**Particle Swarm Optimization Algorithm**

**Tabish, CSE-09/16**

**Department of Computer Science & Engineering, NIT Srinagar**

**Introduction to the Algorithm:**

The Particle Swarm Optimization (PSO) algorithm is a biologically-inspired algorithm motivated by the social/ecological analogy that a flock of birds (termed particles) will work together to find the location of a food source (optimal solution) without a direct leader. Each bird in the flock, flying in a general formation with other birds, uses the knowledge of their experience and the experience of the group as a whole to find the location of food. Therefore, each bird tends (with some randomness) to fly to a location that is a combination of their previous best location and the best location found by the group as a hole. In this sense, the entire flock can converge on the optimal location using swarm intelligence. The metaphor that led to this social/ecological analogy gives three main features for each bird (particle). The particle has: 1) knowledge of its environment (its fitness value); 2) knowledge of its previous state history (particle best); 3) knowledge of the group's previous best state (global best).

In a more mathematical sense, the PSO algorithm is a population-based algorithm that borrows features from genetic and evolutionary algorithms in that a set of potential solutions evolves with each iteration and approaches an optimal solution (or in some cases a set of solutions) for a problem. Being an optimization method, the goal of the algorithm is to obtain a real-valued fitness function (either minimum or maximum) defined in a given search space. One of the simplest forms of the PSO is known as the "continuous" form. This form can use any real-valued multidimensional space as the search space and evolves the position of each particle in that space with each new iteration by updating the particle's position and velocity. The continuous form can be easily expressed by the following two equations:

https://adowney2.public.iastate.edu/projects/The_simplest_Particle_Swarm/image001.png

https://adowney2.public.iastate.edu/projects/The_simplest_Particle_Swarm/image002.png

**Algorithm:**

for each particle i = 1, ..., S do

Initialize the particle's position with a uniformly distributed random vector: xi ~ U(blo, bup)

Initialize the particle's best known position to its initial position: pi ← xi

if f(pi) < f(g) then

update the swarm's best known position: g ← pi

Initialize the particle's velocity: vi ~ U(-|bup-blo|, |bup-blo|)

while a termination criterion is not met do:

for each particle i = 1, ..., S do

for each dimension d = 1, ..., n do

Pick random numbers: rp, rg ~ U(0,1)

Update the particle's velocity: vi,d ← ω vi,d + φp rp (pi,d-xi,d) + φg rg (gd-xi,d)

Update the particle's position: xi ← xi + vi

if f(xi) < f(pi) then

Update the particle's best-known position: pi ← xi

if f(pi) < f(g) then

Update the swarm's best-known position: g ← pi

A basic variant of the PSO algorithm works by having a population (called a swarm) of candidate solutions (called particles). These particles are moved around in the search-space according to a few simple formulae.

The movements of the particles are guided by their own best-known position in the search-space as well as the entire swarm's best-known position. When improved positions are being discovered these will then come to guide the movements of the swarm. The process is repeated and by doing so it is hoped, but not guaranteed, that a satisfactory solution will eventually be discovered.

**Code:**

import numpy as np

import time as time

import matplotlib.pyplot as plt

import matplotlib as mpl

plt.close('all')

#%% Define the inputs to the PSO

# Define the function that will be optimized. Here, the optimal value is x=10, y=20

def Function(x,y):

z = (x-10)\*\*2 + (y-20)\*\*2

return(z)

# Set the parameters for the particle swarm optimization. The parameters set here

# were selected as they provide for a simple-to-read final plot. Experimentation

#with the parameters is encouraged.

swarm\_size = 5 # number of the swarm particles

iterations = 30 # maximum number of iterations

inertia = 0.5 # inertia of a particle

dimensions = 2 # number of values the PSO will optimize

local\_weight = 2 # weighted factor for the particles historical best

global\_weight = 2 # weighted factor for the the global best

max\_velocity = 1 # the highest velocity allowed for a particle

step\_size = 1 # step size for updating each particle, or how far a particle

# travels before its velocity is readjusted

#%% Setup the PSO

# set the x,y location of the particle's (initial guess) scattered around the x=0,y=0

particle\_location = np.random.rand(swarm\_size,dimensions)-0.5

# set the initial velocity of the particles in each direction

particle\_velocity = np.random.rand(swarm\_size,dimensions)

# solve the function for the particle's locations and save as their local best

particle\_best\_value = Function(particle\_location[:,0],particle\_location[:,1])

particle\_best\_location = np.copy(particle\_location)

global\_best\_value = np.min(particle\_best\_value)

global\_best\_location = particle\_location[np.argmin(particle\_best\_value)].copy()

best\_value = [] # for the best fitting value

best\_locaion = [] # for the location of the best fitting value

iteration\_value\_best = [] # for the best fitting value of the iteration

iteration\_locaion\_best = [] # for the location of the best fitting value of the iteration

iteration\_value = [] # for the values of the iteration

iteration\_locaion = [] # for the locations of the iteration

for iteration\_i in range(iterations): # for each iteration

for particle\_i in range(swarm\_size): # for each particle

for dimension\_i in range(dimensions): # for each dimension

# generate 2 random numbers between 0 and 1

u = np.random.rand(dimensions)

error\_particle\_best = particle\_best\_location[particle\_i,dimension\_i] - \

particle\_location[particle\_i,dimension\_i]

error\_global\_best = global\_best\_location[dimension\_i] - \

particle\_location[particle\_i,dimension\_i]

