Name: Nachiket Erlekar

Roll no: 41434 Batch: Q4

Assignment 3

Title: 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 and 0.4.

• Global best variation

Objectives: To understand concept of genetic algorithm and use it for optimization

Outcome: We will understand the concept of a genetic algorithm and use it to minimize the Rosenbrock function

Requirements:

1) Python 3.7+, Jupyter Notebook
2) Windows 64-bit OS

Theory:

Particle swarm optimization (**PSO**) is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. It solves a problem by having a population of candidate solutions, here dubbed particles, and moving these particles around in the search-space according to simple mathematical formula over the particle's position and velocity. Each particle's movement is influenced by its local best known position, but is also guided toward the best known positions in the search-space, which are updated as better positions are found by other particles. This is expected to move the swarm toward the best solutions.

It was first intended for simulating social behavior as a stylized representation of the movement of organisms in a bird flock or fish school. The algorithm was simplified and it was observed to be performing optimization.

PSO is a meta-heuristic as it makes few or no assumptions about the problem being optimized and can search very large spaces of candidate solutions. Also, PSO does not use the gradient of the problem being optimized, which means PSO does not require that the optimization problem be differentiable as is required by classic optimization methods such as gradient descent and quasi-newton methods. However, meta-heuristics such as PSO do not guarantee an optimal solution is ever found. The choice of PSO parameters can have a large impact on optimization performance. Selecting PSO parameters that yield good performance has therefore been the subject of much research

The pseudo code for PSO is as follows:

1. For each particle i = 1, ..., S do
    a. Initialize the particle's position with a uniformly distributed random vector: $x_i \sim U(b_{low}, b_{up})$
    b. Initialize the particle's best known position to its initial position: $p_i \leftarrow x_i$
    c. If $f(p_i) < f(g)$ then

i. Update the swarm's best known position: $g \leftarrow p_i$

d. Initialize the particle's velocity: $vi \sim U(-|b_{low} - b_{up}|, |b_{low} - b_{up}|)$

2. While a termination criterion is not met do:
   a. For each particle i = 1, ..., S do
      i. For each dimension d = 1, ..., n do
         1. Pick random numbers: $r_p, r_g \sim U(0,1)$
         2. Update the particle's velocity: $v_{i,d} \leftarrow \omega * v_{i,d} + \varphi_p * r_p * (p_{i,d} - x_{i,d}) + \varphi_g * r_g * (g_d - x_{i,d})$
      ii. Update the particle's position: $x_i \leftarrow x_i + l_r * v_i$
      iii. F $f(x_i) < f(p_i)$ then
         1. Update the particle's best known position: $p_i \leftarrow x_i$
         2. If $f(p_i) < f(g)$ then
            a. Update the swarm's best known position: $g \leftarrow p_i$

The values $\mathbf{b_{low}}$ and $\mathbf{b_{up}}$ represent the lower and upper boundaries of the search-space respectively. The termination criterion can be the number of iterations performed, or a solution where the adequate objective function value is found. The parameters ω, φ_p, and φ_g are selected by the practitioner and control the behavior and efficacy of the PSO method. l_r represents the learning rate ($0 \leq l_r \leq 1.0$), which is the proportion at which the velocity affects the movement of the particle (where $l_r = 0$ means the velocity will not affect the particle at all and $l_r = 1$ means the velocity will fully affect the particle).

<u>Test Cases</u>:

Consider the Rosenbrock function which is to be minimized

$f(x, y) = (1 - x)^2 + 100 * (y - x^2)^2$ with a known minima of 0 at point (0, 0)

```
iter# 0: Fitness:1.43346, Position:[-0.07025182  0.05860275], Velocity:6.965992491545918
iter# 1: Fitness:1.24624, Position:[-0.11055051  0.0235869 ], Velocity:6.551251842039969
iter# 2: Fitness:1.24624, Position:[-0.11055051  0.0235869 ], Velocity:3.2540270168034016
iter# 3: Fitness:1.23772, Position:[-0.09075492 -0.0136674 ], Velocity:6.801100101725705
iter# 4: Fitness:1.23772, Position:[-0.09075492 -0.0136674 ], Velocity:4.803002401418271
iter# 5: Fitness:0.95866, Position:[0.16316706 0.07745378], Velocity:3.806331022515294
iter# 6: Fitness:0.49026, Position:[0.33357061 0.13274685], Velocity:1.6236587105164022
iter# 7: Fitness:0.49026, Position:[0.33357061 0.13274685], Velocity:5.880571178206715
iter# 8: Fitness:0.36177, Position:[0.67121079 0.50088948], Velocity:3.646983976666997
iter# 9: Fitness:0.29595, Position:[0.5504404  0.27234975], Velocity:5.075754819225084
iter# 10: Fitness:0.29595, Position:[0.5504404  0.27234975], Velocity:1.1992917675600878
iter# 11: Fitness:0.29595, Position:[0.5504404  0.27234975], Velocity:10.0
iter# 12: Fitness:0.24363, Position:[0.54719651 0.31907112], Velocity:1.8834931260943146
iter# 13: Fitness:0.07915, Position:[0.72191398 0.52542727], Velocity:7.03313928585155
iter# 14: Fitness:0.07915, Position:[0.72191398 0.52542727], Velocity:5.484664498552223
iter# 15: Fitness:0.07915, Position:[0.72191398 0.52542727], Velocity:9.096613382842277
iter# 16: Fitness:0.07915, Position:[0.72191398 0.52542727], Velocity:2.527683873632189
iter# 17: Fitness:0.07915, Position:[0.72191398 0.52542727], Velocity:9.999999999999996
iter# 18: Fitness:0.03721, Position:[0.8135662 0.656941 ], Velocity:2.999843163580064
iter# 19: Fitness:0.03721, Position:[0.8135662 0.656941 ], Velocity:5.1208026385172944
iter# 20: Fitness:0.02786, Position:[0.83732193 0.69736978], Velocity:6.790082986478646
iter# 21: Fitness:0.02786, Position:[0.83732193 0.69736978], Velocity:2.7543929012850104
iter# 22: Fitness:0.02019, Position:[0.91106971 0.84112806], Velocity:3.1225046592642185
iter# 23: Fitness:0.00019, Position:[0.99241738 0.9860605 ], Velocity:2.0051063327488428
iter# 24: Fitness:0.00019, Position:[0.99241738 0.9860605 ], Velocity:3.722867174022369
iter# 25: Fitness:0.00019, Position:[0.99241738 0.9860605 ], Velocity:3.1233080553730637
iter# 26: Fitness:0.00019, Position:[0.99241738 0.9860605 ], Velocity:2.6494265907173773
iter# 27: Fitness:0.00019, Position:[0.99241738 0.9860605 ], Velocity:0.9614011198952035
iter# 28: Fitness:0.00019, Position:[0.99241738 0.9860605 ], Velocity:3.702646568715364
iter# 29: Fitness:0.00019, Position:[0.99241738 0.9860605 ], Velocity:2.578325220042802
iter# 30: Fitness:0.00019, Position:[0.99241738 0.9860605 ], Velocity:3.989269166529044
iter# 31: Fitness:0.00019, Position:[0.99241738 0.9860605 ], Velocity:5.954613476375235
iter# 32: Fitness:0.00019, Position:[0.99241738 0.9860605 ], Velocity:1.6954564679827715
iter# 33: Fitness:0.00019, Position:[0.99241738 0.9860605 ], Velocity:4.166101550123726
iter# 34: Fitness:0.00019, Position:[0.99241738 0.9860605 ], Velocity:2.1931433919745014
iter# 35: Fitness:0.00019, Position:[0.99241738 0.9860605 ], Velocity:6.095094213616975
iter# 36: Fitness:0.00019, Position:[0.99241738 0.9860605 ], Velocity:4.694796173084044
iter# 37: Fitness:0.00019, Position:[0.99241738 0.9860605 ], Velocity:2.543309824896172
iter# 38: Fitness:0.00019, Position:[0.99241738 0.9860605 ], Velocity:5.36774960353159
iter# 39: Fitness:0.00019, Position:[0.99241738 0.9860605 ], Velocity:4.814722878924161
iter# 40: Fitness:0.00019, Position:[0.99241738 0.9860605 ], Velocity:1.838512354036487
iter# 41: Fitness:0.00019, Position:[0.99241738 0.9860605 ], Velocity:2.854031450558876
iter# 42: Fitness:0.00012, Position:[0.99356968 0.98804631], Velocity:3.6104007791313792
iter# 43: Fitness:0.00012, Position:[0.99356968 0.98804631], Velocity:2.8587398635355212
iter# 44: Fitness:0.00012, Position:[0.99356968 0.98804631], Velocity:1.446758834147228
iter# 45: Fitness:0.00011, Position:[1.01040923 1.02115711], Velocity:1.2297625826950178
iter# 46: Fitness:0.00005, Position:[1.00709331 1.01442942], Velocity:0.5504222152652438
iter# 47: Fitness:0.00005, Position:[1.00709331 1.01442942], Velocity:1.5111857394955028
iter# 48: Fitness:0.00000, Position:[1.00024019 1.00040484], Velocity:1.4119653916327433
iter# 49: Fitness:0.00000, Position:[1.00024019 1.00040484], Velocity:0.3539617318797534
global best: 6.292610779035639e-07 , global best position: [1.00024019 1.00040484]
```

Conclusion:

Thus I successfully implemented particle swarm optimization and minimized the Rosenbrock function

Source code and Output

Code

```
from __future__ import division
import random
import math


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=[]
        self.velocity_i=[]
        self.pos_best_i=[]
        self.err_best_i=-1
        self.err_i=-1


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


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


        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
```

```
    def update_velocity(self,pos_best_g):
        w=0.5
        c1=2
        c2=2

        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

    def update_position(self,bounds):
        for i in range(0,num_dimensions):
            self.position_i[i]=self.position_i[i]+self.velocity_i[i]

            if self.position_i[i]>bounds[i][1]:
                self.position_i[i]=bounds[i][1]

            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
        pos_best_g=[]
```

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

    i=0
    while i < maxiter:
        for j in range(0,num_particles):
            swarm[j].evaluate(costFunc)

            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)

        for j in range(0,num_particles):
            swarm[j].update_velocity(pos_best_g)
            swarm[j].update_position(bounds)
        i+=1
    print ('FINAL:')
    print (pos_best_g)
    print (err_best_g)


initial=[5,5]
bounds=[(-10,10),(-10,10)]
PSO(func1,initial,bounds,num_particles=15,maxiter=30)
```

Output—

FINAL:

[-0.00016359943983711676, -0.002002243512187047]

4.035743858810139e-06