NEURAL NETWORKS

ASSIGNMENT

UNSUPERVISED LEARNING

Keya Desai

201501012

ASSIGNMENT-1

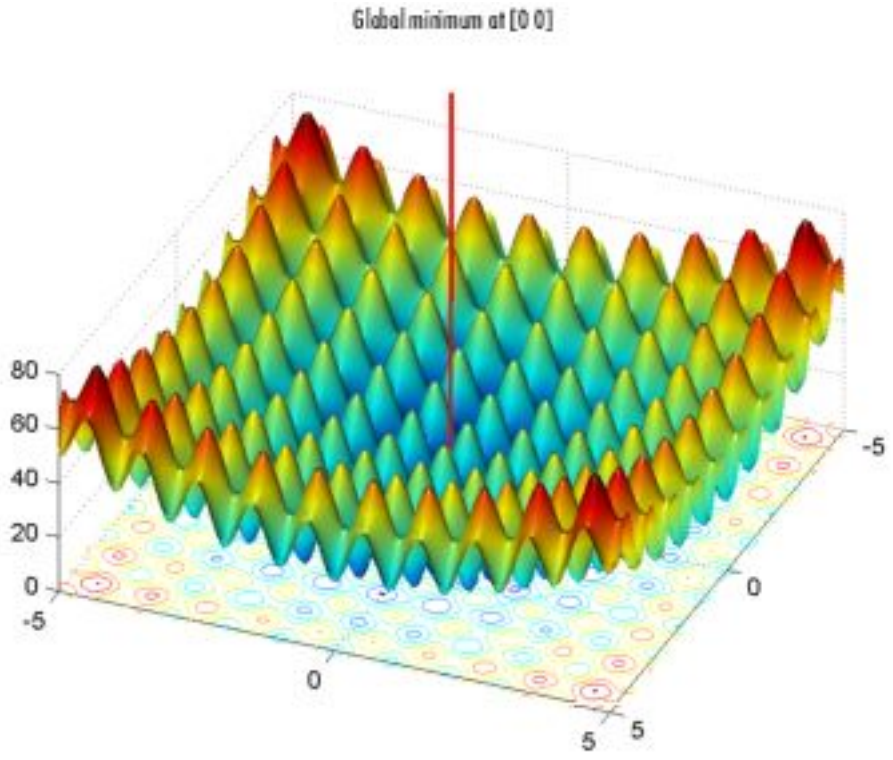
A)Finding minimum of Rastrigin's function

Rastrigin's Function

This section presents an example that shows how to find the minimum of Rastrigin's function, a function that is often used to test the genetic algorithm. For two independent variables, Rastrigin's function is defined as

$$Ras(x)=20+x21+x22-10(\cos 2\pi x1+\cos 2\pi x2).$$

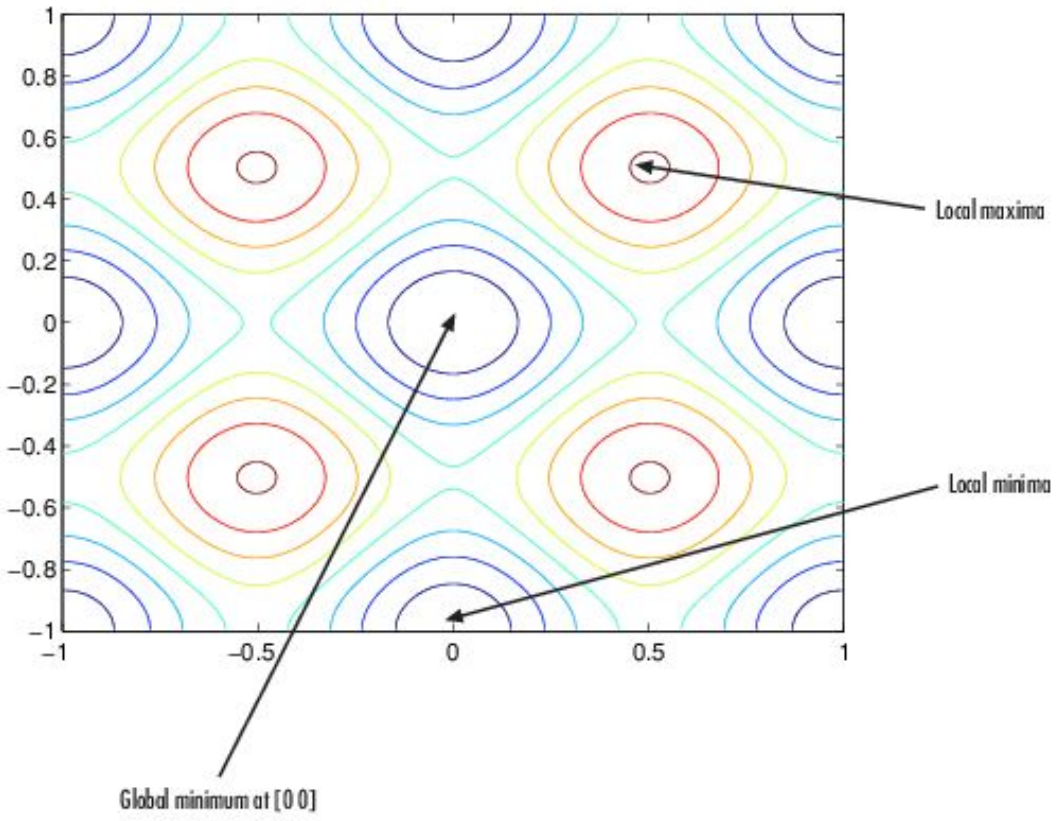
Global Optimization Toolbox software contains the rastriginsfcn.m file, which computes the values of Rastrigin's function. The following figure shows a plot of Rastrigin's function.



As the plot shows, Rastrigin's function has many local minima—the "valleys" in the plot. However, the function has just one global minimum, which occurs at the point [0 0] in the *x*-*y* plane, as indicated by the vertical line in the plot, where the value of the function is 0. At any local minimum other than [0 0], the

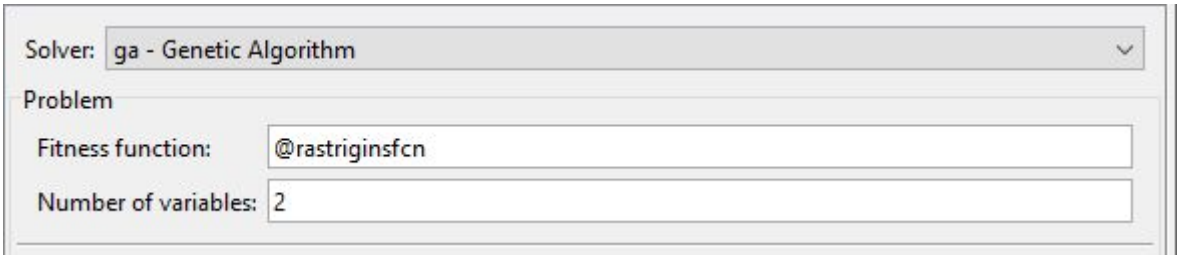
value of Rastrigin's function is greater than 0. The farther the local minimum is from the origin, the larger the value of the function is at that point. Rastrigin's function is often used to test the genetic algorithm, because its many local minima make it difficult for standard, gradient-based methods to find the global minimum.

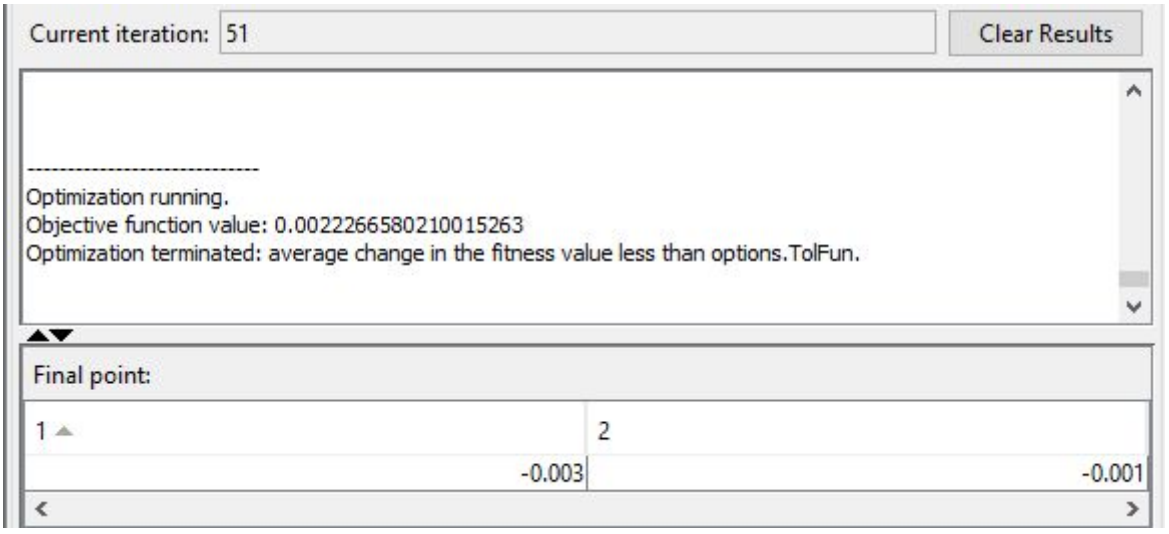
The following contour plot of Rastrigin's function shows the alternating maxima and minima.

