REPORT

**Particle Swarm Optimization (PSO)**

**ABSTRACT:**

**Particle Swarm Optimization (PSO)**

The Particle Swarm Optimization (or famously known as the PSO algorithm), is a stochastic algorithm, based on the population of the subjects involved in the study. In this algorithm, the method recursively processes the inputs and positions of the particles or subjects, which are present in a large quantity to concatenate themselves on a desired common location through constant and repeated comparisons of certain measures within the group. This include the local- best position, which is the closest position of an individual object from the final location and global-best position, which is the position of the object closest to the destination. There are values and rules to be set before the application of this algorithm, which define the paths that the individual objects may take in order to achieve their common goal. This may include the distance one can cover, in what directions they can move and how much of each is to be done.

### INTRODUCTION:

Inspired from nature, the Particle Swarm Optimization algorithm was created after observing the common movement of a flock of birds in the search of food or during migration from one place to another. How they work in unity to find food for their group through common sharing of what they observe about their environments is a key part of this process. This concept of swarm intelligence presents a more reliable way of finding the most well-optimized solution for a specific problem. In this algorithm, the concept of swarm intelligence can be used to find the optimum solution for a problem in a large space. The algorithm uses the particle’s velocity and position to make them move around in the space. The objects use their own best positions and the global best positions to move around the space, which are updated as new overall best positions are found. This recursive process helps the objects to gradually move towards the optimum position in a non-linear fashion.

### PROGRAMMING DESIGN:

**Programming Language:** Python

**Algorithm:**

1. Create objects or particles that will be used to find the optimum solution in the space.
2. Decide every particle’s initial position using a reference function.
3. **If** current position of the particle is **better** than its previous best position, then:
   1. Find the current value.
   2. Update the particles local best position.
4. Find the best particle position overall and set it as the global best position.
5. Update the velocities of the particles using the search strategy.

Vk+1 i = WkVk i + c1r1(Pki - Xki) + c2r2(Pkg - Xki)

where the next velocity is determined by the inertia ‘w’, the locally best location of a particle ‘Pki’ and the globally best location among all particles ‘Pkg’

1. Update the particles’ positions respectively using:

xk+1 i = xk i+vk+1

1. **If** stopping condition is reached or maximum iterations are complete, then end the process.
2. **Else**, repeat process from Step 2.

### Pseudocode:

1. **For each particle:**

Initialize particle Pi

### END

1. **Do:**

### For each particle:

i. Compare positions of Pi

ii. If Pi (Current position) > Pi (Best position):

Pi (Best Position) = Pi (Current Position)

### END

* 1. **For each particle:**
     1. Compare positions of all particles
     2. Calculate velocities of particles using:

Vk+1 i = WkVk i + c1r1(Pki - Xki) + c2r2(Pkg - Xki)

* + 1. Update new positions of particles using:

xk+1 i = xk i+vk+1

### END

1. **While:** Stopping condition is not reached or maximum iterations are not completed.

### Space Complexity:

The space complexity of the algorithm will be **O(i)**, as **‘i’** particles will be initialized and will store coordinates of their positions.

### Time Complexity:

There is a for loop in the beginning of the process repeating **‘i’** times, where i is the number of particles. After that we see a do-while condition, which has the condition of either achieving stopping condition or completing maximum iterations. Since we consider the worst case, we assume that the condition will run for all iterations **‘n’**. Now that do-while loop consists of two for loops each running **‘i’** times, so it becomes **i raised to the power 2 or i2**. So, our estimated time complexity becomes: **O(i+ni2)**.

### CODE:

**import numpy as np**

**import pyswarms as ps**

**from pyswarms.utils.functions import single\_obj as fx import matplotlib.pyplot as plt**

**from pyswarms.utils.plotters import plot\_contour, plot\_surface from pyswarms.utils.plotters.formatters import Designer**

