RAMAIAH UNIVERSITY OF APPLIED SCIENCES

# ASSIGNMENT

| | |
|---|---|
| **Course Code** | 19CSE422A |
| **Course Name** | Computational Intelligence |
| **Programme** | B. Tech. |
| **Department** | Computer Science & Engineering |
| **Faculty** | Faculty of Engineering & Technology |

| | |
|---|---|
| **Name of the Student** | Subhendu Maji |
| **Reg. No** | 18ETCS002121 |
| **Semester/Year** | 7th semester / 2018 batch |
| **Course Leader/s** | Dr. Vaishali R. Kulkarni |

| Declaration Sheet | | | |
|---|---|---|---|
| Student Name | Subhendu Maji | | |
| Reg. No | 18ETCS002121 | | |
| Programme | B. Tech. | Semester/Year | 7th sem /2018 batch |
| Course Code | 19CSE422A | | |
| Course Title | Computational Intelligence | | |
| Course Date | | to | |
| Course Leader | Dr. Vaishali R. Kulkarni | | |

**Declaration**

The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of university regulations and will be dealt with accordingly.

| | | | |
|---|---|---|---|
| Signature of the Student | | Date | |
| Submission date stamp (by Examination & Assessment Section) | | | |

| Signature of the Course Leader and date | Signature of the Reviewer and date |
|---|---|
| | |

**Contents**

Declaration Sheet ........ ii

Contents ........ iii

Marking Scheme ........ 4

Question No. 1 ........ 5

1.1 Steps in Genetic algorithm with respect to a given problem ........ 5

1.2 Software Simulation with sample input and output ........ 9

Question No. 2 ........ 12

2.1 Justification that the recommended algorithm is the best fit to the challenge ........ 12

2.2 Discussion on the biological inspiration for the recommended algorithm ........ 13

2.3 Software Simulation with sample input and output ........ 13

2.4 Conclusion ........ 16

| Assignment-1 | | | |
|---|---|---|---|
| Reg. No. | 18ETCS002121 | Name of Student | Subhendu Maji |

| Sections | | Marking Scheme | Mark | | |
|---|---|---|---|---|---|
| | | | **Max Marks** | First Examiner Marks | Moderator |
| **Part A** | PartA | | | | |
| | A1 | Steps in Genetic algorithm with respect to a given problem | 03 | | |
| | A2 | Software Simulation with sample input and output | 07 | | |
| | | Part-A Max Marks | 10 | | |
| **Part B** | | | | | |
| | B.1 | Justification that the recommended algorithm is the best fit to the challenge | 03 | | |
| | B.2 | Discussion on the biological inspiration for the recommended algorithm | 03 | | |
| | B.3 | Software Simulation with sample input and output | 07 | | |
| | B.4 | Conclusion | 02 | | |
| | | Part- B Max Marks | 15 | | |
| | | Total Assignment Marks | 25 | | |

| Course Marks Tabulation | | | | |
|---|---|---|---|---|
| Component-1 (B) Assignment | First Examiner | Remarks | Moderator | Remarks |
| A | | | | |
| B | | | | |
| Marks (out of 25) | | | | |

**Question No. 1**

**Solution to Question No. 1:**

### 1.1 Steps in Genetic algorithm with respect to a given problem

A genetic algorithm is a search heuristic that is inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.

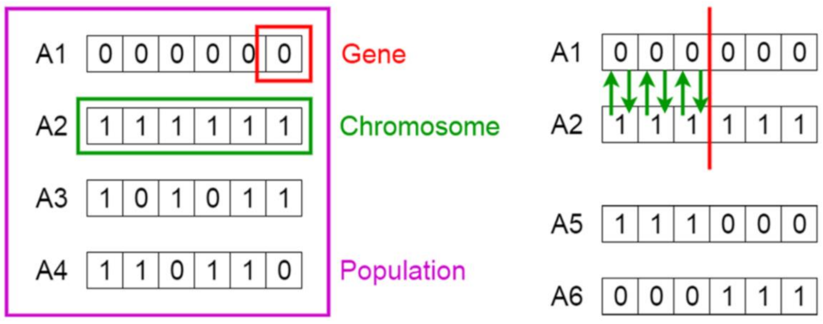


*Figure 1 genetic algorithm*

**Notion of Natural Selection**

The process of natural selection starts with the selection of fittest individuals from a population. They 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.

