# SCOA Assignment 3

Roll No.: 41163

**Title:**

Implement Particle swarm optimization for benchmark function (eg.Square, Rosenbrock function). Initialize the population from the Standard Normal Distribution. Evaluate fitness of all particles.

Use :

1. c1=c2 = 2
2. Inertia weight is linearly varied between 0.9 to 0.4.
3. Global best variation

**Objectives:**

Student will learn:

1. Understand the particle swarm optimization.
2. Implementation of travelling Sqaure function with particle swarm optimization.

**Software Requirements:**

- Jupyter Notebook
- Python 3.8.5
- 64 bit open source OS

**Hardware Requirements:**

- Machine with 64 bit processor

**Theory:**

**Particle swarm optimization**

Particle swarm optimization (PSO) is a population based stochastic optimization technique developed by Dr. Eberhart and Dr. Kennedy in 1995, inspired by social behavior of bird flocking or fish schooling.

PSO shares many similarities with evolutionary computation techniques such as Genetic Algorithms (GA). The system is initialized with a population of random solutions and searches for optima by updating generations. However, unlike GA, PSO has no evolution operators such as crossover and mutation. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles. The detailed information will be given in following sections.

Compared to GA, the advantages of PSO are that PSO is easy to implement and there are few parameters to adjust. PSO has been successfully applied in many areas: function optimization, artificial neural network training, fuzzy system control, and other areas where GA can be applied.

PSO simulates the behaviors of bird flocking. Suppose the following scenario: a group of birds are randomly searching food in an area. There is only one piece of food in the area being searched. All the birds do not know where the food is. But they know how far the food is in each iteration. So what's the best strategy to find the food? The effective one is to follow the bird which is nearest to the food.

PSO learned from the scenario and used it to solve the optimization problems. In PSO, each single solution is a "bird" in the search space. We call it "particle". All of particles have fitness values which are evaluated by the fitness function to be optimized, and have velocities which direct the flying of the particles. The particles fly through the problem space by following the current optimum particles.

PSO is initialized with a group of random particles (solutions) and then searches for optima by updating generations. In every iteration, each particle is updated by following two "best" values. The first one is the best solution (fitness) it has achieved so far. (The fitness value is also stored.) This value is called pbest. Another "best" value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the population. This best value is a global best and called gbest. When a particle takes part of the population as its topological neighbors, the best value is a local best and is called lbest.

**Rosenbrock funtion**
In mathematical optimization, the Rosenbrock function is a non-convex function, introduced by Howard H. Rosenbrock in 1960, which is used as a performance test problem for optimization algorithms. It is also known as Rosenbrock's valley or Rosenbrock's banana function.

The global minimum is inside a long, narrow, parabolic shaped flat valley. To find the valley is trivial. To converge to the global minimum, however, is difficult.

The function is defined by:

f(x,y)=(a-x)^2+b(y-x^2)^2

**Mathematical model** :
1. Each particle in particle swarm optimization has an associated position, velocity, fitness value.
2. Each particle keeps track of the particle_bestFitness_value particle_bestFitness_position.
3. A record of global_bestFitness_position and global_bestFitness_value is maintained.

**Algorithm:**
Parameters of problem:
- Number of dimensions (d)
- Lower bound (minx)
- Upper bound (maxx)

Hyperparameters of the algorithm:
- Number of particles (N)
- Maximum number of iterations (max_iter)
- Inertia (w)
- Cognition of particle (C1)
- Social influence of swarm (C2)

Steps
```
1. Randomly initialize Swarm population of N particles Xi ( i=1, 2, …, n)
2. Select hyperparameter values w, c1 and c2
3. For Iter in range(max_iter):  # loop max_iter times
    3.1.    For i in range(N):  # for each particle:
        3.1.1.    Compute new velocity of ith particle
            3.1.1.1.    swarm[i].velocity = w*swarm[i].velocity + r1*c1*(swarm[i].bestPos - swarm[i].position) + r2*c2*( best_pos_swarm – swarm[i].positin)
        3.1.2.    If velocity is not in range [minx, max] then clip it
```

```
            3.1.2.1.   if swarm[i].velocity < minx:
            3.1.2.2.      swarm[i].velocity = minx
            3.1.2.3.   elif swarm[i].velocity[k] > maxx:
            3.1.2.4.      swarm[i].velocity[k] = maxx
        3.1.3.     Compute new position of ith particle using its new velocity
            3.1.3.1.   swarm[i].position += swarm[i].velocity
        3.1.4.     Update new best of this particle and new best of Swarm
            3.1.4.1.   if swarm[i].fitness < swarm[i].bestFitness:
            3.1.4.2.      swarm[i].bestFitness = swarm[i].fitness
            3.1.4.3.      swarm[i].bestPos = swarm[i].position
            3.1.4.4.   if swarm[i].fitness < best_fitness_swarm
            3.1.4.5.      best_fitness_swarm = swarm[i].fitness
            3.1.4.6.      best_pos_swarm = swarm[i].position
4. Return best particle of Swarm
```

**Test Cases:**

| **Description** | **Expected O/P** | **Actual O/P** |
|---|---|---|
| PSO on Square function | Best position and solution e.g. Best Position: [-0.0017169821808446575, -0.031208182147021327] Best Solution: 0.0009768986607309987 | Successful |
| PSO on rosenbrock function | Global best particle and corresponding solution e.g. Best Position: [2.4707917491912976, 6.048252987696178] Best Solution: -5.056828263971 | Successful |

**Conclusion:**

Compared with genetic algorithms (GAs), the information sharing mechanism in PSO is significantly different. However, PSO does not have genetic operators like crossover and mutation. Particles update themselves with the internal velocity. They also have memory, which is important to the algorithm.