Yash Bhandari (BECOC303)

Assignment No 4

Code:-

```
def cal_pop_fitness(equation_inputs, pop):
    fitness = numpy.sum(pop*equation_inputs, axis=1)
    return fitness


def select_mating_pool(pop, fitness, num_parents):
    parents = numpy.empty((num_parents, pop.shape[1]))
    for parent_num in range(num_parents):
        max_fitness_idx = numpy.where(fitness == numpy.max(fitness))
        max_fitness_idx = max_fitness_idx[0][0]
        parents[parent_num, :] = pop[max_fitness_idx, :]
        fitness[max_fitness_idx] = -99999999999
    return parents


def crossover(parents, offspring_size):
    offspring = numpy.empty(offspring_size)
     crossover_point = numpy.uint8(offspring_size[1]/2)

    for k in range(offspring_size[0]):
        parent1_idx = k%parents.shape[0]
        parent2_idx = (k+1)%parents.shape[0]
        offspring[k, 0:crossover_point] = parents[parent1_idx, 0:crossover_point]
        offspring[k, crossover_point:] = parents[parent2_idx, crossover_point:]
    return offspring


def mutation(offspring_crossover):
    # Mutation changes a single gene in each offspring randomly.
    for idx in range(offspring_crossover.shape[0]):
        # The random value to be added to the gene.
```

```
        random_value = numpy.random.uniform(-1.0, 1.0, 1)
        offspring_crossover[idx, 4] = offspring_crossover[idx, 4] + random_value
    return offspring_crossover


import numpy
import geneticalgorithm
equation_inputs = [4,-2,3.5,5,-11,-4.7]
num_weights = 6


sol_per_pop = 8
num_parents_mating = 4
pop_size = (sol_per_pop,num_weights) # The population will have sol_per_pop chromosome where
each chromosome has num_weights genes.
new_population = numpy.random.uniform(low=-4.0, high=4.0, size=pop_size)
print(new_population)


num_generations = 5
for generation in range(num_generations):
    print("Generation",generation,": - ")
    # Measing the fitness of each chromosome in the population.
    fitness = cal_pop_fitness(equation_inputs, new_population)


    parents = select_mating_pool(new_population, fitness,
                                  num_parents_mating)
    offspring_crossover = crossover(parents,
                                   offspring_size=(pop_size[0]-parents.shape[0], num_weights))


    offspring_mutation = mutation(offspring_crossover)


    new_population[0:parents.shape[0], :] = parents
    new_population[parents.shape[0]:, :] = offspring_mutation
```

Yash Bhandari (BECOC303)

```
    print("Best result : ", numpy.max(numpy.sum(new_population*equation_inputs, axis=1)))


fitness = cal_pop_fitness(equation_inputs, new_population)
best_match_idx = numpy.where(fitness == numpy.max(fitness))


print("Best solution : ", new_population[best_match_idx, :])
print("Best solution fitness : ", fitness[best_match_idx])
```

Output:-

```
Python 3.10.3 (tags/v3.10.3:a342a49, Mar 16 2022, 13:07:40) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==== RESTART: D:/Yash/BE Lab Assignments/SCOA/Assign 4 Crossover Mutation.py ===
[[-2.9292628   1.64894789  0.09994966  3.45525157 -3.87002944 -2.89952331]
 [-2.38020153  3.75291457 -3.95986302 -3.67097681  3.84435358 -2.26182761]
 [-1.29335911 -0.79143925  1.07073492 -1.70247781 -3.12507316  0.92344787]
 [-0.4442401  -2.03183361  1.92513633  0.74448066  2.79980692 -1.84200984]
 [ 1.93671389  2.69737413  0.73211119  1.99663392  3.33581886  0.67562322]
 [-2.28503955 -2.72711911 -2.14622861  0.67541237  2.44541252  1.94611259]
 [ 2.07517991 -0.12172547 -3.7930201   2.11236009 -3.38194312 -3.12044544]
 [-1.97163912  1.00326532 -1.48453772  2.19585008  0.99500309 -2.23591888]]
Generation 0 : -
Best result :  60.38557874181902
Generation 1 : -
Best result :  70.25055369449645
Generation 2 : -
Best result :  72.99719466813931
Generation 3 : -
Best result :  72.99719466813931
Generation 4 : -
Best result :  81.65894208807072
Best solution :  [[[-1.97163912  1.00326532 -1.48453772  3.45525157 -5.98580853
   -2.89952331]]]
Best solution fitness :  [81.65894209]
>>>
```

Yash Bhandari (BECOC303)

