

In [1]:

```
1  # Import PySwarms
2  import numpy as np
3  import pyswarms as ps
4  from pyswarms.utils.functions import single_obj as fx
5  import random
6  import matplotlib.pyplot as plt
7  from matplotlib.animation import FuncAnimation
8
9  import statistics # added for the mean computation
10 from collections import defaultdict # added to compare elements of the list
11 from itertools import tee # to allow pairwise comparisons
12 from scipy.spatial.distance import cosine # to compute cosine distance
```

In [ ]:

```
1
```

In [2]:

```
1  # ... I also made some experiments with PySwarm
```

In [3]:

```
1  # Adapted from: https://machinelearningmastery.com/a-gentle-introduction-to-part
```

In [4]:

```
1  #n_particles = 10
2  #X = np.random.rand(2, n_particles)
3  #V = np.random.randn(2, n_particles)
```

In [5]:

```
1  #n_particles = 3
2  #print(np.random.rand(2, n_particles)*0.1 + 0.2)
```

In [6]:

```
1  n_particles = 3
2  print(np.random.rand(2, n_particles)*0.1 + 0.2)
3  print(np.random.rand(2, n_particles)*0.1 + 0.5)
```

```
[[0.23558849 0.2227348 0.21708076]
 [0.23560405 0.2847416 0.2140623 ]]
[[0.56738909 0.54754209 0.51420347]
 [0.51135421 0.5511132 0.56689559]]
```

In [7]:

```

1
2 def f(x,y):
3     "Objective function"
4     return (x-0.9)**2 + (y-0.5)**2 # new
5
6 # Compute and plot the function in 3D within [0,5]x[0,5]
7 x, y = np.array(np.meshgrid(np.linspace(0,1,100), np.linspace(0,1,100))) # 1, n
8 z = f(x, y)
9
10 # Find the global minimum
11 x_min = x.ravel()[z.argmin()]
12 y_min = y.ravel()[z.argmin()]
13
14 # Hyper-parameter of the algorithm
15 c1 = c2 = 0.1 # 0.1
16 w = 0.8 # 0.8
17
18 # Create particles
19 n_particles = 10 # 20
20 np.random.seed(1000) # take away or leave it here?
21 #X = np.random.rand(2, n_particles)*0.9 # I can generate them randomly but clo
22 #V = np.random.rand(2, n_particles)*0.01
23 X = np.random.rand(2, n_particles)*0.1 + 0.2
24 V = np.random.rand(2, n_particles)*0.1 + 0.2
25
26
27 #X = np.random.rand(2, n_particles) * 5
28 #V = np.random.randn(2, n_particles) * 0.1
29
30
31
32 # 0.2 + 0.2; 0.01 + 0.5
33
34 # with these parameters, we are already on the target:
35 # X = np.random.rand(2, n_particles)* 0.9
36 # V = np.random.rand(2, n_particles)*0.01
37 # also with 0.2, 0.4
38
39
40 #X = np.random.rand(2, n_particles) * 5
41 #V = np.random.randn(2, n_particles) * 0.1
42
43 # Initialize data
44 pbest = X
45 pbest_obj = f(X[0], X[1])
46 gbest = pbest[:, pbest_obj.argmin()]
47 gbest_obj = pbest_obj.min()
48
49 def update():
50     "Function to do one iteration of particle swarm optimization"
51     global V, X, pbest, pbest_obj, gbest, gbest_obj
52     # Update params
53     # r1, r2 = np.random.rand(2)
54     r1, r2 = np.random.rand(2)
55     V = w * V + c1*r1*(pbest - X) + c2*r2*(gbest.reshape(-1,1)-X)
56     X = X + V
57     obj = f(X[0], X[1])
58     pbest[:, (pbest_obj >= obj)] = X[:, (pbest_obj >= obj)]
59     pbest_obj = np.array([pbest_obj, obj]).min(axis=0)

