

## Lab - 2 Particle Swarm Optimization.

### Algorithm

#### Initialize

Swarm size ( $N$ ), problem dimensionality ( $D$ ), max iterations ( $T$ ), and bounds for positions ( $LB, UB$ ).

Assign random positions and velocities for each particle.

Step 1: Randomly initialize swarm population of  $N$  particles  $X_i$  ( $i=1, 2, \dots, N$ )

Step 2: Select hyperparameter values:  
 $w, c1$  and  $c2$

Step 3: For Iter in range (max iter):  
For  $i$  in range ( $N$ ):

a. Compute new velocity of  $i$ th particle.

$$\text{swarm}[i].\text{velocity} = w * \text{swarm}[i].\text{velocity} + c1 * (\text{swarm}[i].\text{bestPos} - \text{swarm}[i].\text{position}) + c2 * c2 * (\text{bestPosSwarm} - \text{swarm}[i].\text{position})$$

b. Compute new position of  $i$ th particle using its new velocity.

$$\text{swarm}[i].\text{position} = \text{swarm}[i].\text{position} + \text{swarm}[i].\text{velocity}$$

c. If position is not in range  $[minx, maxx]$  then clip it

if  $\text{swarm}[i].\text{position} < minx$ :

$$\text{swarm}[i].\text{position} = minx$$

elif  $\text{swarm}[i].\text{position} > maxx$ :

$$\text{swarm}[i].\text{position} = maxx$$

d. Update new best of this particle and new best of swarm.

if  $\text{swarm}[i].\text{fitness} < \text{swarm}[i].\text{bestFitness}$ :  
Best Fitness!



swarm[i].bestFitness = swarm[i].fitness  
 swarm[i].bestPos = swarm[i].position

End - for

End - for

Step 4: return best particle of swarm

Program

import numpy as np

def objectiveFunction(x):  
 return np.sum(x\*\*2)

class Particle:

def \_\_init\_\_(self, dimensions, minx, maxx):

self.position = np.random.uniform(minx, maxx,  
 dimensions)

self.velocity = np.random.uniform(minx, maxx,  
 dimensions)

self.bestPosition = np.copy(self.position)

self.bestFitness = objectiveFunction(self.position)

self.fitness = self.bestFitness

def PSO(n\_particles, dimensions, max\_iter, minx, maxx,  
 w, c1, c2):

swarm = [Particle(dimensions, minx, maxx)  
 for \_ in range(n\_particles)]

bestFitnessSwarm = float('inf')

bestPosSwarm = None



for iter in range(max\_iter):

for particle in swarm:

$r_1, r_2 = \text{np.random.rand}(\text{dimensions})$ ,

$\text{np.random.rand}(\text{dimensions})$

$\text{particle.velocity} = (w * \text{particle.velocity} +$

$c1 * r1 * (\text{particle.best\_position} - \text{particle.position}) +$

$c2 * r2 * (\text{best\_pos\_swarm} -$

$\text{particle.position})$  if best\_pos\_swarm is not None else 0

)

$\text{particle.position} += \text{particle.velocity}$

$\text{particle.position} = \text{np.clip}(\text{particle.position}, \text{minx}, \text{maxx})$

$\text{particle.fitness} = \text{objective\_function}(\text{particle.position})$

if  $\text{particle.fitness} < \text{best\_fitness\_swarm}$ :

$\text{best\_fitness\_swarm} = \text{particle.fitness}$

$\text{best\_pos\_swarm} = \text{np.copy}(\text{particle.position})$

return  $\text{best\_pos\_swarm}, \text{best\_fitness\_swarm}$

$n\_particles = 30$

$\text{dimensions} = 2$

$\text{max\_iter} = 100$

$\text{minx}, \text{maxx} = -10, 10$

$w = 0.5$

$c1 = 1.5$

$c2 = 1.5$

best position, best fitness = pso (p, particles, dimensions,  
max\_iter, min\_x, max\_x, w, l1, l2)  
print ("Best position found:", best position)  
print ("Best fitness found:", best fitness)

Output  
Best fitness found: [-1.988 e-12 -8.12 e-13]  
Best fitness found: 2.33 e-24

~~def pso(p, particles, dimensions, max\_iter, min\_x, max\_x, w, l1, l2):  
 # Initialize particles  
 pos = np.zeros((particles, dimensions))  
 vel = np.zeros((particles, dimensions))  
 pbest\_pos = pos  
 pbest\_fit = np.inf  
 gbest\_fit = np.inf  
 gbest\_pos = pos  
 for i in range(max\_iter):  
 for j in range(particles):  
 # Calculate fitness  
 fit = fitness(pos[j], min\_x, max\_x, l1, l2)  
 # Update pbest  
 if fit < pbest\_fit[j]:  
 pbest\_fit[j] = fit  
 pbest\_pos[j] = pos[j].copy()  
 # Update gbest  
 if fit < gbest\_fit:  
 gbest\_fit = fit  
 gbest\_pos = pos[j].copy()  
 # Update velocity  
 r = np.random.random()  
 vel[j] = w \* vel[j] + r \* (pbest\_pos[j] - pos[j])  
 # Update position  
 pos[j] = pos[j] + vel[j]  
 # Boundary constraints  
 pos[j] = np.clip(pos[j], min\_x, max\_x)  
 return gbest\_pos, gbest\_fit~~

~~p = 10  
particles = 20  
dimensions = 2  
max\_iter = 100  
min\_x = -10  
max\_x = 10  
w = 0.9  
l1 = 0.01  
l2 = 0.01~~