RAMAIAH UNIVERSITY OF APPLIED SCIENCES

ASSIGNMENT

Course Code	19CSE422A
Course Name	Computational Intelligence
Programme	B.Tech
Department	CSE
Faculty	FET

Name of the Student	K Srikanth
Reg. No	17ETCS002124
Semester/Year	7th/ 4th Year
Course Leader/s	Dr. Vaishali R. Kulkarni and Dr. Monika Ravishankar

Declaration Sheet			
Student Name	K Srikanth		
Reg. No	17ETCS002124		
Programme	B.Tech	Semester/Year	7th/ 4th Year
Course Code	19CSE422A		
Course Title	Computational Intelligence		
Course Date		to	
Course Leader	Dr. Vaishali R. Kulkarni and Dr. Monika Ravishankar		

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	Srikanth K	Date	25/11/2021
Submission date stamp (by Examination & Assessment Section)			

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

RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Faculty of Engineering and Technology			
Ramaiah University of Applied Sciences			
Department	Computer Science and Engineering	Programme	B. Tech. in CSE- Summer
Semester/Batch	7/2018		
Course Code	19CSE422A	Course Title	Computational Intelligence
Course Leader	Dr. Vaishali R. Kulkarni and Dr. Monika Ravishankar		

Assignment-1			
Reg.No.		Name of Student	

Sections	Marking Scheme		Marks		
			Max Marks	First Examiner Marks	Moderator
Part A	Part A				
	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)				

Name: K Srikanth Registration Number: 17ETCS002124

Part A

Q-A.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 problems which otherwise would take a lifetime to solve. It is frequently used to solve optimization problems, in research, and in machine learning.

Now let's look at how can we solve one problem using genetic algorithm,

Let us estimate the optimal values of a and b using GA which satisfies for any optimization problem starts with an objective function, so our question this is our objective function,

$$\begin{cases} \max \sqrt{x_1} + \sqrt{x_2} + \sqrt{x_3} \\ \text{subject to:} \\ \qquad x_1^2 + 2x_2^2 + 3x_3^2 \leq 1 \\ \qquad x_1, x_2, x_3 \geq 0. \end{cases}$$

It is understood that the value of the function is 0. This function is our objective function and the aim is to estimate values of a and b such that the value of the objective function gets minimized to zero. The entire optimization process is explained below in four major steps and coded in R for one iteration (or generation).

Initializing Population,

This step starts with guessing of initial sets of x1, x2 and x3values which may or may not include the optimal values. These sets of values are called as '**chromosomes** and the step are called **'initialize population'**. Here population means sets of x1, x2 and x3, [x1, x2, x3,]. Random uniform function is used to generate initial values of x1, x2 and x3

Selecting Chromosomes,

In this step, the value of the objective function for each chromosome is computed. The value of the objective function is also called fitness value. This step is very important and is called 'selection' because fittest chromosomes are selected from the population for subsequent operations.

Based on the fitness values, more suitable chromosomes who have possibilities of producing low values of fitness function (because the value of our objective function needs to be 0) are selected and allowed to survive in succeeding generations. Some chromosomes are discarded to be unsuitable to produce low fitness value.

Now that we know how to initialize the population and select chromosomes, Now let's look at the algorithm with respect to this problem,

<u>Algorithm</u>

```
1. Start
2. IMPORT numpy
3. IMPORT sys
4. SET equation_INPUTs TO [1,1,1]
5. SET num_weights TO 3
6. SET chromosomes TO 30
7. SET num_generations TO 100
8. SET num_parents_mating TO 4
9. SET pop_size TO (chromosomes,num_weights)
10. SET new_population TO numpy.random.uniform(low=0, size=pop_size)
11. SET smallestFitness TO -sys.maxsize - 1
12. new_population[0:5]
13. DEFINE FUNCTION cal_pop_fitness(equation_INPUTs, pop):
    a. SET fitness TO []
    b. FOR i,chromosome IN enumerate(pop):
        • (x1,x2,x3)=chromosome
        • IF (x1*x1+2*x2*x2+3*x3*x3) >1 or x1<0 or x2<0 or x3<0:
          #x1^2+2*x2^2+3*x3^2<=1  # x1,x2,x3>=0
        • fitness.append(smallestFitness )
    • continue
    • chromosome=numpy.sqrt(chromosome)
    • fitness.append(sum(chromosome))
    • RETURN numpy.array(fitness)
```

14. DEFINE FUNCTION select_mating_pool(pop, fitness, num_parents):

```
  a. SET parents TO numpy.empty((num_parents, pop.shape[1]))
  b. FOR parent_num IN range(num_parents):
       • SET max_fitness_idx TO numpy.where(fitness EQUALS numpy.max(fitness))
       • SET max_fitness_idx TO max_fitness_idx[0][0]
       • SET parents[parent_num, :] TO pop[max_fitness_idx, :]
       • SET fitness[max_fitness_idx] TO smallestFitness
       • RETURN parents
```

15. DEFINE FUNCTION crossover(parents, offspring_size):

```
  a. SET offspring TO numpy.empty(offspring_size)
  b. SET crossover_point TO numpy.uint8(offspring_size[1]/2)
  c. FOR k IN range(offspring_size[0]):
       • SET parent1_idx TO k%parents.shape[0]
       • SET parent2_idx TO (k+1)%parents.shape[0]
       • SET offspring[k, 0:crossover_point] TO parents[parent1_idx,
         0:crossover_point]
       • SET offspring[k, crossover_point:] TO parents[parent2_idx, crossover_point:]
       • RETURN offspring
```

