Algorithms between nature and nurture.

Sandra sahnoune, student N° 21039366.

1 INTRODUCTION

Genetic algorithms became famous due to the work of john holland in early 70s where he mentioned on his book "adaptation in natural and artificial systems " how this new science has been founded. The discipline of genetic algorithms has expended into a significant area these last years. Genetic algorithms are a subset of evolutionary algorithms based on Darwin's theory of evolution as a source of inspiration. It is a computational search approach that finds true or approximate answers to optimization and search issues, it is also categorized as a global search heuristic, the idea of solving an optimization problem consists in exploring a search space to maximize (or minimize) a given function. This report explains the implementation of a performed genetic algorithm applied to optimize two mathematical functions. In addition, it will compare various methods and techniques used for optimizations and search problems. There has been a lot of effort done in the field of GAs, both in terms of research and practical applications. GAs may and have been used to solve a variety of complex problems in Image processing which is introduced in the last chapter of this report.

2 BACKGROUND RESEARCH

Genetic algorithms are based on an analogy with genetic structure and behaviour of chromosome of the population.
They are algorithms for finding the minimum of a function $J : \Omega \rightarrow R$ using Darwin's laws of evolution of species. Note that the set Ω can correspond to various mathematical objects (discrete or continuous). They consist in studying the evolution of a population of elements (also called individuals), initially chosen randomly among the elements of Ω. Using the principles of selection, crossover, and mutation, it is observed that after a number of generations, the population is concentrated towards the global minimum of J in Ω. Binary genetic algorithms are the first genetic algorithms to have been introduced historically.
In this case, the set Ω consists of chromosomes, corresponding to a sequence of N bits taking the value 0 or 1. Thus real genetic algorithms were then introduced to handle the case more easily where Ω is a subset of R^n .
Natural selection begins with the selection of the fittest individuals from a random population, followed by the production of children who inherit the traits of their parents and are grouped in the following generation. This approach is done until the optimal solution of the problem is found.

1.Steps:

Genetic algorithm is made in the following stages:

1. Initial population
2. Fitness function to evaluate
3. Select fittest parents
4. Crossover
5. Mutation
6. Best solution

1.1 Initial population:

this process starts with initialising a random population by setting individuals. These individuals are known as chromosome which is a string combination of genes.

1.2 Fitness function:

This function determines how fit an individual (chromosome)is in the population to survive for the next generation. The fitness gives the score to each chromosome and its probability to be selected for the next generation.

1.3 Selection:

The selection phase is based to select the fittest chromosomes from the population. Individuals with the lowest fitness have more chance to be selected for the next generation.

1.4 Crossover:

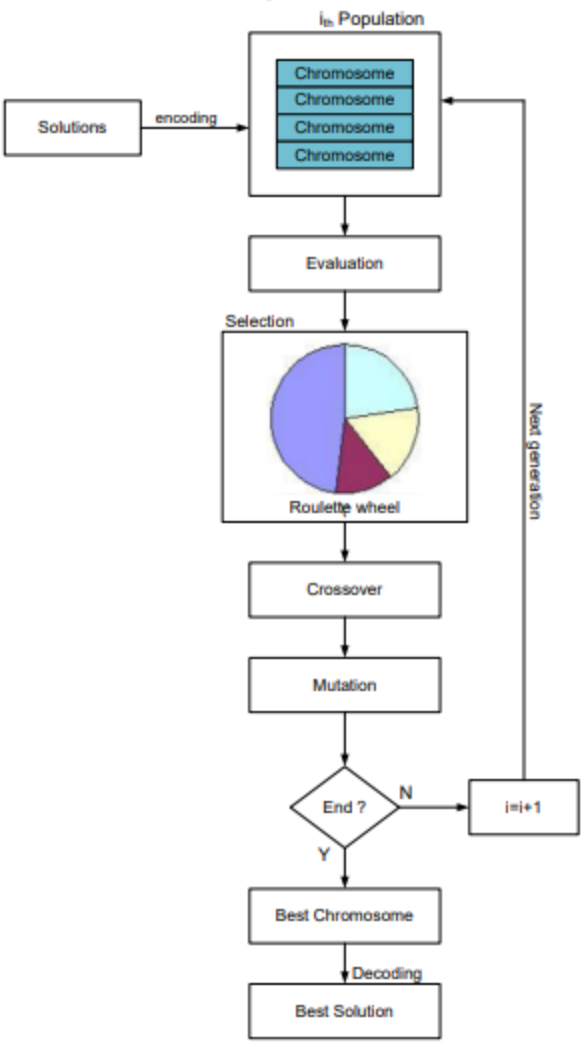
It is the most important step in GA. Crossover point is randomly selected and children are generated by exchanging the genes of the parents. Then the offspring are added to the population and new generation is created.

