# Particle Swarm Optimization

## CL603 Project - Team Ascenters

Abhi Manjunath (190020036)

Deepak Thorat (190020037)

Mihir Nandawat (190020071)

Image Source

# Table of Contents

# Introduction

- PSO algorithm works by having a population (called a swarm) of particles
- Each particle has a position and velocity ∊ R^n. Simple mathematical relations describe movement of particles in search space
- Particle’s movement is guided by its own best-known position so far (pBest), as well as the best-known position of the entire particle population (gBest)
- Process repeated in the hopes of finding a satisfactory solution, but it is not guaranteed
- Generally suitable for a class of optimization problems which are high dimensional and where high accuracy isn't required

Source - PSO TDS

# Velocity Update Formula

$$V_{i,t+1} = \omega * V_{i,t} + c_1 * rand * (pBest_{i,t} - X_{i,t}) + c_2 * rand * (gBest_t - X_{i,t})$$

- Influence of particle’s previous velocity
- ω - Inertia Weight

- Influence of distance between particle’s best known position and current position
- C1 - Cognitive Learning Factor

- Influence of distance between Swarm’s best known position so far and particle’s current location
- C2 - Social Learning Factor

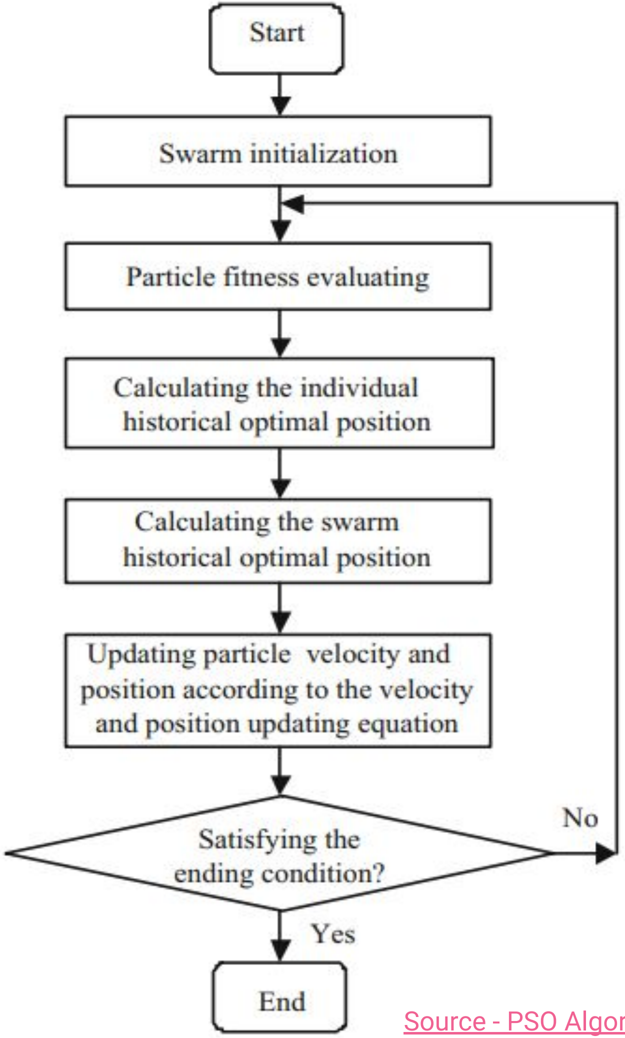
# Algorithm

$$V_{i,t+1} = \omega * V_{i,t} + c_1 * rand * (pBest_{i,t} - X_{i,t}) + c_2 * rand * (gBest_t - X_{i,t})$$

$$X_{i,t+1} = X_{i,t} + V_{i,t+1}$$

Hyperparameters

- W → inertia weight
- $C_1$ → Cognitive Learning Factor
- $C_2$ → Social Learning Factor




Source - PSO Algorithm

# Literature Survey

Variants of Velocity Update:-

- **Basic Algorithm:**

$$V_{i,t+1} = V_{i,t} + 2 * rand * (pBest_{i,t} - X_{i,t}) + 2 * rand * (gBest_t - X_{i,t})$$

- **Introduction of weight inertia (w)**

$$V_{i,t+1} = \omega * V_{i,t} + c_1 * rand * (pBest_{i,t} - X_{i,t}) + c_2 * rand * (gBest_t - X_{i,t})$$

This improved the performance of the algorithm by controlling the extent of exploration and exploitation of particles

- **Introduction of constriction factor ($\chi$)**

$$V_{i,t+1} = \chi * (\omega * V_{i,t} + c_1 * rand * (pBest_{i,t} - X_{i,t}) + c_2 * rand * (gBest_t - X_{i,t}))$$

Constriction factor ensures convergence and increases convergence rate.

