Lince Rumainum

DSA 5113

Advanced Analytics & Metaheuristics

Group 18 - HW 6

Problem 1 – Genetic Algorithm Implementation

Part a – Finalize the code

i. Create code to generate chromosomes in the initial population

```
#create an continuous valued chromosome
def createChromosome(n, d, lBnd, uBnd):
    # n is the increment of population size (needed for seed to create randomness of the choromosome)
    # d is the number of dimension

    #this code as-is expects chromosomes to be stored as a list, e.g., x = []
    x = [] #initialize x as an empty list

    #write code to generate chromosomes, most likely want this to be randomly generated
    #pick a point for each dimension
    for i in range(0,d):
        #create seed for random number
        seed = (i+n+d+populationSize)*100
        chromosome = Random (seed)
        #pick a random point between lower and upper bound
        x.append(chromosome.uniform(lBnd,uBnd))

    return x

#create initial population
def initializePopulation(): #n is size of population; d is dimensions of chromosome
    population = []
    populationFitness = []

    for i in range(0,populationSize):
        population.append(createChromosome(i, dimensions,lowerBound, upperBound))
        populationFitness.append(evaluate(population[i]))

    tempZip = zip(population, populationFitness)
    popVals = sorted(tempZip, key=lambda tempZip: tempZip[1])

    #the return object is a sorted list of tuples:
    #the first element of the tuple is the chromosome; the second element is the fitness value
    #for example:  popVals[0] is represents the best individual in the population
    #popVals[0] for a 2D problem might be  ([-70.2, 426.1], 483.3)  -- chromosome is the list [-70.2, 426.1] and the fitness is 

    return popVals
```

ii. Create code to mutate chromosomes

```
#function to mutate solutions
# a+5 is the increment of population pair index (needed for seed to create random probability)
def mutate(a, x):
    #new random seed for mutation
    mutationSeed = (a+10+dimensions+populationSize)*100
    rndmMutation = Random (mutationSeed)

    #current mutation probability
    pRate = rndmMutation.random()

    #do mutation
    if pRate < mutationRate:
        #10% of dimensions
        #for example: d = 10, mutate 1 random dimension, d = 1000, mutate 100 random dimensions
        numOfMutation = math.floor(0.10 * dimensions)
        #print("mutation")

        for i in range(0,numOfMutation):
            #select random
            selectionSeed = (i+seed+mutationSeed)*10
            mutationSelection = Random(selectionSeed)
            #pick index to do mutation
            indexToMutate = mutationSelection.randint(0, dimensions-1)

            #do mutation
            if x[indexToMutate] < -250: #shift between 0 to +750
                x[indexToMutate] += mutationSelection.uniform(0, 750)
            elif x[indexToMutate] < 0: #shift between 0 to +500
                x[indexToMutate] += mutationSelection.uniform(0, 500)
            elif x[indexToMutate] < 250: #shift between 0 to -500
                x[indexToMutate] -= mutationSelection.uniform(0, 500)
            else: #shift between 0 to -750
                x[indexToMutate] -= mutationSelection.uniform(0, 750)

    return x
```

iii. Implement logic for crossover rate and mutation rate

The implementation logic for the mutation rate can be seen on ***part a-ii*** and the implementation logic for crossover rate can be see below:

```python
#implement a linear crossover
# a+10 is the increment of population pair index (needed for seed to create random probability)
def crossover(a, x1,x2):
    #new random seed for crossover
    crossoverSeed = (a+10+dimensions+populationSize)*100
    rndmCrossover = Random (crossoverSeed)

    #current crossover probability
    pRate = rndmCrossover.random()

    if pRate < crossOverRate:    #do crossover
        #print("crossover")
        d = len(x1) #dimensions of solution

        #choose crossover point

        #we will choose the smaller of the two [0:crossOverPt] and [crossOverPt:d] to be unchanged
        #the other portion be linear combo of the parents

        crossOverPt = myPRNG.randint(1,d-1) #notice I choose the crossover point so that at least 1 element of parent is copied

        beta = myPRNG.random()  #random number between 0 and 1

        #note: using numpy allows us to treat the lists as vectors
        #here we create the linear combination of the soltuions
        new1 = list(np.array(x1) - beta*(np.array(x1)-np.array(x2)))
        new2 = list(np.array(x2) + beta*(np.array(x1)-np.array(x2)))

        #the crossover is then performed between the original solutions "x1" and "x2" and the "new1" and "new2" solutions
        if crossOverPt<d/2:
            offspring1 = x1[0:crossOverPt] + new1[crossOverPt:d]  #note the "+" operator concatenates lists
            offspring2 = x2[0:crossOverPt] + new2[crossOverPt:d]
        else:
            offspring1 = new1[0:crossOverPt] + x1[crossOverPt:d]
            offspring2 = new2[0:crossOverPt] + x2[crossOverPt:d]
    else: # no crossover
        #print("no crossover")
        #keep the same x1 and x2
        offspring1 = x1
        offspring2 = x2
    return offspring1, offspring2  #two offspring are returned
```