1.5 Mutation:

Mutation is implemented to maintain the diversity in the generations, mutation rate is randomly selected where one allele of gene is selected to be replaced by another one to produce new genetic structure.

1.6 The final solution:

The algorithm terminates when the number of iterations is generated and the optimal solution is provided.



3 EXPERIMENTATIONS

We are going to use a genetic algorithm to solve and optimize two mathematical functions to get the lowest fitness possible:

1.Intitialization:

We set individual with gene and fitness in a population, then we define the number of chromosomes and genes.

Then we generate random reel values between 0 and 1 of genes for P chromosomes, so we append the new individual in the population.

Chromosome [0] *P = [0] *N

1.2 Evaluation

For each chromosome generated during the initialization, we are passing it through a fitness function in order to calculate each value of the gene and classify the chromosomes:

Firstly, we put as a parameter chromosome, then we define:

x = chromosome.gene[i]

STYBLINSKY-TANG function:

$$f(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{d}(x_i^4 - 16x_i^2 + 5x_i)$$

where $-5 \le x \le 5$, start with d=20

DIXON-PRICE function:

$$f(\mathbf{x}) = (x_1 - 1)^2 + \sum_{i=2}^{d} i\left(2x_i^2 - x_{i-1}\right)^2$$

where $-10 \le x \le 10$, start with d=20

1.3 Selection:

The chromosomes with the lowest fitness will have a better chance of being chosen for the following generation. To determine fitness probability, we must first determine the fitness of each chromosome by passing the genes through the fitness function according to this equation :

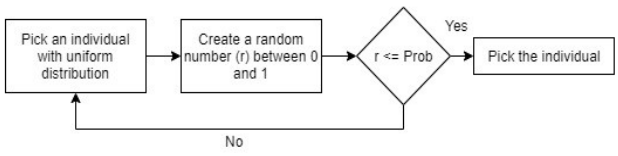
$$P(x_i)=\frac{F(x_i)}{\sum_{k=1}^{N}F(x_k)}$$

Population[gene].fitness=test_function(population[gene])

The total probability is calculated for each chromosome and the fittest ones will be selected as parents for the next generation.

```
chromosome 0 = -30.524152244731475
chromosome 1 = -41.980438985623316
chromosome 2 = -22.47161198750876
chromosome 3 = -22.33691476360019
chromosome 4 = -28.85391716677933
chromosome 5 = -18.10936470432126
chromosome 6 = -16.587753506998588
chromosome 7 = -39.118899259762046
chromosome 8 = -37.2514423772939
chromosome 9 = -37.23233833353789
```

From the example above we find out that chromosome 6 has the lowest fitness so this one has the highest probability to be selected.
By "spinning a Roulette Wheel," we select one individual from our population bag. In this situation, the likelihood of any individual being chosen is precisely proportional to their fitness level.



Roulette Wheel Flowchart

We compare off1.fitness and off2.fitness and we append the fittest one in the population of offspring.
If off1.fitness is lower than off2.fitness in this case we will minimize the value of the function.

```
For i in 0 to P:

  Do
  parent1 <- RANDOM_INT(0 to P-1)
  off1 <- population[parent1]
  parent2 <- RANDOM_INT(0 to P-1)
  off2 <- population[parent2]
  if fitness in off1 < fitness in off2:
    append off1 in offspring
  else
    append off2 in offspring
```

1.4 Crossover

In this genetic algorithm we use single point crossover as we randomly choose a position in the parent chromosome and exchange them. The number of parent Chromosomes is regulated using crossover point crp parameters, and the parent chromosome that will mate is chosen at random.