- **PSO Algorithm Local Version:-**

$$V_{i,t+1} = \omega * V_{i,t} + c_1 * rand * (pBest_{i,t} - X_{i,t}) + c_2 * rand * (pBest_{i,n,t} - X_{i,t})$$

Instead of tracking the Swarm’s optimal position (GBest), track the optimal location found amongst n particles located in topological neighborhood of the particle

# Engineering Applications

- Adaptive PSO algorithms are used to track the changes in the dynamic system automatically
- Environment-based PSO approach is used to optimise an aggregate production plan model
- In wireless sensor networks, PSO is used to optimise performance in terms of the number of alive nodes.
- PSO is used in swarm robotics

# Our Approach

1. **Convex Optimisation**
   Applied PSO on simple unconstrained convex objective function

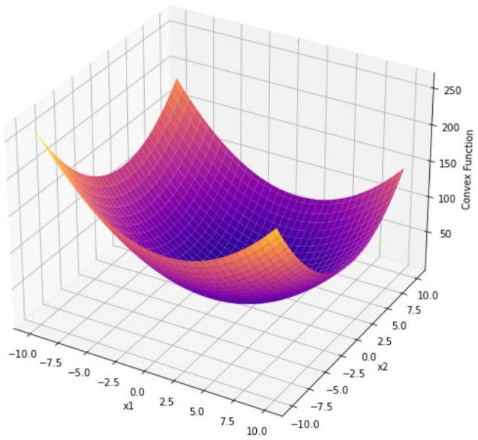
2. **Non Convex Optimisation**
   Applied PSO on a unconstrained non convex objective function

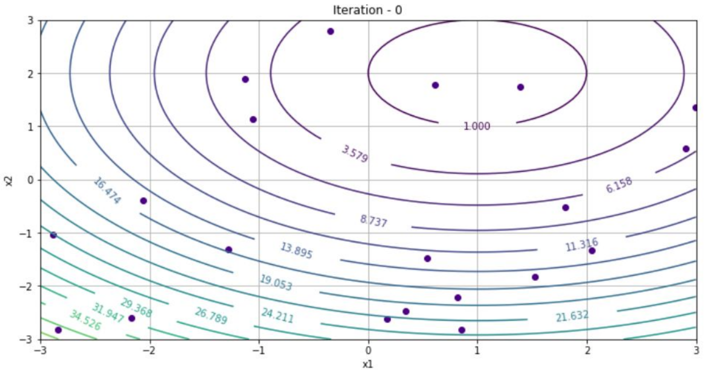
3. **Constrained Optimisation**
   Converted constrained optimisation problem into an unconstrained problem via penalty function to apply PSO

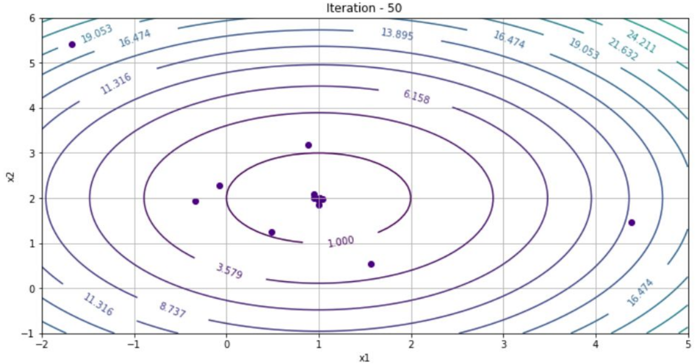
4. **Effects of Hyperparameter**
   Studied the effects of w, c1, c2, X (contraction) on PSO

