In [1]:

```python
# Import PySwarms
import numpy as np
import pyswarms as ps
from pyswarms.utils.functions import single_obj as fx
import random
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

import statistics # added for the mean computation
from collections import defaultdict # added to compare elements of the list
from itertools import tee # to allow pairwise comparisons
from scipy.spatial.distance import cosine # to compute cosine distance
```

In [ ]:

```python

```

In [2]:

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

In [3]:

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

In [4]:

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

In [5]:

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

In [6]:

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

```
[[0.24972397 0.21141381 0.22437356]
 [0.26430362 0.29491385 0.27558969]]
[[0.51106673 0.50734775 0.52648022]
 [0.52810773 0.5554692  0.56918046]]
```

In [7]:

```python
def f(x,y):
    "Objective function"
    return (x-0.9)**2 + (y-0.5)**2 # new

# Compute and plot the function in 3D within [0,5]x[0,5]
x, y = np.array(np.meshgrid(np.linspace(0,1,100), np.linspace(0,1,100))) # 1, n
z = f(x, y)

# Find the global minimum
x_min = x.ravel()[z.argmin()]
y_min = y.ravel()[z.argmin()]

# Hyper-parameter of the algorithm
c1 = c2 = 0.1 # 0.1
w = 0.8 # 0.8

# Create particles
n_particles = 10 # 20
np.random.seed(1000) # take away or leave it here?
X = np.random.rand(2, n_particles)*0.9  # I can generate them randomly but clos
V = np.random.rand(2, n_particles)*0.01
# X = np.random.rand(2, n_particles)*0.1 + 0.2
# V = np.random.rand(2, n_particles)*0.1 + 0.2


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



# 0.2 + 0.2; 0.01 + 0.5

# with these parameters, we are already on the target:
# X = np.random.rand(2, n_particles)* 0.9
# V = np.random.rand(2, n_particles)*0.01
# also with 0.2, 0.4


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

# Initialize data
pbest = X
pbest_obj = f(X[0], X[1])
gbest = pbest[:, pbest_obj.argmin()]
gbest_obj = pbest_obj.min()

def update():
    "Function to do one iteration of particle swarm optimization"
    global V, X, pbest, pbest_obj, gbest, gbest_obj
    # Update params
    # r1, r2 = np.random.rand(2)
    r1, r2 = np.random.rand(2)
    V = w * V + c1*r1*(pbest - X) + c2*r2*(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]).min(axis=0)
```

```python
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
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.
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='
74    gbest_plot = plt.scatter([gbest[0]], [gbest[1]], marker='*', s=100, color='blac
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
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 simula
102   print("The X-coodinates  are: ", X[0]) # Added on September 14
103   print("The Y-coodinates  are: ", X[1]) # Added on September 14
```

```
2022-09-14 18:48:26,469 - matplotlib.animation - WARNING - MovieWriter
imagemagick unavailable; using Pillow instead.
2022-09-14 18:48:26,471 - matplotlib.animation - INFO - Animation.save
using <class 'matplotlib.animation.PillowWriter'>

PSO found best solution at f([0.90005957 0.49987121])=2.01347749083424
05e-08
Global optimal at f([0.8989898989898991, 0.494949494949495])=2.6527905
315783662e-05
The X-coodinates  are:  [0.90009066 0.89985779 0.90069937 0.89986114
0.90044802 0.89940503
 0.89959254 0.90022084 0.89986117 0.89996884]
The Y-coodinates  are:  [0.50076851 0.50001332 0.49987415 0.49921265
0.50012569 0.49930679
 0.50024451 0.50055499 0.50038255 0.50028202]
```
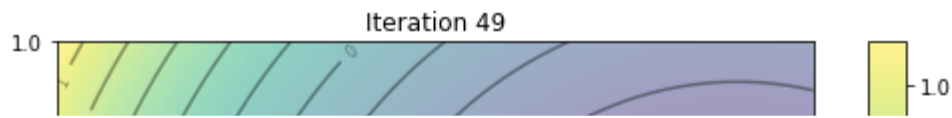
Iteration 49

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], Y
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]:

```python
#listX = list(k.betax for k in Robot._registry)
#listY = list(k.betay for k in Robotx._registry)

listX = X[0]
listY = X[1]

num_of_robots = 10

def Euclidean_distance(T, listX, listY): # the same as distance_A
    sum_x = sum(listX)
    sum_y = sum(listY)
    center_x = sum_x/num_of_robots
    center_y = sum_y/num_of_robots
    return ((T.x - center_x)**2 + (T.y - center_y)**2)**0.5

print("Euclidean", Euclidean_distance(T, listX, listY))

def Manhattan_distance(T, listX, listY):
    sum_x = sum(listX)
    sum_y = sum(listY)
    center_x = sum_x/num_of_robots
    center_y = sum_y/num_of_robots
    return (abs(T.x - center_x) + abs(T.y - center_y))

print("Manhattan", Manhattan_distance(T, listX, listY))

def Cosine_distance(T, listX, listY):
    sum_x = sum(listX)
    sum_y = sum(listY)
    center_x = sum_x/num_of_robots
    center_y = sum_y/num_of_robots
    array_1 = np.array([center_x, T.x])
    array_2 = np.array([center_y, T.y])
    return cosine(array_1, array_2)

print("Cosine", Cosine_distance(T, listX, listY))
```

```
Euclidean 7.652006605708254e-05
Manhattan 7.705837344196276e-05
Cosine 2.9041486993008903e-09
```

In [ ]:

```

```

In [ ]:

```

```