iv. Implement some type of elitism in the insertion step

```python
#insertion step
def insert(pop,kids):
    #replacing the previous generation completely...  probably a bad idea -- please implement some type of elitism
    tempKids = [] #initialize list of temporary kids list

    #combined list of population and kids
    combinedList = pop + kids

    #re-sort the combined list into temporary kids list
    tempKids = sorted(combinedList, key=lambda combinedList: combinedList[1])

    kids = [] #reset kids list
    #pick the best solutions for the new population
    for i in range(0,populationSize):
        kids.append(tempKids[i])

    return kids
```

v. Complete/modify any other logic as you see fit

```python
dimensions = 2    #set dimensions for Schwefel Function search space (should either be 2 or 200 for HM #5)

populationSize = 10000 #size of GA population
Generations = 10   #number of GA generations

crossOverRate = 0.70  #currently not used in the implementation; neeeds to be used.
mutationRate = 0.10   #currently not used in the implementation; neeeds to be used.
```

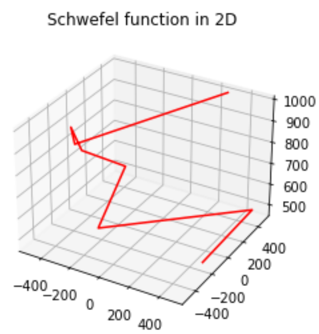
For 2 dimensions, using population size of 10,000 and 10 generations with crossover rate of 0.70 and mutation rate of 0.10, the best solution reached f(x) = 2.5460e-05 at x = (420.9687, 420.9689), which is close to the optimal solution.

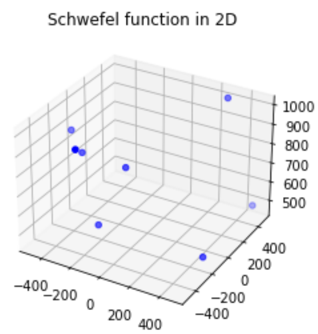
Part b

The empirical decision for population size, stopping criterion, cross over rate, mutation rate, selection, and elitism are based on how close to the global optimum it would reach.
The higher the population size the better chance that it does. The stopping criterion is the number of generation since the more generations, the closer to optimum solution it becomes. The cross over rate needs to be higher than the mutation rate but both higher rates does always mean better solution. The selection and elitism are based on the rank it is in; better solutions are picked for new population.

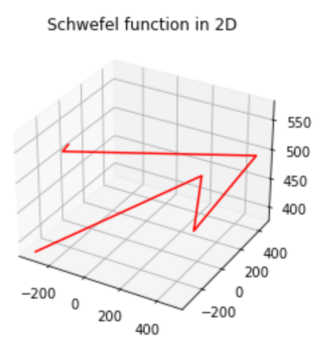
Part c

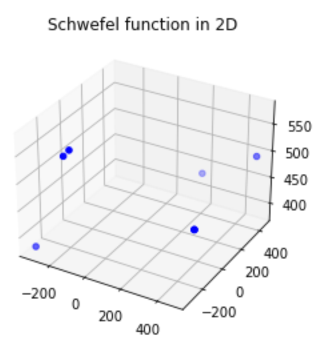
i. **Create a small population of 8 chromosomes and depict their locations on a graph for the initial random set:**





Create a small population of 8 chromosomes and depict their locations on a graph for the first generation:





ii. **Solve the problem as best as possible and provide information on the quality of the best solution and the performance of the approach overall.**