This notion can be applied for a search problem. We consider a set of solutions for a problem and select the set of best ones out of them.

Five phases are considered in a genetic algorithm.

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

**Initialization of Population**

Every gene represents a parameter (variables) in the solution. This collection of parameters that forms the solution is the chromosome. Therefore, the population is a collection of chromosomes.

Order of genes on the chromosome matters. Chromosomes are often depicted in binary as 0's and 1's, but other encodings are also possible.

**Fitness Function**

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.

**Selection**

The idea of **selection** phase is to select the fittest individuals and let them pass their genes to the next generation.

Two pairs of individuals (**parents**) are selected based on their fitness scores. Individuals with high fitness have more chance to be selected for reproduction.

**Crossover**

**Crossover** is the most significant phase in a genetic algorithm. For each pair of parents to be mated, **a crossover point** is chosen at random from within the genes.

There are 3 major types of crossovers.

- Single Point Crossover: A point on both parents' chromosomes is picked randomly and designated a 'crossover point'. Bits to the right of that point are exchanged between the two parent chromosomes.
- Two-Point Crossover: Two crossover points are picked randomly from the parent chromosomes. The bits in between the two points are swapped between the parent organisms.
- Uniform Crossover: In a uniform crossover, typically, each bit is chosen from either parent with equal probability.

For example, consider the crossover point to be 3 as shown below.

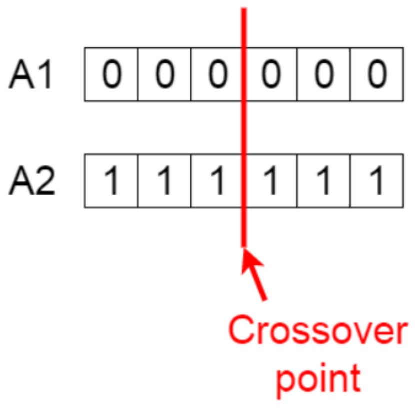


*Figure 2 Crossover Point*

**Offspring** are created by exchanging the genes of parents among themselves until the crossover point is reached.

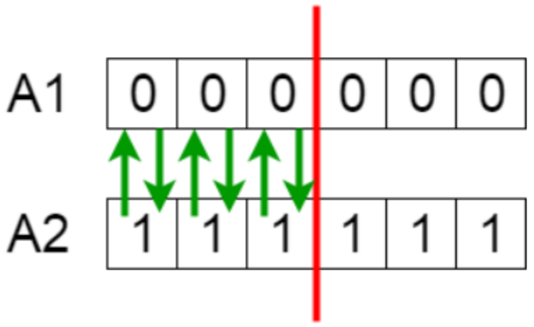


*Figure 3 Exchanging genes among parents*

The new offspring are added to the population.

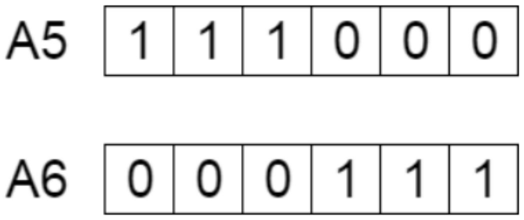


*Figure 4 new offspring*

**Mutation**

In certain new offspring formed, some of their genes can be subjected to a **mutation** with a low random probability. This implies that some of the bits in the bit string can be flipped.

Note: Mutation occurs to maintain diversity within the population and prevent premature convergence.

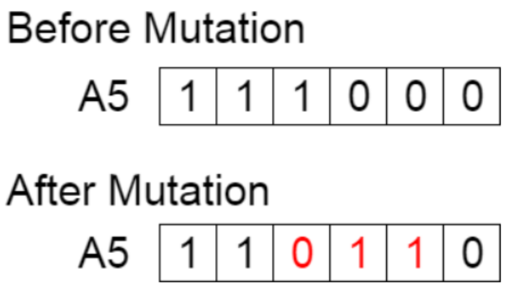


*Figure 5 Mutation before and after*

**Termination**

The algorithm terminates if the population has converged (does not produce offspring which are significantly different from the previous generation). Then it is said that the genetic algorithm has provided a set of solutions to our problem.

