Name: Tania Rajabally

Roll No: 43

UID: 2017130047

Batch: C

Course Code: CEL71 (AI and Soft Computing Lab)

Date:18/10/2020

**Experiment No 8**

**Genetic Algorithm**

**Aim:** To implement genetic algorithm.

**Theory:**

Genetic Algorithm (GA) is a search-based optimization technique based on the principles of Genetics and Natural Selection. It is frequently used to find optimal or near-optimal solutions to difficult problems which otherwise would take a lifetime to solve. It is frequently used to solve optimization problems, in research, and in machine learning.

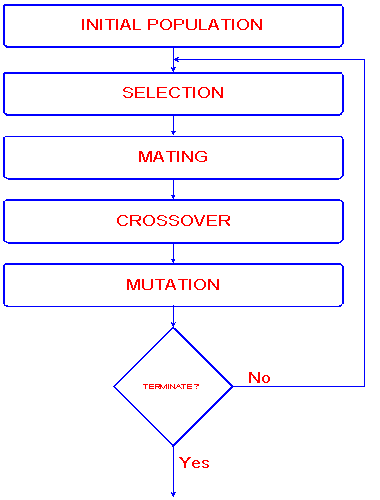
Nature has always been a great source of inspiration to all mankind. Genetic Algorithms (GAs) are search based algorithms based on the concepts of natural selection and genetics. GAs are a subset of a much larger branch of computation known as Evolutionary Computation.

GAs were developed by John Holland and his students and colleagues at the University of Michigan, most notably David E. Goldberg and has since been tried on various optimization problems with a high degree of success.

In GAs, we have a pool or a population of possible solutions to the given problem. These solutions then undergo recombination and mutation (like in natural genetics), producing new children, and the process is repeated over various generations. Each individual (or candidate solution) is assigned a fitness value (based on its objective function value) and the fitter individuals are given a higher chance to mate and yield more “fitter” individuals. This is in line with the Darwinian Theory of “Survival of the Fittest”.

In this way we keep “evolving” better individuals or solutions over generations, till we reach a stopping criterion.

Genetic Algorithms are sufficiently randomized in nature, but they perform much better than random local search (in which we just try various random solutions, keeping track of the best so far), as they exploit historical information as well.



**Procedure:**

* [Start] Generate random population of n chromosomes (suitable solutions for the problem)
* [Fitness] Evaluate the fitness f(x) of each chromosome x in the population
* [New population] Create a new population by repeating following steps until the new population is complete
  + [Selection] Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)
  + [Crossover] With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.
  + [Mutation] With a mutation probability mutate new offspring at each locus (position in chromosome).
  + [Accepting] Place new offspring in a new population
* [Replace] Use new generated population for a further run of algorithm
* [Test] If the end condition is satisfied, stop, and return the best solution in current population
* [Loop] Go to step 2

**Code:**

import numpy as np

# Initializing n = 6

n = 6

# Initialization of chromosomes

# chromosome

chromosome = np.random.randint(0,35 ,(n,3))

print("chromosomes :",chromosome)

epoch = 0

while epoch < 200 :

# Computation of fitness function

objective = abs(35 - 2\*chromosome[:,0] - 3\*chromosome[:,1] -4\*chromosome[:,2] )

print("Fitness object :", objective)

# Selection of fittest chromosome

fitness = 1/(1 + objective)

print("Fitness :",fitness)

# Calculating the total of fitness function

total = fitness.sum()

print("Total :",total)

# Calculating Probablility for each chromosome

prob = fitness/total

print("Probability :",prob)

# Selection using Roulette Wheel And Calculating Cumulative Probability

cum\_sum = np.cumsum(prob)

print("Cumulative Sum :", cum\_sum)

# Generating Random Numbers in the range 0-1

Ran\_nums = np.random.random((chromosome.shape[0]))

print("Random Numbers :",Ran\_nums)

