

Task 4 Particle Swarm Optimization

Target: To find the minimum of $f(x,y)$

$$f(x, y) = (x - 3.14)^2 + (y - 2.72)^2 + \sin(3x + 1.41) + \sin(4y - 1.73)$$

The Objective function need not to be differentiable because PSO does not use the gradient decent technique.

Step1:

We start with P number of particles. We need initial position and velocity as a starting point for every particle.

Step2:

In every iteration we update the new position using previous position and velocity of each particle. We also update the velocity at each iteration. Particle best position (Personal Best) and Global best positions are also updated in each iteration.

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

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

Particle update [Original Image]

Algorithm Testing:

Step 1: Plot the objective function



Task4.ipynb ☆

File Edit View Insert Runtime Tools Help

+ Code + Text



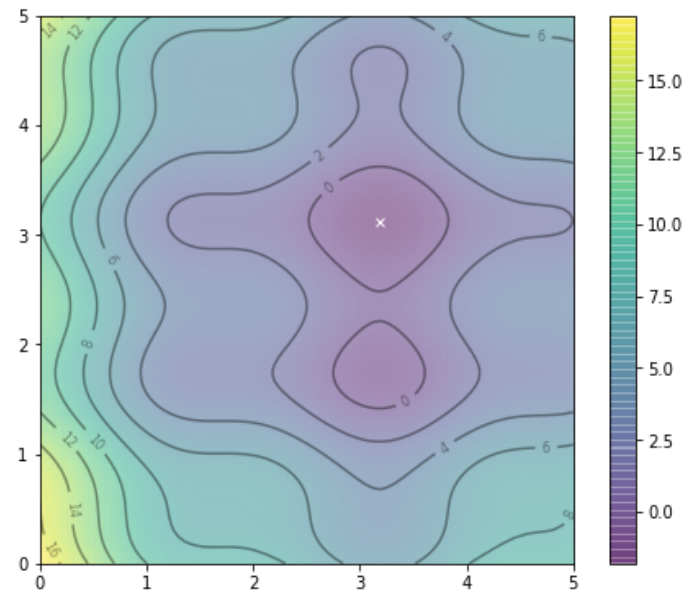
{x}



```
import numpy as np
import matplotlib.pyplot as plt

def f(x,y):
    "Objective function"
    return (x-3.14)**2 + (y-2.72)**2 + np.sin(3*x+1.41) + np.sin(4*y-1.73)

# Contour plot: With the global minimum showed as "X" on the plot
x, y = np.array(np.meshgrid(np.linspace(0,5,100), np.linspace(0,5,100)))
z = f(x, y)
x_min = x.ravel()[z.argmin()]
y_min = y.ravel()[z.argmin()]
plt.figure(figsize=(8,6))
plt.imshow(z, extent=[0, 5, 0, 5], origin='lower', cmap='viridis', alpha=0.5)
plt.colorbar()
plt.plot([x_min], [y_min], marker='x', markersize=5, color="white")
contours = plt.contour(x, y, z, 10, colors='black', alpha=0.4)
plt.clabel(contours, inline=True, fontsize=8, fmt="%.0f")
plt.show()
```



Step 2: Create particles (here 20), initial position and velocity

```
[3] #####Initialization

n_particles = 20
X = np.random.rand(2, n_particles) * 5
V = np.random.randn(2, n_particles) * 0.1
```

Step 3: Calculate the initial best personal and global positions for the particles