16. DEFINE FUNCTION mutation(offspring_crossover, num_mutations=1):

```
a. SET mutations_counter TO numpy.uint8(offspring_crossover.shape[1] /
   num_mutations)
b. FOR idx IN range(offspring_crossover.shape[0]):
     • SET gene_idx TO mutations_counter - 1
     • FOR mutation_num IN range(num_mutations):
         ▪ SET random_value TO numpy.random.uniform(-1.0, 1.0, 1)
         ▪ SET offspring_crossover[idx, gene_idx] TO offspring_crossover[idx, gene_idx]
           + random_value
     • SET gene_idx TO gene_idx + mutations_counter
• RETURN offspring_crossover
```

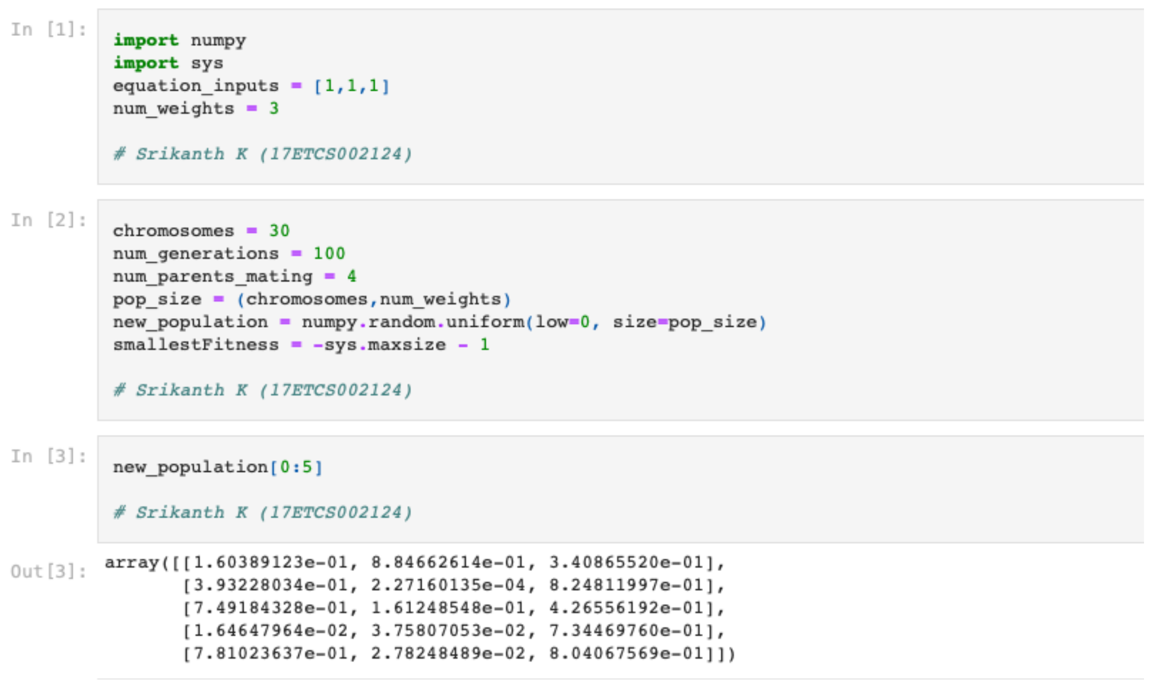
17. FOR generation IN range(num_generations):

```
• OUTPUT(f"\nGeneration : {generation}")
• SET fitness TO cal_pop_fitness(equation_INPUTs, new_population)
• SET parents TO select_mating_pool(new_population, fitness,num_parents_mating)
• OUTPUT("Parents selected FOR crossover")
• OUTPUT(parents)
• SET offspring_crossover TO crossover(parents,offspring_size=(pop_size[0]-
  parents.shape[0], num_weights))
• OUTPUT("Crossover results")
• OUTPUT(offspring_crossover)
• SET offspring_mutation TO mutation(offspring_crossover)
• OUTPUT("Mutation results")
```

- OUTPUT(offspring_mutation)
- SET new_population[0:parents.shape[0], :] TO parents
- SET new_population[parents.shape[0]:, :] TO offspring_mutation
- OUTPUT(f"\nBest result FOR iteration {generation} : {numpy.max(numpy.sum(new_population*equation_INPUTs, axis=1))} ")
- SET fitness TO cal_pop_fitness(equation_INPUTs, new_population)
- SET best_match_idx TO numpy.where(fitness EQUALS numpy.max(fitness))
- OUTPUT("Best solution : ", new_population[best_match_idx, :])
- OUTPUT("Best solution fitness : ", fitness[best_match_idx])