Assignment No 5

Code:-

```
import random
import math    # cos() for Rastrigin
import copy    # array-copying convenience
import sys     # max float

def show_vector(vector):
  for i in range(len(vector)):
    if i % 8 == 0: # 8 columns
      print("\n", end="")
    if vector[i] >= 0.0:
      print(' ', end="")
    print("%.4f" % vector[i], end="") # 4 decimals
    print(" ", end="")
  print("\n")
def error(position):
  err = 0.0
  for i in range(len(position)):
    xi = position[i]
    err += (xi * xi) - (10 * math.cos(2 * math.pi * xi)) + 10
  return err

class Particle:
  def __init__(self, dim, minx, maxx, seed):
    self.rnd = random.Random(seed)
    self.position = [0.0 for i in range(dim)]
    self.velocity = [0.0 for i in range(dim)]
    self.best_part_pos = [0.0 for i in range(dim)]
    for i in range(dim):
```

```
      self.position[i] = ((maxx - minx) *
        self.rnd.random() + minx)
      self.velocity[i] = ((maxx - minx) *
        self.rnd.random() + minx)

    self.error = error(self.position) # curr error
    self.best_part_pos = copy.copy(self.position)
    self.best_part_err = self.error # best error

def Solve(max_epochs, n, dim, minx, maxx):
  rnd = random.Random(0)
  swarm = [Particle(dim, minx, maxx, i) for i in range(n)]

  best_swarm_pos = [0.0 for i in range(dim)] # not necess.
  best_swarm_err = sys.float_info.max # swarm best
  for i in range(n): # check each particle
    if swarm[i].error < best_swarm_err:
      best_swarm_err = swarm[i].error
      best_swarm_pos = copy.copy(swarm[i].position)

  epoch = 0
  w = 0.729    # inertia
  c1 = 1.49445 # cognitive (particle)
  c2 = 1.49445 # social (swarm)

  while epoch < max_epochs:
    if epoch % 10 == 0 and epoch > 1:
      print("Epoch = " + str(epoch) +
        " best error = %.3f" % best_swarm_err)
    for i in range(n): # process each particle
```

```
      # compute new velocity of curr particle
      for k in range(dim):
        r1 = rnd.random()    # randomizations
        r2 = rnd.random()
        swarm[i].velocity[k] = ( (w * swarm[i].velocity[k]) +
          (c1 * r1 * (swarm[i].best_part_pos[k] -
          swarm[i].position[k])) +
          (c2 * r2 * (best_swarm_pos[k] -
          swarm[i].position[k])) )
        if swarm[i].velocity[k] < minx:
          swarm[i].velocity[k] = minx
        elif swarm[i].velocity[k] > maxx:
          swarm[i].velocity[k] = maxx
      for k in range(dim):
        swarm[i].position[k] += swarm[i].velocity[k]
      swarm[i].error = error(swarm[i].position)
      if swarm[i].error < swarm[i].best_part_err:
        swarm[i].best_part_err = swarm[i].error
        swarm[i].best_part_pos = copy.copy(swarm[i].position)

      if swarm[i].error < best_swarm_err:
        best_swarm_err = swarm[i].error
        best_swarm_pos = copy.copy(swarm[i].position)
    epoch += 1
  return best_swarm_pos

print("\nBegin particle swarm optimization using Python demo\n")
dim = 3
print("\nGoal is to solve Rastrigin's function in " + str(dim) + " variables")
print("\nFunction has known min = 0.0 at (", end="")
```

```
for i in range(dim-1):
  print("0, ", end="")
print("0)")
num_particles = 50
max_epochs = 100
print("\nSetting num_particles = " + str(num_particles))
print("Setting max_epochs    = " + str(max_epochs))
print("\nStarting PSO algorithm\n")
best_position = Solve(max_epochs, num_particles,
 dim, -10.0, 10.0)
print("\nPSO completed\n")
print("\nBest solution found:")
show_vector(best_position)
err = error(best_position)
print("Error of best solution = %.6f" % err)
print("\nEnd particle swarm demo\n")
```

Output:-

```
Python 3.10.3 (tags/v3.10.3:a342a49, Mar 16 2022, 13:07:40) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>
==== RESTART: D:/Yash/BE Lab Assignments/SCOA/Assign 5 Swarm Optimization.py ===

Begin particle swarm optimization using Python demo


Goal is to solve Rastrigin's function in 3 variables

Function has known min = 0.0 at (0, 0, 0)

Setting num_particles = 50
Setting max_epochs    = 100

Starting PSO algorithm

Epoch = 10 best error = 8.463
Epoch = 20 best error = 4.792
Epoch = 30 best error = 2.223
Epoch = 40 best error = 0.251
Epoch = 50 best error = 0.251
Epoch = 60 best error = 0.061
Epoch = 70 best error = 0.007
Epoch = 80 best error = 0.005
Epoch = 90 best error = 0.000

PSO completed


Best solution found:

 0.0006  0.0000  0.0006

Error of best solution = 0.000151

End particle swarm demo

>
```