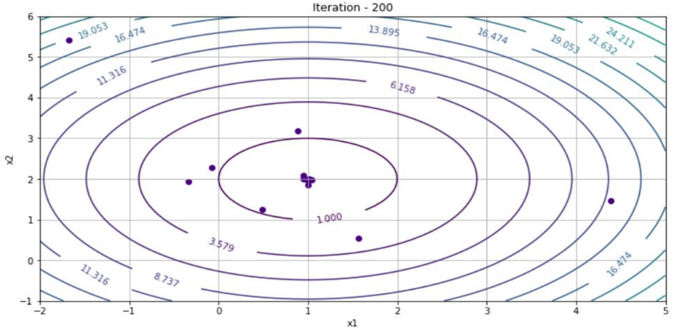
# Convex Optimisation

$$f(x_1, x_2) = (x_1 - 1)^2 + (x_2 - 2)^2$$




No of Particles - 20
W initial = 0.9 , W := W*0.98
c1, c2 = 2, 2
Iterations - 200

```
g_best
```

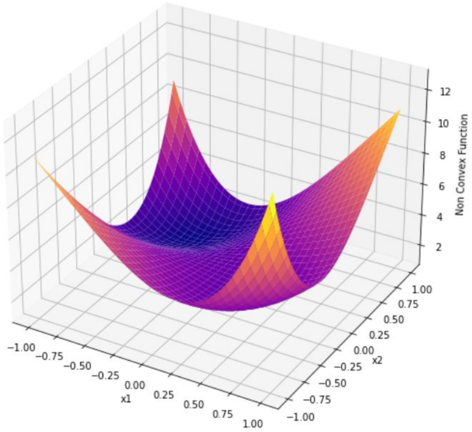
```
array([1.00794822, 1.99879916])
```

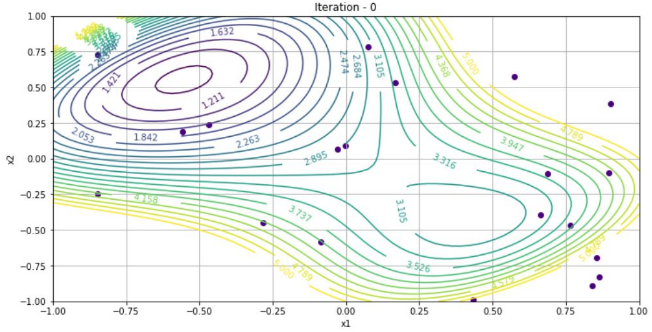
```
function(g_best)
```

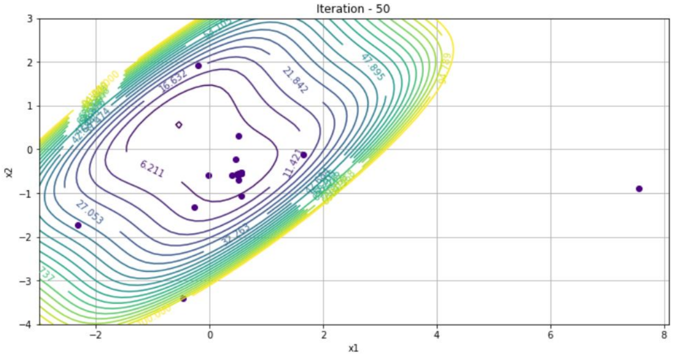
```
6.461616763419825e-05
```

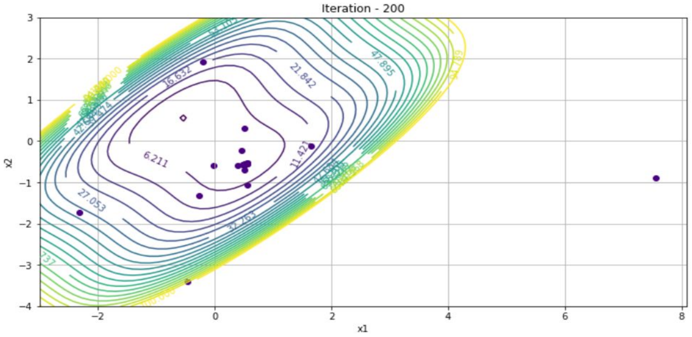
# Non Convex Optimisation

$$f(x_1, x_2) = (x_2 - x_1)^4 + 8x_1x_2 - x_1 + x_2 + 3$$




No of Particles - 20
W initial = 0.9 , W := W*0.98
c1, c2 = 2, 2
Iterations - 200

```
g_best
```

```
array([ 0.55357859, -0.55375177])
```

```
function(g_best)
```

```
0.9438273304158256
```