Q-A-2

Python Code

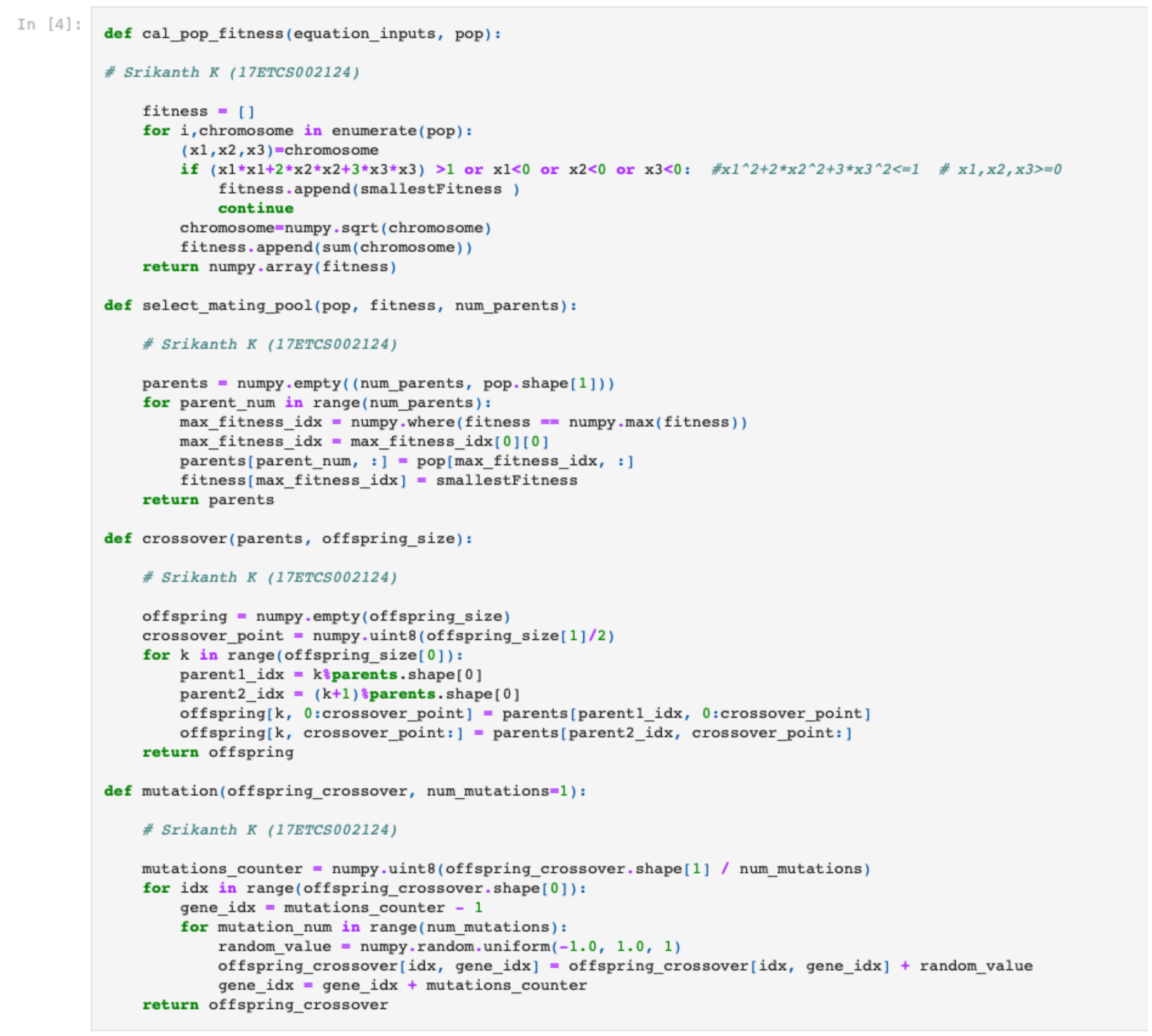


```
In [1]:
import numpy
import sys
equation_inputs = [1,1,1]
num_weights = 3

# Srikanth K (17ETCS002124)
```

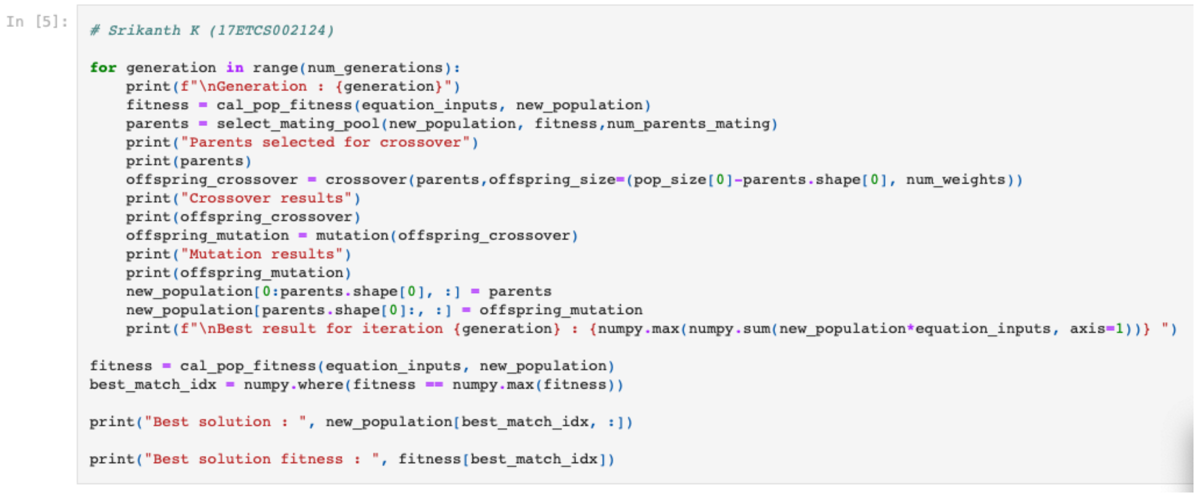
```
In [2]:
chromosomes = 30
num_generations = 100
num_parents_mating = 4
pop_size = (chromosomes,num_weights)
new_population = numpy.random.uniform(low=0, size=pop_size)
smallestFitness = -sys.maxsize - 1

# Srikanth K (17ETCS002124)
```

```
In [3]:
new_population[0:5]

# Srikanth K (17ETCS002124)
```

```
Out[3]: array([[1.60389123e-01, 8.84662614e-01, 3.40865520e-01],
               [3.93228034e-01, 2.27160135e-04, 8.24811997e-01],
               [7.49184328e-01, 1.61248548e-01, 4.26556192e-01],
               [1.64647964e-02, 3.75807053e-02, 7.34469760e-01],
               [7.81023637e-01, 2.78248489e-02, 8.04067569e-01]])
```