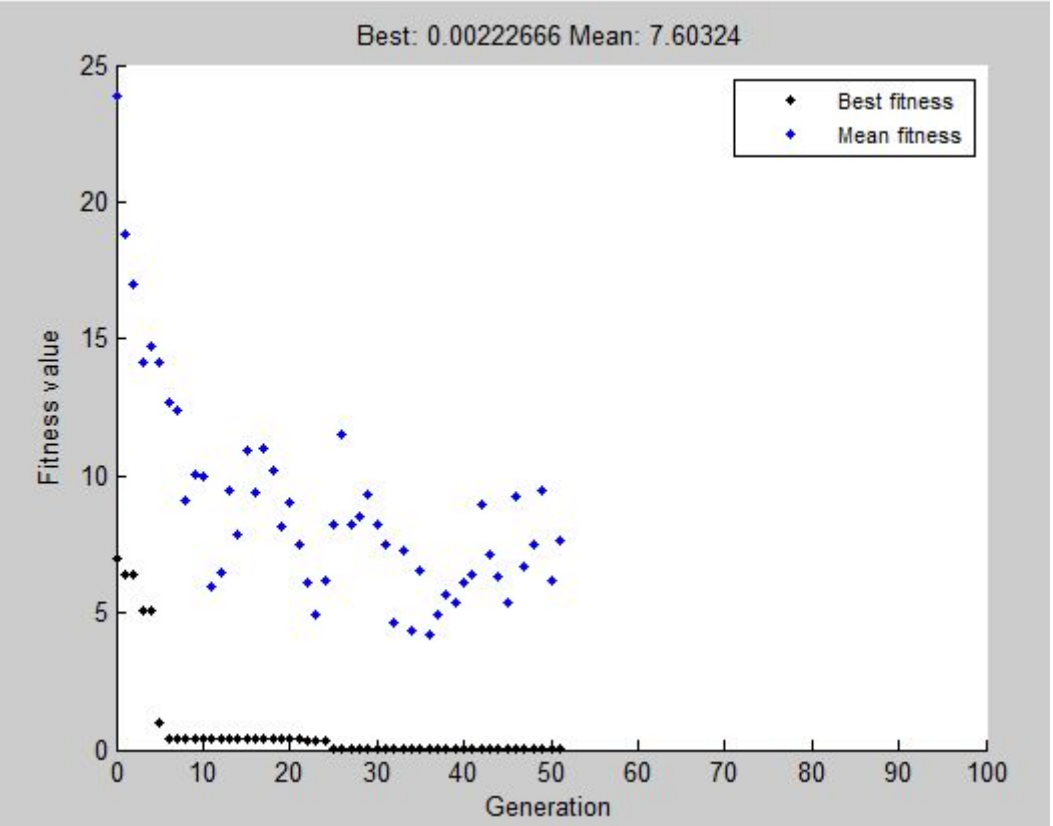


i)GRADIENT FREE METHOD-

We use the Optimization tool for finding where the minima lies in the rastrigin function.







The display shows:

- The final value of the fitness function when the algorithm terminated:
- Objective function value: 0.0022266580210015263
- Note that the value shown is very close to the actual minimum value of Rastrigin's function, which is 0. Setting the Initial Range, Setting the Amount of Mutation, and Set Maximum Number of Generations describe some ways to get a result that is closer to the actual minimum.
- Optimization terminated: maximum number of generations exceeded.
- The final point, which in this example is [-0.003 -0.001]

ii)Gradient Based Method

Code :

```
Rastrigin Function:
function y = Rastrigin (X)
    n = 2;
    m = 0;
    for i = 1:n
        m = m + X(i)^2 - 10*cos(2*p
    end
    y = 10*n + m;
end

%Randomize function:

function [V]=rand1(X,M,F,m)
    R=randperm(M);
    j=R(1);
    k=R(2);
    p=R(3);
    u=R(4);
    v=R(5);
    if j==m
       j=R(6);
     elseif k==m
        k=R(6);
     elseif p==m
        p=R(6);
     elseif u==m
        u=R(6);
     elseif v==m
        v=R(6);
     end
     V=X(j,:)+F*(X(k,:)-X(p,:));
 Running code:
 N=2;         % Number of variables
     M=50;        % Populations size
     F=0.5;       % Mutation factor
     C=0.9;       % Crossover rate
     I_max=200;   % Max iteration time
         Run=1;       % The number of test time
     X_max=[5,5];
     X_min=[-5,-5];

     Func=@Rastrigin;
```

```
45      for r=1:Run
46          iter=0;
47          % 1.Generate MxN matrix
48          for m=1:M
49              for n=1:N
50                  X(m,n)=X_min(n)+rand()*(X_max(n)-X_min(n));
51              end
52          end
53
54
55
56          for i=1:I_max
57              iter=iter+1;
58              for m=1:M % For each individual
59                  % Mutation
60                [V]=rand1(X,M,F,m);
61                      for n=1:N
62                      if V(1,n)>X_max(1,n)
63                          V(1,n)=X_max(1,n);
64                      end
65                      if V(1,n)<X_min(1,n)
65      C:\Users\ONKAR\Desktop\matlab work\for_fuzzyCmeans.m
66                          V(1,n)=X_min(1,n);
67                      end
68                  end
69
70
71
72                  jrand=floor(rand()*N+1);
73                  for n=1:N
74                      R1=rand();
75                      if (R1<C || n==jrand)
76                          U(1,n)=V(1,n);
77                      else
78                          U(1,n)=X(m,n);
79                      end
80                  end
81
82
83                  % Selection
84
85                  if Func(U(1,:)) < Func(X(m,:))
```

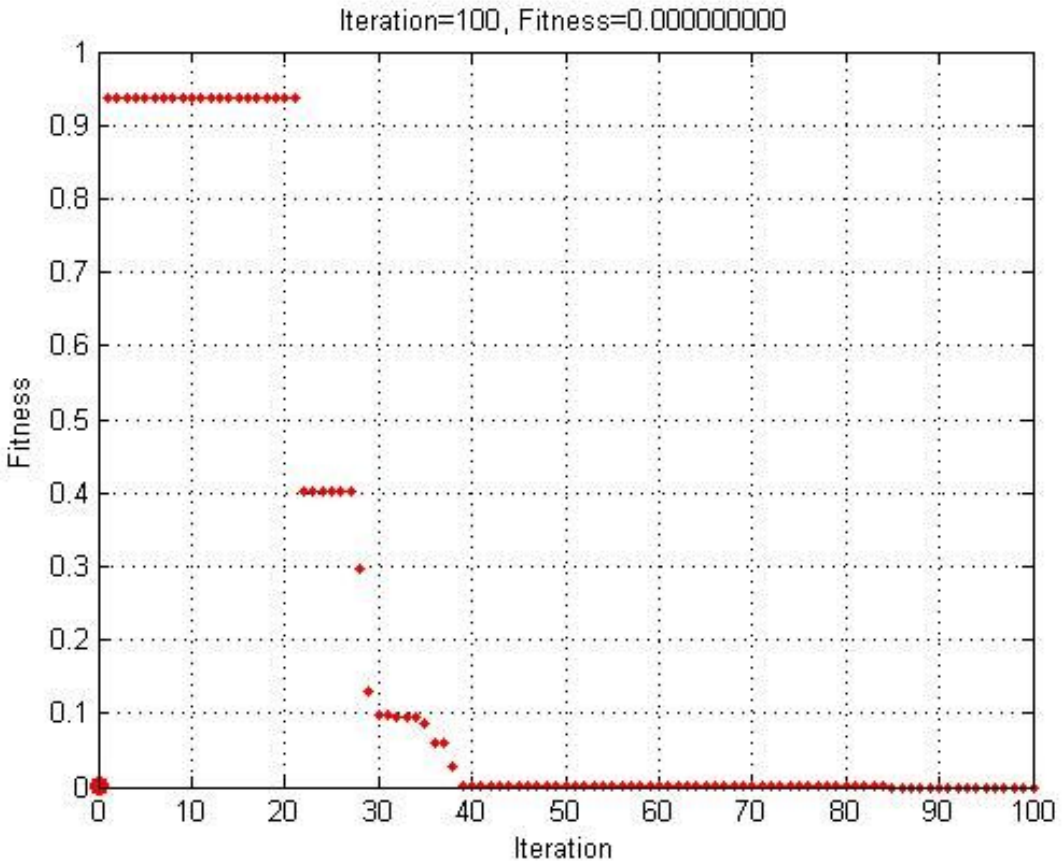
```
84
85                  if Func(U(1,:)) < Func(X(m,:))
86                      Tr=U(1,:);
87                  else
88                      Tr=X(m,:);
89                  end
90                  % Use the selection result to replace the m row
91                  X(m,:)=Tr;
92                  Y(m,1)=Func(X(m,:));
93              end
```

Observation:

The error value either remains constant or decreases and the decrease takes more number of steps but it changes drastically to a lower value. After almost 40 iterations the error value remains constant.

