EXPERIMENT NO . 5

Genetic Algorithm

i) Without Using Inbuilt Library

Code:-

```
from random import randint
def selection(li):
    dec = list(map(lambda x : int(x, 2), li))
    fit = list(map(lambda x : x*x*x, dec))
    s = sum(fit)
    prob = list(map(lambda x : round(x/s, 3), fit))
    avg = s/n
    exe = list(map(lambda x : round(x/avg, 3), fit))
    ac = list(map(lambda x : round(x), exe))
    return dec, fit, prob, exe, ac
def pp(li, ac, n):
    co = []
    temp = []
    index = []
    for i in range(n):
        if ac[i] == 1:
            co.append(li[i])
        elif ac[i] >= 2:
            for j in range(ac[i] - 1):
                temp.append(li[i])
            co.append(li[i])
        elif ac[i] == 0 and len(temp) != 0:
            co.append(temp[0])
            temp.pop(0)
        elif ac[i] == 0 and len(temp) == 0:
            index.append(i)
    if len(index) != 0 and len(temp) != 0:
        for i in index:
            co.insert(i, temp[0])
            temp.pop(0)
    elif len(index) != 0 and len(temp) == 0:
        co.insert(i, li[i])
    return co
def cr(x):
    s = 0
    for i in x:
        if i == '1':
            s = s + 1
    return s
def crossing(li, n):
    crossed = []
    for i in range(0, n, 2):
        temp1 = li[i]
```

```
        j = i + 1
        temp2 = li[j]
        crosspoint = cr(temp1)
        print("The crosspoint for pair " + str(i) + " is " + str(crosspoint))
        temp3 = temp1[crosspoint: ]
        temp4 = temp2[crosspoint: ]
        temp1 = temp1[0 : crosspoint] + temp4
        temp2 = temp2[0 : crosspoint] + temp3
        crossed.append(temp1)
        crossed.append(temp2)
    return crossed
def mutation(li, n):
    mut = []
    for i in li:
        j = randint(0, n - 1)
        print("For pair " + str(i) + ", the bit that will be changed is "
+str(j))
        if i[j] == '1':
            i = i[0 : j] + '0' +i[j + 1 : ]
            mut.append(i)
        elif i[j] == '0':
            i = i[0 : j] + '1' +i[j + 1 : ]
            mut.append(i)
    return mut
n = int(input("Enter number of samples: "))
sam = []
for i in range(n):
    sam.append(input("Enter gene: "))
m = int(input("Enter number of generations to be computed: "))
crossed = sam.copy()
for i in range(m):
    dec, fit, prob, exe, ac = selection(crossed)
    s = sum(ac)
    if s < n:
        maxi = max(ac)
        k = ac.index(maxi - 1)
        ac[k] += 1
    if s > n:
        maxi = max(ac)
        k = ac.index(maxi)
        ac[k] -= 1
    print("\n---------------------------------------- GENERATION ",i, "--------
----------------------------------")
    print("Initial Population\tX Value\t\tFitness Value\tProbability\tExpected
Count\t\tActual Count")
    for j in range(n):
        print(crossed[j], "\t\t", dec[j], "\t\t", fit[j], "\t",
prob[j],"\t\t", exe[j], "\t\t\t", ac[j])
```

```python
    co = pp(crossed, ac, n)
    print("\nSelected Genes for Crossover - \n", co)
    crossed = crossing(co, n)
    print("\nCrossover - \n", crossed)
    crossed = mutation(crossed, n)
    print("\nMutated - \n", crossed)
print("\nGENERATION ", (m + 1), " - ", crossed)
```

