossover technique used, consider that each parent consists of a selection of polynomial terms. Crossover should somehow randomly distribute the terms of the parents to terms in the offspring, i.e., offspring number 1 would have some of parent l's terms and some of parent 2's terms, and likewise with offspring 2. The crossover technique must also take into account that repeated terms are not allowed. Suppose the two selected parents are:

{(5,2,1), (0,2,3), (1,7,0), (0,0,0)} for parent 1
and
{(0,0,8), (0,2,3), (0,2,4), (3,0,1)} for parent 2

Then for each chromosome/term in parent 1 a random number between 0 and 1 is picked. For the first chromosome in parent 1, (5,2,1), suppose the random number picked is 0.75, then (5,2,1) will go to child 1 with probability 0.75 and child 2 with probability 0.25. This is repeated for all chromosomes in parent 1, so we may finish with offsprings:

{(0,2,3), (0,0,0), ... Child 1
{(5,2,1), (1,7,0), ... Child 2

The same procedure is repeated for the chromosomes of parent 2. When the chromosome (0,2,3) is picked from parent 2 (note that it already exists in child 1), probabilities are not used because repeat terms are not allowed, and it automatically goes to child 2. Mutation is performed by randomly introducing a completely new term in the polynomial, checking that this particular term does not already exist [17].

Genetic algorithms are a type of optimization algorithm inspired by the process of natural selection. They are widely used in various fields, including machine learning, engineering, and optimization. Polynomials, on the other hand, are mathematical expressions involving one or more variables raised to a power and multiplied by coefficients. They are widely used to approximate complex functions and to model real-world phenomena. Genetic algorithms can be used to find the coefficients of a polynomial that best fits a given set of data points. The process involves encoding the coefficients as chromosomes, and using genetic operators such as selection, crossover, and mutation to evolve a population of candidate solutions. The fitness function is typically defined as the sum of the squared errors between the polynomial and the data points.

3. RESULTS

The genetic algorithm iteratively generates new candidate solutions by selecting the fittest individuals from the current population, applying genetic operators to generate new offspring, and evaluating their fitness. The process continues until a termination criterion is met, such as reaching a maximum number of generations or achieving a certain level of fitness.
Genetic algorithm generally may not give the best solution at the first instance. For a better solution, the operators, the number of iterations or the crossover and mutation rates may need to be changed. The Griewank function is a function widely used to test the convergence of optimization functions. It contains a lot of local optimums. The Griewank function of order *n* is defined by

$$f(x) = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=0}^{n} cos\left(\frac{x}{\sqrt{i}}\right) \tag{5}$$

or

$$f(x) = \sum_{i=1}^{n} \frac{1}{4000}\, x_i^2 - \prod_{i=1}^{n} cos\left(\frac{x}{\sqrt{i}}\right) + 1 \tag{6}$$