Sr. No.	Maximum Error Value	Iteration
1	3	38
2	7	44
3	5	30
4	6	24
5	6	27
6	3.3	18
7	1	38
8	5.2	19
9	6	28
10	2	34

Output:



The graph for best fitness (i.e. Minimum error)

B)Density Scatter Plot for satellite and ground observation and Correlation Coefficient:

1. Code Snippet:

```
%Specifing data paths
ground_file = 'C:\Users\DELL\Documents\data\Ground_observation'
satellite_file = 'C:\Users\DELL\Documents\data\Satellite_observation'

%Coverting data from excel sheet into a single column vector
g_data = xlsread(ground_file, 'B:B')
s_data = xlsread(satellite_file,'B:B')


x=[70:150];
y1=x;   % one-one line with slope=1
y2=0.5284.*x+55.45;  % regression model

%Finding Correlation Coefficient
R = corrcoef(g_data,s_data)

% Scatter plot showing both the lines
figure(1)
scatter(g_data,s_data,50,'filled');
xlabel('Ground Observation')
ylabel('Satellite Observation')
hold on;
plot(x,y,'-r',0.5284*x+55.45,y,'.b');

```

2. Regression Model fitting the data:

- We find the linear regression model for the given 150 data-sets by plotting ground-data on x-axis and satellite data on the y-axis.
- We have used the Curve-Fitting tool in MATLAB and got the parameters as shown below:

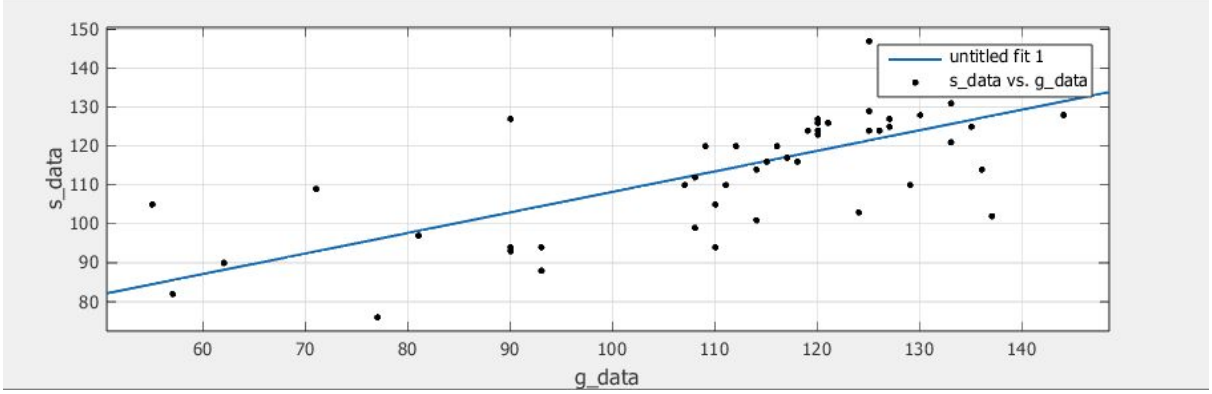
Results

Linear model Poly1:
f(x) = p1*x + p2
Coefficients (with 95% confidence bounds):
p1 = 0.5284 (0.3758, 0.681)
p2 = 55.45 (37.93, 72.96)

Goodness of fit:
SSE: 6305
R-square: 0.5026
Adjusted R-square: 0.4922
RMSE: 11.46

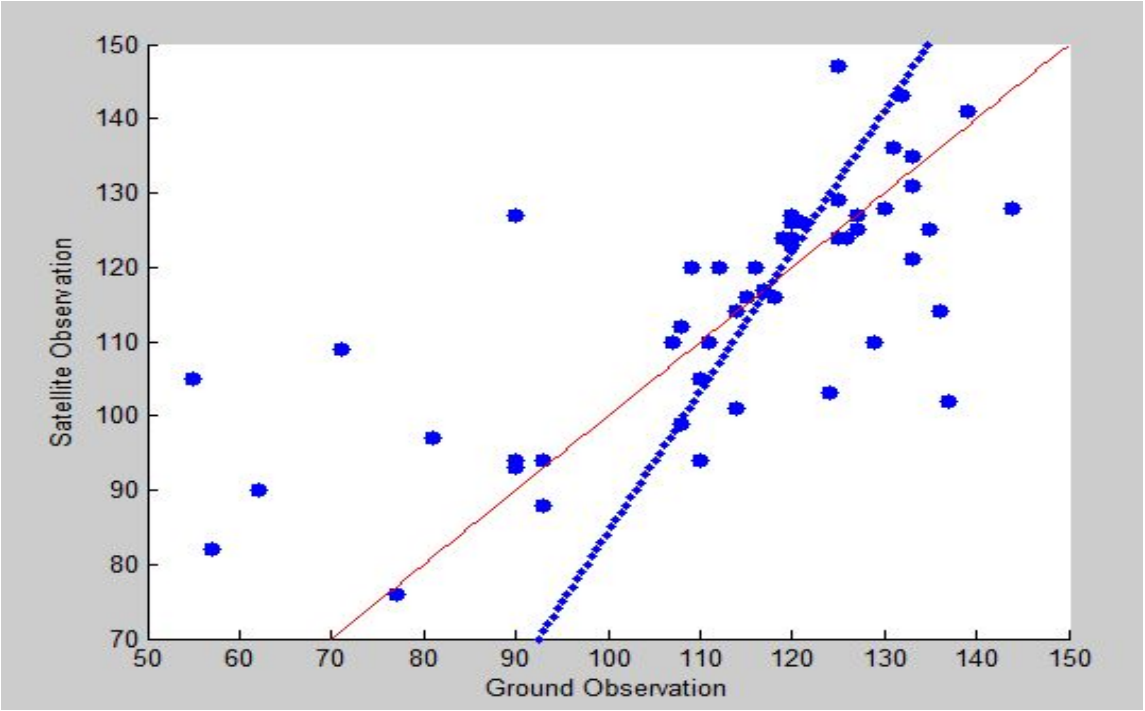
- **Y= F(x) = 0.5284*x + 55.45**

3. Plot of Regression Model:



Y= F(x) = 0.5284*x + 55.45

4. Scatter plot showing both the Regression Line and One-One line:



Visual Analysis:

- The regression line is close to the one-one line which shows that the two data sets are closely related.
- Positive slope of both the lines shows that as ground observation increases, satellite observation increases too and thus there is positive correlation.
- Since there is little difference in the slopes of both regression and one-one line, the data-sets are strongly related.
- We will further justify these observations by calculating the correlation coefficient.

5. Correlation Coefficient:

A. What is Correlation Coefficient?

- The sample correlation coefficient, denoted **r**, ranges between -1 and +1 and quantifies the direction and strength of the linear association between the two variables. The correlation between two

variables can be positive (i.e., higher levels of one variable are associated with higher levels of the other) or negative (i.e., higher levels of one variable are associated with lower levels of the other).

- The sign of the correlation coefficient indicates the direction of the association. The magnitude of the correlation coefficient indicates the strength of the association.
- A correlation close to zero suggests no linear association between two continuous variables.

B. **Implementation of Correlation Coefficient and Observations:**

- We have found out the correlation coefficient between the ground data and satellite data by using " corrcoef " function in MATLAB which returns a matrix representing the r values.
- The element Aij represents the r value of i with respect to data j.
- **Observations:** The value found from coefficient matrix comes out to be **0.7089** which shows strong positive correlation between the given data-set of ground and satellite data.
- **Correlation Coefficient Matrix:**

```
R =

    1.0000    0.7089
    0.7089    1.0000
```

ASSIGNMENT-2

A)AIC / BIC model selection:

- Akaike's Information Criterion (AIC) provides a measure of model quality by simulating the situation where the model is tested on a different data set. After computing several different models, you can compare them using this criterion. According to Akaike's theory, the most accurate model has the smallest AIC.
- Similarly, lower the value of BIC, better is the model. BIC is also used for penalizing data points used for model fitting.
- The known model that we are using is:

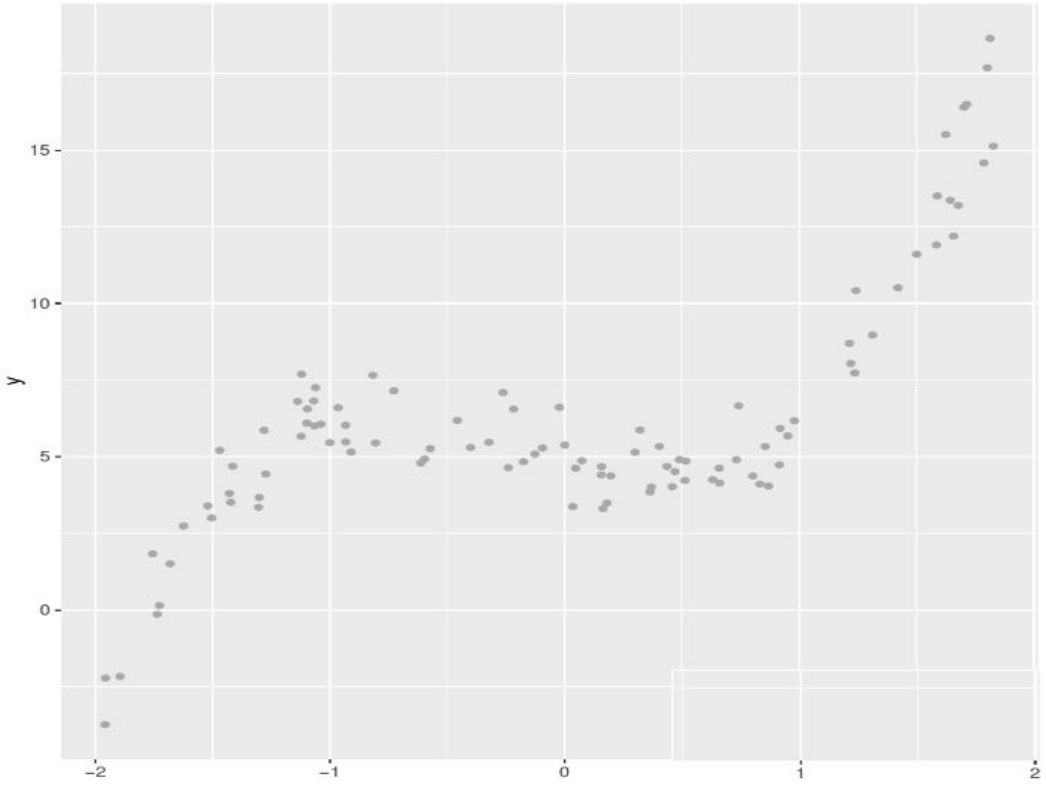
Y= 5 + 2*x + x*x + 2*x*x*x + E { E – normal (mean =0, variance = 1) }

It is clearly a third degree polynomial and thus in the result we should get minimum values of AIC and BIC for 3rd degree polynomial**.**