For the 2D, best optimize solution I could find is 2.5460e-05, which is zero (the global optimum solution). Details of results are shown below:

```
Generation:  1
(2.0367231676913207, 533.9652625956385, 21795.38319812716)
Generation:  2
(0.0981894968749657, 372.3774241775205, 13594.708267273802)
Generation:  3
(0.0981894968749657, 247.29251305772294, 9947.179171252366)
Generation:  4
(0.0981894968749657, 127.94542843268218, 4970.48715105363)
Generation:  5
(0.0001440245125650108, 35.151515301941856, 591.0759500875487)
Generation:  6
(0.0001440245125650108, 7.807969677306758, 20.771820110088466)
Generation:  7
(0.0001440245125650108, 2.290541802900547, 2.257419072835992)
Generation:  8
(2.5632751658122288e-05, 0.40370222088064184, 0.1060491727409461)
Generation:  9
(2.5460164124524454e-05, 0.06697564007579192, 0.0013071742225892074)
Generation:  10
(2.5460164124524454e-05, 0.01814986442104539, 0.00013054035640284255)

--------SOLUTIONS PROBLEM 1--------
Number of dimension: 2
Population size:  10000
Number of generations:  10

Crossover Rate: 0.70
Mutation Rate: 0.10
Using 100% top elitism every generation
Best Solution In Population:
([420.96873595860126, 420.96894577427827], 2.5460164124524454e-05)
```

Part d

For the 200D, best optimize solution I could find is 45,050.6153. Details of results are shown below:

```
summary fitness:
(45050.615319020515, 45112.138185556934, 90.76609425503237)

--------SOLUTIONS PROBLEM 1--------
Number of dimension: 200
Population size:  50000
Number of generations:  100

Crossover Rate: 0.75
Mutation Rate: 0.15
Using 100% top elitism every generation
Best Solution In Population:
([441.8631851283465, 436.40637682666016, 150.85485975784897, 388.768115452
8448, 178.70122164851202, -319.4793577325116, -299.0117372069598, 415.5398
860632186, -282.8109575577295, -339.2059698212887, -267.01079931781675, 41
1.41324905990916, -2.453643961484202, 243.5613662589489, -379.789817445115
66, -347.23326092668833, -274.7435018850818, -298.0105488122612, 437.58561
40860624, -171.75055066313809, -330.46736158306226, -306.53993614625887, 4
33.75212750955836, -291.49613756251, -145.62922953013867, -298.36047329022
2, 373.82630423482453, 225.3242278466453, -290.24730387788225, 170.0382930
3614205, 46.57118640511983, -42.81566661235613, 53.997419875891765, -99.01
918711519204, 412.6631008479825, -130.4781563340097, -301.1335460554521, -
157.97876372473937, -295.2933297661823, 384.159281285215, -25.276382313215
79, -109.3832094944531, 218.0657599052961, -328.7578393960174, 228.1959124
0292579, 394.6994959612565, 65.20238118423413, 426.01242647947953, 200.860
6713223952, -335.4413724725402, -120.30392735788561, 94.5999692671944, 44.
640408327405964, 230.49507137900991, -100.90364128286859, 437.278213210797
```

```
, -323.24623668365894, 439.27629146080136, -86.9770478450339, 207.19344804
419555, 401.2539804415081, 431.2009070975511, -81.20869789567584, 376.7488
471548551, 193.24194859406143, 116.48052452224391, -261.2659585394703, -28
6.3194502929093, -141.3386614017101, 243.85502340204204, 218.9305503787854
3, 422.8260467799353, -279.7541213408971, -282.26429688446467, -119.004726
02149185, 425.2205311979324, 2.1976078915900534, -280.7712029889866, 419.5
362756908224, 59.91436098625972, 429.00664084012345, -337.48989145007243,
-28.40148407682541, 428.61117951458897, -315.977554883902, 424.85054543447
68, 27.921350593784855, 365.9392335124395, 415.97724561472376, -113.236480
55704784, -150.0401404330302, -279.8646570482169, 396.5319037288391, -125.
8933906259486, -35.22367706938021, 206.02212437909878, 371.16954310144314
, 161.2579353284347, -139.692156348678, 55.47037648455929, -325.4107527923
0614, -37.19861690672001, -291.84754545269493, -279.22559413674804, 425.94
127040201903, -273.7309604457903, -275.3498458629729, -261.2184658925739,
54.87772280900782, -161.15752098861165, 62.31435183261919, -273.0697892418
1607, 206.92025586682703, 139.066164364424, -261.1778585294885, -153.27049
852250005, -103.82111258745509, -282.548371973924, -288.3479624583846, 85.
48335648103426, -120.33639343554599, 405.2890932051655, -99.55619876357154
, 85.48857201333755, -320.9014057872542, -304.7469108849207, 390.281847764
73864, 431.3197631315747, 186.4903675789032, -294.3769350891543, 404.32102
93732518, 225.1619084611O618, 51.238275206417995, 392.46587024797566, 401.
3527773360683, 40.4330344109053, 179.88497604887704, 225.1996932695683, -3
08.6553118O979815, -72.43691220177226, 359.76827959704127, -59.44791956484
863, 238.91153401495535, 398.42810639937034, -308.00279105559315, 175.7080
0751756923, 426.49539598831814, -331.58548379855813, -134.38635647621652,
424.0387801498827, -302.3121511496489, 403.6531589054008, 206.049701108652
6, -317.5795106271222, -122.40864204878207, -279.74935953088806, 203.16442
73160503, -33.489086772626216, 407.64353177519604, 387.3272194116185, -301
.2516929541853, -33.555462553530255, 207.77659974608636, -84.871440929402
24, -281.02840810040783, 229.98025644149905, 392.5104566947086, 195.003189
81276584, 429.27392900799964, 429.9774560852206, 404.1045745529145, 217.39
961241260713, -316.3912917944206, -330.7207222234652, 433.07594632203404,
-18.468221881806866, 221.57861481416018, 425.8078777503574, -283.954011283
63517, -127.7841677637974, -101.35813036829069, -159.61544310894894, -279.
5281399769709, -321.0614440662145, 405.35820336370017, 175.2912688978353,
-293.22699011624854, 180.53596730844043, -302.7398331510039, 401.868297869
69923, -92.3531712722712, 378.2979180106528, 417.84195532152563, -305.4229
0408020335, 409.8685256719028, 419.7460721429373, -148.56946116570558, -33
8.85204422001254, -331.1532854852098, -288.13751029199733], 45050.61531902
0515)
```