# Making a new matrix of chromosome for calculation purpose

chromosome\_2 = np.zeros((chromosome.shape[0],3))

for i in range(Ran\_nums.shape[0]):

for j in range(chromosome.shape[0]):

if Ran\_nums[i] < cum\_sum[j]:

chromosome\_2[i,:] = chromosome[j,:]

break

chromosome = chromosome\_2

print("Chromosomes after updation :",chromosome)

# crossover

R = [np.random.random() for i in range(n)]

print("Random Values :",R)

# Crossover Rate

pc = 0.25

flag = Ran\_nums < pc

print("Flagged Values :",flag)

# Determining the cross chromosomes

cross\_chromosome = chromosome[[(i == True) for i in flag]]

print("Cross chromosome :",cross\_chromosome)

len\_cross\_chrom = len(cross\_chromosome)

# Calculating cross values

cross\_values = np.random.randint(1,3,len\_cross\_chrom)

print("Cross Values :",cross\_values)

cpy\_chromosome = np.zeros(cross\_chromosome.shape)

# Performing Cross-Over

# Copying the chromosome values for calculations

for i in range(cross\_chromosome.shape[0]):

cpy\_chromosome[i , :] = cross\_chromosome[i , :]

if len\_cross\_chrom == 1:

cross\_chromosome = cross\_chromosome

else :

for i in range(len\_cross\_chrom):

c\_val = cross\_values[i]

if i == len\_cross\_chrom - 1 :

cross\_chromosome[i , c\_val:] = cpy\_chromosome[0 , c\_val:]

else :

cross\_chromosome[i , c\_val:] = cpy\_chromosome[i+1 , c\_val:]

print("Crossovered Chromosome :",cross\_chromosome)

index\_chromosome = 0

index\_newchromosome = 0

for i in flag :

if i == True :

chromosome[index\_chromosome, :] = cross\_chromosome[index\_newchromosome, :]

index\_newchromosome = index\_newchromosome + 1

index\_chromosome = index\_chromosome + 1

print("New Chromosomes:", chromosome)

# Calculating the total no. of generations

a ,b = chromosome.shape[0] ,chromosome.shape[1]

total\_gen = a\*b

print("Total Generations :",total\_gen)

#mutation rate = pm

pm = 0.1

no\_of\_mutations = int(np.round(pm \* total\_gen))

print("No. of Mutations :" ,no\_of\_mutations)

# Calculating the Generation number

gen\_num = np.random.randint(0,total\_gen - 1, no\_of\_mutations)

print(" Generated Random Numbers : " , gen\_num)

# Generating a random number which can replace the selected chromosome to be mutated

Replacing\_num = np.random.randint(0,30, no\_of\_mutations)

print(" Numbers to be replaced : " , Replacing\_num)

for i in range(no\_of\_mutations):

a = gen\_num[i]

