Simulation of Particle Swarm Optimization Algorithm

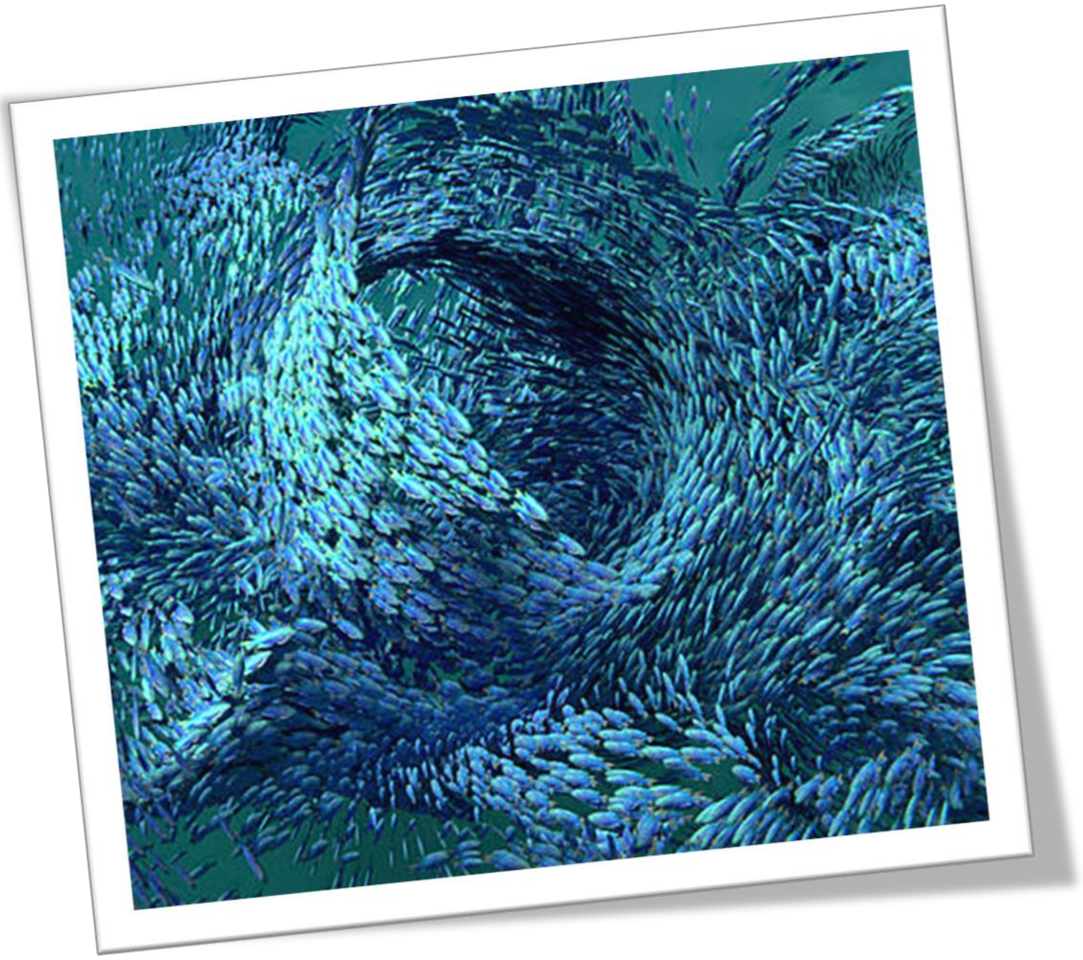
Submitted by:

Aneesh Panchal - 2K20/MC/21
Ayushi sagar - 2K20/MC/35

Submitted to:

Ms. Neha Punetha

Department of Applied Mathematics
Delhi Technological University



Certificate

I hereby certify that the project dissertation titled “Simulation of Particle Swarm Optimization Algorithm” which is submitted by Aneesh Panchal (2K20/MC/21) and Ayushi Sagar (2K20/MC/35) of Mathematics and Computing Department, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Bachelor of Technology, is a record of the project work carried out by the students. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this university or elsewhere.

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

Acknowledgement

We, Aneesh Panchal(2K20/MC/21) and Ayushi Sagar (2K20/MC/35) would like to express our special thanks of gratitude to Ms. Neha Punetha, Department of Applied Mathematics, Delhi Technological University for his able guidance and support in completing our project report.

We came to know about many new things, we are really thankful to him. We would also like to thank our classmates who have also helped whenever we were stuck at some point.

Thanking You

Aneesh Panchal (2K20/MC/21)
Ayushi Sagar (2K20/MC/35)