Chromosome[6]>< Chromosomes[5]

1.5 Mutation

The generation at random point is replaced with a new value throughout the mutation process. The steps are as follows:
So, initially, we must determine the length of the chromosomes in population. Then we pass through all the genes in each chromosome, we determine the MUTRATE and MUTSTEP in parameters, if the value of the allele in the gene is more than the maximum value, the gene will take its value, else if it is less than the minimum value it will take its value. Finally, we append all the new individuals on the population of offspring.

1.6 Experimental Results & parameters

The method of optimisation by genetic algorithms requires the definition of several parameters of major importance to achieve good results.
the number of iterations and the size of the population on the one hand and the probabilities of
and mutation on the other.

Case 1:

```
N = 20  # number of genes
P = 10  # number of chromosomes
iteration = 50  # number of generations
crp = 0  # crossover point
```

```
MUTRATE = 0.01  # mutation rate
Min = 0.0
Max = 1.0
MUTSTEP = 5 #mutation step
```

When we run the styblinski-tang function, the graph shows as follow:

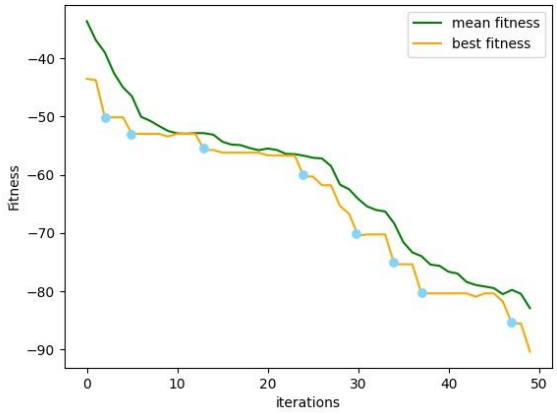


Figure1

the optimal fitness:
mean value of x = -82.91950811887736
best value of x = -90.1
the blue dots represent the best values that the functions get over 50 generations.
We can see that the function is steadily decreasing which mean that as long as the fitness is low the solution is getting better.

Case 2:

```
N = 20  # number of genes
P = 20  # number of chromosomes
iteration = 70  # number of generations
crp = 0  # crossover point
MUTRATE = 0.01  # mutation rate
Min = 0.0
Max = 1.0
MUTSTEP = 5 #mutation step
```

here we change the number of chromosomes to 20 and iterations to 70 :

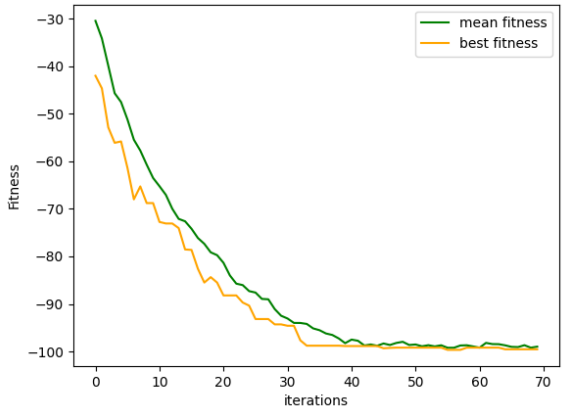


Figure 2

These results show an improvement over the first case, we can clearly notice that over the last 40 generations the mean fitness and best fitness are lightly close and produce better values than the graphs in *figure 1*.
This means that the performance of populations increases with their size.

Case 3:

To check the effect of mutation rate and crossover point in the search space, we set:
MUTRATE = 0.06
Crp = 2

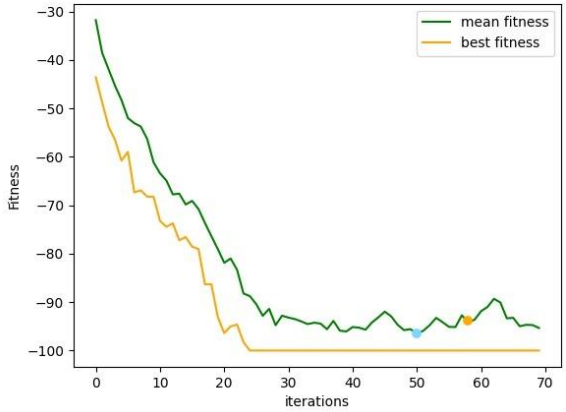


Figure 3

The graph show the fluctuation of the mean fitness, where it appears a global minimum (optimal solution) as the blue dot.
It means that the mutation rate and crossover point bring some diversity in the population which improve the search space.