row = a//3

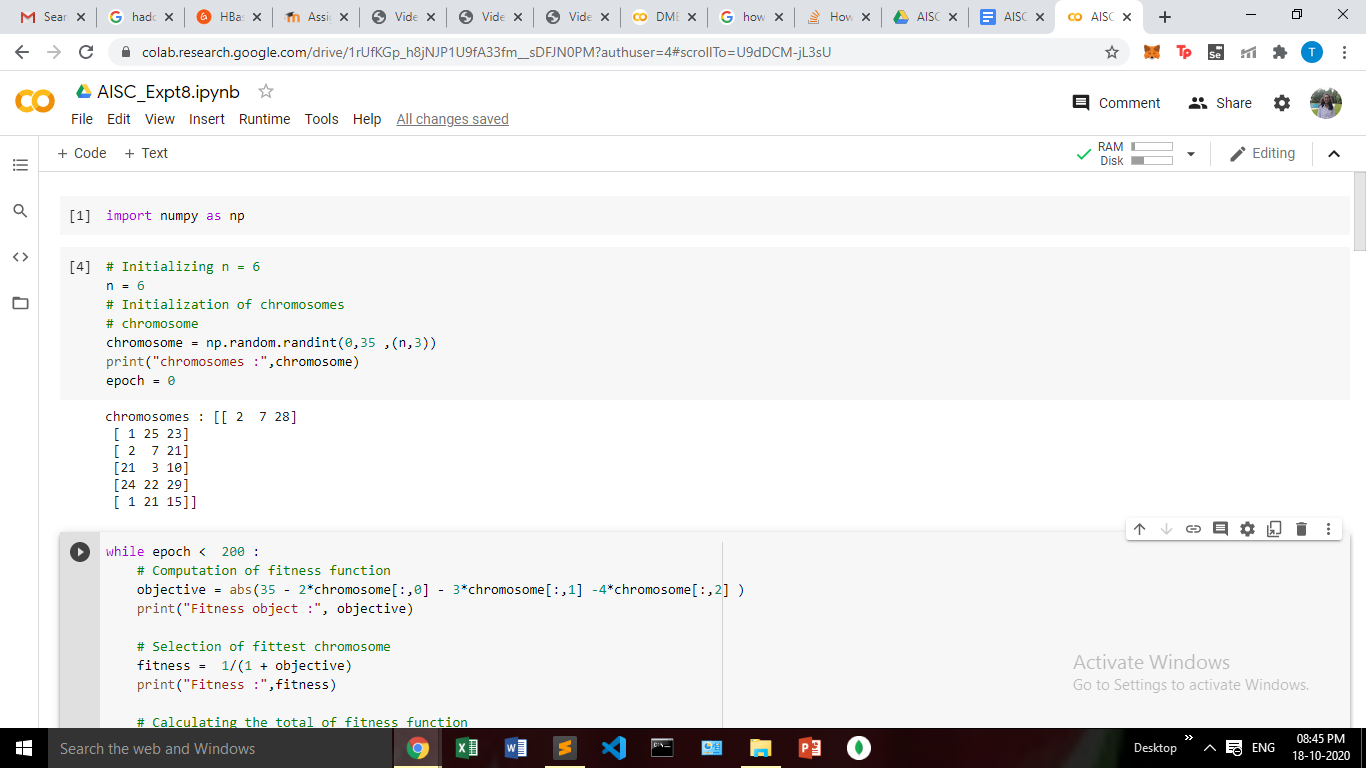
col = a%3