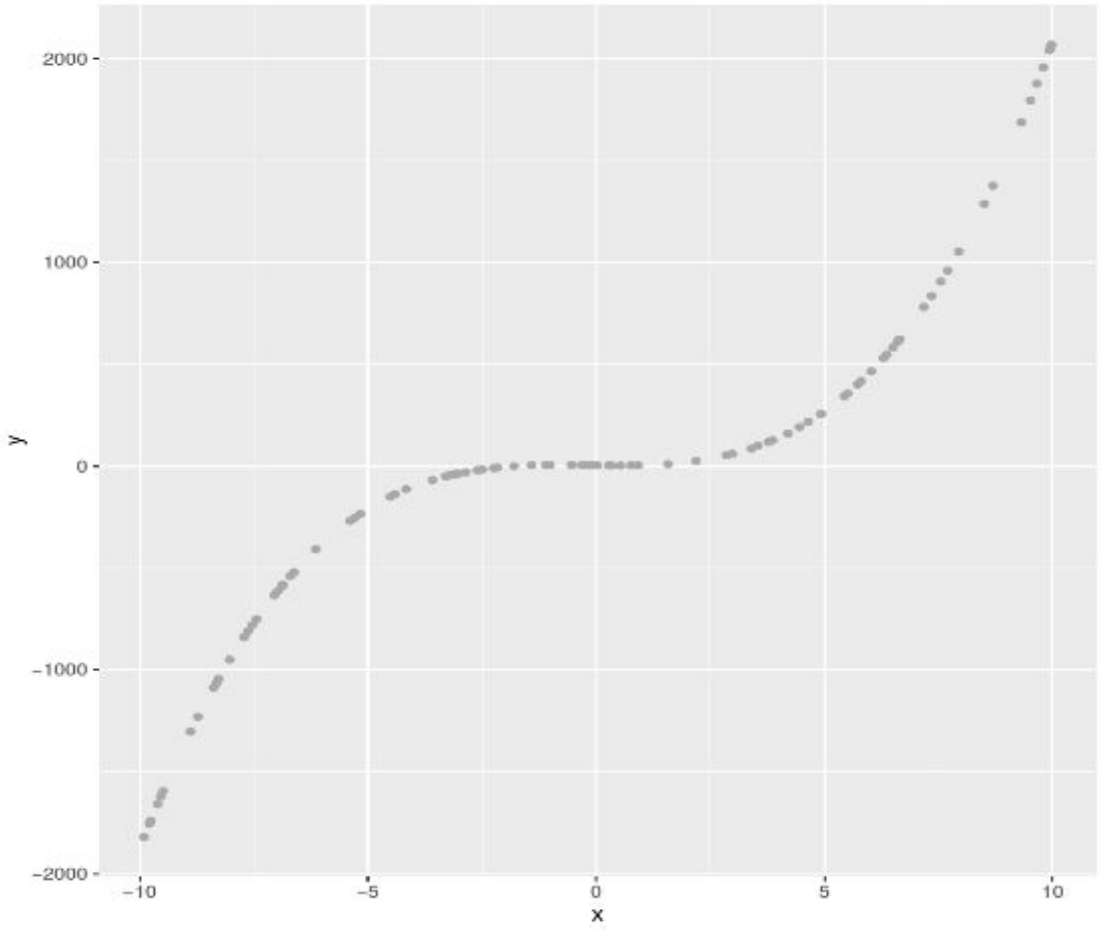
1.Data generated:

For x- values uniformly distributed between :

1. [-2,2]

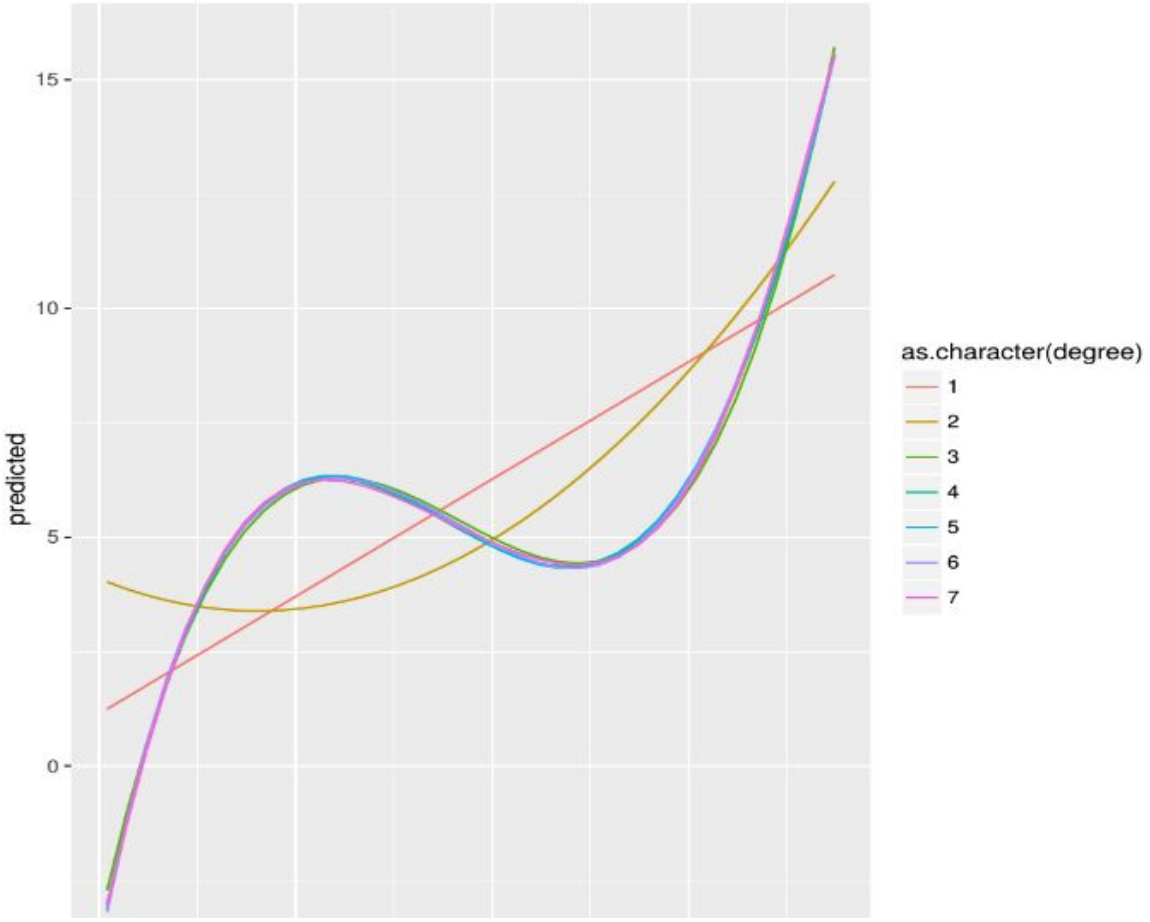


2. **[-10,10]**

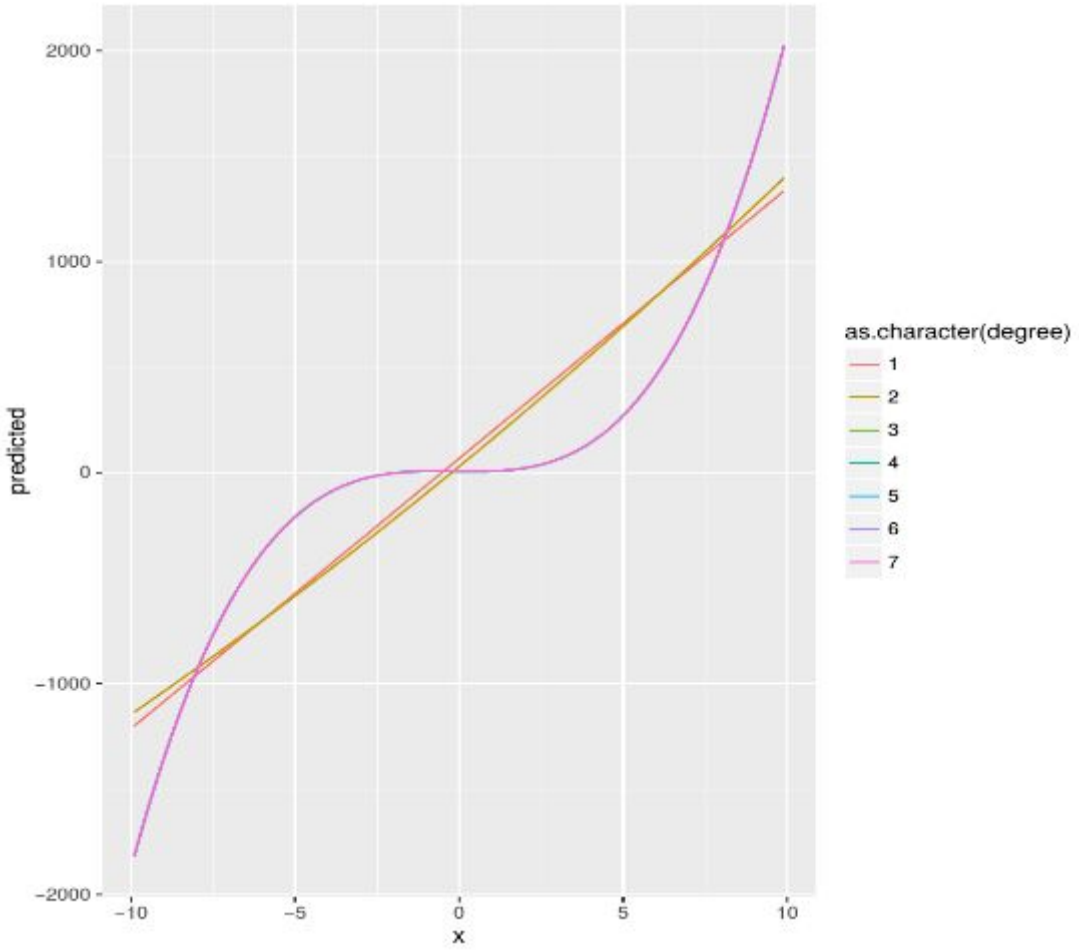


Graph of different degree models:

1. **[-2,2]**

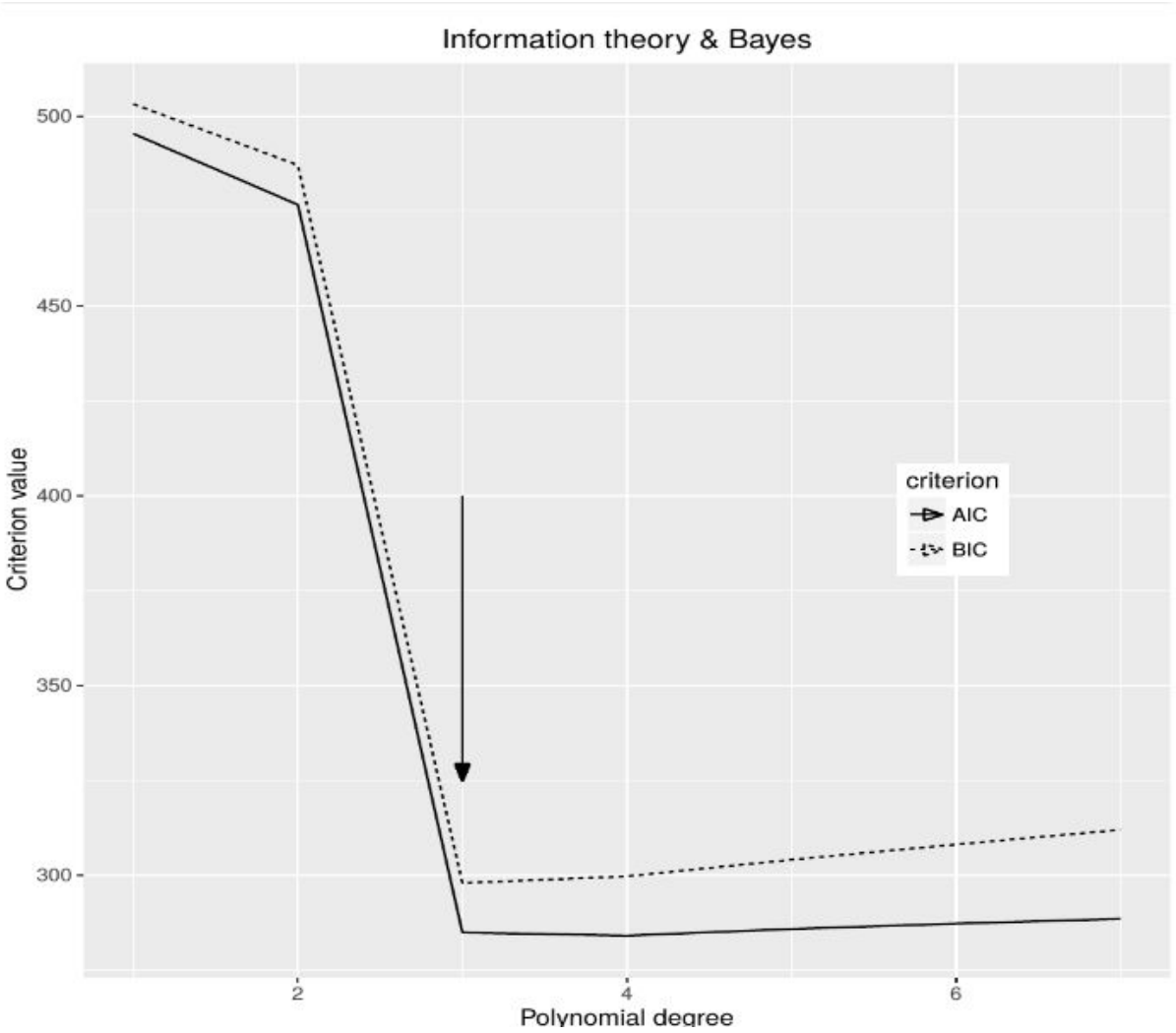


2. [-10,10]

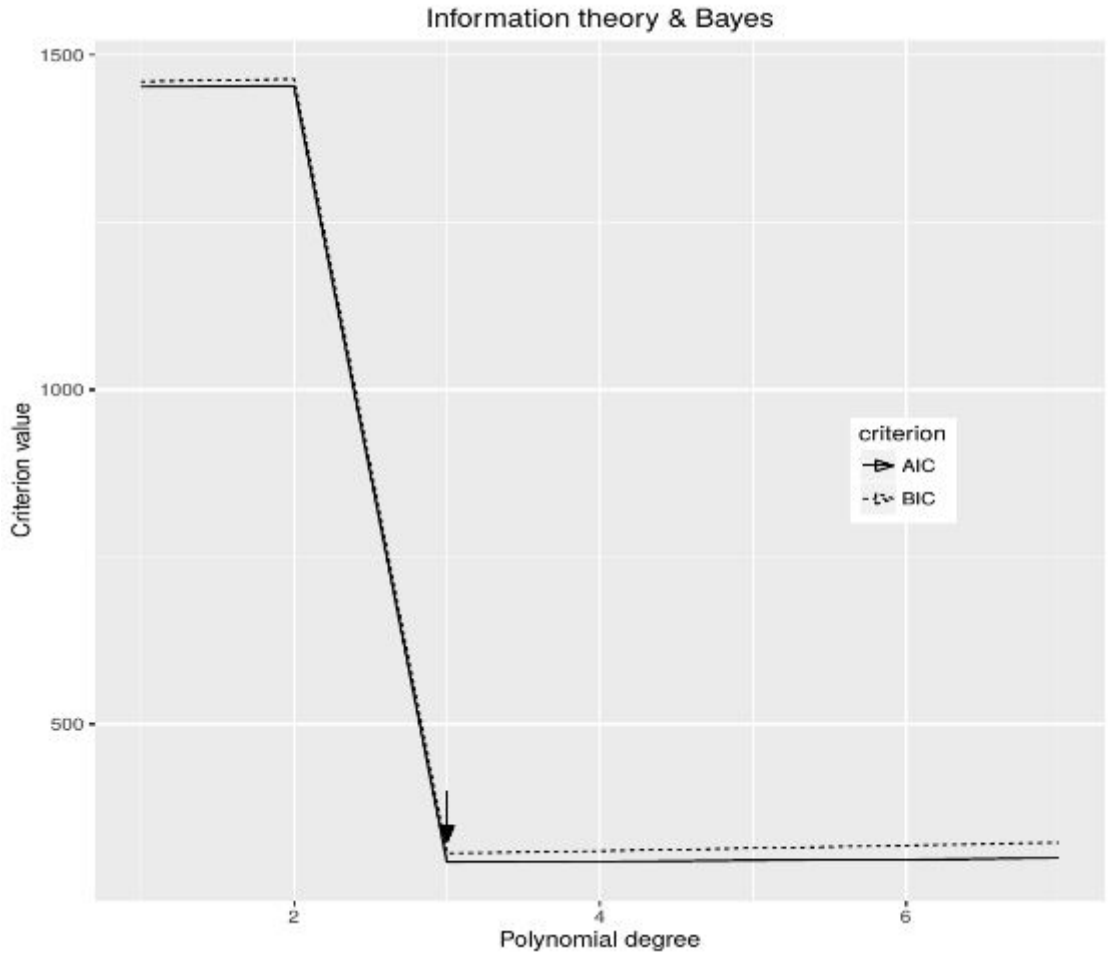

2000
1000
0
−1000
−2000
predicted
−10
−5
0
5
10
x
as.character(degree)
1
2
3
4
5
6
7

3. **Plot of AIC BIC vs Degree of Polynomial :**

    1. **[-2,2]**



2. **[-10,10]**



Observation:

- We can clearly see that for both data-range the minimum value of both AIC and BIC is at 3rd Degree Polynomial, which means that the 3rd degree polynomial is the best fit for the given model.
- We see that both AIC and BIC take a sharp dip and decrease till 3rd degree and then after 3rd degree start increasing, indicating lower fitting models as we further increase the polynomial degree.

Conclusion:

- We are correctly able to identify the optimum model by using AIC and BIC values.
- This is used for model selection whenever the model is not known previously.

Code Snippets:

```r
#R version 3.3.2

# the figures require ggplot2 library and
# all packages it depends on
library(ggplot2)

# generate the x predictor
x <- runif(100,-10,10)
# generate the y response
y <- 2*x^3 + x^2 - 2*x +5 + rnorm(100)
xy <- data.frame(x=x, y=y)
# specify the maximum polynomial degree that will be explored
max.poly <- 7

# cretaing data.frame which will store model predictions
# that will be used for the smooth curves in Fig. 1
x.new <- seq(min(x), max(x), by=0.1)
degree <- rep(1:max.poly, each=length(x.new))
predicted <- numeric(length(x.new)*max.poly)
new.dat <- data.frame(x=rep(x.new, times=max.poly),
                      degree,
                      predicted)

# fitting lm() polynomials of increasing complexity
# (up to max.degree) and storing their predictions
# in the new.dat data.frame
for(i in 1:max.poly)
{
  sub.dat <- new.dat[new.dat$degree==i,]
  new.dat[new.dat$degree==i,3] <- predict(lm(y~poly(x, i)),
                                          newdata=data.frame(x=x.new))
}

# plotting the data and the fitted models
p <- ggplot()
p + geom_point(aes(x, y), xy, colour="darkgrey")
p + geom_line(aes(x, predicted,
                  colour=as.character(degree)),
```

```
                    colour=as.character(degree)),
                    new.dat)
p + scale_colour_discrete(name = "Degree")
p

# creating empty data.frame that will store
# AIC and BIC values of all of the models
AIC.BIC <- data.frame(criterion=c(rep("AIC",max.poly),
                                  rep("BIC",max.poly)),
                      value=numeric(max.poly*2),
                      degree=rep(1:max.poly, times=2))

# calculating AIC and BIC values of each model
for(i in 1:max.poly)
{
  AIC.BIC[i,2] <- AIC(lm(y~poly(x,i)))
  AIC.BIC[i+max.poly,2] <- BIC(lm(y~poly(x,i)))
}


# plotting AIC and BIC against model complexity
# (which is the polynomial degree)
AIC.plot <- qplot(degree, value, data=AIC.BIC,
                  geom="line", linetype=criterion) +
                  xlab("Polynomial degree") +
                  ylab("Criterion value") +
                  labs(title="Information theory & Bayes")+
                  geom_segment(aes(x=3, y=400,
                                   xend=3, yend=325),
                  arrow = arrow(length = unit(0.3, "cm"),
                  angle=20, type="closed")) +
                  theme(legend.position=c(0.8,0.5))
AIC.plot
```

B)Implement K-Means and Fuzzy C -Means for:
i)Iris dataset
ii) Satellite image

K-Means:

- K-means is an **unsupervised learning algorithm** that solve the clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed a priori. The main idea is to define k centroids, one for each cluster.