same parameters applied for the Dixon-price function:

Case 1:

We set parameters as :

```
N = 20  # number of genes
P = 10  # number of chromosomes
iteration = 50  # number of generations
crp = 0  # crossover point
MUTRATE = 0.01  # mutation rate
Min = 0.0
Max = 1.0
MUTSTEP = 10 #mutation step
```

The graph shows as follow:

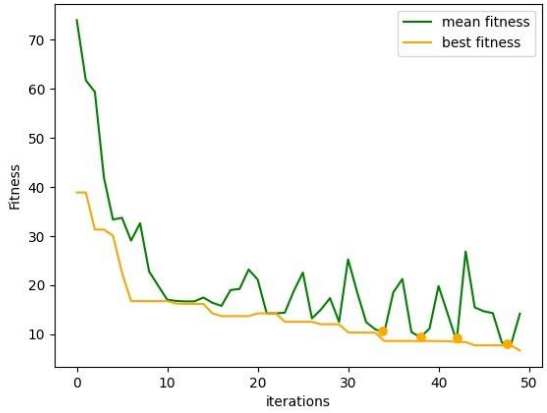


Figure 4

the graph shows various fluctuation with multiple local minima, which explains why the search space is stuck on a not necessarily optimal solution.

To overcome this issue, we change the parameters of:

MUTRATE to 0.02

Crp to 2

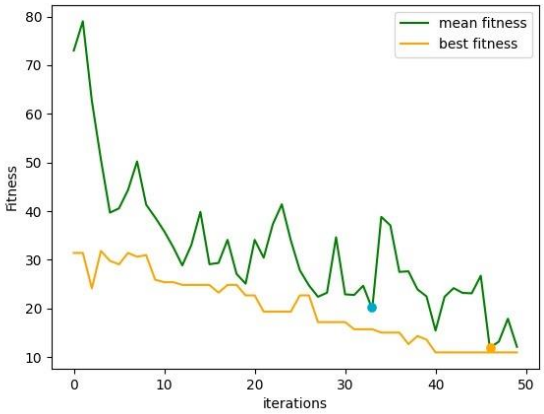


Figure 5

The figure 5 represents the new fitness values in the search space over 50 iterations, it illustrates that the changes of the mutation rate and crossover point had improve the results, as it shows the global minima appeared as the orange dot which define the optimal solution found in the search space.

1.7 Optimization problems:

Genetic algorithms are modern optimization methods for complex problems but how could problems be solved in the past?

For instance, the traveling salesman problem consider solving it with an exhaustive search algorithm. The total number of viable solutions is limited since the salesman travels in a closed loop and the direction of travel between cities is irrelevant (free circular permutation), and the possible solutions would be :

$$S = \frac{(n-1)!}{2} = \frac{19!}{2} = 60,822,550,204,416,000\,.$$

This is a huge number to test every candidate solution, and the time would take forever to find the best solution.

Furthermore, genetic algorithms are used to optimize more complexe function with two variables :

$$g(x,y) = (3/2)e^{\frac{1}{1+(x-1)^2+(y-1)^2}} - (5/2)e^{\frac{1}{1+(1/4)(x+1/2)^2+(1/36)(y-1)^2}} + 2e^{\frac{1}{1+(x-2)^2+(y-2)^2}} + 2e^{\frac{1}{1+(x-1)^2+(y+1)^2}}$$

In this case, because each chromosome must now account for two components (x, y) in order to be a function in R*, each chromosome must have a multiple size of two. It also accepts a two-component vector as a parameter in the target function. The

approach remains the same, with a population of 300 people, a chromosomal size of 50 (which must now be even), and a maximum number of cycles of 1000.