Output:-

```
PS D:\AI pracs> python genetic.py
Enter number of samples: 4
Enter gene: 01010
Enter gene: 11010
Enter gene: 11111
Enter gene: 01010
Enter number of generations to be computed: 2

---------------------------------------- GENERATION  0 ------------------------------------------------
Initial Population      X Value         Fitness Value   Probability     Expected Count          Actual Count
01010           10              1000    0.02            0.081                   0
11010           26              17576   0.356           1.424                   2
11111           31              29791   0.603           2.414                   2
01010           10              1000    0.02            0.081                   0

Selected Genes for Crossover -
 ['11111', '11010', '11111', '11010']
The crosspoint for pair 0 is 5
The crosspoint for pair 2 is 5

Crossover -
 ['11111', '11010', '11111', '11010']
For pair 11111, the bit that will be changed is 3
For pair 11010, the bit that will be changed is 2
For pair 11111, the bit that will be changed is 1
For pair 11010, the bit that will be changed is 1

Mutated -
 ['11101', '11110', '10111', '10010']
```

```
---------------------------------------- GENERATION  1 ------------------------------------------------
Initial Population      X Value         Fitness Value   Probability     Expected Count          Actual Count
11101           29              24389   0.351           1.406                   1
11110           30              27000   0.389           1.556                   2
10111           23              12167   0.175           0.701                   1
10010           18              5832    0.084           0.336                   0

Selected Genes for Crossover -
 ['11101', '11110', '10111', '11110']
The crosspoint for pair 0 is 4
The crosspoint for pair 2 is 4

Crossover -
 ['11100', '11111', '10110', '11111']
For pair 11100, the bit that will be changed is 0
For pair 11111, the bit that will be changed is 1
For pair 10110, the bit that will be changed is 0
For pair 11111, the bit that will be changed is 2

Mutated -
 ['01100', '10111', '00110', '11011']

GENERATION  3  -  ['01100', '10111', '00110', '11011']
```

ii) Using Inbuilt Library

Code:-

```
import numpy
class ga:
    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):
        for idx in range(offspring_crossover.shape[0]):
            random_value = numpy.random.uniform(-1.0, 1.0, 1)
            offspring_crossover[idx, 4] = offspring_crossover[idx, 4] +
random_value
        return offspring_crossover
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)
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)
    fitness = ga.cal_pop_fitness(equation_inputs, new_population)
    print("fitness : ", fitness)
    parents = ga.select_mating_pool(new_population,
fitness,num_parents_mating)
    print("parents : ", parents)
```

```python
    offspring_crossover = ga.crossover(parents,offspring_size=(pop_size[0]-
parents.shape[0],num_weights))
    offspring_mutation = ga.mutation(offspring_crossover)
    print("mutation : ", offspring_mutation)
    new_population[0:parents.shape[0], :] = parents
    new_population[parents.shape[0]:, :] = offspring_mutation
    print("new population : ", new_population)
    print("Best result : ",
    numpy.max(numpy.sum(new_population*equation_inputs, axis=1)))
    fitness = ga.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:-

```
PS D:\AI pracs> python genetic2.py
[[illegible]
 [illegible]]
Generation :  0
fitness :  [illegible]
parents :  [illegible]
D:\AI pracs\genetic2.py:26: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single elem
ent from your array before performing this operation. (Deprecated NumPy 1.25.)
  offspring_crossover[idx, 4] = offspring_crossover[idx, 4] + random_value
mutation :  [illegible]
new_population :  [illegible]
Best result :  [illegible]
```

```
Generation :  1
fitness :  [44.96130455 22.55202293 15.37389268 13.12018769 -2.41115469 27.93473036
  3.58698902 62.95099675]
parents :  [[-1.13485758 -2.02190254 -2.29191793  2.66211143 -4.22609838 -2.48312669]
 [-1.71256754  0.86148456 -3.81787088  2.66211143 -3.81053085 -2.48312669]
 [ 3.57545542  2.68210442 -0.6054428  -3.5598302  -2.4705732  -2.49766236]
 [ 3.57545542  2.68210442 -0.6054428   1.58908781 -1.34129309  1.48217088]]
mutation :  [[-1.13485758 -2.02190254 -2.29191793  2.66211143 -3.45557446 -2.48312669]
 [-1.71256754  0.86148456 -3.81787088 -3.5598302  -2.41689029 -2.49766236]
 [ 3.57545542  2.68210442 -0.6054428   1.58908781 -1.18818886  1.48217088]
 [ 3.57545542  2.68210442 -0.6054428   2.66211143 -4.02457721 -2.48312669]]
new_population :  [[-1.13485758 -2.02190254 -2.29191793  2.66211143 -4.22609838 -2.48312669]
 [-1.71256754  0.86148456 -3.81787088  2.66211143 -3.81053085 -2.48312669]
 [ 3.57545542  2.68210442 -0.6054428  -3.5598302  -2.4705732  -2.49766236]
 [ 3.57545542  2.68210442 -0.6054428   1.58908781 -1.34129309  1.48217088]
 [-1.13485758 -2.02190254 -2.29191793  2.66211143 -3.45557446 -2.48312669]
 [-1.71256754  0.86148456 -3.81787088 -3.5598302  -2.41689029 -2.49766236]
 [ 3.57545542  2.68210442 -0.6054428   1.58908781 -1.18818886  1.48217088]
 [ 3.57545542  2.68210442 -0.6054428   2.66211143 -4.02457721 -2.48312669]]
Best result :  76.0701649800664
```