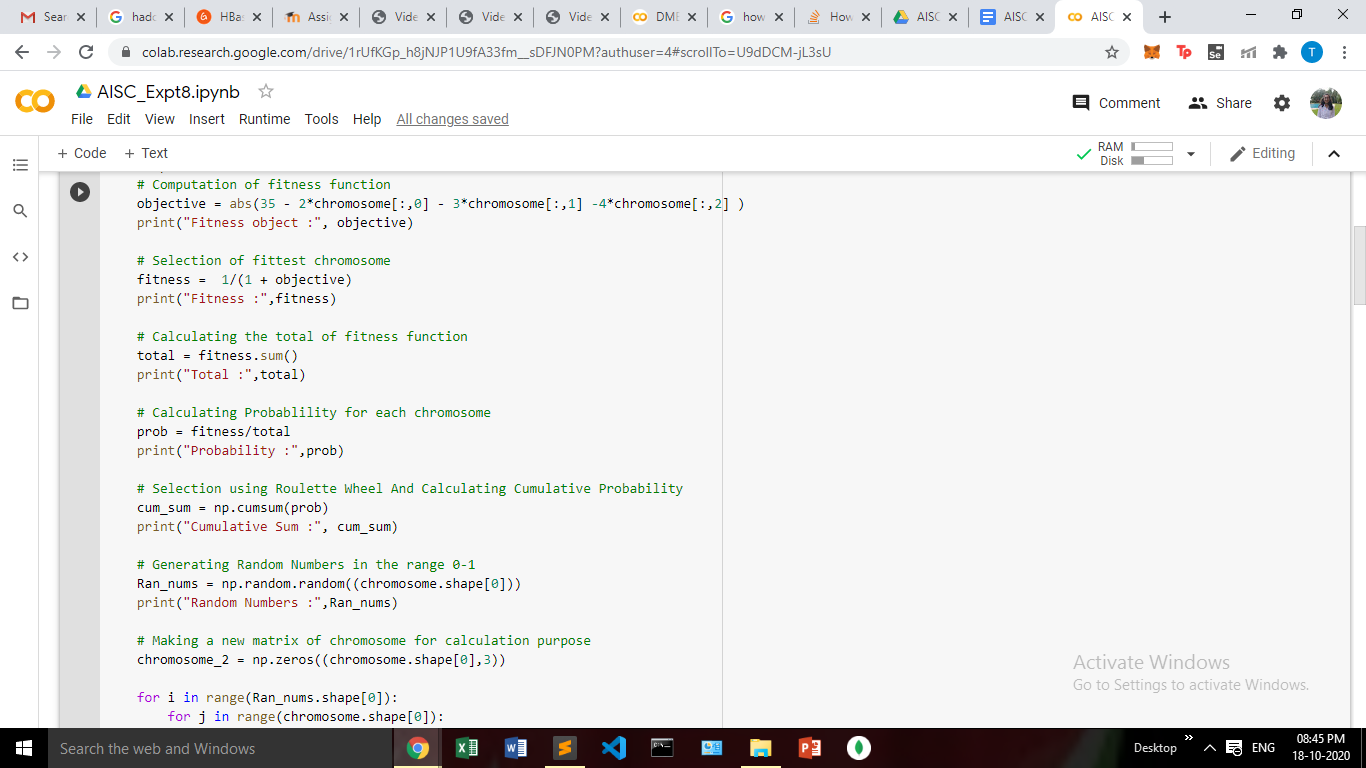
chromosome[row , col] = Replacing\_num[i]