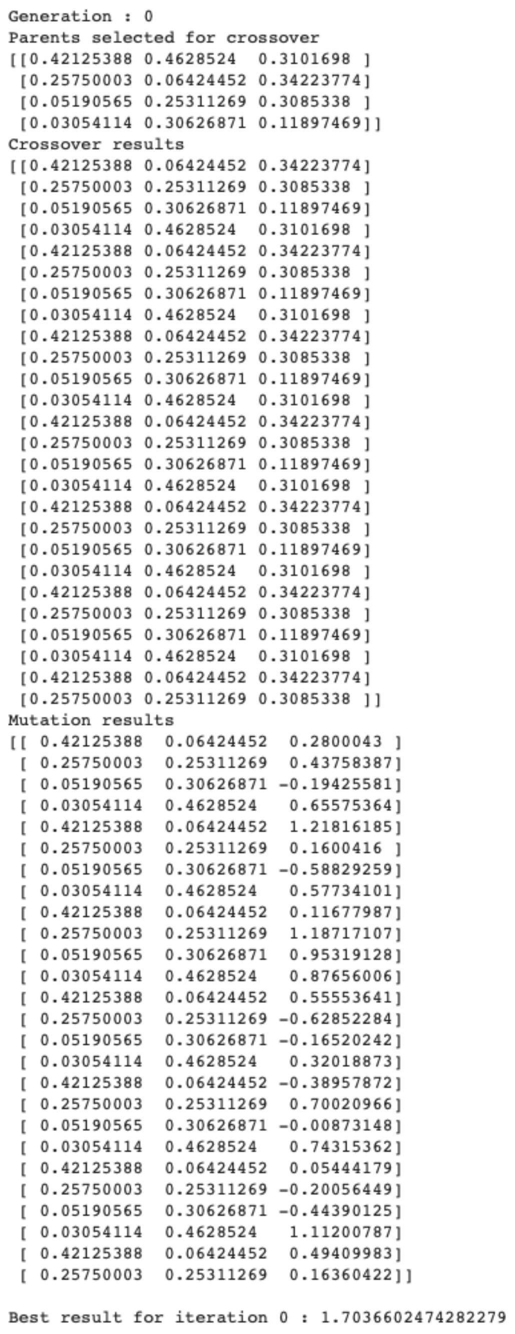


```
In [4]:
def cal_pop_fitness(equation_inputs, pop):

# Srikanth K (17ETCS002124)

    fitness = []
    for i,chromosome in enumerate(pop):
        (x1,x2,x3)=chromosome
        if (x1*x1+2*x2*x2+3*x3*x3) >1 or x1<0 or x2<0 or x3<0:  #x1^2+2*x2^2+3*x3^2<=1  # x1,x2,x3>=0
            fitness.append(smallestFitness )
            continue
        chromosome=numpy.sqrt(chromosome)
        fitness.append(sum(chromosome))
    return numpy.array(fitness)

def select_mating_pool(pop, fitness, num_parents):

    # Srikanth K (17ETCS002124)

    parents = numpy.empty((num_parents, pop.shape[1]))
    for parent_num in range(num_parents):
        max_fitness_idx = numpy.where(fitness == numpy.max(fitness))
        max_fitness_idx = max_fitness_idx[0][0]
        parents[parent_num, :] = pop[max_fitness_idx, :]
        fitness[max_fitness_idx] = smallestFitness
    return parents

def crossover(parents, offspring_size):

    # Srikanth K (17ETCS002124)

    offspring = numpy.empty(offspring_size)
    crossover_point = numpy.uint8(offspring_size[1]/2)
    for k in range(offspring_size[0]):
        parent1_idx = k%parents.shape[0]
        parent2_idx = (k+1)%parents.shape[0]
        offspring[k, 0:crossover_point] = parents[parent1_idx, 0:crossover_point]
        offspring[k, crossover_point:] = parents[parent2_idx, crossover_point:]
    return offspring

def mutation(offspring_crossover, num_mutations=1):

    # Srikanth K (17ETCS002124)

    mutations_counter = numpy.uint8(offspring_crossover.shape[1] / num_mutations)
    for idx in range(offspring_crossover.shape[0]):
        gene_idx = mutations_counter - 1
        for mutation_num in range(num_mutations):
            random_value = numpy.random.uniform(-1.0, 1.0, 1)
            offspring_crossover[idx, gene_idx] = offspring_crossover[idx, gene_idx] + random_value
            gene_idx = gene_idx + mutations_counter
    return offspring_crossover
```