Problem 2 – Particle Swarm Optimization

Part a – Finalize the code

Limit particle position between the lower bound of -500 and upper bound of 500.
Limit particle velocity to between zero and the distance between the current position to the optimal solution of 420.9687.
Stopping when best value is less than 0.0001 or the maximum number of iteration is reached

Part b – Swarm of size 5 and solve the 2D Schwefel problem

POSITION – 1

Particle #	x_1	x_2	Value
1	-21.4416	-203.9708	1018.4361
2	-334.3105	-117.0672	543.5962
3	-243.3289	12.3857	868.9945
4	78.8758	-41.1791	802.6695
5	211.8135	334.9881	818.4738

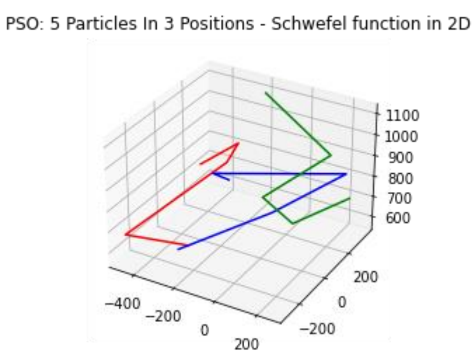
POSITION – 2

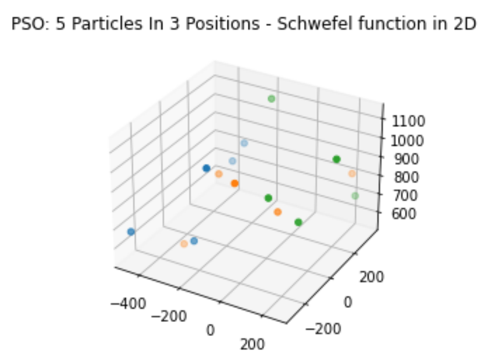
Particle #	x_1	x_2	Value
1	45.7946	-60.4775	876.9543
2	-160.8897	288.4684	1132.9413
3	257.3743	116.3226	1036.3252
4	165.8987	-21.7752	765.0054
5	220.5952	353.3591	689.6994

POSITION – 3

Particle #	x_1	x_2	Value
1	102.8124	50.0681	869.7001
2	-27.2883	388.3383	520.6962
3	380.6808	193.8196	414.7700
4	230.7843	-3.1860	727.1330
5	229.0081	367.8049	594.7062