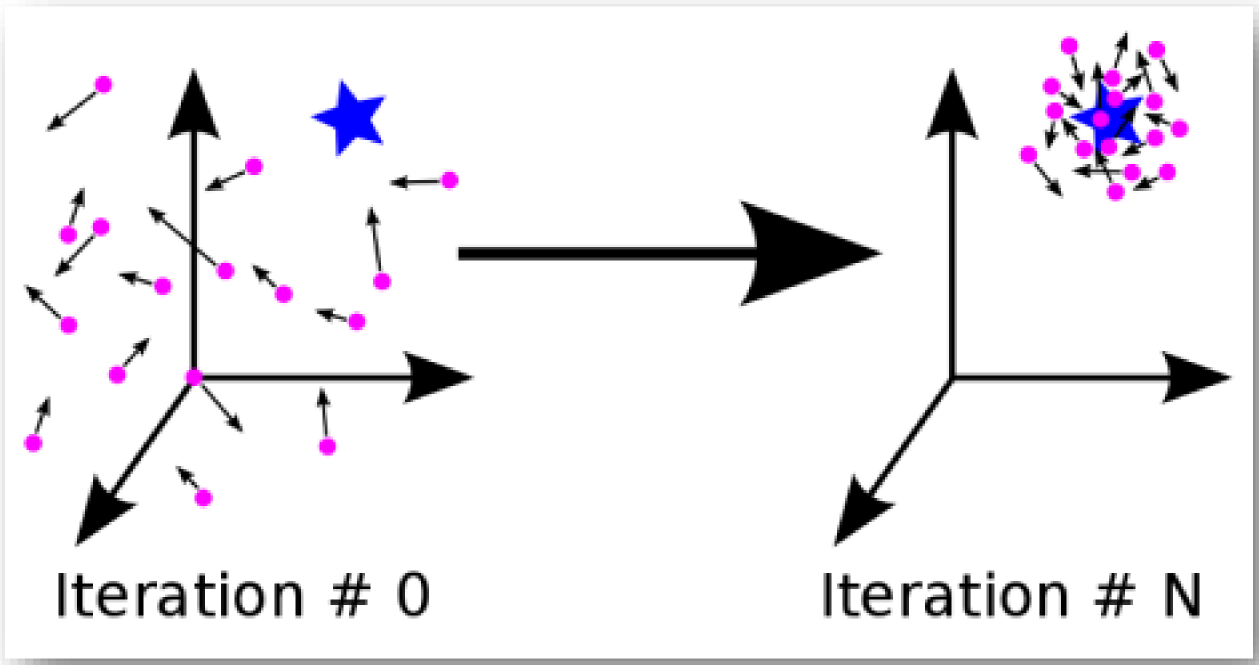
Introduction:

The Particle Swarm Optimization (PSO) Algorithm is a population-based search algorithm based on the simulation of the social behaviour of birds within a flock. The initial intent of the particle swarm concept was to graphically simulate the graceful and unpredictable choreography of a bird flock, to discover patterns that govern the ability of birds to fly synchronously, and to suddenly change direction by regrouping in an optimal formation. From this initial objective, the concept evolved into a simple and efficient optimization algorithm.

It is based on the idea of multiple schemes capable of solving complex mathematical problems that exist in engineering. It is very important to note that dealing with PSO has some advantages compared to other efficiency algorithms, if it has a few adjustment parameters, and those that should be set are discussed in detail in the literature.

Increasing lead or reducing losses has always been a problem in engineering problems. In various fields of knowledge, the difficulty of the problem of efficiency increases as science and technology advance. Often, examples of engineering problems that may need to be addressed are in the transformation and distribution of energy, in the construction of machinery, in the handling of equipment, and in reloading nuclear reactors.

In computer science, particle swarm optimization PSO is a computer-based solution that solves a problem by trying to improve the selection solution with a certain level of quality. It solves the problem by having a large number of student solutions, here called particles, and moving these particles into the search space according to a simple mathematical formula above the particle size and speed. The movement of each particle is influenced by a well-known local position, but is also directed to the most well-known positions in the search field, which is updated as the best positions obtained by other particles. This is expected to move the crowd to more advanced solutions.



Overview:

PSO is a swarm intelligence algorithm that is inspired by the behaviour of a group of animals, for example herds of birds or fish schools. Similar to the genetic algorithms, it is a human-based approach, that is, it represents the nature of the algorithm by humans, which is subtly modified until the termination factor is satisfied.
In Particle Swarm Optimization (PSO) Algorithms, the number of particles
P = {p1, …, pn}
of possible solutions is often referred to as mass. Possible solutions p1, …, pn are called particles. The PSO approach considers the set of possible solutions as a “space” in which the particles “move”. To solve the obvious problems, the number of particles is usually selected depending upon the size of the problem.

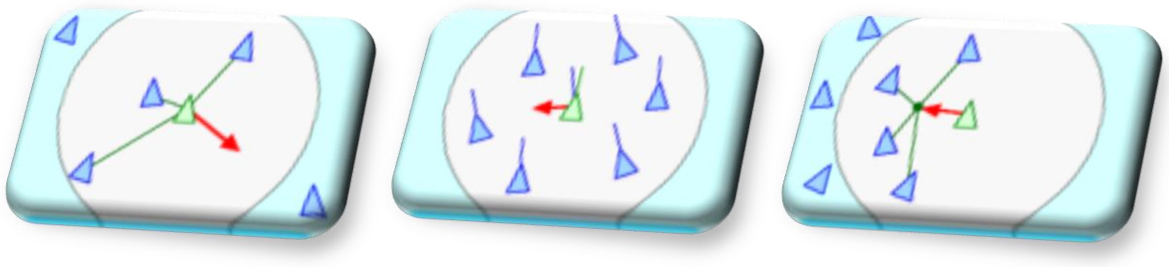
It is encouraged by the social behaviour of bird migration and school fish and their way to search for food. Suppose a group of birds search for food in an area Only a piece of food is present Birds have no knowledge of the availability of food but they know how far food is from the area.

Birds usually follow three movements while searching for food, separation, alignment and cohesion to avoid crowding of local flock mates, to move towards the average heading of the flock mates and to move towards the average position of the local flock mates respectively.

Birds Swarming: **Fish Schooling:**

In computer science, Particle Swarm Optimization (PSO) is a computer program that solves a problem by trying to improve the selection solution with a certain level of quality. It solves the problem by having a large number of probable solutions, here called particles, and moving these particles into the search space according to a simple mathematical formula known as the fitness function.

Separation **Alignment** **Cohesion**



The movement of particles is influenced by a well-known local position, but is also directed to the most well-known positions in the search field, which are updated as the best positions obtained by other particles.

This is expected to move the crowd to more advanced solutions. The PSO was originally said to have been created by Kennedy, Eberhart and Shi and was originally intended to mimic social behaviour, such as an image created in the wildlife movement in a flock or in a school of fish.

The algorithm was simplified and proven to work well. Kennedy and Eberhart's book describe many aspects of the PSO's philosophy and the multiplicity of spies. An in-depth study of PSO applications by Poli, Recently, a comprehensive review of the PSO's ideas and evaluation activities was published by Bonyadi and Michalewicz.