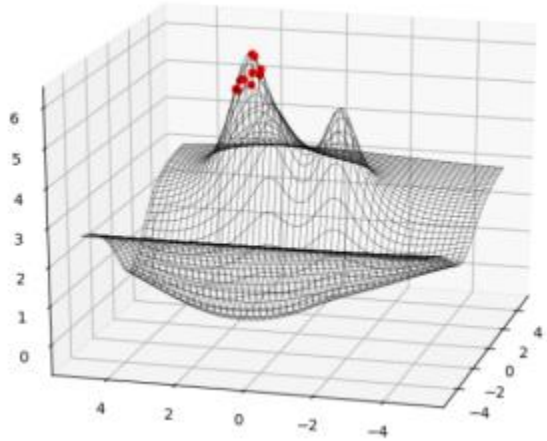


Figure 6

Genetic algorithms are now used in various domain to solve complex problems such as mathematical functions, science research, finance, industry.

For example: Image Segmentation.

Let's apply the GA to investigate the solution space in the sense that each pixel is grouped into other pixels using a distance function based on both local and global segments that have already been computed. In order to develop a GA for picture segmentation, two fundamental difficulties must be addressed:

•1) determining an appropriate representation for each potential solution;

•2) determining an appropriate fitness function in terms of weight. The genetic system can dynamically adjust the parameters to get the optimal performance. Almost every image segmentation method comprises parameters that are used to influence the segmentation results.

Figure 7

the process of image segmentation is based on the image simililarity including the distance function (fitness).

$$\gamma - \frac{1}{K}\sum_{k=1}^{K}\beta_k \times \frac{\#agr_k}{P_k}$$

Figure 8

The same steps are used for single-point crossover , mutation and selection to generate random values.

Parameter	Value
Population size	50 individuals
Gene length	32 bits
Maximum generation	200
The Prob. of Mutation P_m	0.01
The Prob. of Crossover P_c	0.8

Figure 9

It can be observed that the GAproposed approach yields the best results, but at the cost of a lot of processing time.

Figure 10

The graph shows the search space with the optimal solution found:

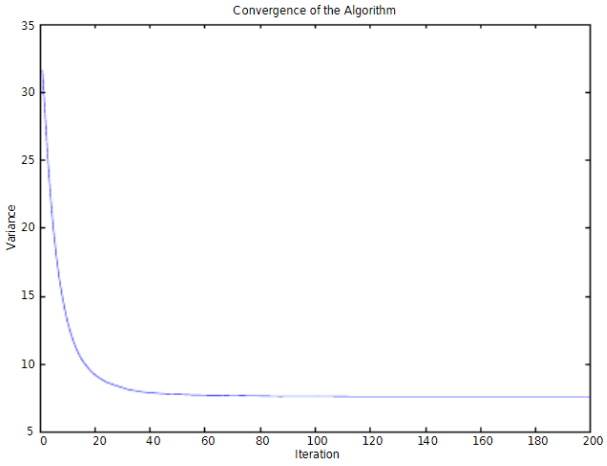


Figure 11.

The ability to use AG to identify the maximum or minimum of real variable functions is just the beginning of their vast potential for addressing mathematical issues. However, GAs may be applied in many other fields of mathematics, such as finding system coefficients of linear equations or solving linear programming issues.

1.8 Ethical issues of using genetic algorithms in AI and its impacts

Despite of all the amazing invention and work achieved in AI using GA, this domain faces several issues some of them are:

1. Autonomous car is one that can perceive its surroundings and moving with little or no human intervention. For the vehicle to operate safely and understand its driving environment, a massive amount of data must be always collected by a variety of sensors located throughout the vehicle. The vehicle's autonomous driving computer system then processes these. The autonomous vehicle must also undergo extensive training to comprehend the data it collects and make the best judgement possible in any given traffic condition. Every day, everyone makes moral decisions. When a driver slams on the brakes to avoid hitting a jaywalker, they are making the moral decision to shift risk from the pedestrian to the passengers in the vehicle.

2. As artificial intelligence (AI) becomes more widely used in legal systems around the world, more ethical issues arise. AI could allegedly evaluate cases and administer justice more effectively, quickly, and efficiently than a judge.AI technologies have the potential to have a tremendous impact in a variety of domains, from legal professions and the judiciary to assisting legislative and administrative public bodies in their decision-making. They can, for example, boost lawyers' efficiency and accuracy in both counselling and litigation, benefiting lawyers, their clients, and society. To assist judges in formulating new rulings, existing software systems can be supplemented and advanced by AI techniques. The automatization of justice has been coined to characterise the tendency toward more usage of autonomous technologies.