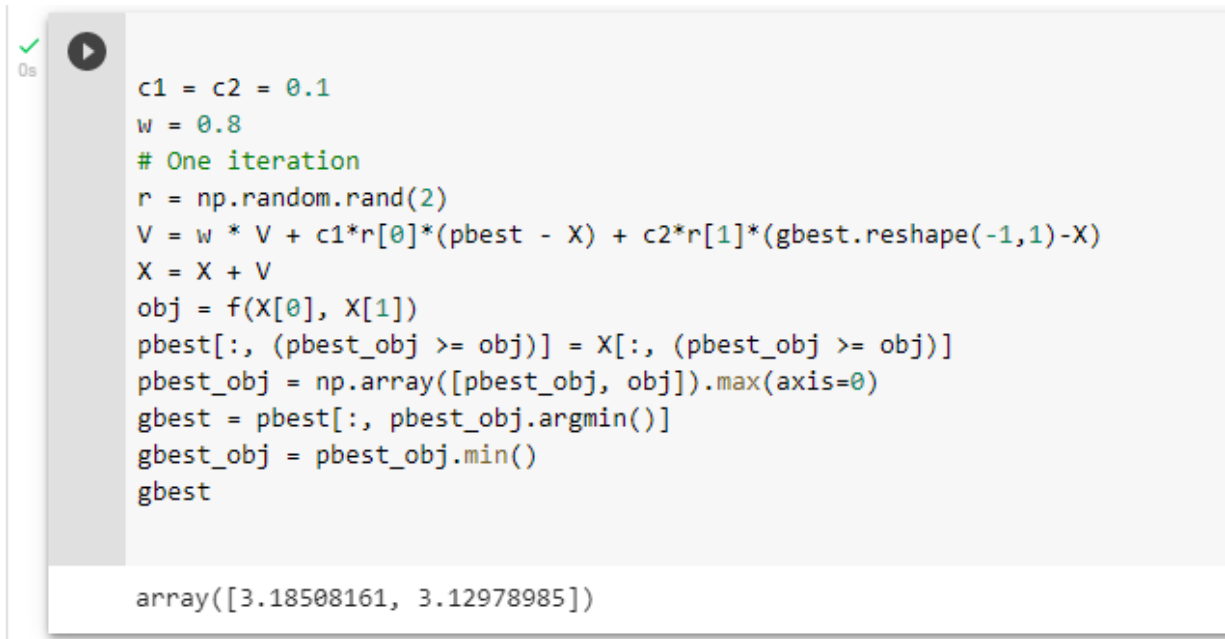
```
✓ 0s
▶
pbest = X
pbest_obj = f(X[0], X[1])
gbest = pbest[:, pbest_obj.argmin()]
gbest_obj = pbest_obj.min()
```

Step 4: Update the best positions in each iteration. Also using hyper parameters ($c1=c2=0.1$ and $w=0.8$)

Result of First Iteration:

```
✓ 1s
▶
c1 = c2 = 0.1
w = 0.8
# One iteration
r = np.random.rand(2)
V = w * V + c1*r[0]*(pbest - X) + c2*r[1]*(gbest.reshape(-1,1)-X)
X = X + V
obj = f(X[0], X[1])
pbest[:, (pbest_obj >= obj)] = X[:, (pbest_obj >= obj)]
pbest_obj = np.array([pbest_obj, obj]).max(axis=0)
gbest = pbest[:, pbest_obj.argmin()]
gbest_obj = pbest_obj.min()
gbest
gbest_obj
-0.7704175796382481
```

Step 5: Result of successive Iterations:



```
✓ 0s
c1 = c2 = 0.1
w = 0.8
# One iteration
r = np.random.rand(2)
V = w * V + c1*r[0]*(pbest - X) + c2*r[1]*(gbest.reshape(-1,1)-X)
X = X + V
obj = f(X[0], X[1])
pbest[:, (pbest_obj >= obj)] = X[:, (pbest_obj >= obj)]
pbest_obj = np.array([pbest_obj, obj]).max(axis=0)
gbest = pbest[:, pbest_obj.argmin()]
gbest_obj = pbest_obj.min()
gbest

array([3.18508161, 3.12978985])
```

We found the Global best position at particle position of (3.185,3.1297)

Answer the following questions:

Define following:

1. pbest
It is the personal best position of the particle at any iteration.
2. pbest_obj
When we put the personal best position of any particle into the objective function it gives the lowest personal best found by that particle in the objective function.
3. gbest
4. It is the global best position of the particle at any iteration.
5. gbest_obj

When we put the Global best position of the global best particle into the objective function it gives the Global best position found by that particle in the objective function.

6. V

It is the new update of the particle in its previous position.

6. update() function

This function calculates the new positions for the current iteration for particles.

$$V = w * V + c1*r1*(pbest - X) + c2*r2*(gbest.reshape(-1,1)-X)$$

$$X = X + V$$

7. Your thoughts on how we can apply PSO to the current problem that we are working on.

We want to select a subset of relevant features for use in model construction, in order to make prediction faster and more accurate. We will be using Particle Swarm Optimization to search for the optimal subset of features.

Our solution vector will represent a subset of features:

$$x = [x_1, x_2, \dots, x_d]; x_i \in [0, 1]$$

Where d is the total number of features in the dataset. We will then use a threshold of 0.5 to determine whether the feature will be selected:

$$x_i = \begin{cases} 1, & \text{if } x_i > 0.5 \\ 0, & \text{otherwise} \end{cases}$$

The function we'll be optimizing is the classification accuracy penalized by the number of features selected, that means we'll be minimizing the following function:

$$f(x) = \alpha \times (1 - P) + (1 - \alpha) \times \frac{N_{selected}}{N_{features}}$$

Where α is the parameter that decides the tradeoff between classifier performance P (classification accuracy in our case) and the number of selected features with respect to the number of all features.

This problem is implemented using UCI Breast Cancer detection with the Code attached and the reference is presented at the end.

Reference:

https://niapy.org/en/stable/tutorials/feature_selection.html