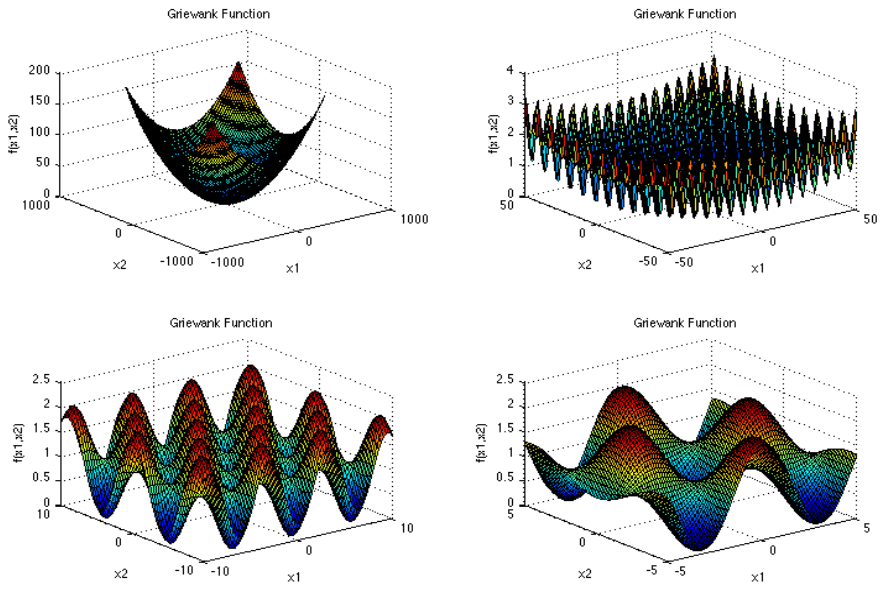

Fig 7. The Girewank function

The function is usually evaluated on the hypercube $x_i \in [-600, 600]$, for all i = 1, ..., d. It is defined as:

$$f(x) = 1 + (1/4000 * \text{sum}(xi^2) - \text{prod}(\cos(xi/\sqrt{i}))) \quad (7)$$

where xi is the ith element of the vector x, and i ranges from 1 to the dimension of the vector x. The function has a global minimum of 0 at x = (0, 0, ..., 0). The Griewank function shown in fig 7 above is commonly used to test the performance of optimization algorithms because it has many local minima and a large search space. It is also known for having a high degree of correlation between its variables, which can make optimization more challenging. The function's complexity increases with the dimension of the input vector x, which means that it becomes increasingly difficult to find the global minimum as the number of variables increases. A fitness function quantifies the optimality of a solution (chromosome) so that that particular solution may be ranked against all the other solutions. A fitness value is assigned to each solution depending on how close it actually is to solving the problem. Ideal fitness function correlates closely to goal and is quickly computable. Since we have determined the fitness function, let's determine the chromosome type, fitness function, how many genes the chromosome will consist of, maximum iteration, selection type, population size and elitism. Elitism refers to the copying of the best chromosomes or chromosomes within the population as they are to the new population. In this work, we did not apply it for this example, but values such as 1–2 can be given. The crossover ratio is 0.8 and the mutation rate is 0.1. We can change those values if we want to. The selection type is the roulette wheel, the crossover type is single-point, and the mutation type is uniform random.

4. DISCUSSION

Genetic algorithms are a type of metaheuristic optimization algorithm inspired by the process of natural selection. They are commonly used to solve optimization problems that involve a large search space or multiple constraints. Polynomials are mathematical functions that involve a sum of powers in one or more variables. They are widely used in various fields of mathematics and engineering to approximate complex functions. Genetic algorithms can be used to find the coefficients of a polynomial that best fits a set of data points. This process involves encoding the coefficients as chromosomes and using genetic operators such as selection, crossover, and mutation to evolve a population of candidate solutions. The fitness function in this case is typically the sum of the squared errors between the polynomial and the data points. The genetic algorithm iteratively generates new candidate solutions by selecting the fittest individuals from the current population, applying genetic operators to generate new offspring, and evaluating their fitness. This process continues until a termination criterion is met, such as reaching a maximum number of generations or achieving a certain level of fitness. The results obtained from this experiment shows that though genetic algorithm can be used as a method of solving polynomials, they are expensive and take a little longer time to converge than using other known methods like Bisection, Regular-falsi and the rest. This can however be shortened by adjusting the population size and efficient fitness functions.

5. CONCLUSION

Using a genetic algorithm to fit a polynomial can be a computationally intensive process, especially for high-degree polynomials and large datasets. The process is repeated for several generations until a stopping criterion is met, such as a

maximum number of generations or a satisfactory level of fitness. The final solution is the chromosome with the highest fitness value. PGAs can be effective for solving a wide range of optimization problems, particularly those that involve polynomial functions. However, they can be computationally expensive and may require careful tuning of parameters such as population size, mutation rate, and crossover rate to achieve good results.

REFERENCES

[1] BajPai, P., Kumar, M. (1999). Genetic Algorithm – an Approach to Solve Global Optimization Problems, *Indian Journal of Computer Science and Engineering*, Vol 1 No 3 199-206.