```
Generation :  2
fitness :  [62.95099675 44.96130455 27.93473036 22.55202293 54.47523371 -1.41013206
 20.86787643 76.07016498]
parents :  [[ 3.57545542  2.68210442 -0.6054428   2.66211143 -4.02457721 -2.48312669]
 [-1.13485758 -2.02190254 -2.29191793  2.66211143 -4.22609838 -2.48312669]
 [-1.13485758 -2.02190254 -2.29191793  2.66211143 -3.45557446 -2.48312669]
 [-1.71256754  0.86148456 -3.81787088  2.66211143 -3.81053085 -2.48312669]]
mutation :  [[ 3.57545542  2.68210442 -0.6054428   2.66211143 -4.15548491 -2.48312669]
 [-1.13485758 -2.02190254 -2.29191793  2.66211143 -3.39007318 -2.48312669]
 [-1.13485758 -2.02190254 -2.29191793  2.66211143 -3.55270573 -2.48312669]
 [-1.71256754  0.86148456 -3.81787088  2.66211143 -4.59928874 -2.48312669]]
new_population :  [[ 3.57545542  2.68210442 -0.6054428   2.66211143 -4.02457721 -2.48312669]
 [-1.13485758 -2.02190254 -2.29191793  2.66211143 -4.22609838 -2.48312669]
 [-1.13485758 -2.02190254 -2.29191793  2.66211143 -3.45557446 -2.48312669]
 [-1.71256754  0.86148456 -3.81787088  2.66211143 -3.81053085 -2.48312669]
 [ 3.57545542  2.68210442 -0.6054428   2.66211143 -4.15548491 -2.48312669]
 [-1.13485758 -2.02190254 -2.29191793  2.66211143 -3.39007318 -2.48312669]
 [-1.13485758 -2.02190254 -2.29191793  2.66211143 -3.55270573 -2.48312669]
 [-1.71256754  0.86148456 -3.81787088  2.66211143 -4.59928874 -2.48312669]]
Best result :  77.51014960696165
```

```
Generation :  3
fitness :  [76.07016498 62.95099675 54.47523371 44.96130455 77.51014961 53.84271956
 55.54367767 53.63764137]
parents :  [[ 3.57545542  2.68210442 -0.6054428   2.66211143 -4.15548491 -2.48312669]
 [ 3.57545542  2.68210442 -0.6054428   2.66211143 -4.02457721 -2.48312669]
 [-1.13485758 -2.02190254 -2.29191793  2.66211143 -4.22609838 -2.48312669]
 [-1.13485758 -2.02190254 -2.29191793  2.66211143 -3.55270573 -2.48312669]]
mutation :  [[ 3.57545542  2.68210442 -0.6054428   2.66211143 -4.74575549 -2.48312669]
 [ 3.57545542  2.68210442 -0.6054428   2.66211143 -5.04797017 -2.48312669]
 [-1.13485758 -2.02190254 -2.29191793  2.66211143 -4.38725664 -2.48312669]
 [-1.13485758 -2.02190254 -2.29191793  2.66211143 -3.23252107 -2.48312669]]
new_population :  [[ 3.57545542  2.68210442 -0.6054428   2.66211143 -4.15548491 -2.48312669]
 [ 3.57545542  2.68210442 -0.6054428   2.66211143 -4.02457721 -2.48312669]
 [-1.13485758 -2.02190254 -2.29191793  2.66211143 -4.22609838 -2.48312669]
 [-1.13485758 -2.02190254 -2.29191793  2.66211143 -3.55270573 -2.48312669]
 [ 3.57545542  2.68210442 -0.6054428   2.66211143 -4.74575549 -2.48312669]
 [ 3.57545542  2.68210442 -0.6054428   2.66211143 -5.04797017 -2.48312669]
 [-1.13485758 -2.02190254 -2.29191793  2.66211143 -4.38725664 -2.48312669]
 [-1.13485758 -2.02190254 -2.29191793  2.66211143 -3.23252107 -2.48312669]]
Best result :  87.32748757431156
```

