Assignment No. 3

Title: Implement Particle swarm optimization

Problem Statement:

Implement Particle swarm optimization for benchmark function (eg. Square, Rosenbrock function). Initialize the population from the Standard Normal Distribution. Evaluate fitness of all particles.

Use:

c1=c2 = 2

Inertia weight is linearly varied between 0.9 to 0.4.

Global best variation

Objectives:

1. To learn swarm algorithm
2. To learn about optimization algorithm.

Software Requirements:

- Ubuntu 16.04

Hardware Requirements:

- Pentium IV system with latest configuration

Theory:

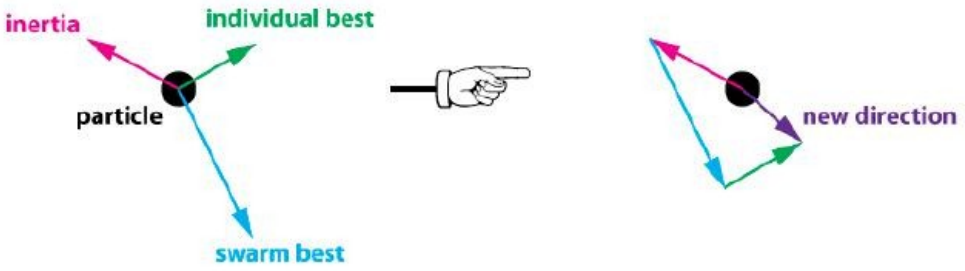
Particle swarm optimization (PSO) is a population based stochastic optimization technique developed by Dr. Eberhart and Dr. Kennedy in 1995, inspired by social behavior of bird flocking or fish schooling.

PSO shares many similarities with evolutionary computation techniques such as Genetic Algorithms (GA). The system is initialized with a population of random solutions and searches for optima by updating generations. However, unlike GA, PSO has no evolution operators such as crossover and mutation. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles. Compared to GA, the advantages of PSO are that PSO is easy to implement and there are few parameters to adjust. PSO has been successfully applied in many areas: function optimization, artificial neural network training, fuzzy system control, and other areas where GA can be applied.

PSO uses a bunch of particles called *the swarm*. These particles are allowed to move around & explore the search-space.

These particles move in a direction which is guided by —

1. The particle's own previous velocity (*Inertia*)
2. Distance from the individual particles' best known position (*Cognitive Force*)
3. Distance from the swarms best known position (*Social Force*)



Essentially the particles collectively communicate with each other to converge faster. The swarm doesn't fully explore the search space but potentially finds a better solution. Interestingly the overall direction of the swarm movement can be changed at any point of time when a particle's individual best is better than the swarm best. This allows a lot of *disorder* and more chances of getting close to the global minima of the cost function.

Algorithm

PSO is initialized with a group of random particles (solutions) and then searches for optima by updating generations. In every iteration, each particle is updated by following two "best" values. The first one is the best solution (fitness) it has achieved so far. (The fitness value is also stored.) This value is called pbest. Another "best" value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the population. This best value is a global best and called gbest. When a particle takes part of the population as its topological neighbours, the best value is a local best and is called lbest. After finding the two best values, the particle updates its velocity and positions with following equation (a) and (b).

v[] = v[] + c1 * rand() * (pbest[] - present[]) + c2 * rand() * (gbest[] - present[])
……..(a)
present[] = persent[] + v[] ……(b)

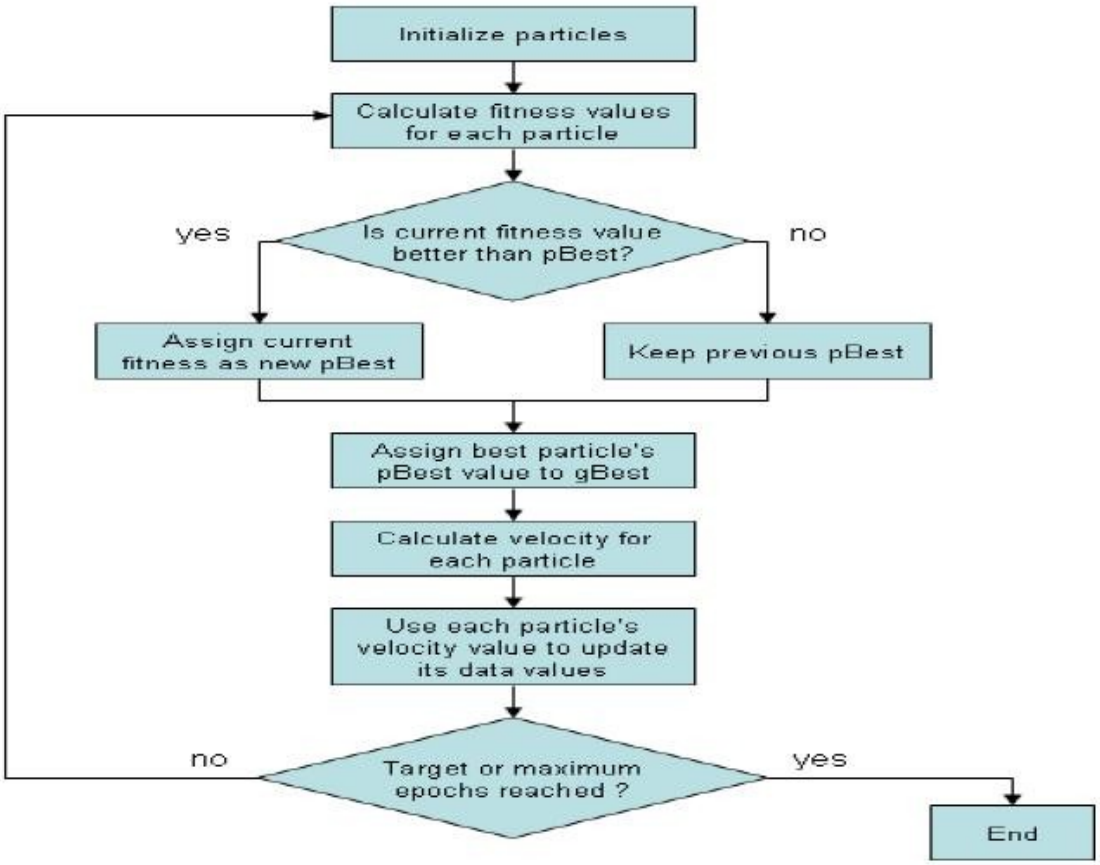
v[] is the particle velocity, persent[] is the current particle (solution). pbest[] and gbest[] are defined as stated before. rand () is a random number between (0,1). c1, c2 are learning factors. Usually
c1 = c2 = 2.

