PBA3

September 20, 2021

I only managed to do the low level PSO due to unforseen circumstances and time constraints. Hopefully the low level PSO presented still shows sufficient understanding of the work to be marked partially.

[1]:

```
import numpy as np
import random
```

[2]:

```
def objective(x,y):
    ans = -(y+47)*np.sin(np.sqrt(np.abs(y+(x/2)+47)))-x*np.sin(np.sqrt(np.
 ↪abs(x-(y+47))))
    return ans

def fitness(particle): #the fitness score is just the negative of the objective␣
 ↪function (lower the obj funct the better)
    x,y = particle.coords()
    return (-objective(x,y))


class particle: #particle class containing postition and the ability to update␣
 ↪or fetch its coordinates

    def __init__(self,x,y):
        self.x = x
        self.y = y

    def update(self,x,y):
        self.x = x
        self.y = y

    def coords(self):
        return(self.x,self.y)

    def setbest(self,best):
        self.best = best

    def returnbest(self):
        return best
```

[3]:

```
print(objective(512,404.2319)) #test that the obj func correctly
```

```
-959.6406627106155
```

[4]:

```
a = particle(512,404.2319)
b = particle(500,400)
print(fitness(a))
print(fitness(b)) #testing that the given coords in the question are a good␣
 ↪fitness score against a nearby area
#therefore the coords are at least a local minima and trust that it is the␣
 ↪global minima within the bounds
```

```
959.6406627106155
846.5692073973108
```

[5]:

```
class PSO:

    def __init__(self,particles,fitness,itermax,w,c1,c2,vel):
        #Initilise and store all parameters such as w,c,r for use later
        self.particles =  particles
        self.fitness = fitness
        self.itermax = itermax
        self.w = w
        self.c1 = c1
        self.c2 = c2
        self.vel = vel

        self.best = []
        self.best_values = []
        for i in range(len(self.particles)):
            self.best.append(self.particles[i].coords())
            self.best_values.append(self.fitness(self.particles[i]))

        self.globalbest = self.best[0]
        self.globalbest_value = self.best_values[0]
        self.iter = 0
        self.run = True
        print("starting best:",self.globalbest)
        self.best_memory = self.globalbest_value
        self.best_tally=0
        self.update()

    def step(self):
        if self.iter>0: #dont move on step 0.
            self.move() #Updates all particles positions (see method below)
            self.update() #Updates the global and personal bests of the␣
 ↪particles (if necessary)
```

```
            if(self.globalbest_value>self.best_memory): #Testing if any␣
↪improvement has been made
                self.best_memory = self.globalbest_value
                self.best_tally = 0
            else:
                self.best_tally+=1
        self.iter +=1

        if(self.iter<self.itermax): #If within itermax keep going
            self.run = True
        elif(self.iter>=self.itermax): #stop conditions
            self.run = False
            print("run ended due to iter")
        if(self.best_tally>50):
            self.run = False
            print("run ended due to no new improvement")
        return self.run

    def move(self):
        r1_list = np.random.random(len(self.particles))#list of len self.length␣
↪of rand num 0-1
        r2_list = np.random.random(len(self.particles))
        for i,particle in enumerate(self.particles):

            x,y = particle.coords()
            #Below we calculate the new x and y velocity components according␣
↪to the equations using w,c,r etc
            new_velx = self.w*self.vel[i][0] + self.c1*r1_list[i]*(self.
↪best[i][0]-x)
            new_velx += self.c2*r2_list[i]*(self.globalbest[0]-x)
            new_vely = self.w*self.vel[i][1] + self.c1*r1_list[i]*(self.
↪best[i][0]-y)
            new_vely += self.c2*r2_list[i]*(self.globalbest[1]-y)
            self.vel[i] = (new_velx,new_vely)
            newx = x+new_velx
            newy = y+new_vely

            if(newx>512): #Keeping wtihin the bounds
                newx = 512
            if(newx<-512):
                newx = -512

            if(newy>512):
                newy = 512
            if(newy<-512):
                newy = -512
            particle.update(newx,newy) #update the particle instance with its new location

    def update(self):
        for i, (particle,bestvalue) in enumerate(zip(self.particles,self.best_values)):
            #step through both the particles and the list of best fitness scores
            currentfit = fitness(particle) #fitness of the current particle under consideration
            if(currentfit>bestvalue): #check against personal best
                bestvalue = currentfit
                self.best[i] = particle.coords()

                if(currentfit>self.globalbest_value): #check against global best
                    self.globalbest_value = currentfit
                    self.globalbest = particle.coords()

    def result(self): #some end messages
        print("iter reached:",self.iter)
        print("coords:",self.globalbest)
        print("fitness",self.globalbest_value)
        return
```

[6]:

```
num_particle = 50 #50 particlces low level pso
particles = []
velocities = []
for i in range(num_particle): #initialise all particles coords and their velocities as random
    randx = random.randrange(-512,512)
    randy = random.randrange(-512,512)
    velx = random.randrange(-20,20)
    vely = random.randrange(-20,20)
    vel = (velx,vely)
    velocities.append(vel)
    particles.append(particle(randx,randy))

pso = PSO(particles,fitness,itermax=20000,w=0.9,c1=0.5,c2=1.5,vel=velocities) #initialise the PSO class
run = True
while(run): #run PSO
    run = pso.step()
pso.result() #Some stats
```

```
starting best: (74, 281)
run ended due to no new improvement
iter reached: 68
coords: (512, 404.21982984856396)
```

```
fitness 959.6404997045429
```