# update the velocity vector in a given dimension

v\_new = inertia\*particle\_velocity[particle\_i,dimension\_i] + \

local\_weight\*u[0]\*error\_particle\_best + \

global\_weight\*u[1]\*error\_global\_best

# bound a particle's velocity to the maximum value set above

if v\_new < -max\_velocity:

v\_new = -max\_velocity

elif v\_new > max\_velocity:

v\_new = max\_velocity

particle\_location[particle\_i,dimension\_i] = particle\_location[particle\_i,dimension\_i] + \

v\_new\*step\_size

# update the particle velocity

particle\_velocity[particle\_i,dimension\_i] = v\_new

# for the new location, check if this is a new local or global best

v = Function(particle\_location[particle\_i,0],particle\_location[particle\_i,1])

# update if its a new local best

if v < particle\_best\_value[particle\_i]:

particle\_best\_value[particle\_i]=v

particle\_best\_location[particle\_i,:] = particle\_location[particle\_i,:].copy()

# update if its a new global best

if v < global\_best\_value:

global\_best\_value=v

global\_best\_location = particle\_location[particle\_i,:].copy()

# print the current best location to the console

print('solution at x='+'%.2f' % global\_best\_location[0]+', y='+'%.2f' % global\_best\_location[1])

# update the lists

best\_value.append(global\_best\_value.copy())

best\_locaion.append(global\_best\_location.copy())

iteration\_value.append(v)

iteration\_locaion.append(particle\_location.copy())

v = Function(particle\_location[:,0].copy(),particle\_location[:,1].copy())

iteration\_value\_best.append(np.min(v))

iteration\_locaion\_best.append(particle\_location[np.argmin(v),:])

plt.figure(figsize=(5,5))

plt.grid('on')

plt.rc('axes', axisbelow=True)

plt.scatter(10,20,100,marker='\*',facecolors='k', edgecolors='k')

#plt.text(10,21,'optimal solution',horizontalalignment='center')

for i in range(len(iteration\_locaion)):

plt.scatter(iteration\_locaion[i][:,0],iteration\_locaion[i][:,1],10,marker='x')

plt.scatter(best\_locaion[i][0],best\_locaion[i][1],50,marker='o',facecolors='none',edgecolors='k',linewidths=0.4)

plt.text(best\_locaion[i][0]+0.1,best\_locaion[i][1]+0.1,str(i),fontsize=8)

plt.xlim(-1,15)

plt.ylim(-1,23)

plt.xlabel('$x$')

plt.ylabel('$y$')

plt.title('2-dimensional particle swarm optimization')

plt.show()

**Output Observed for different Inputs:**

* **Input passed:**

particle\_location chosen randomly via - np.random.rand(swarm\_size,dimensions)-0.5

particle\_velocity = np.random.rand(swarm\_size,dimensions)

function = x1\*\*2 + x2\*\*2

swarm\_size = 5 # number of the swarm particles

iterations = 56 # maximum number of iterations

inertia = 0.5 # inertia of a particle

dimensions = 2 # number of values the PSO will optimize

local\_weight = 2 # weighted factor for the particles historical best

global\_weight = 2 # weighted factor for the the global best

max\_velocity = 1 # the highest velocity allowed for a particle

step\_size = 1 # step size for updating each particle

* **Output Obtained:**

solution at x=1.48, y=1.16

solution at x=2.05, y=2.11

solution at x=3.05, y=3.11

solution at x=4.05, y=4.11

solution at x=5.05, y=5.11

solution at x=6.05, y=5.98

solution at x=6.57, y=6.98

solution at x=6.78, y=7.85

solution at x=7.75, y=8.85

solution at x=8.33, y=9.85

solution at x=9.33, y=10.85

solution at x=10.33, y=11.74

solution at x=11.14, y=12.62

solution at x=12.13, y=13.62

solution at x=12.48, y=14.62

solution at x=12.90, y=15.62

solution at x=13.11, y=16.12

solution at x=13.29, y=16.98

solution at x=13.18, y=17.82

solution at x=12.67, y=18.82

solution at x=11.67, y=19.01

solution at x=10.67, y=19.28

solution at x=10.36, y=20.07

solution at x=9.98, y=20.28

solution at x=9.98, y=20.28

solution at x=9.87, y=19.99

solution at x=9.87, y=19.99

solution at x=9.87, y=19.99

solution at x=9.87, y=19.99

solution at x=9.96, y=19.89