**from pyswarms.utils.plotters.formatters import Mesher**

**c1 = float(input("Input value of Personal Best(c1)= "))**

**c2 = float(input("Input value of Global Best(c2)= ")) inertia = float(input("Input value of Inertia(W)= "))**

**options = {'c1': c1, 'c2': c2, 'w':inertia}**

**optimizer = ps.single.GlobalBestPSO(n\_particles=10, dimensions=2, options=options)**

**cost, pos = optimizer.optimize(fx.sphere, iters=1000)**

**m = Mesher(func=fx.sphere)**

**animation = plot\_contour(pos\_history=optimizer.pos\_history,mesher=m,mark=(0,0)) plt.show()**

**pos\_history\_3d = m.compute\_history\_3d(optimizer.pos\_history)**

**d = Designer(limits=[(-1,1), (-1,1), (-0.1,1)], label=['x-axis', 'y-axis', 'z-axis'])**

**animation3d**

**=**

**plot\_surface(pos\_history=pos\_history\_3d,**

**mesher=m,**

**designer=d,mark=(0,0,0))**

**plt.show()**

**EXPERIMENTAL SETUP:**

We take multiple variations in the values of c1 and c2, where c1 is the confidence a particle has in itself or the local best and c2 is the confidence a particle has in its neighbors or the globally best position. Also, we take small variations in the total values of inertia of the particles in order to achieve our desired results. These changes in values present a wide range of results where the particles present different optimum positions and costs. Our code was able to graph the results using the matplotlib library. Some of the results are shown in the next chapter.