Few rules which are followed which tell when to stop is as follows:

- When there is no improvement in the solution quality after completing a certain number of generations set beforehand.
- When a hard and fast range of generations and time is reached.

- Till an acceptable solution is obtained.

**Pseudocode**

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

**Conclusion**

The population has a fixed size. As new generations are formed, individuals with least fitness die, providing space for new offspring.

The sequence of phases is repeated to produce individuals in each new generation which are better than the previous generation.

## 1.2 Software Simulation with sample input and output

```
In [1]: # importing packages
        import numpy as np
        import random
        import matplotlib.pyplot as plt
```

## Genetic Algorithm

```
In [2]: def generate_population(size, constraint):
            population = []

            while len(population)!=size:
                x1 = random.uniform(0,1)
                x2 = random.uniform(0,1)
                x3 = random.uniform(0,1)
                if constraint(x1,x2,x3):
                    individual = {
                        "x1": x1,
                        "x2": x2,
                        "x3": x3,
                    }
                    population.append(individual)

            return population
```

```
In [3]: def constraint(x1,x2,x3):
            c1 = (x1**2 + 2*(x2**2) + 3*(x3**2))<=1
            c2 = x1>=0 and x2>=0 and x3>=0
            return c1 and c2
```

```
In [4]: def objective_function(individual):
            x1 = individual["x1"]
            x2 = individual["x2"]
            x3 = individual["x3"]
            return x1**0.5 + x2**0.5 + x3**0.5
```

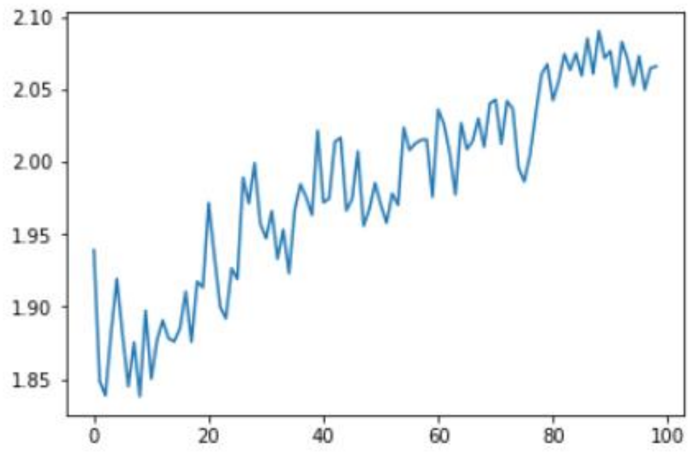
```
In [5]: def choice_by_roulette(sorted_population, fitness_sum):

            lowest_fitness = objective_function(sorted_population[0])
            draw = random.uniform(0, 1)
            accumulated = 0

            for individual in sorted_population:
                fitness = objective_function(individual)
                probability = fitness / fitness_sum
                accumulated += probability

                if draw <= accumulated:
                    return individual
```

```
In [9]: def sort_population_by_fitness(population):
            return sorted(population, key=objective_function)


        def crossover(individual_a, individual_b):

            w1 = w2 =0.5

            x1a = individual_a["x1"]
            x2a = individual_a["x2"]
            x3a = individual_a["x3"]

            x1b = individual_b["x1"]
            x2b = individual_b["x2"]
            x3b = individual_b["x3"]

            return {"x1": (w1*x1a + w2*x1b), "x2": (w1*x2a + w2*x2b) , "x3": (w1*x3a+w2*x3b)}


        def mutate(individual,constraint):
            next_x1 = individual["x1"] + random.uniform(-0.05, 0.05)
            next_x2 = individual["x2"] + random.uniform(-0.05, 0.05)
            next_x3 = individual["x3"] + random.uniform(-0.05, 0.05)

            return {"x1": next_x1, "x2": next_x2, "x3": next_x3}
```

```
In [7]: generations = 100

        population = generate_population(size=30,constraint=constraint)

        best_ones = []

        i = 1
        while True:
        #     print(f"GENERATION {i}")

        #     for individual in population:
        #         print(individual, objective_function(individual))

            if i == generations:
                break
            i += 1

            best_ones.append(objective_function(sort_population_by_fitness(population)[-1]))
            population = make_next_generation(population)


        best_individual = sort_population_by_fitness(population)[-1]
        print("\n FINAL RESULT")
        print(best_individual, objective_function(best_individual))
```

```
 FINAL RESULT
{'x1': 0.5272776531094844, 'x2': 0.40647668914634816, 'x3': 0.451299302224095} 2.0354822481674546
```

```
In [8]: plt.plot(best_ones)

Out[8]: [<matplotlib.lines.Line2D at 0x1bde3077580>]
```