```
Generation :  4
fitness :  [77.51014961 76.07016498 62.95099675 55.54367767 84.00312603 87.32748757
 64.72373761 52.02164632]
parents :  [[ 3.57545542  2.68210442 -0.6054428   2.66211143 -5.04797017 -2.48312669]
 [ 3.57545542  2.68210442 -0.6054428   2.66211143 -4.74575549 -2.48312669]
 [ 3.57545542  2.68210442 -0.6054428   2.66211143 -4.15548491 -2.48312669]
 [ 3.57545542  2.68210442 -0.6054428   2.66211143 -4.02457721 -2.48312669]]
mutation :  [[ 3.57545542  2.68210442 -0.6054428   2.66211143 -5.73266178 -2.48312669]
 [ 3.57545542  2.68210442 -0.6054428   2.66211143 -3.59006272 -2.48312669]
 [ 3.57545542  2.68210442 -0.6054428   2.66211143 -4.96592221 -2.48312669]
 [ 3.57545542  2.68210442 -0.6054428   2.66211143 -5.22776043 -2.48312669]]
new_population :  [[ 3.57545542  2.68210442 -0.6054428   2.66211143 -5.04797017 -2.48312669]
 [ 3.57545542  2.68210442 -0.6054428   2.66211143 -4.74575549 -2.48312669]
 [ 3.57545542  2.68210442 -0.6054428   2.66211143 -4.15548491 -2.48312669]
 [ 3.57545542  2.68210442 -0.6054428   2.66211143 -4.02457721 -2.48312669]
 [ 3.57545542  2.68210442 -0.6054428   2.66211143 -5.73266178 -2.48312669]
 [ 3.57545542  2.68210442 -0.6054428   2.66211143 -3.59006272 -2.48312669]
 [ 3.57545542  2.68210442 -0.6054428   2.66211143 -4.96592221 -2.48312669]
 [ 3.57545542  2.68210442 -0.6054428   2.66211143 -5.22776043 -2.48312669]]
Best result :  94.85909525344171
Best solution :  [[[ 3.57545542  2.68210442 -0.6054428   2.66211143 -5.73266178
   -2.48312669]]]
Best solution fitness :  [94.85909525]
```

HILLCLIMBING

Implementation:-

```
exp4.py > randomSolution
import random
def randomSolution(tsp):
    cities = list(range(len(tsp)))
    solution = []
    for i in range(len(tsp)):
        randomCity = cities[random.randint(0, len(cities) - 1)]
        solution.append(randomCity)
        cities.remove(randomCity)
    return solution

def routeLength(tsp, solution):
    routeLength = 0
    for i in range(len(solution)):
        routeLength += tsp[solution[i - 1]][solution[i]]
    return routeLength

def getNeighbours(solution):
    neighbours = []
    for i in range(len(solution)):
        for j in range(i + 1, len(solution)):
            neighbour = solution.copy()
            neighbour[i] = solution[j]
            neighbour[j] = solution[i]
            neighbours.append(neighbour)
    return neighbours

def getBestNeighbour(tsp, neighbours):
    bestRouteLength = routeLength(tsp, neighbours[0])
    bestNeighbour = neighbours[0]
    for neighbour in neighbours:
        currentRouteLength = routeLength(tsp, neighbour)
        if currentRouteLength < bestRouteLength:
            bestRouteLength = currentRouteLength
            bestNeighbour = neighbour
    return bestNeighbour, bestRouteLength

def hillClimbing(tsp):
    currentSolution = randomSolution(tsp)
    currentRouteLength = routeLength(tsp, currentSolution)
    neighbours = getNeighbours(currentSolution)
    bestNeighbour, bestNeighbourRouteLength = getBestNeighbour(tsp,neighbours)
    while bestNeighbourRouteLength < currentRouteLength:
        currentSolution = bestNeighbour
        currentRouteLength = bestNeighbourRouteLength
        neighbours = getNeighbours(currentSolution)
        bestNeighbour, bestNeighbourRouteLength = getBestNeighbour(tsp,neighbours)
    return currentSolution, currentRouteLength

def main():
    tsp = [
  [0, 100, 700, 50],
  [100, 0, 330, 1200],
  [700, 330, 0, 400],
  [50, 1200, 400, 0] ]
    print(hillClimbing(tsp))
    if __name__ == "__main__":
        main()
```

Output:-

```
[3,2,1,0] 880
```