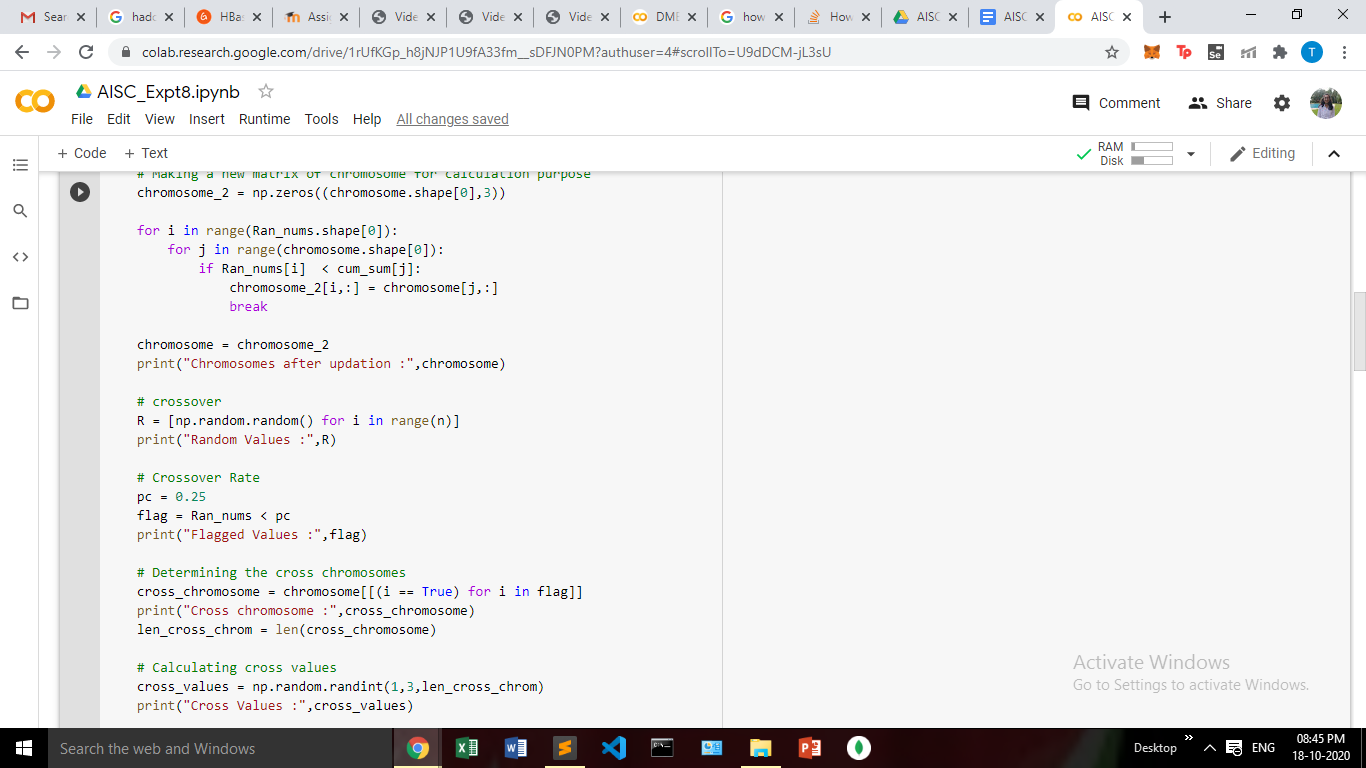
print(" Chromosomes After Mutation : " , chromosome)