The PSO is metaheuristic as it thinks little or no about the problem at hand and can search for enormous gaps in student solutions. Also, the PSO does not use the gradient of the correction problem, which means that the PSO does not require the performance problem to be categorized as required by older operating methods such as gradient reduction and quasi-newton methods.

Algorithm:

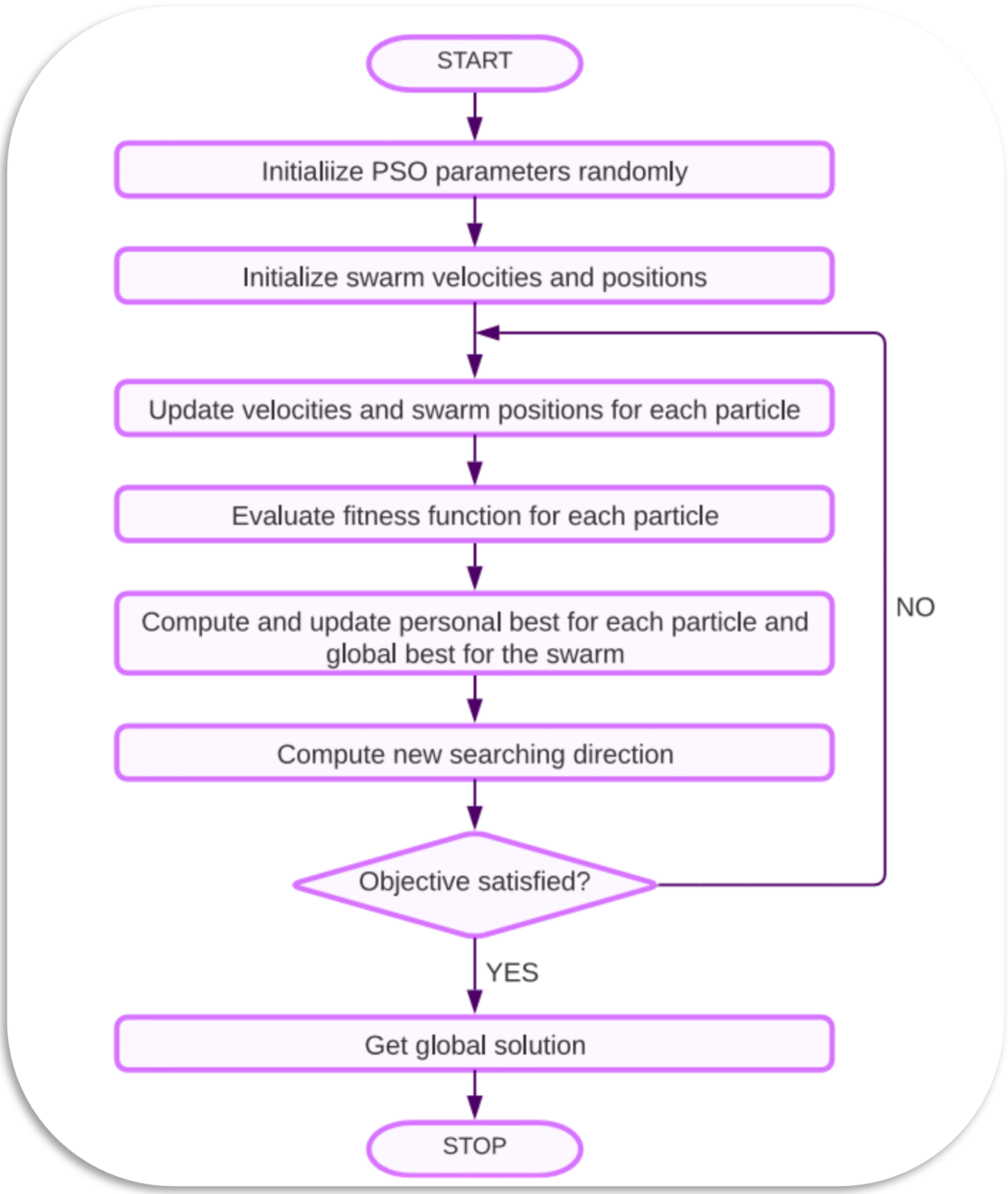
The basic algorithm works by considering a population of random solutions called particles. These particles move around the space following some simple formulae. The movement of the particles depends on its best optimum position so far and the best optimum position of the swarm so far. As the particles move in the search for a space they find better optimal positions and iteratively the personal best position of the particle and global best position of the swarm is updated. The process keeps on repeating until a satisfactory solution is achieved which is not guaranteed by the algorithm.

Let's consider a multi-variate cost function F which needs be minimized, then the goal of the algorithm is to find a vector a such that $F(a) \leq F(b)$ for all vectors b lying in the search space.

Let S be the number of particles in the swarm, each having a position $x_i \in R^n$ in the search-space and a velocity $v_i \in R^n$. Let p_i be the best known position of particle i and let g be the best known position of the entire swarm. A basic PSO algorithm is can be given as:

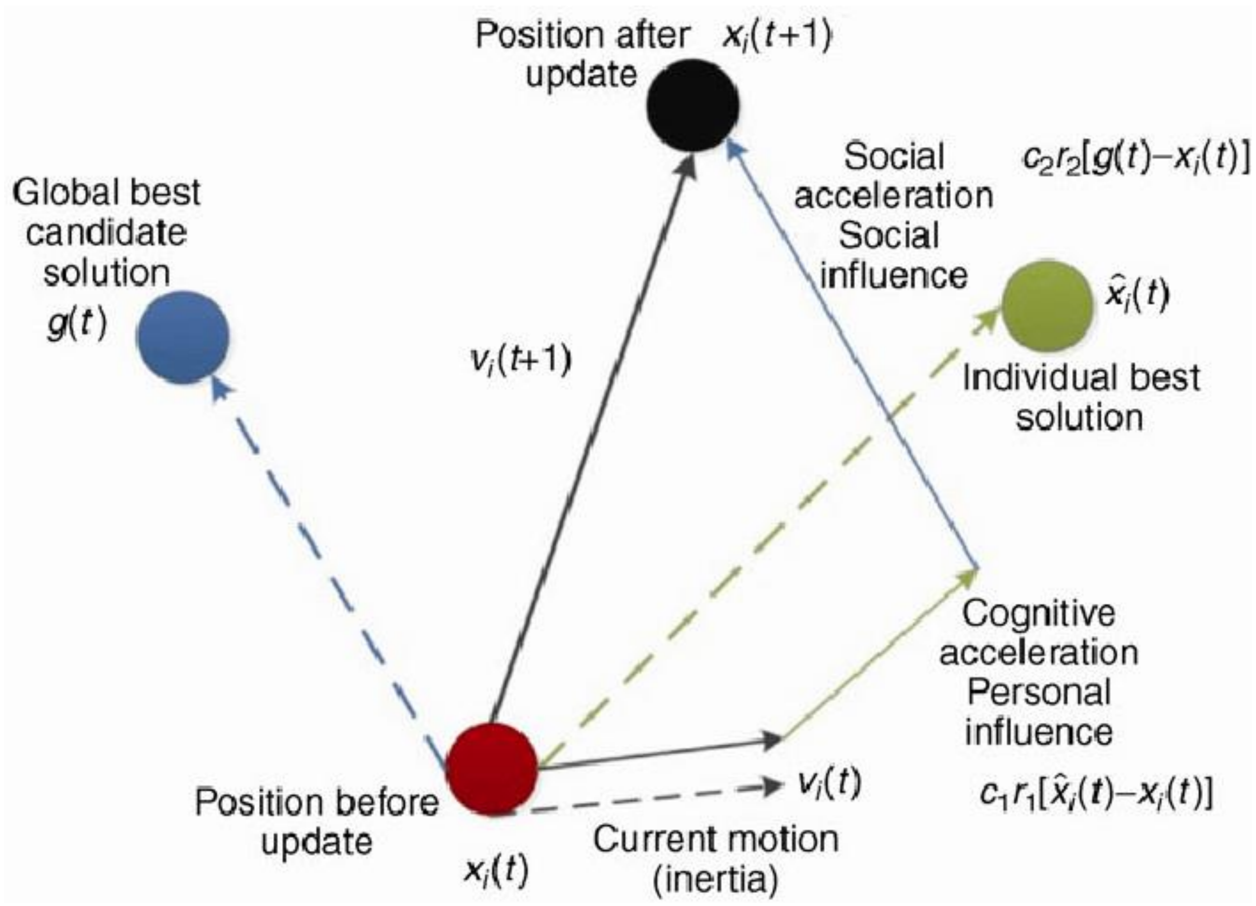
Algorithm:

```
for each particle i = 1, ..., S do
    Initialize the particle's position with a uniformly distributed random vector: xi ~ U(blo, bup)
    Initialize the particle's best known position to its initial position: pi ← xi
    if f(pi) < f(g) then
        update the swarm's best known position: g ← pi
    Initialize the particle's velocity: vi ~ U(-|bup-blo|, |bup-blo|)
while a termination criterion is not met do:
    for each particle i = 1, ..., S do
        for each dimension d = 1, ..., n do
            Pick random numbers: rp, rg ~ U(0,1)
            Update the particle's velocity: vi,d ← w vi,d + ϕp rp (pi,d-xi,d) + ϕg rg (gd-xi,d)
        Update the particle's position: xi ← xi + vi
        if f(xi) < f(pi) then
            Update the particle's best known position: pi ← xi
            if f(pi) < f(g) then
                Update the swarm's best known position: g ← pi
```


START
Initialiize PSO parameters randomly
Initialize swarm velocities and positions
Update velocities and swarm positions for each particle
Evaluate fitness function for each particle
Compute and update personal best for each particle and global best for the swarm
Compute new searching direction
Objective satisfied?
NO
YES
Get global solution
STOP

Flowchart of the Algorithm

The termination criteria can be the number of iterations performed or the optimum solution is achieved. The hyperparameters w, c1 and c2 needs to be tuned to maintain the exploration exploitation trade-off.



The next velocity of each particle is calculated by random scaling and summing up vectors in the global best direction, local best direction and current direction of motion of the particle. The next position vector is updated by adding the next velocity vector to the previous position vector.

$$P_i^{t+1} = P_i^t + V_i^{t+1}$$

$$V_i^{t+1} = \underbrace{wV_i^t}_{\text{Inertia}} + \underbrace{c_1r_1(P_{best(i)}^t - P_i^t)}_{\text{Cognitive (Personal)}} + \underbrace{c_2r_2(P_{bestglobal}^t - P_i^t)}_{\text{Social (Global)}}$$

c1, c2 and w are constants. r1 and r2 are random number in range 0 and 1. P is the position and V is the velocity.