**Solution to Question No. 2:**

## 2.1 Justification that the recommended algorithm is the best fit to the challenge

Particle swarm optimization (PSO) is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. It solves a problem by having a population of candidate solutions, here dubbed particles, and moving these particles around in the search-space according to simple mathematical formula over the particle's position and velocity. Each particle's movement is influenced by its local best-known position, but is also guided toward the best-known positions in the search-space, which are updated as better positions are found by other particles. This is expected to move the swarm toward the best solutions.

A single base learner is a weak learner. But, when we combine all these vulnerable learners, they become strong learners. They become strong learners because their predictive power, accuracy, precision are high. And the error rate is less. We call this type of combined model 'Meta-learning' in machine learning. It refers to learning algorithms that can learn from other learning algorithms. It decreases variance, decreases bias, and improves prediction.

Here we are speaking about finding the optimal solution in a high-dimensional solution space. It talks about Maximizing earns or minimizing losses. So, we are looking to maximize or minimize a function to find the optimum solution. A function can have multiple local maximum and minimum. But there can be only one global maximum as well as a minimum. If your function is very complex, then finding the global maximum can be a very daunting task. PSO tries to capture the global maximum or minimum. Even though it cannot capture the exact global maximum/minimum, it goes very close to it. It is the reason we called PSO a heuristic model.

In the optimization problem, we have a variable represented by a vector $X=[x_1x_2x_3 \ldots x_n]$ that minimizes or maximizes cost function depending on the proposed optimization formulation of the function f(X). X is known as **position vector**; it represents a variable model. It is an n dimensions vector, where n represents the number of variables determined in a problem. We can call it **latitude and the longitude** in the problem of choosing a point to land by a flock of birds. The function f(X) is called the **fitness function or objective function**. The job of f(X) is to assess how good or bad a position X is; that is, how perfect a certain landing points a bird thinks after finding a suitable place. Here, the evaluation, in this case, is performed through several survival criteria.

## 2.2 Discussion on the biological inspiration for the recommended algorithm

PSO is originally attributed to Kennedy, Eberhart and Shi and was first intended for simulating social behaviors, as a stylized representation of the movement of organisms in a bird flock or fish school. The algorithm was simplified and it was observed to be performing optimization.

PSO is a metaheuristic as it makes few or no assumptions about the problem being optimized and can search very large spaces of candidate solutions. Also, PSO does not use the gradient of the problem being optimized, which means PSO does not require that the optimization problem be differentiable as is required by classic optimization methods such as gradient descent and quasi-newton methods. However, metaheuristics such as PSO do not guarantee an optimal solution is ever found.

Swarm intelligence (SI) is the collective behavior of decentralized, self-organized systems, natural or artificial. The concept is employed in work on artificial intelligence. SI systems consist typically of a population of simple agents interacting locally with one another and with their environment.

The inspiration often comes from nature, especially biological systems. The agents follow very simple rules, and although there is no centralized control structure dictating how individual agents should behave, local, and to a certain degree random, interactions between such agents lead to the emergence of "intelligent" global behavior, unknown to the individual agents.

Examples of swarm intelligence in natural systems include ant colonies, bee colonies, bird flocking, hawks hunting, animal herding, bacterial growth, fish schooling and microbial intelligence.