- The **algorithm** is composed of the following steps:

1. *Place K points into the space represented by the objects that are being clustered. These points represent initial group centroids.*

2. *Assign each object to the group that has the closest centroid.*

3. *When all objects have been assigned, recalculate the positions of the K centroids.*

4. *Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.*

- There are some methods for choosing an accurate value of K(i.e the number of clusters).

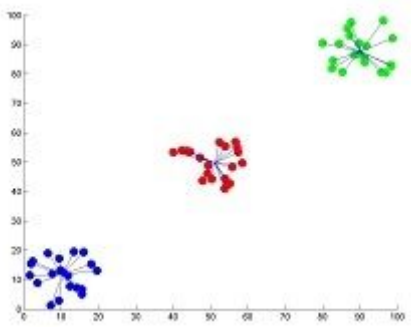
1. ***Elbow method:****mean distance to the centroid as a function of K is plotted and the "elbow point," where the rate of decrease sharply shifts, can be used to roughly determine K*

2. *information criteria*
3. *information theoretic jump method*
4. *silhouette method*
5. *G-means algorithm*

Advantages

1) Fast, robust and easier to understand.

2) Gives best result when data set are distinct or well separated from each other.



K-means for

i) Iris dataset

- Then following the algorithm, the code classifies the input data as one of the three clusters.
- In the code , firstly k centroids for each cluster is chosen randomly
- Until convergence,nearest centroid to each data point is calculated and assigned to the cluster
- For each cluster,new centroid is calculated
- The above cycle is repeated until the maximum iterations is reached or the error is below a certain value(here,0.005)
- Here, we have chosen the number of clusters to be 3(i.e k=3)

  Maximum iterations=100

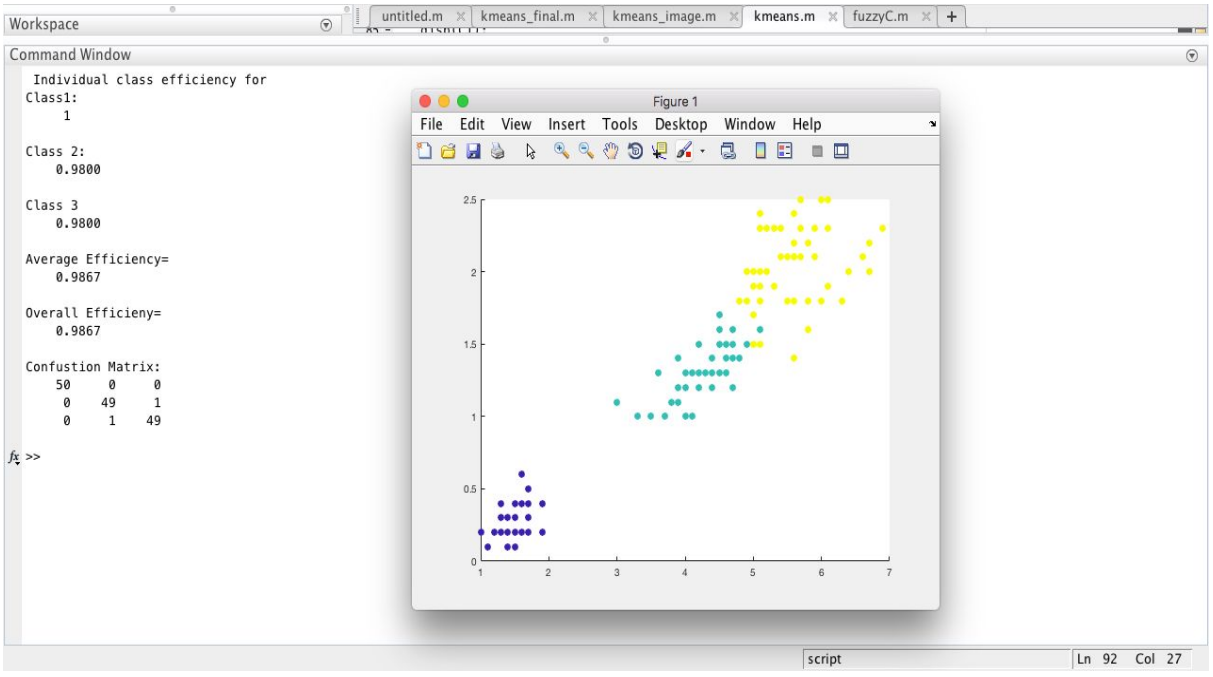
- **Code Snippet:**

```
clc;
load iris_dat.dat;
  X = iris_dat;
K=3;
maxIter=100;
TOL=0.005;

% number of vectors in X
[vectors_num, dim] = size(X);

% compute a random permutation of all input vectors
R = randperm(vectors_num);

%construct indicator matrix (each entry corresponds to the cluster of each point in X)
I = zeros(vectors_num, 1);

% construct centers matrix
C = zeros(K, dim);

% take the first K points in the random permutation as the center sead
for k=1:K
    C(k,:) = X(R(k),:);
end

iter = 0;     % iteration count
% compute new clustering while the cumulative intracluster error in kept
% below the maximum allowed error, or the iterative process has not
% exceeded the maximum number of iterations permitted
while 1
    % find closest point
    for n=1:vectors_num
        % find closest center to current input point
```

```
        minIdx = 1;
        minVal = norm(X(n,:) - C(minIdx,:), 1);
        for j=1:K
            dist = norm(C(j,:) - X(n,:), 1);
            if dist < minVal
                minIdx = j;
                minVal = dist;
            end
        end

        % assign point to the closter center
        I(n) = minIdx;
    end

    % compute centers
    for k=1:K
        C(k, :) = sum(X(find(I == k), :));
        C(k, :) = C(k, :) / length(find(I == k));
    end

    % compute RSS error
    RSS_error = 0;
    for idx=1:vectors_num
        RSS_error = RSS_error + norm(X(idx, :) - C(I(idx),:), 2);
    end
    RSS_error = RSS_error / vectors_num;

    % increment iteration
    iter = iter + 1;

    % check stopping criteria
    if 1/RSS_error < TOL
```

```
64 -        if 1/RSS_error < TOL
65 -            break;
66 -        end
67
68 -        if iter > maxIter
69 -            iter = iter - 1;
70 -            break;
71 -        end
72 -    end
73 -    C = confusionmat(I,X(:,5));
74 -    C1 = C(1,1)/50;
75 -    C2 = C(2,2)/50;
76 -    C3 = C(3,3)/50;
77 -    disp(' Individual class efficiency for');
78 -    disp('Class1:');
79 -    disp(C1);
80 -    disp('Class 2:')
81 -    disp(C2);
82 -    disp('Class 3');
83 -    disp(C3);
84 -    SUM = C1 + C2 + C3;
85 -    SUM1 = SUM/3;
86 -    disp('Average Efficiency=');
87 -    disp(SUM1);
88 -    SUM2=(C(1,1)+C(2,2)+C(3,3))/150;
89 -    disp('Overall Efficieny=');
90 -    disp(SUM2);
91 -    disp('Confustion Matrix:');
92 -    disp(C);
93
94 -    scatter(X(:,3),X(:,4),50,I,'filled');
95
```

- <u>Output</u>



Individual class efficiency for
Class1:
1
Class 2:
0.9800
Class 3
0.9800
Average Efficiency=
0.9867
Overall Efficiency=
0.9867

Confusion Matrix:

50	0	0
0	49	1
0	1	49

- The efficiency obtained is thus very high indicating that a high number of input data were classified correctly
- On increasing the number of iterations,a higher efficiency is obtained
- Also the figure shows the data points being part of 3 different clusters

ii) Satellite Image

TO EXTRACT WATER FROM THE SATELLITE IMAGE USING K MEANS CLUSTERING.