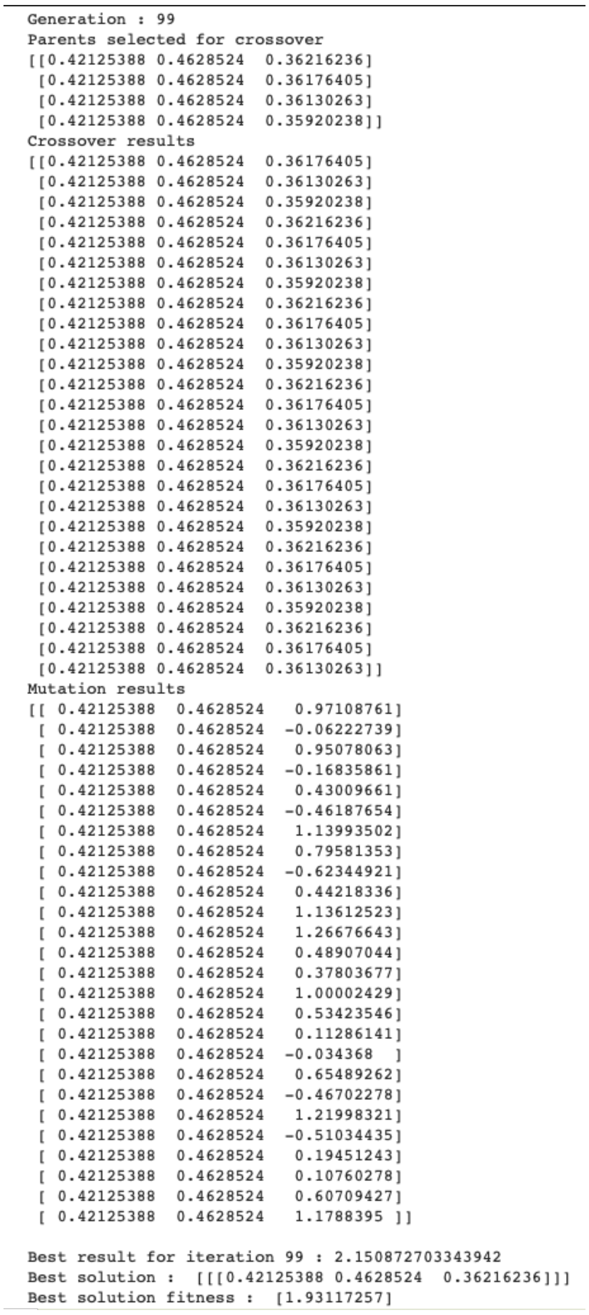


```
In [5]:
# Srikanth K (17ETCS002124)

for generation in range(num_generations):
    print(f"\nGeneration : {generation}")
    fitness = cal_pop_fitness(equation_inputs, new_population)
    parents = select_mating_pool(new_population, fitness,num_parents_mating)
    print("Parents selected for crossover")
    print(parents)
    offspring_crossover = crossover(parents,offspring_size=(pop_size[0]-parents.shape[0], num_weights))
    print("Crossover results")
    print(offspring_crossover)
    offspring_mutation = mutation(offspring_crossover)
    print("Mutation results")
    print(offspring_mutation)
    new_population[0:parents.shape[0], :] = parents
    new_population[parents.shape[0]:, :] = offspring_mutation
    print(f"\nBest result for iteration {generation} : {numpy.max(numpy.sum(new_population*equation_inputs, axis=1))} ")

fitness = cal_pop_fitness(equation_inputs, new_population)
best_match_idx = numpy.where(fitness == numpy.max(fitness))

print("Best solution : ", new_population[best_match_idx, :])

print("Best solution fitness : ", fitness[best_match_idx])
```