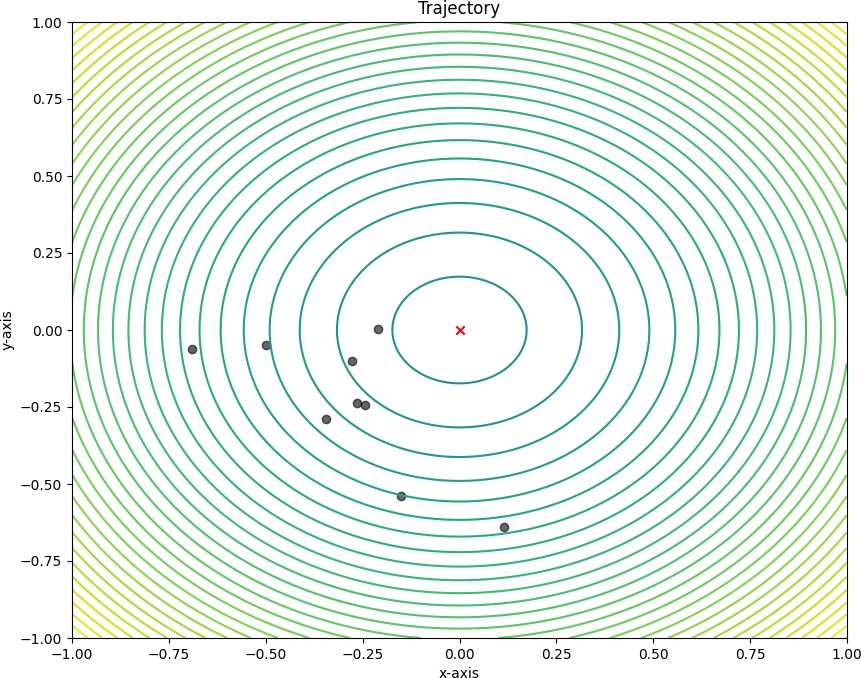
**RESULTS AND DISCUSSION:**

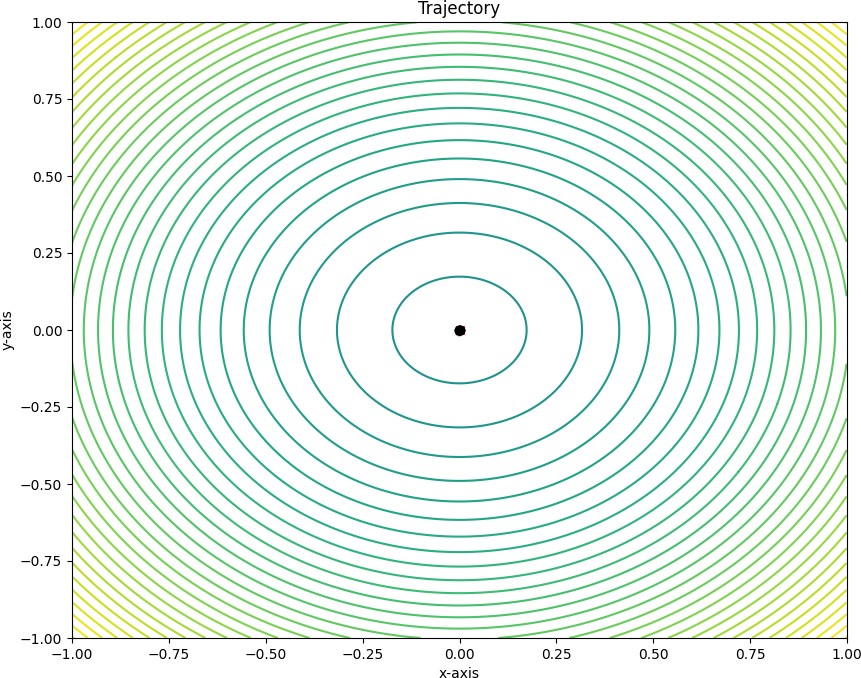
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Personal Best (c1)** | **Global Best (c2)** | **Inertia (w)** | **Best Cost** | **Best Position** |
| 0.3 | 0.5 | 0.9 | 6.45e-41 | 8.03291123e-21 -  4.40796249e-23 |
| 0.5 | 0.7 | 0.9 | 5.23e-38 | 1.94380996e-19 -  1.20384879e-19 |
| 0.4 | 0.6 | 0.9 | 2.68e-39 | 4.54673205e-20 -  2.48089416e-20 |
| 0.6 | 0.8 | 0.9 | 1.27e-33 | -6.28914938e-18  -3.50421682e-17 |
| 0.45 | 0.66 | 0.8 | 2.58e-91 | 4.97271318e-46  1.05467024e-46 |
| 0.145 | 0.334 | 0.7 | 1.63e-954 | -1.19188950e-77  -4.56872055e-78 |
| 0.223 | 0.335 | 0.9 | 1.28e-44 | -5.87788150e-24  1.12831713e-22 |
| 0.534 | 0.672 | 0.9 | 1.07e-36 | 9.72473810e-19  3.46866491e-19 |
| 0.7 | 0.8 | 0.6 | 9.28e-206 | -1.97431617e- 103  2.32003145e-103 |
| 0.523 | 0.772 | 0.9 | 6.23e-39 | 7.84412489e-20 -  8.80905215e-21 |
| 0.721 | 0.61 | 0.6 | 8.69e-218 | -2.29800095e- 109  1.84510914e-109 |
| 0.1212 | 0.4 | 0.9 | 1.39e-43 | 1.09466868e-22 -  3.55889874e-22 |
| 0.8 | 0.12 | 0.9 | 2.58e-42 | -6.44251247e-22  1.47011773e-21 |
| 0.61 | 0.58 | 0.9 | 4.77e-39 | 4.57301236e-20  5.17234005e-20 |
| 0.412 | 0.321 | 0.8 | 1.48e-95 | 3.46141499e-48  1.66687276e-48 |
| 0.74 | 0.45 | 0.9 | 1.84e-35 | 3.27141497e-18  2.76826675e-18 |
| 0.231 | 0.7 | 0.9 | 3.06e-38 | -1.18588196e-19  1.28550765e-19 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0.771 | 0.634 | 0.8 | 9.79e-87 | -4.88504771e-44  8.60465492e-44 |
| 0.845 | 0.2 | 0.9 | 8.54e-41 | -7.48957726e-21  5.41751464e-21 |
| 0.234 | 0.783 | 0.9 | 1.41e-36 | -4.74530124e-19  -1.08705433e-18 |

## Example:

For c1 = 0.3, c2 = 0.5 and w = 0.9





As we see all particles have collected on the best position represented by the collection of black dots.

## CONCLUSION:

The algorithm discussed above is one of the most used and reliable algorithms among scientists around the globe and can be used to solve a lot of optimization problems that may exist. It has been modified to fit other uses as well such as Hybridization, Gradient- based PSO algorithms, Alleviate premature convergence and others. Regardless, it also has its disadvantages as it is prone to fall in the local optimum in a large space and due to its recursive process, may have a convergence rate lower than other optimization algorithms.

## REFERENCES:

1. Lec 8 : Particle Swarm Optimization (PSO) (January 2021): NPTEL IIT Guwahati:

Youtube.com <https://www.youtube.com/watch?v=8dpYjgRskgQ>

1. Learn Particle Swarm Optimization (PSO) in 20 minutes (March 2018): Ali Mirjalli:

Youtube.com <https://www.youtube.com/watch?v=JhgDMAm-imI>

1. Particle swarm optimization: Wikipedia.com

[https://en.wikipedia.org/wiki/Particle\_swarm\_optimization#:~:text=In%20computational](https://en.wikipedia.org/wiki/Particle_swarm_optimization#%3A~%3Atext%3DIn%20computational%20science%2C%20particle%20swarm%2Ca%20given%20measure%20of%20quality)

[%20science%2C%20particle%20swarm,a%20given%20measure%20of%20quality](https://en.wikipedia.org/wiki/Particle_swarm_optimization#%3A~%3Atext%3DIn%20computational%20science%2C%20particle%20swarm%2Ca%20given%20measure%20of%20quality).

1. D. Wang · D. Tan · L. Liu (2017): Particle swarm optimization algorithm: an overview.

DOI 10.1007/s00500-016-2474-6