Python program for the Algorithm to find the Minimum of Rastrigin Function:

```
import random
import math
import numpy as np
import matplotlib.pyplot as plt
import sys

num_particles = 50
iterations = 50

bounds = [(-4,4),(-4,4)]
initial = 8 * np.random.rand(num_particles, 2) - 4
error = []

colors = np.array([[1, 1, 1]] * num_particles)


class Particle:
  def __init__(self,initial):
    self.pos=initial.tolist()
    self.vel=[]
    self.best_pos=[]
    self.best_error=-1
    self.error=-1
    for i in range(0,num_dimensions):
      self.vel.append(random.uniform(-1,1))

  def update_velocity(self,global_best_position):
    w = 0.5
    c1 = 1
    c2 = 2

    for i in range(0,num_dimensions):
      r1=random.random()
      r2=random.random()

      cog_vel=c1*r1*(self.best_pos[i]-self.pos[i])
      social_vel=c2*r2*(global_best_position[i]-self.pos[i])
      self.vel[i]=w*self.vel[i]+cog_vel+social_vel
  def update_position(self,bounds):
    for i in range(0,num_dimensions):
      self.pos[i]=self.pos[i]+self.vel[i]


      if self.pos[i]>bounds[i][1]:
        self.pos[i]=bounds[i][1]


      if self.pos[i] < bounds[i][0]:
        self.pos[i]=bounds[i][0]


  def evaluate_fitness(self,fitness_function):
    self.error=fitness_function(self.pos)
    print("ERROR------->",self.error)

    if self.error < self.best_error or self.best_error==-1:
      self.best_pos=self.pos
      self.best_error=self.error


def fitness_function(x):
  A = 3
  total=0
  total+= A*len(x) + sum([(i**2 -
 A * np.cos(2 * math.pi * i)) for i in x])
  return total


import time

class Interactive_PSO():
  def __init__(self,fitness_function,initial,bounds,num_particles):
    global num_dimensions

    num_dimensions = 2
    global_best_error=-1
    global_best_position=[]
    self.gamma = 0.0001
    swarm=[]
    for i in range(0,num_particles):
      swarm.append(Particle(initial[i]))

    i=0
    for i in range(iterations):
      xlist = np.linspace(-5.0, 5.0, 100)
      ylist = np.linspace(-5.0, 5.0, 100)
      X, Y = np.meshgrid(xlist, ylist)

      A = 3
      Z = A*2 + (X**2 - A * np.cos(2 * np.pi * X)) + (Y**2 -
 A * np.cos(2 * np.pi * Y))
      cp = plt.contourf(X, Y, Z)
      for j in range(0,num_particles):
        swarm[j].evaluate_fitness(fitness_function)
        print('global_best_position',swarm[j].error,global_best_error)


        if swarm[j].error < global_best_error or global_best_error == -1:
          global_best_position=list(swarm[j].pos)
          global_best_error=float(swarm[j].error)
          error.append(global_best_error)
          plt.title("Particle Swarm Optimization, Particles:{}, Error:{}".
format(num_particles,round(global_best_error,1)))

        if i%2==0:
          global_best_error=-1
          global_best_position = list([swarm[j].pos[0]+self.gamma*(swarm[j
].error)*random.random() ,swarm[j].pos[1]+self.gamma*(swarm[j].error)*rand
om.random() ])


      pos_0 = {}
      pos_1 = {}
      for j in range(0,num_particles):
        pos_0[j] = []
        pos_1[j] = []

      for j in range(0,num_particles):
        swarm[j].update_velocity(global_best_position)
        swarm[j].update_position(bounds)


        pos_0[j].append(swarm[j].pos[0])
        pos_1[j].append(swarm[j].pos[1])
        plt.xlim([-5, 5])
        plt.ylim([-5, 5])
      for j in range(0,num_particles):
        plt.plot(pos_0[j], pos_1[j],  color = colors[j],marker = '^'  )


      x,y = 0, 0
      plt.plot(float(x), float(y),  color = 'orange',marker = 'x'  )
      plt.pause(0.01)
      if i != iterations - 1:
        plt.clf()
      else:
        plt.show()
        plt.plot([i for i in range(len(error))], error)
        plt.show()
    print ('Results')
    print ('Best Position:',global_best_position)
    print( 'Best Error:',global_best_error)


Interactive_PSO(fitness_function,initial,bounds,num_particles=num_particle
s)
if __name__ == "__Interactive_PSO__":
    main()
```

Time Complexity: **O(S*n)**   where, S is number of particles
                                                    n is number of dimensions

Classification:

P and NP are classes of decision problems.

Optimization problems aren't decision problems, so they're not in any of those classes by definition.

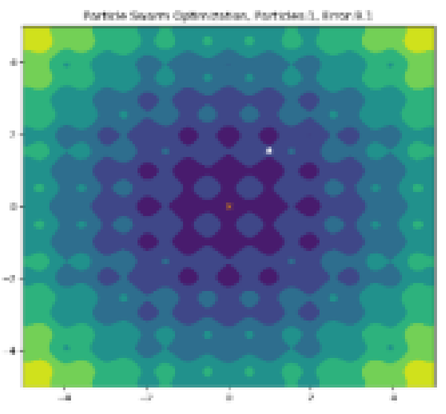
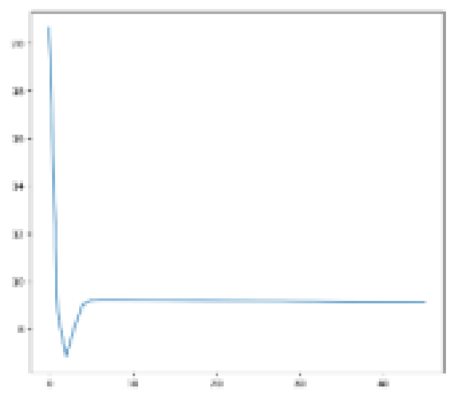
Simulation and Result:

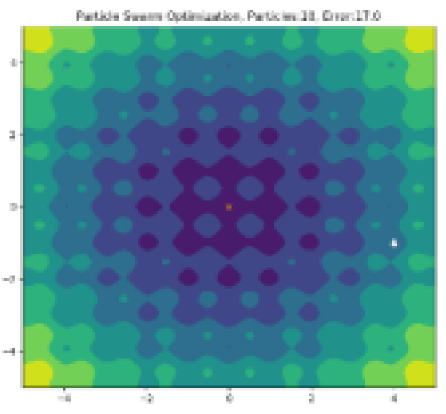
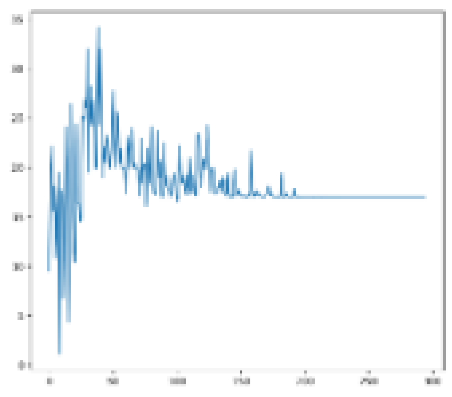
Simulation was conducted to find the minimum of the Rastrigin Function as it is used as a benchmark for testing optimization algorithms. The number of particles was taken to be 1, 10, 20 and 50 and the algorithm was run for 50 iterations each time.