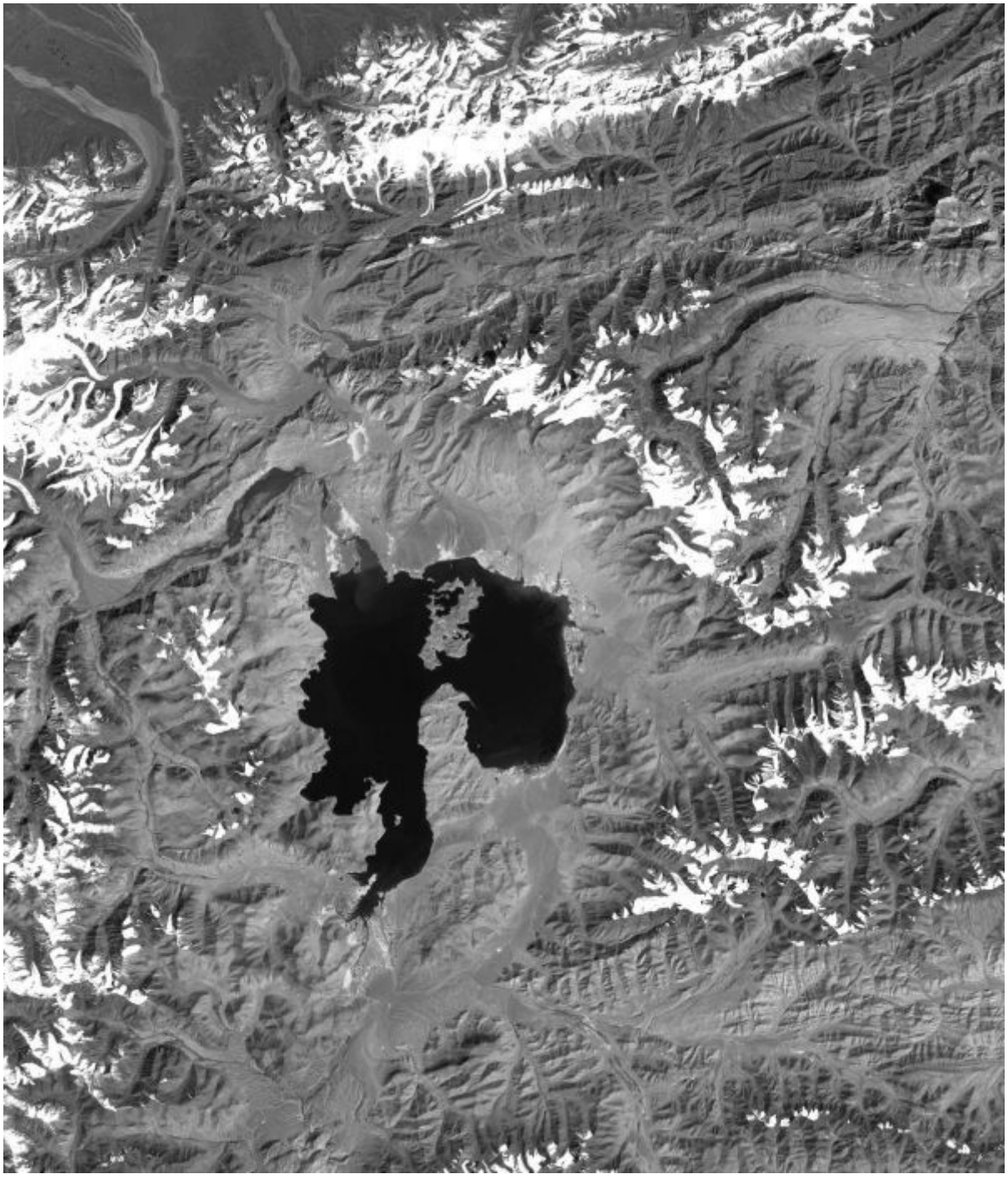
Code Snippet

```
clear all;
clc;
no_centroids = 3;

img = imread('Landsat.jpg');
img(:,:,2:3) = [];
img = cast(img,'int32');
centroid = [0,40,180];
answer = zeros(701,600);
answer = cast(answer,'int32');
count = zeros(3);
sum = zeros(3);

for it = 1:1
for i = 1:701
      for j=1:600
          min = abs(centroid(1) - img(i,j));
          answer(i,j) = 1;

          for k=2:3
             temp = abs(centroid(k)-img(i,j));
              if temp<min
                  min = temp;
                  answer(i,j) = k;
              end
```

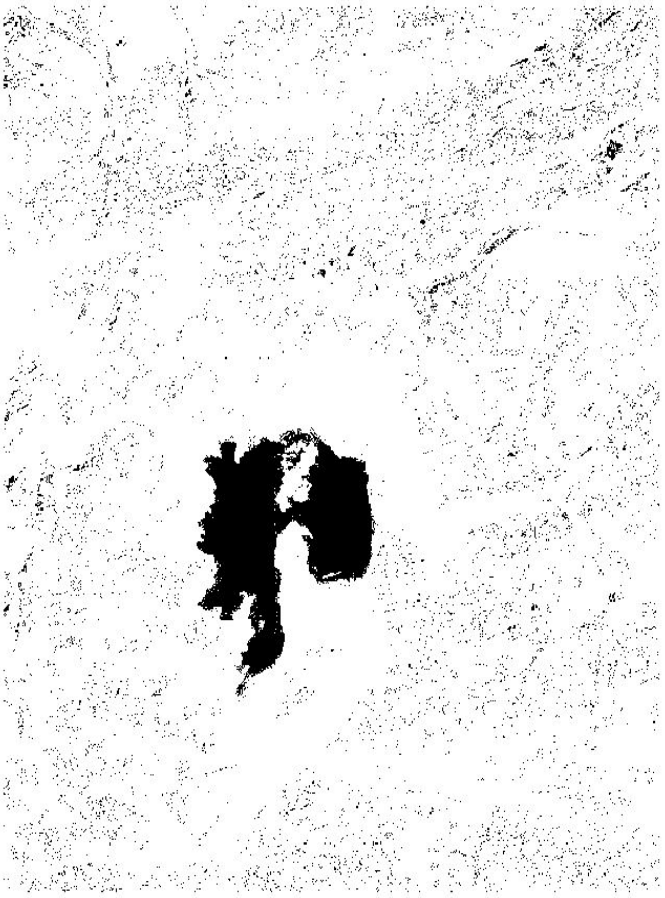
```
            end

            count(answer(i,j)) = count(answer(i,j)) + 1;
            sum(answer(i,j)) = sum(answer(i,j)) + img(i,j);

        end
    end

    centroid = sum./count;

    end


    for i = 1:701
        for j = 1:600
            if answer(i,j) == 2
                answer(i,j) = 3;
            end
        end
    end

    ans_img = mat2gray(answer,[1 3])+40;
    imshow(ans_img);
    %BW = imbinarize(ans_img);
```

- **OUTPUT-**

ORIGINAL IMAGE



OUTPUT AFTER CLUSTERING



DRAWBACKS OF K MEANS-

- Euclidean distance is used as a metric and variance is used as a measure of cluster scatter.
- The number of clusters *k* is an input parameter: an inappropriate choice of *k* may yield poor results. That is why, when performing k-means, it is important to run diagnostic checks for determining the number of clusters in the data set.

- A key limitation of *k*-means is its cluster model. The concept is based on spherical clusters that are separable in a way so that the mean value converges towards the cluster center. The clusters are expected to be of similar size, so that the assignment to the nearest cluster center is the correct assignment.

Fuzzy C-Means

- In Fuzzy C-means (FCM) clustering a dataset is grouped into n clusters with every data point in the dataset belonging to every cluster to a certain degree.
- Fuzzy C-Means is similar to K-means. The only difference is, instead of assigning a data point exclusively to only one cluster, it can have some sort of fuzziness or overlap with other clusters.

The **algorithm** is composed of the following steps:

1. *Initialize $U=[u_{ij}]$ matrix, $U^{(0)}$*

2. *At k-step: calculate the centers vectors $C^{(k)}=[c_j]$ with $U^{(k)}$*

$$c_j = \frac{\sum_{i=1}^{N} u_{ij}^{m} \cdot x_i}{\sum_{i=1}^{N} u_{ij}^{m}}$$

3. *Update $U^{(k)}$, $U^{(k+1)}$*

$$u_{ij} = \frac{1}{\sum_{k=1}^{C} \left(\frac{\|x_i - c_j\|}{\|x_i - c_k\|} \right)^{\frac{2}{m-1}}}$$

4. *If $|| U^{(k+1)} - U^{(k)} || < \varepsilon$ then STOP; otherwise return to step 2.*

Advantages

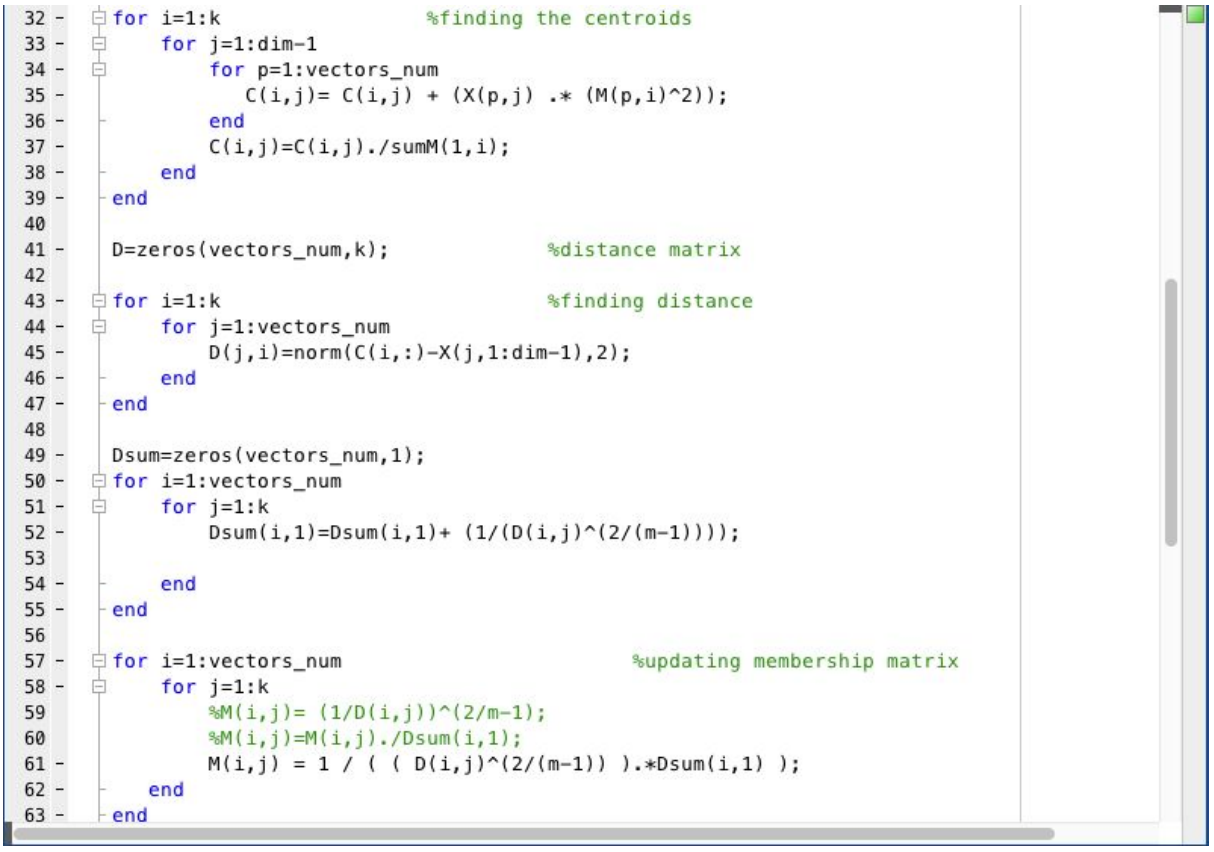
- Allows a data point to be in multiple clusters.
- Algorithmically easy to implement