<u>Output of Python Code</u>



```
Generation : 0
Parents selected for crossover
[[0.42125388 0.4628524  0.3101698 ]
 [0.25750003 0.06424452 0.34223774]
 [0.05190565 0.25311269 0.3085338 ]
 [0.03054114 0.30626871 0.11897469]]
Crossover results
[[0.42125388 0.06424452 0.34223774]
 [0.25750003 0.25311269 0.3085338 ]
 [0.05190565 0.30626871 0.11897469]
 [0.03054114 0.4628524  0.3101698 ]
 [0.42125388 0.06424452 0.34223774]
 [0.25750003 0.25311269 0.3085338 ]
 [0.05190565 0.30626871 0.11897469]
 [0.03054114 0.4628524  0.3101698 ]
 [0.42125388 0.06424452 0.34223774]
 [0.25750003 0.25311269 0.3085338 ]
 [0.05190565 0.30626871 0.11897469]
 [0.03054114 0.4628524  0.3101698 ]
 [0.42125388 0.06424452 0.34223774]
 [0.25750003 0.25311269 0.3085338 ]
 [0.05190565 0.30626871 0.11897469]
 [0.03054114 0.4628524  0.3101698 ]
 [0.42125388 0.06424452 0.34223774]
 [0.25750003 0.25311269 0.3085338 ]
 [0.05190565 0.30626871 0.11897469]
 [0.03054114 0.4628524  0.3101698 ]
 [0.42125388 0.06424452 0.34223774]
 [0.25750003 0.25311269 0.3085338 ]
 [0.05190565 0.30626871 0.11897469]
 [0.03054114 0.4628524  0.3101698 ]
 [0.42125388 0.06424452 0.34223774]
 [0.25750003 0.25311269 0.3085338 ]]
Mutation results
[[ 0.42125388  0.06424452  0.2800043 ]
 [ 0.25750003  0.25311269  0.43758387]
 [ 0.05190565  0.30626871 -0.19425581]
 [ 0.03054114  0.4628524   0.65575364]
 [ 0.42125388  0.06424452  1.21816185]
 [ 0.25750003  0.25311269  0.1600416 ]
 [ 0.05190565  0.30626871 -0.58829259]
 [ 0.03054114  0.4628524   0.57734101]
 [ 0.42125388  0.06424452  0.11677987]
 [ 0.25750003  0.25311269  1.18717107]
 [ 0.05190565  0.30626871  0.95319128]
 [ 0.03054114  0.4628524   0.87656006]
 [ 0.42125388  0.06424452  0.55553641]
 [ 0.25750003  0.25311269 -0.62852284]
 [ 0.05190565  0.30626871 -0.16520242]
 [ 0.03054114  0.4628524   0.32018873]
 [ 0.42125388  0.06424452 -0.38957872]
 [ 0.25750003  0.25311269  0.70020966]
 [ 0.05190565  0.30626871 -0.00873148]
 [ 0.03054114  0.4628524   0.74315362]
 [ 0.42125388  0.06424452  0.05444179]
 [ 0.25750003  0.25311269 -0.20056449]
 [ 0.05190565  0.30626871 -0.44390125]
 [ 0.03054114  0.4628524   1.11200787]
 [ 0.42125388  0.06424452  0.49409983]
 [ 0.25750003  0.25311269  0.16360422]]

Best result for iteration 0 : 1.7036602474282279
```



```
Generation : 99
Parents selected for crossover
[[0.42125388 0.4628524  0.36216236]
 [0.42125388 0.4628524  0.36176405]
 [0.42125388 0.4628524  0.36130263]
 [0.42125388 0.4628524  0.35920238]]
Crossover results
[[0.42125388 0.4628524  0.36176405]
 [0.42125388 0.4628524  0.36130263]
 [0.42125388 0.4628524  0.35920238]
 [0.42125388 0.4628524  0.36216236]
 [0.42125388 0.4628524  0.36176405]
 [0.42125388 0.4628524  0.36130263]
 [0.42125388 0.4628524  0.35920238]
 [0.42125388 0.4628524  0.36216236]
 [0.42125388 0.4628524  0.36176405]
 [0.42125388 0.4628524  0.36130263]
 [0.42125388 0.4628524  0.35920238]
 [0.42125388 0.4628524  0.36216236]
 [0.42125388 0.4628524  0.36176405]
 [0.42125388 0.4628524  0.36130263]
 [0.42125388 0.4628524  0.35920238]
 [0.42125388 0.4628524  0.36216236]
 [0.42125388 0.4628524  0.36176405]
 [0.42125388 0.4628524  0.36130263]
 [0.42125388 0.4628524  0.35920238]
 [0.42125388 0.4628524  0.36216236]
 [0.42125388 0.4628524  0.36176405]
 [0.42125388 0.4628524  0.36130263]
 [0.42125388 0.4628524  0.35920238]
 [0.42125388 0.4628524  0.36216236]
 [0.42125388 0.4628524  0.36176405]
 [0.42125388 0.4628524  0.36130263]]
Mutation results
[[ 0.42125388  0.4628524   0.97108761]
 [ 0.42125388  0.4628524  -0.06222739]
 [ 0.42125388  0.4628524   0.95078063]
 [ 0.42125388  0.4628524  -0.16835861]
 [ 0.42125388  0.4628524   0.43009661]
 [ 0.42125388  0.4628524  -0.46187654]
 [ 0.42125388  0.4628524   1.13993502]
 [ 0.42125388  0.4628524   0.79581353]
 [ 0.42125388  0.4628524  -0.62344921]
 [ 0.42125388  0.4628524   0.44218336]
 [ 0.42125388  0.4628524   1.13612523]
 [ 0.42125388  0.4628524   1.26676643]
 [ 0.42125388  0.4628524   0.48907044]
 [ 0.42125388  0.4628524   0.37803677]
 [ 0.42125388  0.4628524   1.00002429]
 [ 0.42125388  0.4628524   0.53423546]
 [ 0.42125388  0.4628524   0.11286141]
 [ 0.42125388  0.4628524  -0.034368  ]
 [ 0.42125388  0.4628524   0.65489262]
 [ 0.42125388  0.4628524  -0.46702278]
 [ 0.42125388  0.4628524   1.21998321]
 [ 0.42125388  0.4628524  -0.51034435]
 [ 0.42125388  0.4628524   0.19451243]
 [ 0.42125388  0.4628524   0.10760278]
 [ 0.42125388  0.4628524   0.60709427]
 [ 0.42125388  0.4628524   1.1788395 ]]

Best result for iteration 99 : 2.150872703343942
Best solution :  [[[0.42125388 0.4628524  0.36216236]]]
Best solution fitness :  [1.93117257]
```

<u>Conclusion</u>

As we can see that after 100 iterations (Couldn't include with the space limitation) the best solution that was generated with the chromosomes are

- **0.42125388**
- **0.4628524**
- **0.36216236**

With the best fitness solution being **1.93117257.**

Part B

Q-B.1

The algorithm I'm going to choose to solve this problem is: **Particle Swarm Optimization.**

Introduction

PSO is old and is the most used swarm intelligence algorithm. The general idea of PSO is inspired by a flying swarm of birds searching for food.). The task of object tracking is considered as a numerical optimization problem, where a PSO is used to track the local mode of the similarity measure and to seek a good local minimum, and then the conjugate gradient is utilized to find the local minimum accurately. But the ordinary PSO is not well suited for multiple object tracking. The algorithm introduces two new components to PSO: a self-adapting component, which is robust against drastic brightness changes of the image sequence, and a self-splitting component, which decides to track the scene as one connected object, or as more stand-alone objects.

Justification

PSO is best used to find the maximum or minimum of a function defined on a multidimensional vector space. Assume we have a function f(X) that produces a real value from a vector parameters radius and height and Radius can take on virtually any value in the space then we can apply PSO. The PSO algorithm will return the parameter X it found that produces the minimum f(X).

Let's start with the following function,

$$2\pi rh + 2(\pi r^2)$$

It is not a **convex function** and therefore it is hard to find its minimum because a **local minimum** found is not necessarily the **global minimum**. So how can we find the minimum point in this function? For sure, we can resort to exhaustive search: If we check the value of function for every point on the plane, we can find the minimum point. Or we can just randomly find some sample points on the plane and see which one gives the lowest value on function if we think it is too expensive to search every point. However, we also note from

the shape of function that if we have found a point with a smaller value of function it is easier to find an even smaller value around its proximity.
The main advantages of the PSO algorithm are summarized as: simple concept, easy implementation, robustness to control parameters, and computational efficiency when compared with mathematical algorithm and other heuristic optimization techniques.
maximum iteration number, Iteration current iteration number.