# Constrained Optimisation (1)

$$F(x) = f(x) + h(k)\,H(x)$$

$$H(x) = \sum_{i=1}^{m} \theta(q_i(x))\, q_i(x)^{\gamma(q_i(x))}$$

$$q_i(x) \;=\; \max\{0, g_i(x)\},\; i \;=\; 1, \ldots, m$$

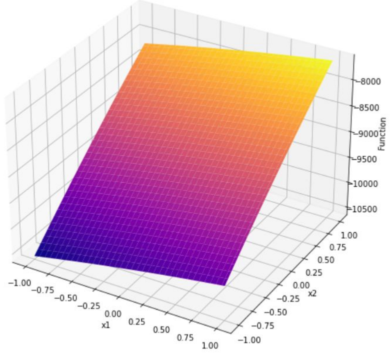
$\gamma$ - Power of Penalty Function
if $q_i(x) < 1$, then $\gamma(q_i(x)) = 1$,
otherwise $\gamma(q_i(x)) = 2$

$\theta$ - Multistage Assignment Function
if $q_i(x) < 0.001$ then $\theta(q_i(x)) = 10$,
else if $q_i(x) \leq 0.1$ then $\theta(q_i(x)) = 20$,
else if $q_i(x) \leq 1$ then $\theta(q_i(x)) = 100$,
else $\theta(q_i(x)) = 300$

h(k) - Dynamically Modified Penalty Value
h(k) = k* $\sqrt{(k)}$

# Constrained Optimisation (2)

$$f(x) = (x_1 - 10)^3 + (x_2 - 20)^3$$




$$100 - (x_1 - 5)^2 - (x_2 - 5)^2 \leqslant 0$$

$$(x_1 - 6)^2 + (x_2 - 5)^2 - 82.81 \leqslant 0$$

$$13 \leq x_1 \leq 100 \qquad 0 \leqslant x_2 \leqslant 100$$

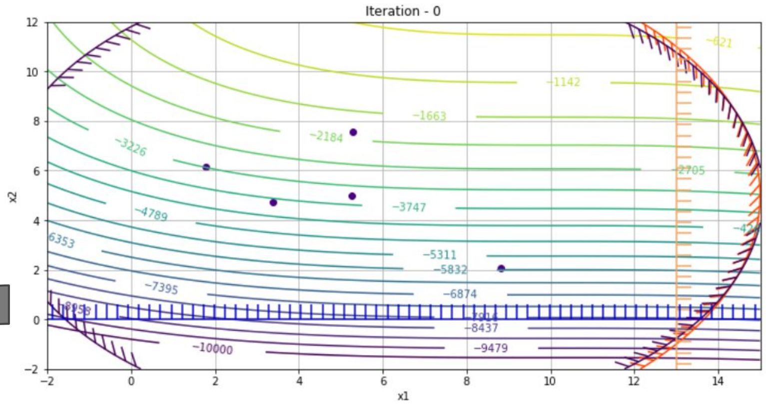
No of Particles - 5
W initial = 1.2 , W := W*0.98
c1, c2 = 2, 2
Iterations - 200

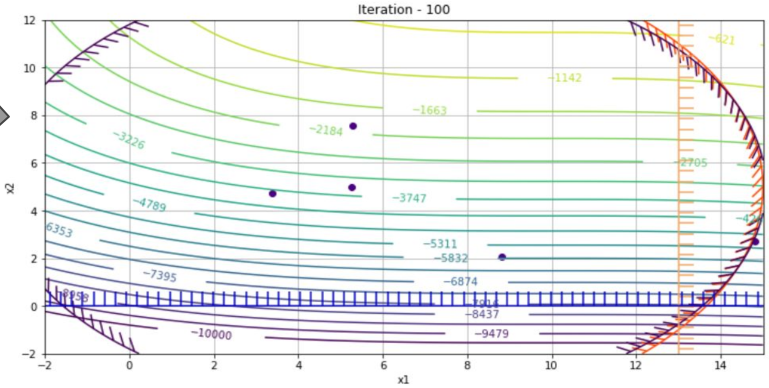
```
g_best
```

```
array([13.59561606, -0.10930529])
```

```
print(constrain_function(g_best,iteration))
```