[2] Hoffmeister, F., Bäck, T. (1991). Genetic Algorithms and evolution strategies: Similarities and differences. *Lecture Notes in Computer Science*, vol 496. Springer, Berlin, Heidelberg. https://doi.org/10.1007 /BFb0029787.

[3] Dao, S.D., Abhary, K. & Marian, R. (2017). An improved genetic algorithm for multidimensional optimization of precedence-constrained production planning and scheduling. *J Ind Eng Int* **13**, 143–159. https://doi.org/10.1007/s40092-016-0181-7

[4] Clegg J., Dawson J., Stuart P., Barley M. (2005). The use of a genetic algorithm to optimize the functional form of a multi- dimensional polynomial fit to experimental data. In: IEEE Congress on Evolutionary Computation, Edinburgh. IEEE Congress on Evolutionary Computation, Edinburgh , pp. 928-934.

[5] Goldbeg, D. (1999). Genetic Algorithm in Search Optimization and Machine Learning. Addison-Wesley Publishing Co.inc.

[6] Chudasama, C., Shah, S., Panchal, M. (2011). Comparison of Parents Selection Methods of Genetic Algorithm for TSP. Proceeding published by International Journal of Computer Application (IJCA). *International Conference on Computer Communication and Network* CSI-COMNET-2011.

[7] Diaz-Gomez, P., Hougen, D. F. (2007). Initial population for genetic algorithms: a metric approacs. In: *Proceedings of the 2007 International Conference on Genetic and Evolutionary Methods*, GEM, Nevada, USA. 2007; pp. 55–63.

[8] Kour, H., Sharma, P., Abrol, P. (2015). Analysis of fitness function in genetic algorithms, *Journal of Scientific and Technical Advancements*, Volume 1, Issue 3, pp. 87-89, ISSN: 2454-1532.

[9] Fan, W., Fox, E., Pathak, P., Wu, H. (2004). The effects of fitness functions on genetic programming-based ranking discovery for Web search, *Journal of the Association for Information Science and Technology*, https://doi.org/10. 1002/asi.20009.

[10] Varnamkhasti, J., Lee, L. (2012). A Fuzzy Genetic Algorithm Based on Binary Encoding for Solving Multidimensional Knapsack Problems, *Journal of Applied Mathematics* Volume 2012, Article ID 703601, http://dx.doi.org/10.1155 /2012/703601.

[11] Wetterstrand, K. (2023). Crossing Over, National Human Genome Research Institute, *https://www.genome.gov/genetics-glossary/Crossing-Over*, accessed on 2023-02-13.

[12] Dutta, A. (2019). Crossover in Genetic Algorithm, *https://www.geeksforgeeks.org/ crossover-in-genetic-algorithm/*, accessed on 2023-03-02.

[13] Hassanat, A., Almohammadi, K., Alkafaween, E., Abunawas, E., Hammouri, A., Prasath, V. (2019). *Choosing Mutation and Crossover Ratios for Genetic Algorithms—A Review with a New Dynamic Approach. Information*, 10(12), 390. MDPI AG. Retrieved from http://dx.doi.org/ 10.3390/ info10120390, accessed on 2023-03-18.

[14] Mallawaarachchi, V. (2017). Introduction to Genetic Algorithms, *https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code*-e396e98d8bf3, accessed on 2023-02-15.

[15] Gad, A. (2018). Introduction to Optimization with Genetic Algorithm *https:// towardsdatascience.com/introduction-to-optimization-with-genetic-algorithm*-2f5001d9964b, accessed on 2023-01-11.

[16] Mishra, S., Sahoo S., Das, M. (2017). Genetic Algorithm: An Efficient Tool for Global Optimization, *Advances in Computational Sciences and Technology*, ISSN 0973-6107 Volume 10, Number 8 (2017) pp. 2201-2211.

[17] Clegg, J., Dawson, J., Porter, S., Barley, M. (2005). The use of a genetic algorithm to optimize the functional form of a multi- dimensional polynomial fit to experimental data. In: IEEE Congress on Evolutionary Computation, Edinburgh. *IEEE Congress on Evolutionary Computation*, 02-05 Sep 2005 IEEE , Edinburgh , pp. 928-934.