Q-B.2

How does this algorithm work? in *mathematical terms,

Assume we have P particles and we denote the position of particle ‘I’ at iteration t as Xi(t), which in the example of above, we have it as a coordinate,

Xi(t)=(xi(t),yi(t))

Besides the position, we also have a velocity for each particle, denoted as

Vi(t)=(vxi(t),vyi(t)).

At the next iteration, the position of each particle would be updated as

$$X^i(t+1) = X^i(t) + V^i(t+1)$$

or, equivalently,

$$x^i(t+1) = x^i(t) + v_x^i(t+1)$$
$$y^i(t+1) = y^i(t) + v_y^i(t+1)$$

and at the same time, the velocities are also updated by the rule

$$V^i(t+1) = wV^i(t) + c_1r_1(\text{pbest}^i - X^i(t)) + c_2r_2(\text{gbest} - X^i(t))$$

where r1 and r2 are random numbers **between 0 and 1, constants w, c1, and c2** are parameters to the PSO algorithm, and **pbest** is the position that gives the best f(X) value ever explored by particle **i** and **gbest** is that explored by all the particles in the swarm.

Note that **pbest** and Xi(t) are two position vectors and the difference **pbest–Xi(t)** is a vector subtraction. Adding this subtraction to the original velocity Vi(t) is to bring the particle back to the position **pbest.** Similar are for the difference **gbest–Xi(t).**

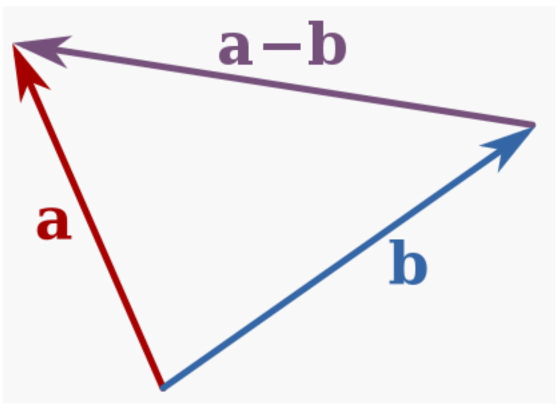


Figure 1 Vector Subtraction of PSO

We call the parameter w the inertia weight constant. It is between 0 and 1 and determines how much should the particle keep on with its previous velocity (i.e., speed and direction of the search). The parameters c1 and c2 are called the cognitive and the social coefficients respectively. They controls how much weight should be given between refining the search result of the particle itself and recognizing the search result of the swarm. We can consider these parameters controls the trade-off between **exploration** and **exploitation**.
The positions **pbest** and **gbest** are updated in each iteration to reflect the best position ever found thus far.

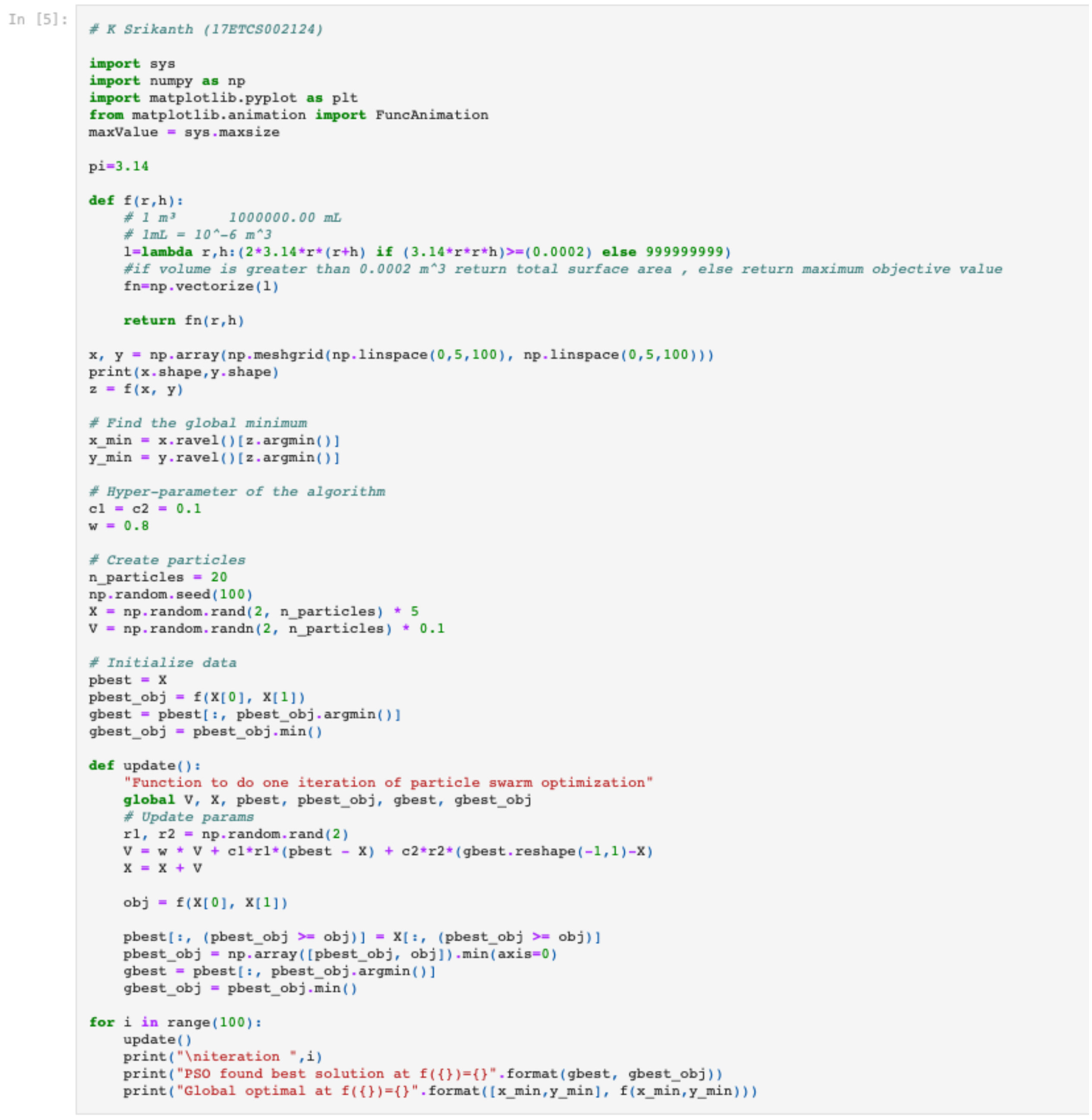
One interesting property of this algorithm that distinguish it from other optimization algorithms is that it does not depend on the gradient of the objective function. In gradient descent, for example, we look for the minimum of a function f(X) by moving X to the direction of $-\nabla f(X)$ as it is where the function going down the fastest. For any particle at the position X at the moment, how it moves does not depend on which direction is the “downhill” but only on where are **pbest** and **gbest.** This makes PSO particularly suitable if differentiating f(X) is difficult.