The application of swarm principles to robots is called swarm robotics while swarm intelligence refers to the more general set of algorithms. Swarm prediction has been used in the context of forecasting problems. Similar approaches to those proposed for swarm robotics are considered for genetically modified organisms in synthetic collective intelligence.

## 2.3 Software Simulation with sample input and output

Given,
We need to design a cylindrical that can hold 200 ml of a soft drink. We need to minimize the surface area.
We know,

$$volume\ of\ a\ cylinder = A = \pi r^2 h = 200$$
$$or, h = \frac{200}{\pi r^2}$$

We also know,

$$surface\ area\ of\ a\ cylinder = 2\pi rh + 2\pi r^2$$

substituting value of h from above,

$$= 2\pi r * \left(\frac{200}{\pi r^2}\right) + 2\pi r^2$$
$$= \frac{400}{r} + 2\pi r^2$$

Hence, our objective function becomes $\frac{400}{r} + 2\pi r^2$ and we need to minimize it.

Flowchart of Particle Swarm Optimization

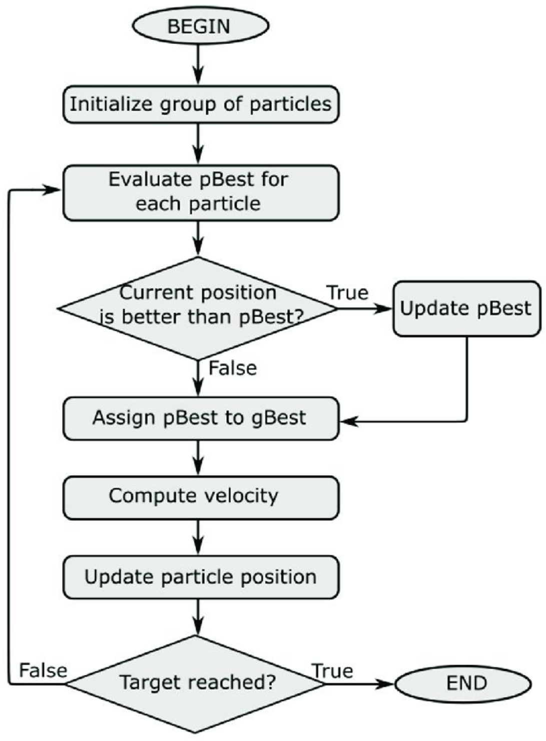
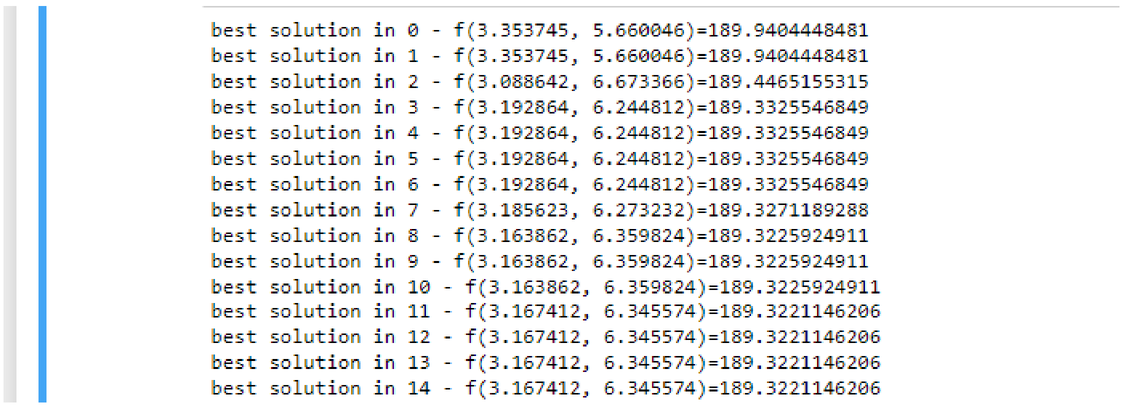