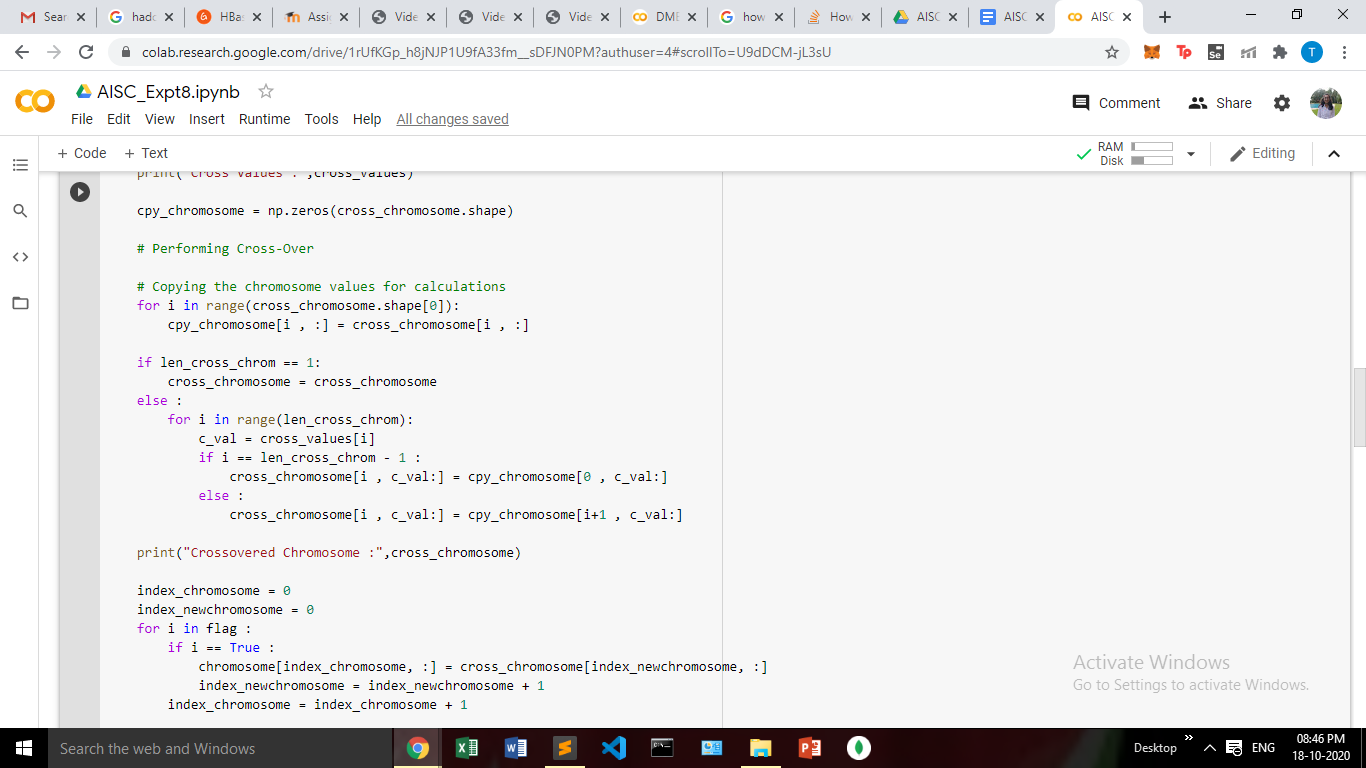
epoch = epoch + 1

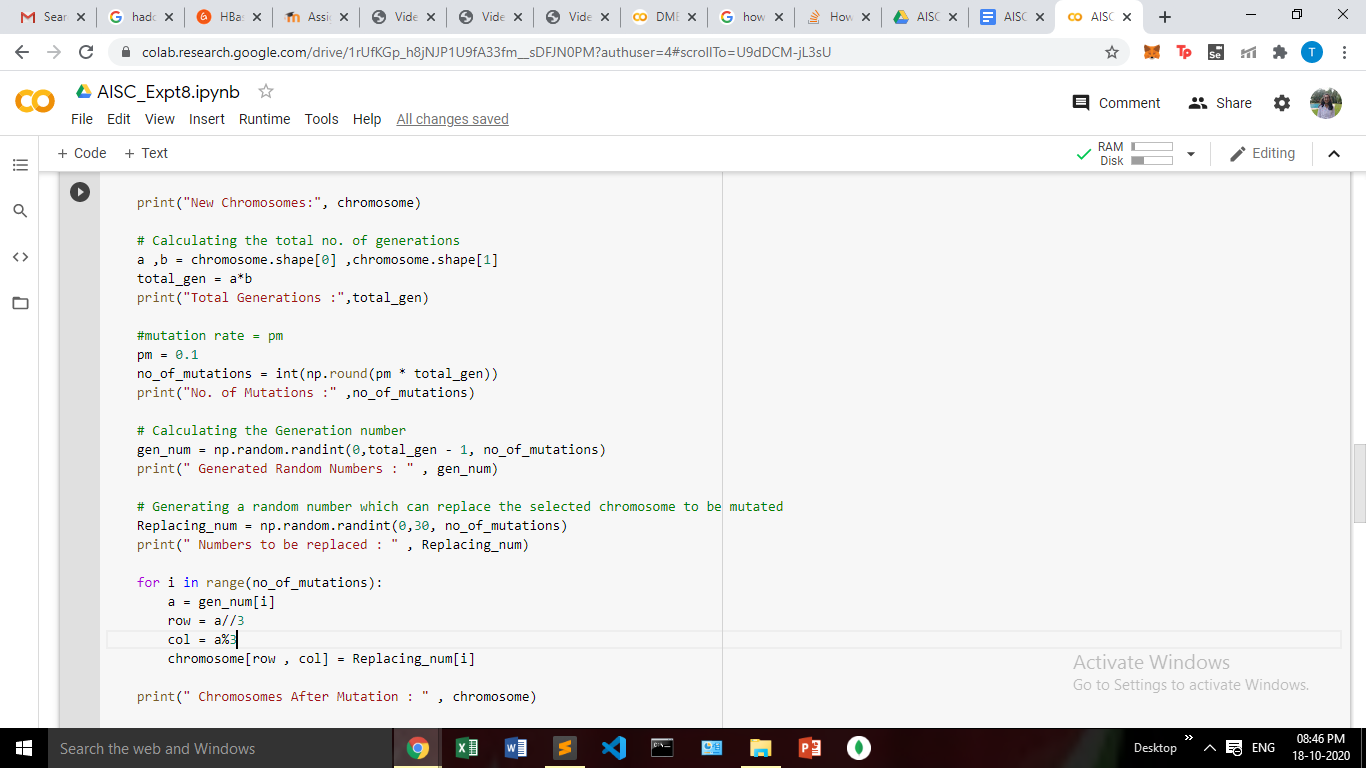