```

```

60     gbest = pbest[:, pbest_obj.argmin()]
61     gbest_obj = pbest_obj.min()
62
63     # Set up base figure: The contour map
64     fig, ax = plt.subplots(figsize=(8,6))
65     fig.set_tight_layout(True)
66     img = ax.imshow(z, extent=[0, 1, 0, 1], origin='lower', cmap='viridis', alpha=0.5)
67     fig.colorbar(img, ax=ax)
68     ax.plot([x_min], [y_min], marker='x', markersize=5, color="white")
69     contours = ax.contour(x, y, z, 10, colors='black', alpha=0.4)
70     ax.clabel(contours, inline=True, fontsize=8, fmt="%.0f")
71     pbest_plot = ax.scatter(pbest[0], pbest[1], marker='o', color='black', alpha=0.5)
72     p_plot = ax.scatter(X[0], X[1], marker='o', color='blue', alpha=0.5)
73     p_arrow = ax.quiver(X[0], X[1], V[0], V[1], color='blue', width=0.005, angles='xy')
74     gbest_plot = plt.scatter([gbest[0]], [gbest[1]], marker='*', s=100, color='black')
75     ax.set_xlim([0,1])
76     ax.set_ylim([0,1])
77
78
79     def animate(i):
80         "Steps of PSO: algorithm update and show in plot"
81         title = 'Iteration {:02d}'.format(i)
82         # Update params
83         update()
84         # Set picture
85         ax.set_title(title)
86         pbest_plot.set_offsets(pbest.T)
87         p_plot.set_offsets(X.T)
88         p_arrow.set_offsets(X.T)
89         p_arrow.set_UVC(V[0], V[1])
90         gbest_plot.set_offsets(gbest.reshape(1,-1))
91         return ax, pbest_plot, p_plot, p_arrow, gbest_plot
92
93     anim = FuncAnimation(fig, animate, frames=list(range(1,50)), interval=500, blit=False)
94     anim.save("PSO.gif", dpi=120, writer="imagemagick")
95
96     print("PSO found best solution at f({})={}".format(gbest, gbest_obj))
97     print("Global optimal at f({})={}".format([x_min,y_min], f(x_min,y_min)))
98
99
100
101     # putting these commands over there, we get the values at the end of the simulation
102     print("The X-coordinates are: ", X[0]) # Added on September 14
103     print("The Y-coordinates are: ", X[1]) # Added on September 14

```

2022-09-14 18:52:31,469 - matplotlib.animation - WARNING - MovieWriter  
 imagemagick unavailable; using Pillow instead.

2022-09-14 18:52:31,471 - matplotlib.animation - INFO - Animation.save  
 using <class 'matplotlib.animation.PillowWriter'>

PSO found best solution at f([0.79994233 0.59251027])=0.01856968734445  
 0914

Global optimal at f([0.8989898989898991, 0.494949494949495])=2.6527905  
 315783662e-05

The X-coordinates are: [0.79994233 0.79942311 0.80055935 0.79998989  
 0.79961004 0.79992951  
 0.79933117 0.7993422 0.79960289 0.8000102 ]

The Y-coordinates are: [0.59251027 0.59248745 0.59403333 0.59348771  
 0.59312173 0.59321301  
 0.59270495 0.5931111 0.5924977 0.59359378]



In [ ]:

```
1
2
```

In [8]:

```
1 # define a class Robot_PSO with the x, y instances... or, directly work with the
2 # Not needed classes here, we already have the position outputs, and the X[0], X[1]
3
4 # class of the target (here: minimum of the objective function)
5
6 class Target:
7     def __init__(self, name, x, y): # no indetermination in the target's position
8         self.name = name
9         self.x = x
10        self.y = y
11
12 T = Target("T", 0.9, 0.5) # deep in the ocean
13
```

In [9]:

```
1  #listX = list(k.betax for k in Robot._registry)
2  #listY = list(k.betay for k in Robotx._registry)
3
4  listX = X[0]
5  listY = X[1]
6
7  num_of_robots = 10
8
9  def Euclidean_distance(T, listX, listY): # the same as distance_A
10     sum_x = sum(listX)
11     sum_y = sum(listY)
12     center_x = sum_x/num_of_robots
13     center_y = sum_y/num_of_robots
14     return ((T.x - center_x)**2 + (T.y - center_y)**2)**0.5
15
16 print("Euclidean", Euclidean_distance(T, listX, listY))
17
18 def Manhattan_distance(T, listX, listY):
19     sum_x = sum(listX)
20     sum_y = sum(listY)
21     center_x = sum_x/num_of_robots
22     center_y = sum_y/num_of_robots
23     return (abs(T.x - center_x) + abs(T.y - center_y))
24
25 print("Manhattan", Manhattan_distance(T, listX, listY))
26
27 def Cosine_distance(T, listX, listY):
28     sum_x = sum(listX)
29     sum_y = sum(listY)
30     center_x = sum_x/num_of_robots
31     center_y = sum_y/num_of_robots
32     array_1 = np.array([center_x, T.x])
33     array_2 = np.array([center_y, T.y])
34     return cosine(array_1, array_2)
35
36 print("Cosine", Cosine_distance(T, listX, listY))
```

Euclidean 0.13677864696322908

Manhattan 0.19330203449006922

Cosine 0.010327404086080127

In [ ]:

1