4 CONCLUSIONS

Genetic algorithms are powerful adaptive search procedure use to optimize a complex problem into an optimal solution.

This report shows the result of optimizing two complex mathematical function by computing the evolution’s theory and illustrates how the parameters can affect the search space.

The algorithm uses a one-point crossover which provided good results, however it would be a good practice to use a uniform crossover to make the process more efficient and get better results and more diversity on the search space.

In addition, the fitness function evaluate all the individuals in the population, and select the fittest ones, elitism is a good strategy to copy the good genes in the new generation of population.

Other future improvement would be to choose another method of selection for instance neuron network with other metaheuristics such as ant colony or swarm optimization.

REFERENCES

D. Beasley, D.R.Bull, R.R.Martin, "An Overview of Genetic Algorithms: Part 1, Fundamentals", Techn.
Report of Univ. of Cardiff, UK, University Computing, 15(2), 1993, pp. 58-69

Holland, J., 2010. *Adaptation in natural and artificial systems*. Cambridge, Mass: MIT Press.

Ahn, Chang Wook, and Rudrapatna S. Ramakrishna. "Elitism-based compact genetic algorithms." *IEEE Transactions on Evolutionary Computation* 7.4 (2003): 367-385.

Sfu.ca. 2021. *Dixon-Price Function*. [online] Available at: <https://www.sfu.ca/~ssurjano/dixonpr.html> [Accessed 16 December 2021].

Sfu.ca. 2021. *Styblinski-Tang Function*. [online] Available at: <https://www.sfu.ca/~ssurjano/stybtang.html> [Accessed 16 December 2021].

Goldberg, D., 2012. *Genetic algorithms in search, optimization, and machine learning*. Boston: Addison-Wesley.

Kosak C, Marks J and Schieber S, A Parallel Genetic Algorithm for Network Design Layout. Proceedings of the Fourth International Conference on Genetic Algorithms, pp. 458-465, Morgan-Kaufmann Publishers, 1991. Division of Applied Sciences, Harvard University, USA.

Citeseerx.ist.psu.edu. 2021. [online] Available at: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.2.3551&rep=rep1&type=pdf> [Accessed 16 December 2021].

Iopscience.iop.org. 2021. [online] Available at: <https://iopscience.iop.org/article/10.1088/1742-6596/1448/1/012020/pdf> [Accessed 16 December 2021].

Kiyoumarsi F 2015 Mathematics programming based on genetic algorithms education Procedia-Social and Behavioral Sciences 192 70

ResearchGate. 2021. *(PDF) On benchmarking functions for genetic algorithm*. [online] Available at: <https://www.researchgate.net/publication/27382766_On_benchmarking_functions_for_genetic_algorithm> [Accessed 16 December 2021].

Denny Hermawanto, "Genetic Algorithm for Solving Simple Mathematical Equality Problem", Indonesian Institute of Sciences (LIPI).

Iopscience.iop.org. 2021. [online] Available at: <https://iopscience.iop.org/article/10.1088/1742-6596/1448/1/012020/pdf> [Accessed 16 December 2021].

Whitman.edu. 2021. [online] Available at: <https://www.whitman.edu/documents/academics/mathematics/2014/carrjk.pdf> [Accessed 16 December 2021].

Core.ac.uk. 2021. [online] Available at: <https://core.ac.uk/download/pdf/82810964.pdf> [Accessed 16 December 2021].

UNESCO. 2021. *Artificial Intelligence: examples of ethical dilemmas*. [online] Available at: <https://en.unesco.org/artificial-intelligence/ethics/cases> [Accessed 16 December 2021].

Ummto.dz. 2021. [online] Available at: <https://www.ummto.dz/dspace/bitstream/handle/ummto/3367/Belhassani%2C%20Saliha.pdf?sequence=1&isAllowed=y> [Accessed 16 December 2021].

ResearchGate. 2021. *(PDF) Genetic Algorithms: A tool for image segmentation*. [online] Available at: <https://www.researchgate.net/publication/261262060_Genetic_Algorithms_A_tool_for_image_segmentation> [Accessed 16 December 2021].

