**GENETIC ALGORITHM**

**CODE AND OUTPUTS:**

import numpy

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

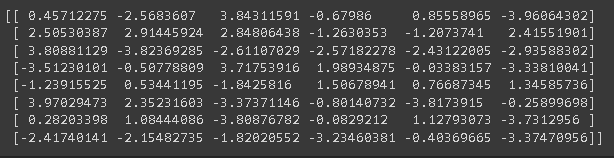
sol\_per\_pop = 8

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 = cal\_pop\_fitness(equation\_inputs, new\_population)

    parents = select\_mating\_pool(new\_population, fitness,

                                      num\_parents\_mating)

    offspring\_crossover = crossover(parents,

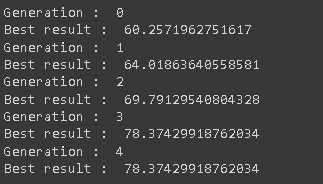
                                       offspring\_size=(pop\_size[0]-parents.shape[0], num\_weights))

    offspring\_mutation = mutation(offspring\_crossover)

    new\_population[0:parents.shape[0], :] = parents

    new\_population[parents.shape[0]:, :] = offspring\_mutation

    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])