*Figure 6 Flowchart of PSO*

## Particle Swarm Optimization

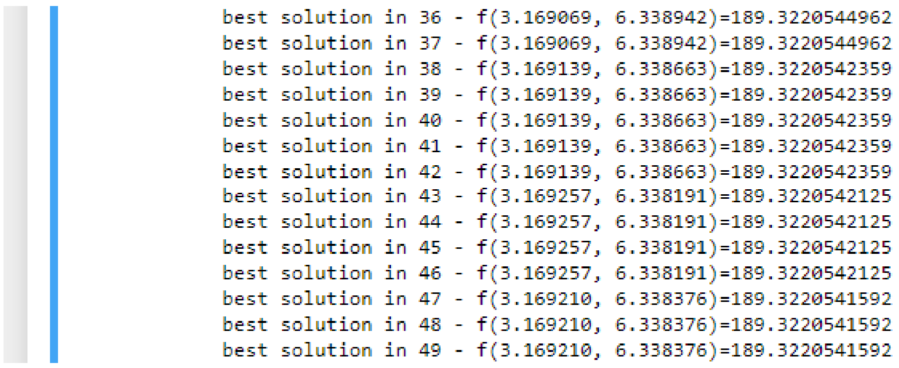
In [14]:
```python
def objective_function(x):
    return 400/x + 2*np.pi*(x**2)

x= np.linspace(0,200,100)
z = objective_function(x)

# Hyper-parameter of the algorithm
c1 = c2 = 0.1
w = 0.8

# Create particles
n_particles = 20
np.random.seed(100)

# initialize position and velocity
X = np.random.rand(n_particles) * 5
V = np.random.randn(n_particles) * 0.1

# Initialize data
pbest = X.copy()
pbest_obj = objective_function(X)
gbest = pbest[pbest_obj.argmin()]
gbest_obj = pbest_obj.min()

#Function to do one iteration of particle swarm optimization
def update():
    global V, X, pbest, pbest_obj, gbest, gbest_obj

    # Update params
    r1, r2 = np.random.rand(2)

    # update position and velocity
    V = w * V + c1*r1*(pbest - X) + c2*r2*(gbest-X)
    X = X + V

    # update personal_best and global_best
    obj = objective_function(X)
    pbest[(pbest_obj >= obj)] = X[(pbest_obj >= obj)]
    pbest_obj = np.array([pbest_obj, obj]).min(axis=0)
    gbest = pbest[pbest_obj.argmin()]
    gbest_obj = pbest_obj.min()

for i in range(50):
    update()
    print("best solution in {} - f({:.6f}, {:.6f})={:.10f}".format(i,gbest,200/(np.pi*gbest**2), gbest_obj))
```

Output:

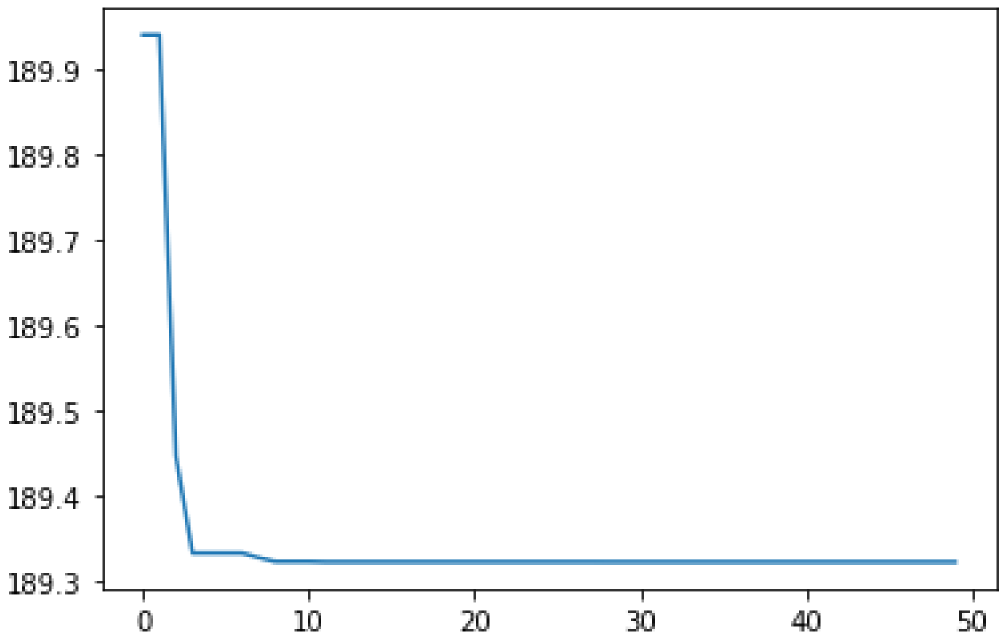


```
best solution in 0 - f(3.353745, 5.660046)=189.9404448481
best solution in 1 - f(3.353745, 5.660046)=189.9404448481
best solution in 2 - f(3.088642, 6.673366)=189.4465155315
best solution in 3 - f(3.192864, 6.244812)=189.3325546849
best solution in 4 - f(3.192864, 6.244812)=189.3325546849
best solution in 5 - f(3.192864, 6.244812)=189.3325546849
best solution in 6 - f(3.192864, 6.244812)=189.3325546849
best solution in 7 - f(3.185623, 6.273232)=189.3271189288
best solution in 8 - f(3.163862, 6.359824)=189.3225924911
best solution in 9 - f(3.163862, 6.359824)=189.3225924911
best solution in 10 - f(3.163862, 6.359824)=189.3225924911
best solution in 11 - f(3.167412, 6.345574)=189.3221146206
best solution in 12 - f(3.167412, 6.345574)=189.3221146206
best solution in 13 - f(3.167412, 6.345574)=189.3221146206
best solution in 14 - f(3.167412, 6.345574)=189.3221146206
```

```
                              :::    :::
                              :::    :::
                              :::    :::
```



```
best solution in 36 - f(3.169069, 6.338942)=189.3220544962
best solution in 37 - f(3.169069, 6.338942)=189.3220544962
best solution in 38 - f(3.169139, 6.338663)=189.3220542359
best solution in 39 - f(3.169139, 6.338663)=189.3220542359
best solution in 40 - f(3.169139, 6.338663)=189.3220542359
best solution in 41 - f(3.169139, 6.338663)=189.3220542359
best solution in 42 - f(3.169139, 6.338663)=189.3220542359
best solution in 43 - f(3.169257, 6.338191)=189.3220542125
best solution in 44 - f(3.169257, 6.338191)=189.3220542125
best solution in 45 - f(3.169257, 6.338191)=189.3220542125
best solution in 46 - f(3.169257, 6.338191)=189.3220542125
best solution in 47 - f(3.169210, 6.338376)=189.3220541592
best solution in 48 - f(3.169210, 6.338376)=189.3220541592
best solution in 49 - f(3.169210, 6.338376)=189.3220541592
```



### 2.4 Conclusion

Particle swarm optimization (PSO) is considered one of the most important methods in swarm intelligence. PSO is related to the study of swarms; where it is a simulation of bird flocks. It can be used to solve a wide variety of optimization problems such as unconstrained optimization problems, constrained optimization problems, nonlinear programming, multi-objective optimization, stochastic programming and combinatorial optimization problems.

Genetic Algorithms (GAs) and PSOs are both used as cost functions, they are both iterative, and they both have a random element. They can be used on similar kinds of problems. The difference between PSO and Genetic Algorithms (GAs) is that GAs it does not traverse the search space like birds flocking, covering the spaces in between. The operation of GAs is more like Monte Carlo, where the candidate solutions are randomized, and the best solutions are picked to compete with a new set of randomized solutions. Also, PSO algorithms require normalization of the input vectors to reach faster "convergence" (as heuristic algorithms, both don't truly converge). GAs can work with features that are continuous or discrete.

***Advantages :***

1. Insensitive to scaling of design variables.
2. Easily parallelized for concurrent processing.
3. Derivative free.
4. Very few algorithm parameters.
5. A very efficient global search algorithm.

***Disadvantages :***

1. PSO's optimum local searchability is weak

## Bibliography

1. https://en.wikipedia.org/wiki/Particle_swarm_optimization
2. https://en.wikipedia.org/wiki/Swarm_intelligence
3. https://www.analyticsvidhya.com/blog/2021/10/an-introduction-to-particle-swarm-optimization-algorithm/