Source code as an appendix:

```
import random as rd
import copy
import matplotlib.pyplot as plt
import math


class individual:
    def __init__(self):
        self.gene = [0]*N
        self.fitness = 0


population = []
offspring = []
avg_fitness = []
bstfit = []

# GA parameters

N = 20  # number of genes
P = 10  # number of chromosomes
iteration = 50  # number of generations
crp = 0  # crossover point
MUTRATE = 0.01  # mutation rate
Min = 0.0
Max = 1.0
MUTSTEP = 5  # mutation step
pi = math.pi
# initialize population
for x in range(0, P, 1):
    tempgene = []
    for y in range(0, N, 1):
        tempgene.append(rd.uniform(Min,
Max))
    newind = individual()
```

```
    newind.gene = tempgene.copy()
    population.append(newind)
    # print("chromosome",x,"\n
gene=\n",tempgene)
# evaluate the population based on the
fitness


def test_function(chromosome):
    fitness = 0
    for i in range(0, N):
        # fitness = fitness +
chromosome.gene[i]
        # **** worksheet 3 ***
        # fitness +=
((chromosome.gene[i])**2 - 10 *
        #             math.cos(2*pi*(chr
omosome.gene[i]))) + 10*N
        # *** equation 1 styblinski ***
        fitness +=
((chromosome.gene[i])**4 - 16 *
                    ((chromosome.gene[i]
)**2) + 5*(chromosome.gene[i])) * 1/2
        # *** equation 2 dixon-price **
*
        # fitness += i * \
        #     (((2*((chromosome.gene[i])
**2))-(chromosome.gene[i-1]))
        #      ** 2) +
((chromosome.gene[1])-1)**2

    return fitness


# loop through all the population in
order to get the fitness of the
chromosomes
for gene in range(P):
    population[gene].fitness =
test_function(population[gene])
    print("chromosome", gene, "=",
population[gene].fitness)

for z in range(0, iteration):
    # selection of parents
    offspring = []
    for i in range(0, P):
        parent1 = rd.randint(0, P-1)
        off1 = population[parent1]
        parent2 = rd.randint(0, P-1)
        off2 = population[parent2]
        if off1.fitness < off2.fitness:
            offspring.append(off1)
        else:
            offspring.append(off2)

    # crossover
    toff1 = individual()
    toff2 = individual()
    temp = individual()  # create
temporary object for individual class
    for i in range(0, P, 2):
        toff1 =
copy.deepcopy(offspring[i])
        toff2 =
copy.deepcopy(offspring[i+1])
        temp =
copy.deepcopy(offspring[i])
        crosspoint = rd.randint(crp, N)
        for j in range(crosspoint, N):
            toff1.gene[j] =
toff2.gene[j]
            toff2.gene[j] = temp.gene[j]
        offspring[i] =
copy.deepcopy(toff1)
        offspring[i+1] =
copy.deepcopy(toff2)

    # mutation:
    for i in range(0, P):
        newind = individual()
        newind.gene = []
        for j in range(0, N):
            gene = offspring[i].gene[j]
            mutprob = rd.random()
            if mutprob < MUTRATE:
                alter = rd.uniform(-
MUTSTEP, MUTSTEP)
                gene = gene + alter
```

```
                if gene > Max:
                    gene = Max
                if gene < Min:
                    gene = Min
            newind.gene.append(gene)
        offspring[i] =
copy.deepcopy(newind)

    for n in range(0, P):
        offspring[n].fitness =
test_function(offspring[n])

    total = 0
    bestfit = offspring[0].fitness
    for i in range(0, P):
        total += offspring[i].fitness
        if offspring[i].fitness <
bestfit:
            bestfit =
offspring[i].fitness
    print("generation", z, "=",
(total/P))

    avg_fitness.append(total/P)
    bstfit.append(bestfit)
    population =
copy.deepcopy(offspring)
    # print(i in bstfit)

plt.plot(avg_fitness, color="Green")
plt.xlabel("iterations")
plt.ylabel("Fitness")
plt.plot(bstfit, color="Orange")
plt.legend(["mean fitness", "best
fitness"])
plt.show()
```