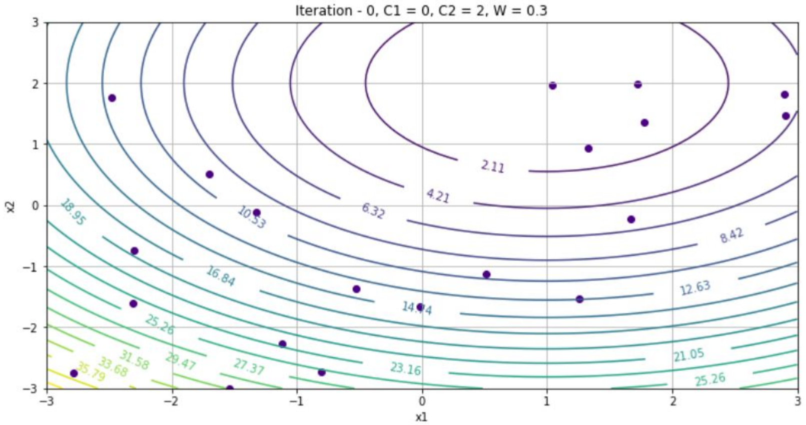
```
-9709.465995599347
```

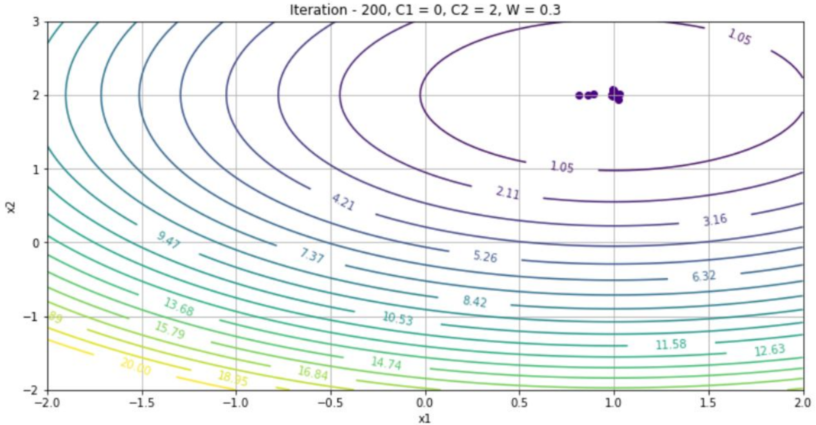


# Effects of Hyperparameters (1)

## $C_1$ (Cognitive Learning Factor) - Controls the effect of Particle's best known location on Particle Velocity

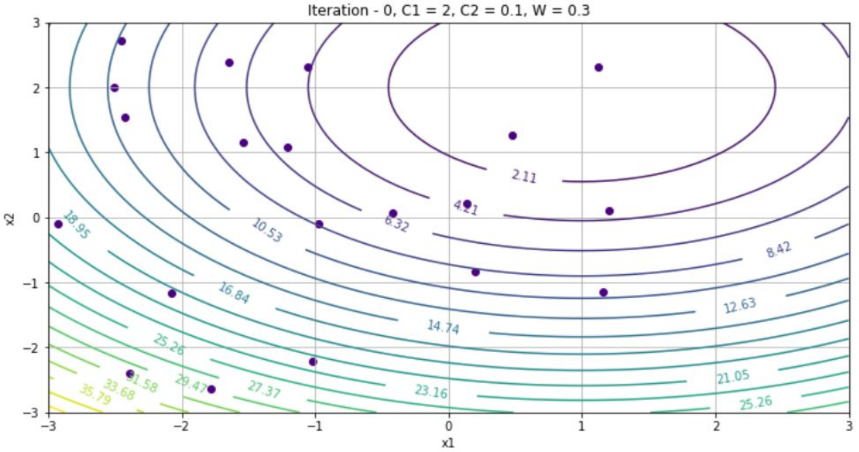


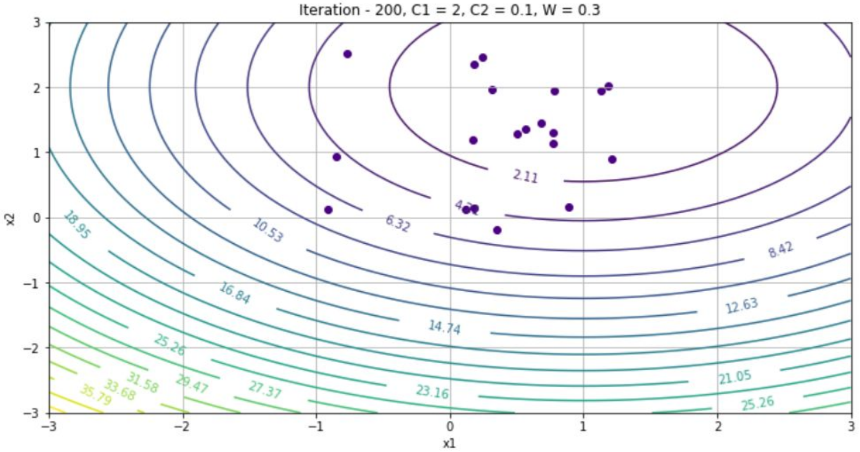




Particles converge at the cost of exploration

# Effects of Hyperparameters (2)

$C_2$ (Social Learning Factor) - Controls the effect of Group's best known location on Particle Velocity

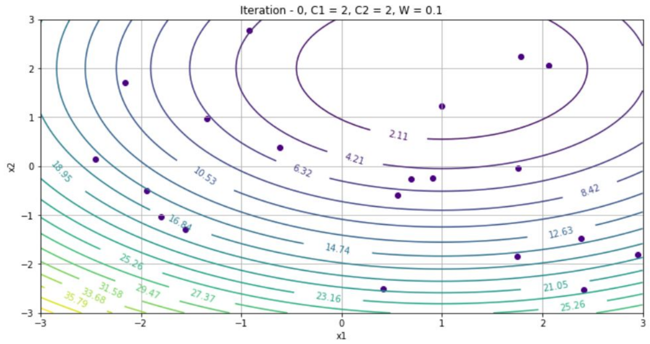


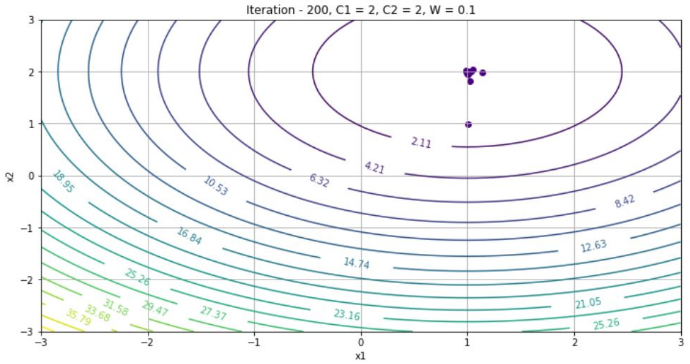



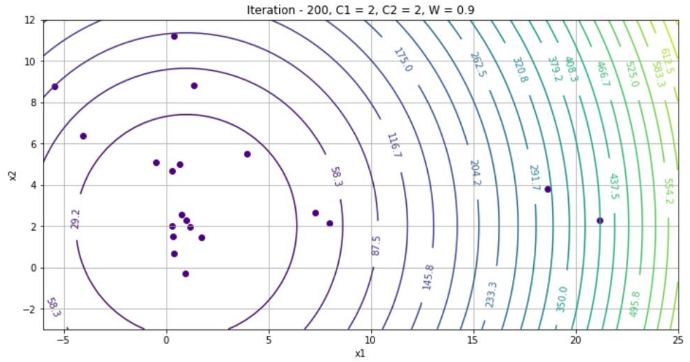

Particles do not converge since Exploration ↑ and Exploitation ↓

# Effects of Hyperparameters (3)

## ω (Inertia Weight) - Controls the effect of prev. iterations velocity on the next one




- Trade Off between Exploitation and Exploration.
- We used W := W*k where k ∈ [0.95,0.98]

# Remarks

**Potential Further Work:-**

- ➢ Experimentation with different conditional criterias for loop termination other than maximum iterations
- ➢ Tuning the algorithm’s parameters conveniently and effectively for peak performance
- ➢ Effective strategy can be build to balance the global explorations and local exploitation

# Conclusion:-

Advantages of the PSO algorithm:

- ➢ Derivative free algorithm
- ➢ Easy to implement
- ➢ Limited number of parameters
- ➢ Less reliance on starting points

Limitations :-

- ➢ Large possibility of falling into local optimum in high dimensional space
- ➢ Less accuracy of the optimum solution

# References


- ➢ Particle swarm optimization algorithm: an overview By Dongshu Wang, Dapei Tan, Lei Liu
- ➢ Particle Swarm Optimization Method for Constrained Optimization Problems By Michael N.
- ➢ Particle Swarm Optimization Wiki
- ➢ youtube.com/watch?v=JhgDMAm-imI&t=980s

# Thank You