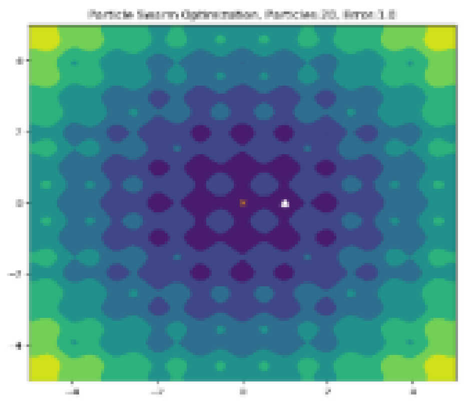
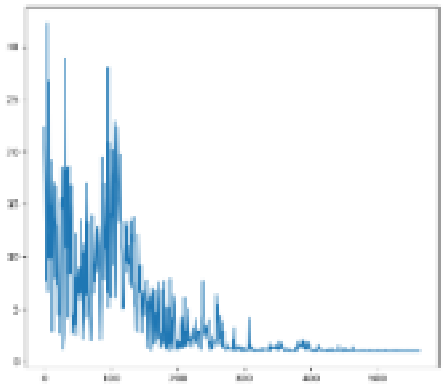
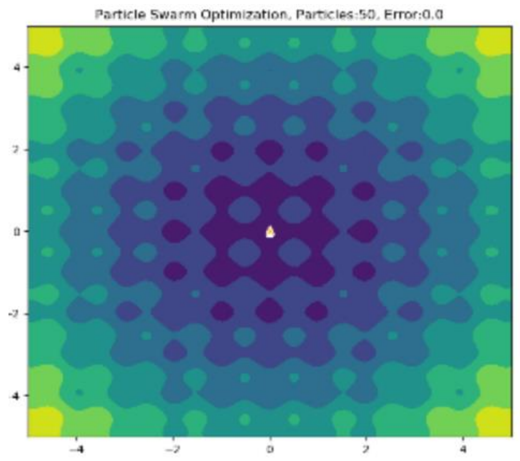
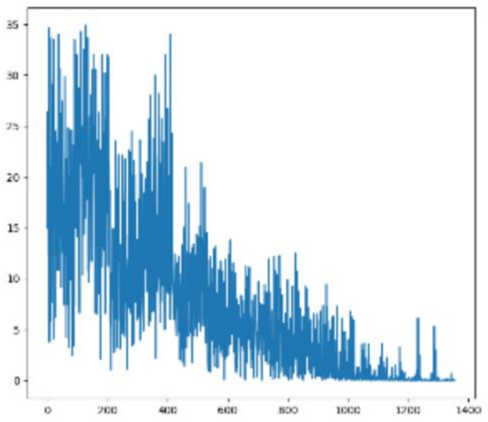
Rastrigin Function:

$$F(x, y) = 2A + \left(x^2 - A\cos(2\pi x)\right) + \left(y^2 - A\cos(2\pi y)\right)$$

$$A = 3$$

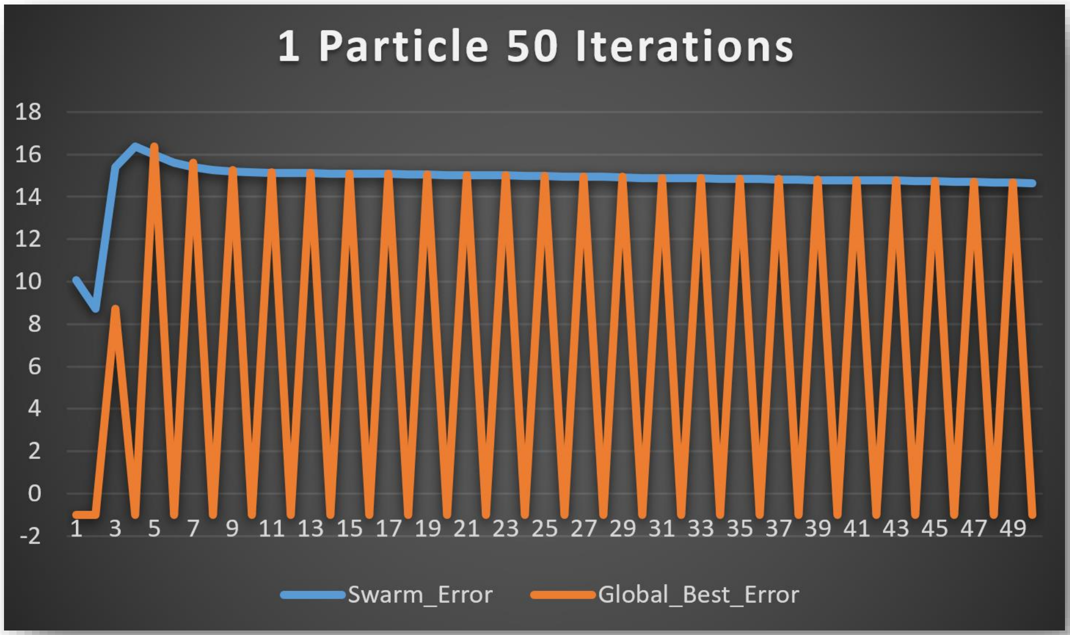
Contour plots and error plots of test runs to find the minimum of Rastrigin Function.	
	
A single particle isn't able to find a minimum.	

	
10 particles find a local minimum.	