LINE AND SCATTER PLOTS OF THE 5 PARTICLES IN 3 POSITIONS





NOTE: Red/Orange – Position 1, Blue – Position 2, and Green – Position 3

Part d

For 2D:

```
Best Position:
[420.9687, 420.94978514178 27]
Best Value:
7.082155866555695e-05
```

For 200D:

```
Best Position:
```

```
[420.96869999999984, 420.9686999999999, 420.9687, 420.96869999999996, 420.
9686999999994, 420.9686999999999, 420.9687, 420.9686999999994, 420.9686999
999999, 420.9687, 420.96869999999996, 420.9686995145817, 420.968699999999
6, 420.96869999999996, 420.9687, 420.9686999999999, 420.9687, 420.9687, 42
0.9686999999999, 420.9687, 420.9687, 420.9687, 420.8013137304989, 420.9687
, 420.96869999999996, 420.96869999999996, 420.96869999999996, 420.9687, 42
0.96869999999956, 420.9687, 420.96869998621275, 420.9687, 420.968690109982
16, 420.9687, 420.9687, 420.9687, 420.9687, 420.96869999999996, 420.968699
99999996, 420.9687, 420.9687, 420.9687, 420.9687, 420.9686999999998, 420.9
687, 420.9687, 420.96869999999996, 420.9686999998729, 420.9687, 420.9687,
420.9686999999993, 420.9687, 420.96869999999996, 420.9687, 420.96869999999
996, 420.96869999999996, 420.96869999999996, 420.9687, 420.96869999999996,
420.9687, 420.9686999999999, 420.9686999999999, 420.9686999999999, 420.968
69999999996, 420.9687, 420.96869999999996, 420.96869999999996, 420.9687, 4
20.9687, 420.96869999999967, 420.9686999999996, 420.9686992592476, 420.968
6999999997, 420.96869999999996, 420.9687, 420.9687, 420.96869999999996, 42
0.9687, 420.9687, 420.9687, 420.9687, 420.96869999999944, 420.968699999999
96, 420.9687, 420.9687, 420.96869999999996, 420.96869999999996, 420.9687,
420.9687, 420.96869999999996, 420.9686999999996, 420.9687, 420.96869999999
933, 420.9687, 420.9687, 420.96869999999996, 420.9687, 420.9687, 420.9687,
420.96869999999996, 420.9687, 420.9687, 420.9687, 420.9687, 420.9687, 420.
64489787454846, 420.96869999999996, 420.9686999999996, 420.96869999999984,
420.9687, 420.96869999999996, 420.9687, 420.96869999999996, 420.9686999999
9996, 420.9687, 420.9687, 420.9687, 420.9687, 420.96869999999996, 420.9687
, 420.9686999999996, 420.9687, 420.9687, 420.96869999999984, 420.968699999
9999, 420.96869999999996, 420.9687, 420.9687, 420.9687, 420.9687, 420.9686
999999999, 420.9687, 420.9686999999997, 420.96869999999984, 420.9687, 420.
96869999999967, 420.96869999999996, 420.9687, 420.9686999999999, 420.96869
99999999, 420.9687, 420.9687, 420.9687, 420.96869999999996, 420.9687, 420.
9687, 420.9686999999999, 420.96869999999984, 420.9687, 420.96869999999996,
420.9687, 420.9687, 420.9687, 420.96869999999996, 420.9687, 420.9687, 420.
96869999999984, 420.9687, 420.9687, 420.9687, 420.9686999999993, 420.9687,
420.96869999999996, 420.9687, 420.96869999999996, 420.96869999999956, 420.
9687, 420.96869046954873, 420.96869999999996, 420.9687000000001, 420.96869
999999984, 420.96869999999996, 420.9687, 420.96869999999996, 420.968699999
99996, 420.9687, 420.9687, 420.9687, 420.9687, 420.9687, 420.9687, 420.968
7, 420.96869999999996, 420.9687, 420.9687, 420.9687, 420.96869999999996, 4
20.96869999999996, 420.9686999999999, 420.9687, 420.9687, 420.968699999997
3, 420.16755434302644, 420.9687, 420.96869999999996, 420.9687, 420.9687, 4
20.96869999999984, 420.96870000000007, 420.9686999999997]

Best Value:
0.10028557426994666
```

Code files:
Group18_HW6_p1c.ipynb, Group18_HW6_p1d.ipynb, Group18_HW6_p2.ipynb.