The pseudo code of the procedure is as follows:

For each particle
Initialize particle
END

Do
For each particle

Calculate fitness value
If the fitness value is better than the best fitness value (pBest) in history
set current value as the new pBest
End



Choose the particle with the best fitness value of all the particles as the
gBest For each particle
Calculate particle velocity according equation (a)
Update particle position according equation (b)
End
While maximum iterations or minimum error criteria is not attained

Particles' velocities on each dimension are clamped to a maximum velocity Vmax. If the sum of accelerations would cause the velocity on that dimension to exceed Vmax, which is a parameter specified by the user. Then the velocity on that dimension is limited to Vmax.

Conclusion

Thus we learnt implementation of particle swarm optimization for benchmark function.

```
from __future__ import division
import random
import math

#--- COST FUNCTION
def func1(x):
    total=0
    for i in range(len(x)):
        total+=x[i]**2
    return total

class Particle:
    def __init__(self,x0):
        self.position_i=[]          # particle position
        self.velocity_i=[]          # particle velocity
        self.pos_best_i=[]          # best position individual
        self.err_best_i=-1          # best error individual
        self.err_i=-1               # error individual

        for i in range(0,num_dimensions):
            self.velocity_i.append(random.uniform(-1,1))
            self.position_i.append(x0[i])

    # evaluate current fitness
    def evaluate(self,costFunc):
        self.err_i=costFunc(self.position_i)

        # check to see if the current position is an individual best
        if self.err_i < self.err_best_i or self.err_best_i==-1:
            self.pos_best_i=self.position_i
            self.err_best_i=self.err_i

    # update new particle velocity
    def update_velocity(self,pos_best_g):
        w=0.5       # constant inertia weight (how much to weigh the previous veloc
        c1=2        # cognative constant
        c2=2        # social constant

        for i in range(0,num_dimensions):
            r1=random.random()
            r2=random.random()

            vel_cognitive=c1*r1*(self.pos_best_i[i]-self.position_i[i])
            vel_social=c2*r2*(pos_best_g[i]-self.position_i[i])
            self.velocity_i[i]=w*self.velocity_i[i]+vel_cognitive+vel_social

    # update the particle position based off new velocity updates
    def update_position(self,bounds):
        for i in range(0,num_dimensions):
            self.position_i[i]=self.position_i[i]+self.velocity_i[i]

            # adjust maximum position if necessary
            if self.position_i[i]>bounds[i][1]:
                self.position_i[i]=bounds[i][1]
```

```
                self.position_i[i]=bounds[i][1]

            # adjust minimum position if neseccary
            if self.position_i[i] < bounds[i][0]:
                self.position_i[i]=bounds[i][0]

class PSO():
    def __init__(self,costFunc,x0,bounds,num_particles,maxiter):
        global num_dimensions

        num_dimensions=len(x0)
        err_best_g=-1                   # best error for group
        pos_best_g=[]                   # best position for group

        # establish the swarm
        swarm=[]
        for i in range(0,num_particles):
            swarm.append(Particle(x0))

        # begin optimization loop
        i=0
        while i < maxiter:
            #print i,err_best_g
            # cycle through particles in swarm and evaluate fitness
            for j in range(0,num_particles):
                swarm[j].evaluate(costFunc)

                # determine if current particle is the best (globally)
                if swarm[j].err_i < err_best_g or err_best_g == -1:
                    pos_best_g=list(swarm[j].position_i)
                    err_best_g=float(swarm[j].err_i)

            # cycle through swarm and update velocities and position
            for j in range(0,num_particles):
                swarm[j].update_velocity(pos_best_g)
                swarm[j].update_position(bounds)
            i+=1

        # print final results
        print ('FINAL:')
        print (pos_best_g)
        print (err_best_g)

if __name__ == "__PSO__":
    main()

initial=[5,5]               # initial starting location [x1,x2...]
bounds=[(-10,10),(-10,10)]  # input bounds [(x1_min,x1_max),(x2_min,x2_max)...]
PSO(func1,initial,bounds,num_particles=15,maxiter=30)
```

```
FINAL:
[-0.004017286255555613, -0.0031876698735276484]
2.6299828081671816e-05
<__main__.PSO at 0x7f500653f750>
```