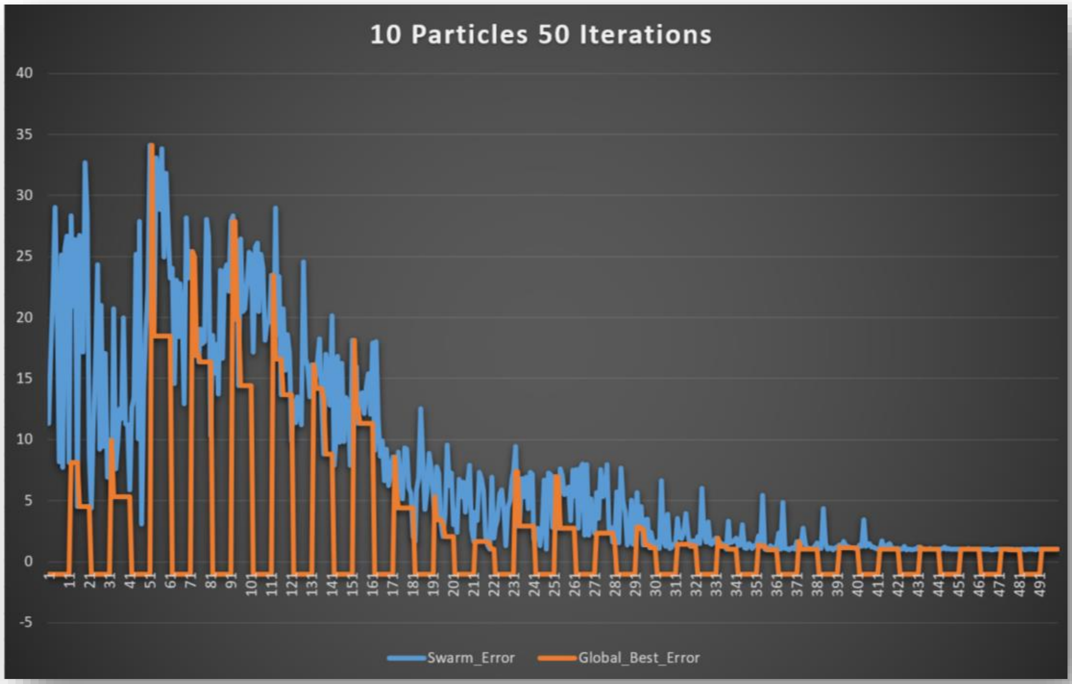
	
20 particles find a local minimum.	
	
50 particles find the global minimum.	

Extracted Dataset Exploration using Excel:

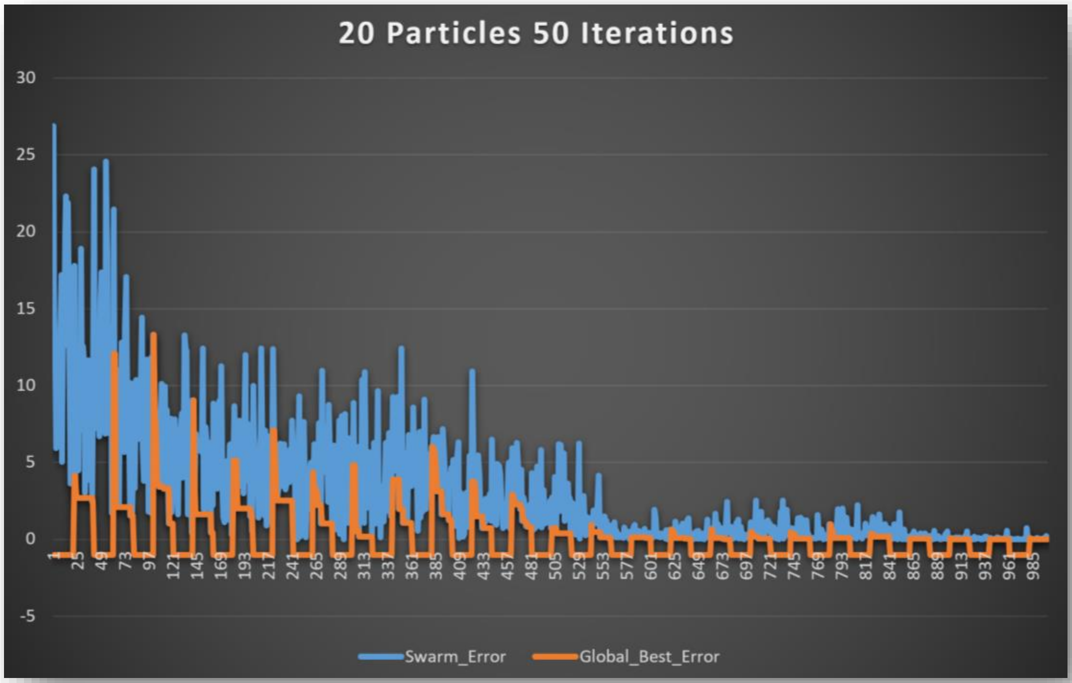
1. 1 Particle 50 Iterations (50 Points)