Another property of PSO is that it can be parallelized easily. As we are manipulating multiple particles to find the optimal solution, each particles can be updated in parallel and we only need to collect the updated value of **gbest** once per iteration. This makes map-reduce architecture a perfect candidate to implement PSO.

This is how a particle swarm optimization does. Similar to the flock of birds looking for food, we start with a number of random points on the plane (call them **particles**) and let them look for the minimum point in random directions. At each step, every particle should search around the minimum point it ever found as well as around the minimum point found by the entire swarm of particles. After certain iterations, we consider the minimum point of the function as the minimum point ever explored by this swarm of particles.

Q-B.3

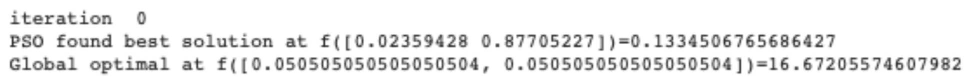
Python Code for PSO



```
In [5]:
# K Srikanth (17ETCS002124)

import sys
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
maxValue = sys.maxsize

pi=3.14

def f(r,h):
    # 1 m³       1000000.00 mL
    # 1mL = 10^-6 m^3
    l=lambda r,h:(2*3.14*r*(r+h) if (3.14*r*r*h)>=(0.0002) else 999999999)
    #if volume is greater than 0.0002 m^3 return total surface area , else return maximum objective value
    fn=np.vectorize(l)

    return fn(r,h)

x, y = np.array(np.meshgrid(np.linspace(0,5,100), np.linspace(0,5,100)))
print(x.shape,y.shape)
z = f(x, y)

# Find the global minimum
x_min = x.ravel()[z.argmin()]
y_min = y.ravel()[z.argmin()]

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

# Create particles
n_particles = 20
np.random.seed(100)
X = np.random.rand(2, n_particles) * 5
V = np.random.randn(2, n_particles) * 0.1

# Initialize data
pbest = X
pbest_obj = f(X[0], X[1])
gbest = pbest[:, pbest_obj.argmin()]
gbest_obj = pbest_obj.min()

def update():
    "Function to do one iteration of particle swarm optimization"
    global V, X, pbest, pbest_obj, gbest, gbest_obj
    # Update params
    r1, r2 = np.random.rand(2)
    V = w * V + c1*r1*(pbest - X) + c2*r2*(gbest.reshape(-1,1)-X)
    X = X + V

    obj = f(X[0], X[1])

    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(100):
    update()
    print("\niteration ",i)
    print("PSO found best solution at f({})={}".format(gbest, gbest_obj))
    print("Global optimal at f({})={}".format([x_min,y_min], f(x_min,y_min)))
```

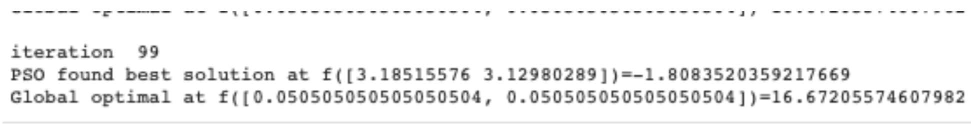
Output for PSO

At 1st iteration



```
iteration  0
PSO found best solution at f([0.02359428 0.87705227])=0.1334506765686427
Global optimal at f([0.050505050505050504, 0.050505050505050504])=16.67205574607982
```

At 100th iteration



```
iteration  99
PSO found best solution at f([3.18515576 3.12980289])=-1.8083520359217669
Global optimal at f([0.050505050505050504, 0.050505050505050504])=16.67205574607982
```

Q-B.4

In the present work, we have used PSO Algorithm to find out best fit for our problem and optimized the solution with global best being **16.67205574607982** and PSO best being -**1.8083520359217669.** As discussed earlier, it is impracticable to say that the result obtained by an optimization method such as PSO is the global maximum or minimum, so some authors call the results as the most likely optimal global. Thus, some strategies can be employed in order to verify the validity of the optimal results obtained. One of the strategies is to compare with the results obtained by other optimization algorithms, as used in the present work. In the absence of optimal data available, due to either computational limitations or even lack of results of the subject, it is possible to use as strategy the comparison of information from real physical models, that is, that were not obtained through optimization algorithms, but instead good engineering practice and judgment gained through technical experience.

In addition, it was possible to apply the PSO algorithm to different engineering problems. The first involves the spacer grid of the fuel element and the second involves the optimization of the cost function of a cogeneration system. In both problems, satisfactory results were obtained demonstrating the efficiency of the PSO method.