```
%copyright Charles Davi 2022

clear all
clc

%=================================================================
===
%Generates the populations / dataset
%=================================================================
===

N = 50; %number of basepairs / columns for all populations
M = 100; %size of each population / number of rows in the population

%population 1 is highly homogenous (2.5% noise)

population_1 = randi(4,[1,N]); %generates a random core sequence of bases
population_1_matrix = population_1.*ones(M,N); %stores M identical sequences in a
matrix

population_1_matrix(:,N+1) = 1; %hidden classifier for the population

noise_percentage = .025; %the percentage of bases that will randomized to create
differentiation in the population
num_noisey_bases = floor(noise_percentage*N); %the number of randomized /
differentiated bases
noisey_bases = randperm(N,num_noisey_bases); %generates random base indexes
that will be differentiated through randomization

population_1_matrix(:,noisey_bases) = randi(4,[M,num_noisey_bases]);

%-----------------------------------------------------------------

%population 2 is less homogenous (5% noise)

population_2 = randi(4,[1,N]); %generates a random core sequence of bases
population_2_matrix = population_2.*ones(M,N); %stores M identical sequences in a
matrix

population_2_matrix(:,N+1) = 2; %hidden classifier for the population

noise_percentage = .05; %the percentage of bases that will randomized to create
differentiation in the population
num_noisey_bases = floor(noise_percentage*N); %the number of differentiated bases
noisey_bases = randperm(N,num_noisey_bases); %generates random base indexes
that will be differentiated through randomization
```

```
population_2_matrix(:,noisey_bases) = randi(4,[M,num_noisey_bases]);

%-----------------------------------------------------------------

%combines the populations into a single dataset
dataset = [population_1_matrix; population_2_matrix];
scramble = randperm(2*M); %generates a permutation vector to scramble the rows of
the dataset to avoid inadvertant cheating
dataset = dataset(scramble,:);

%================================================================
===
%Clusters the dataset
%================================================================
===

cluster_matrix = Genetic_Cluster_Algorithm(dataset,N);

%================================================================
===
%Calculates the accuracy of the clusters
%================================================================
===

for i = 1 : M

  x = find(cluster_matrix(i,:) == 1); %finds all rows in the cluster
  class_vector = dataset(x,N+1); %loads the classes for those rows
  actual_class = dataset(i,N+1); %this is the true class of the row in question (i)
  y = class_vector != actual_class; %these two lines count the number of times the
cluster's classes don't equal the class for row i
  num_errors = sum(y);

  %if the cluster is not empty, we calculate accuracy
  if(!isempty(x))

    accuracy(i) = 1 - num_errors/size(x,2); %accuracy is 1 less the number of errors
divided by the size of the cluster

  %if the cluster is empty, we cannot calculate accuracy and flag it with -1
  else

    accuracy(i) = -1;

  endif

endfor
```

```
x = find(accuracy == -1); %finds all empty clusters
final_accuracy_vector = accuracy;
num_empty_clusters = size(x,2) %displays the number of empty clusters
final_accuracy_vector(x) = []; %deletes all empty cluster flags

mean(final_accuracy_vector) %displys the average accuracy over all non-empty
clusters

sum(cluster_matrix,2)
```