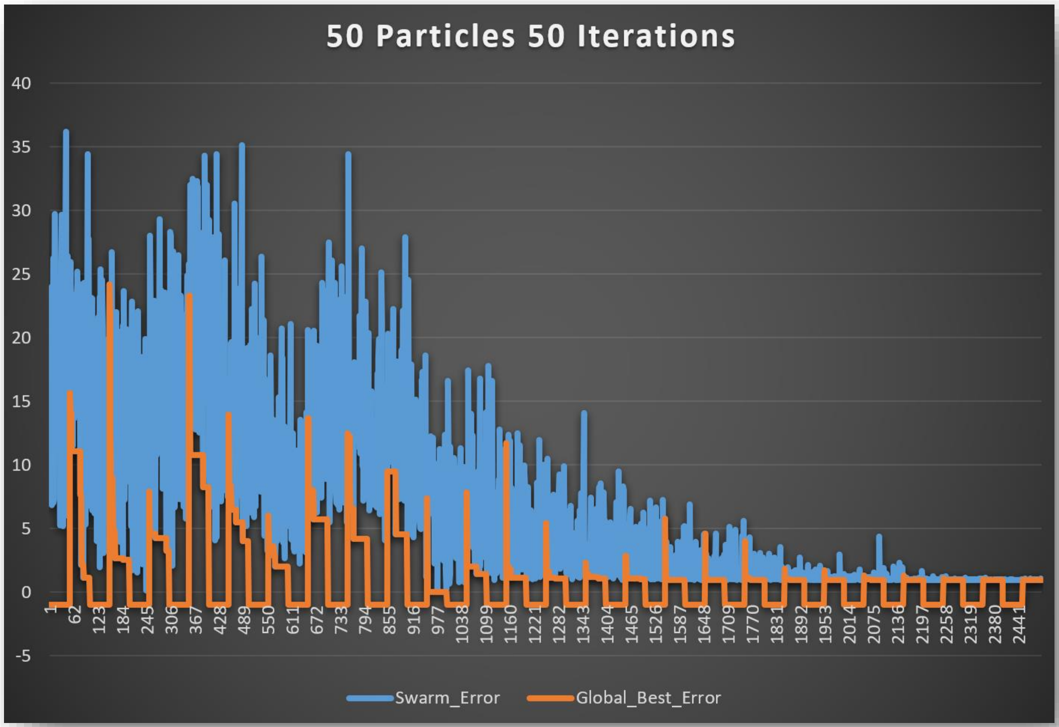
2. 10 Particles 50 Iterations (500 Points)



3. 20 Particles 50 Iterations (1000 Points)



4. 50 Particles 50 Iterations (2500 Points)

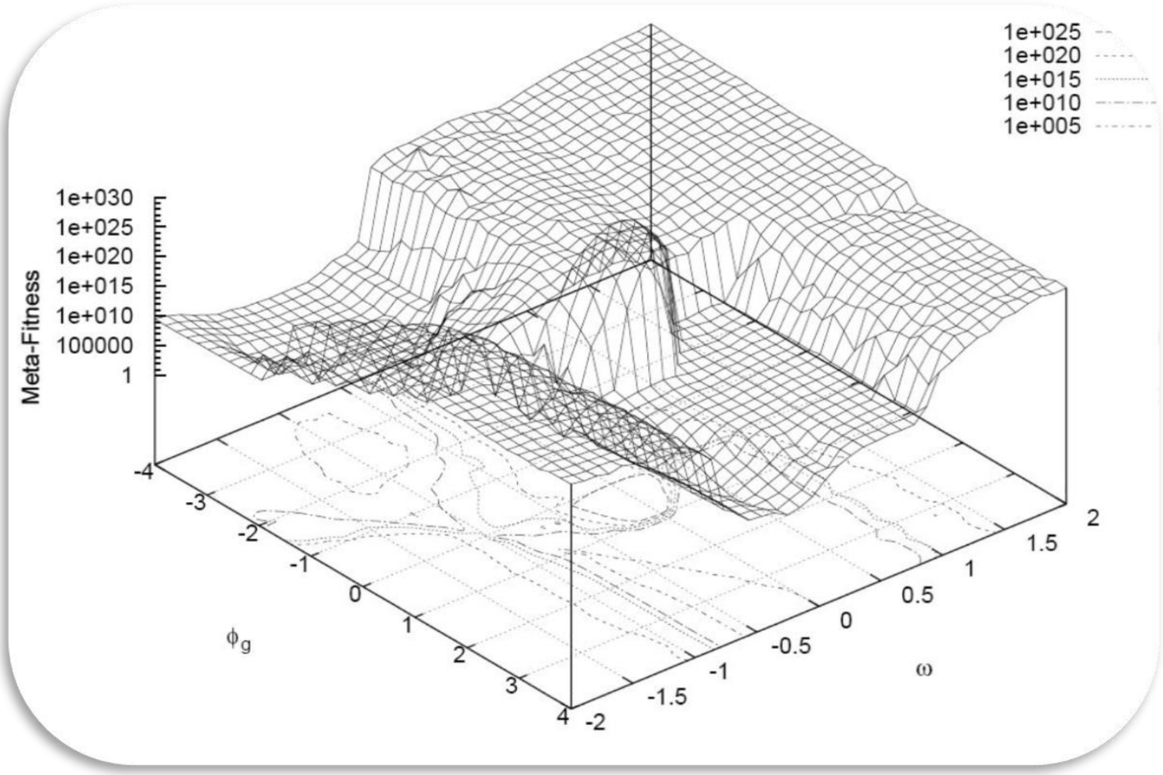


Conclusion:

We have successfully demonstrated the Particle Swarm Optimization algorithm to find the minimum of Rastrigin Function. We have demonstrated how the change in number of particles results in change of the output of the algorithm. Our program can be applied to other differentiable or non-differentiable mathematical functions as well by just modifying the fitness function.
The performance of PSO can be improved by varying the hyperparameters. It is close to evolutionary algorithm so there are many hybrid versions of the algorithm also available. Some popular ones are Multi-Objective PSO, Hybrid of Genetic algorithm and PSO, Adaptive PSO and Discrete PSO.

References:

- Introduction to Particle Swarm Optimization(PSO) - GeeksforGeeks
- A Gentle Introduction to Particle Swarm Optimization (machinelearningmastery.com)
- Particle Swarm Optimization (PSO) Visually Explained | by ★ Axel Thevenot | Towards Data Science
- https://kpfu.ru/staff_files/F_1407356997/overview.pdf


1e+025
1e+020
1e+015
1e+010
1e+005
Meta-Fitness
1e+030
1e+025
1e+020
1e+015
1e+010
ϕ_g
ω