**Output:**

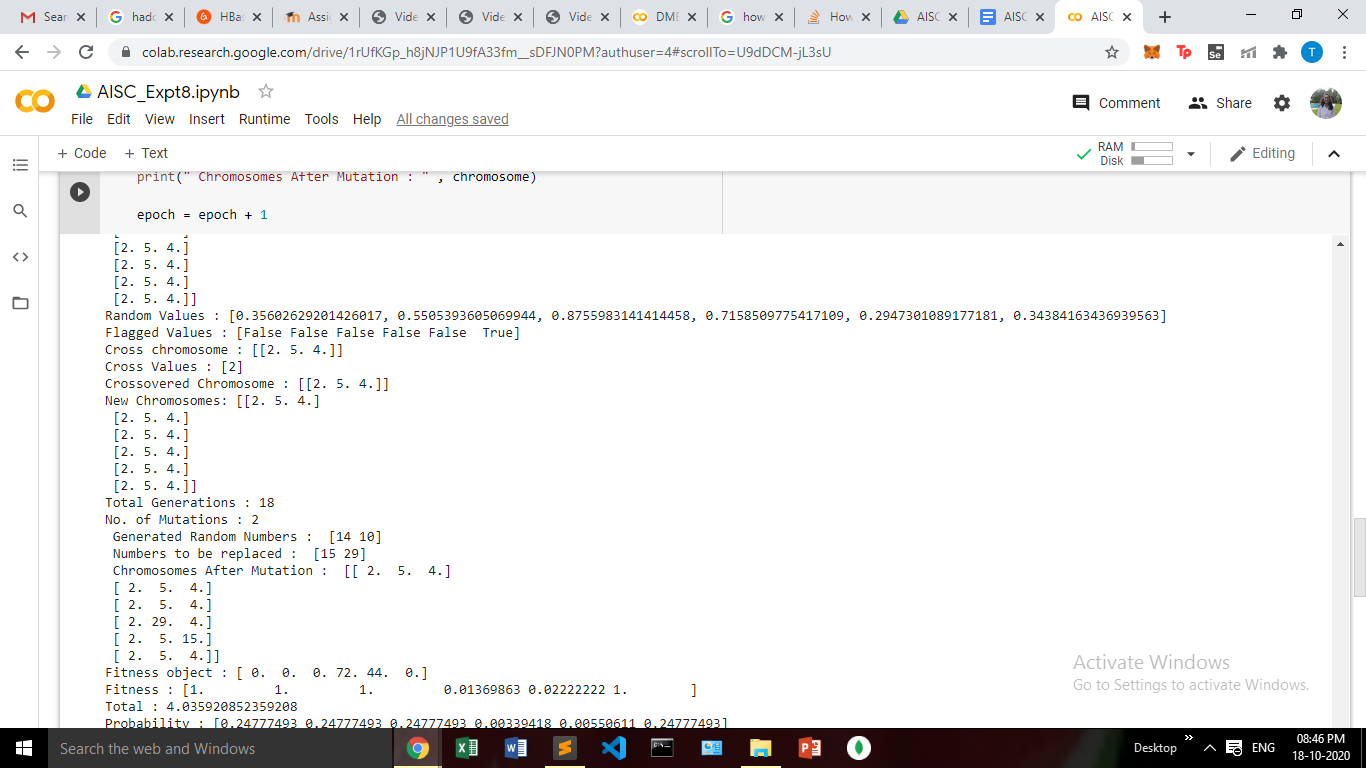


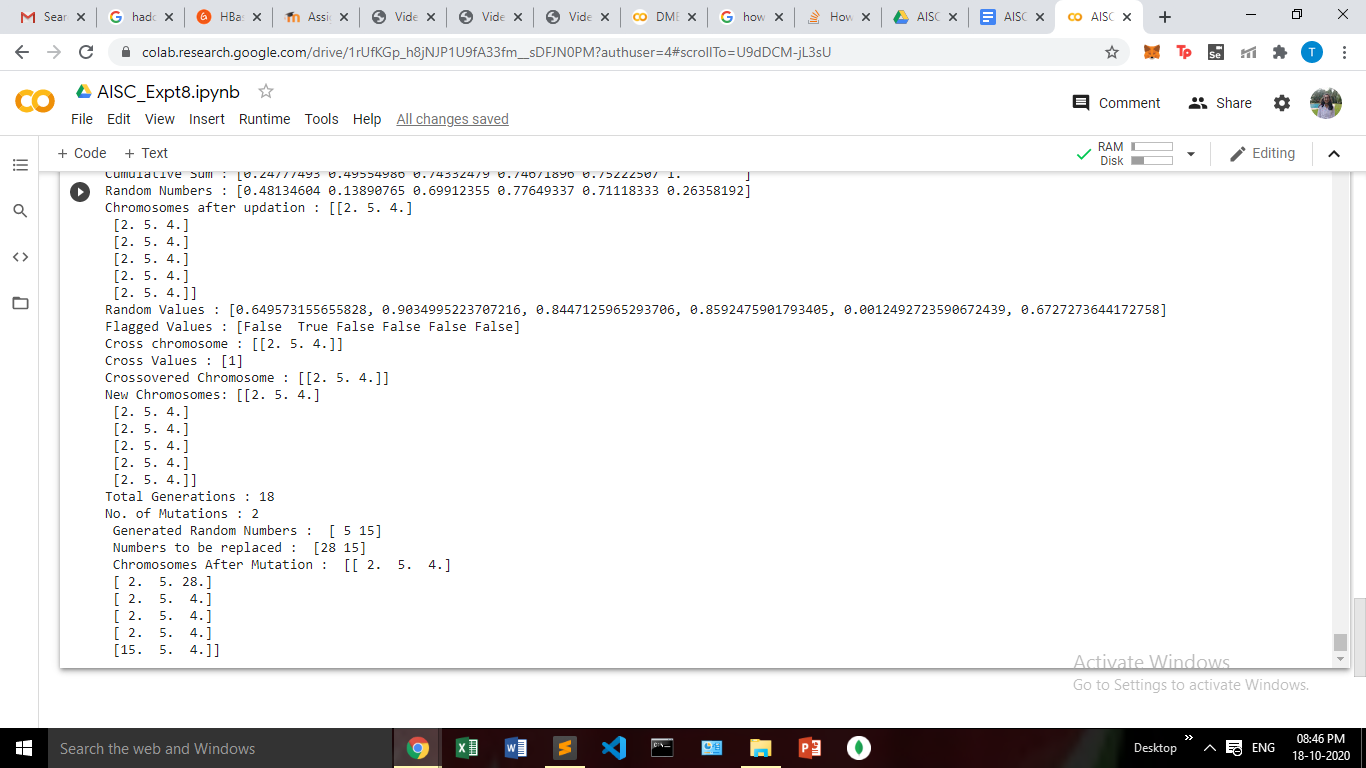












**Conclusion:**

In this experiment, we have implemented genetic algorithm for the given equation.