Fuzzy C-means for
i) Iris dataset

- First, the membership matrix is initialized with random values such that the sum of row equals 1.
- Then, Centroid for each cluster is calculated
- dissimilarly between the data points and Centroid using the Euclidean distance is calculated
- Membership matrix is then updated using the new values

Code Snippet

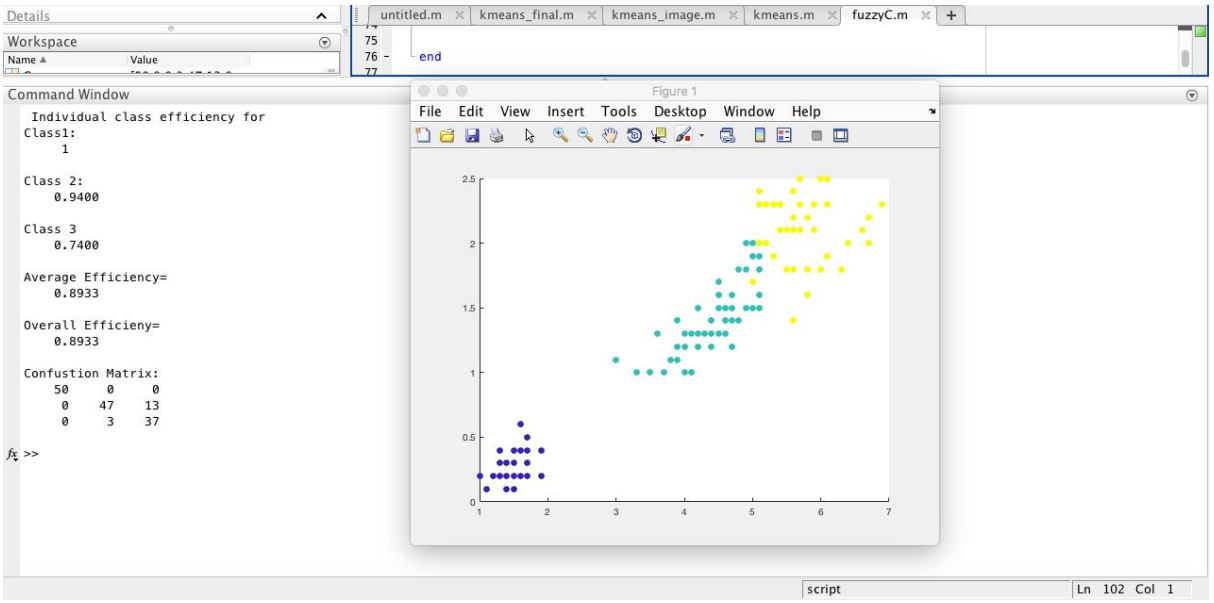
```
%fuzzyC implementation
clc;
load iris_dat.dat;
  X = iris_dat;

[vectors_num, dim] = size(X);

M=zeros(vectors_num,3); %membership matrix

r1 =rand(vectors_num,1);
r2 =rand(vectors_num,1);
r3 =rand(vectors_num,1);
sum=r1+r2+r3;

M(:,1)=r1./sum;       %initialising membership matrix with random values
M(:,2)=r2./sum;
M(:,3)=r3./sum;

k=3;                %no. of clusters;
m=2;                %fuzziness index;
max_itr=10;         %maximum iterations

for q=1:max_itr

C=zeros(k,dim-1);        %centroid matrix
sumM=zeros(1,3);
for i=1:k
    for j=1:vectors_num
        sumM(1,i)= sumM(1,i)+ M(j,i)^2;
    end
end
for i=1:k                %finding the centroids
```



```
for i=1:k                %finding the centroids
    for j=1:dim-1
        for p=1:vectors_num
           C(i,j)= C(i,j) + (X(p,j) .* (M(p,i)^2));
        end
        C(i,j)=C(i,j)./sumM(1,i);
    end
end

D=zeros(vectors_num,k);             %distance matrix

for i=1:k                           %finding distance
    for j=1:vectors_num
        D(j,i)=norm(C(i,:)-X(j,1:dim-1),2);
    end
end

Dsum=zeros(vectors_num,1);
for i=1:vectors_num
    for j=1:k
        Dsum(i,1)=Dsum(i,1)+ (1/(D(i,j)^(2/(m-1))));

    end
end

for i=1:vectors_num                      %updating membership matrix
    for j=1:k
        %M(i,j)= (1/D(i,j))^(2/m-1);
        %M(i,j)=M(i,j)./Dsum(i,1);
        M(i,j) = 1 / ( ( D(i,j)^(2/(m-1)) ).*Dsum(i,1) );
   end
end
```

```
64 -    max_itr=max_itr-1;
65 -    end
66
67 -    I=zeros(vectors_num,1);
68
69 -    for i=1:vectors_num              %assigning to cluster with highest value in membership
70
71 -           [U,I(i,1)]=max(M(i,:));
72 -    end
73
74 -    C = confusionmat(I,X(:,5));
75 -    C1 = C(1,1)/50;
76 -    C2 = C(2,2)/50;
77 -    C3 = C(3,3)/50;
78 -    disp(' Individual class efficiency for');
79 -    disp('Class1:');
80 -    disp(C1);
81 -    disp('Class 2:')
82 -    disp(C2);
83 -    disp('Class 3');
84 -    disp(C3);
85 -    SUM = C1 + C2 + C3;
86 -    SUM1 = SUM/3;
87 -    disp('Average Efficiency=');
88 -    disp(SUM1);
89 -    SUM2=(C(1,1)+C(2,2)+C(3,3))/150;
90 -    disp('Overall Efficieny=');
91 -    disp(SUM2);
92 -    disp('Confustion Matrix:');
93 -    disp(C);
94
95 -    scatter(X(:,3),X(:,4),50,I,'filled');
```

Output



- What we analys from the output is that the class 1 is clustered completely and the class 2 and class 3 are overlapping.

Class1:

1

Class 2:

0.9400

Class 3

0.7400

Average Efficiency=

0.8933

Overall Efficiency=

0.8933

Confusion Matrix:

50 0 0

0 47 13

0 3 47

ii)SATELLITE IMAGE

- AIM- TO EXTRACT WATER FROM THE SATELLITE IMAGE USING FUZZY C MEANS CLUSTERING.

Implementation-

1) For implementation first we create a “for_fuzzycmeans” as a generic function for clustering data using fuzzy c means method.
2) The arguments to pass to this function are the data matrix and the number of clusters.

Code Snippet

```
function [ U, centers ] = for_fuzzyCmeans( data, n_clusters
    nRows = size(data, 1);
    nDim = size(data, 2);
    nIteration = 10;
    m = 2;

    U = zeros(nRows, n_clusters);
    centers = zeros(n_clusters, nDim);

    for i=1:nRows
        U(i,:) = rand(1,n_clusters);
        total = sum(U(i,:));
        U(i,:) = U(i,:)/total;
    end
```

```
    for itr=1:nIteration

        dist = zeros(nRows, n_clusters);

        for c=1:n_clusters
            temp1 = U(:,c).^m;
            temp2 = temp1*ones(1,nDim);
            temp2 = data(:,:) .* temp2;
            temp2 = sum( temp2 );
            temp1 = sum( temp1 );
            centers(c,:) = temp2 ./ temp1;

            temp1 = data(:,:)-(ones(nRows,1)*centers(c,:));
            temp1 = temp1 .* temp1;
            temp1 = temp1*ones(nDim,1);
            dist(:,c) = sqrt(temp1);
        end
```

```
35 -          for i=1:nRows
36 -              temp=0;
37 -              for k=1:n_clusters
38 -                  temp = temp + 1/( dist(i,k) ^ (2/(m-1)) );
39 -              end
```

In this function we calculate the distance of the points from the clusters using Euclidean distance formula.
We return the matrix containing indices and the centroids.

Next we Load the image and call the for_fuzzycmeans function to get the cluster indexes.

```
1 -  clc
2 -  image = imread('Landsat.jpg');
3
4 -  image = rgb2gray(image);
5
6 -  image = double(image(:,:));
7
8 -  nrows = size(image,1);
9 -  ncols = size(image,2);
10
11 -  image = reshape(image,nrows*ncols,1);
12
13 -  nColors = 3;
14
15 -  [cluster_idx, cluster_center] = my_fuzzyCmeans(image,nColors);
16
```

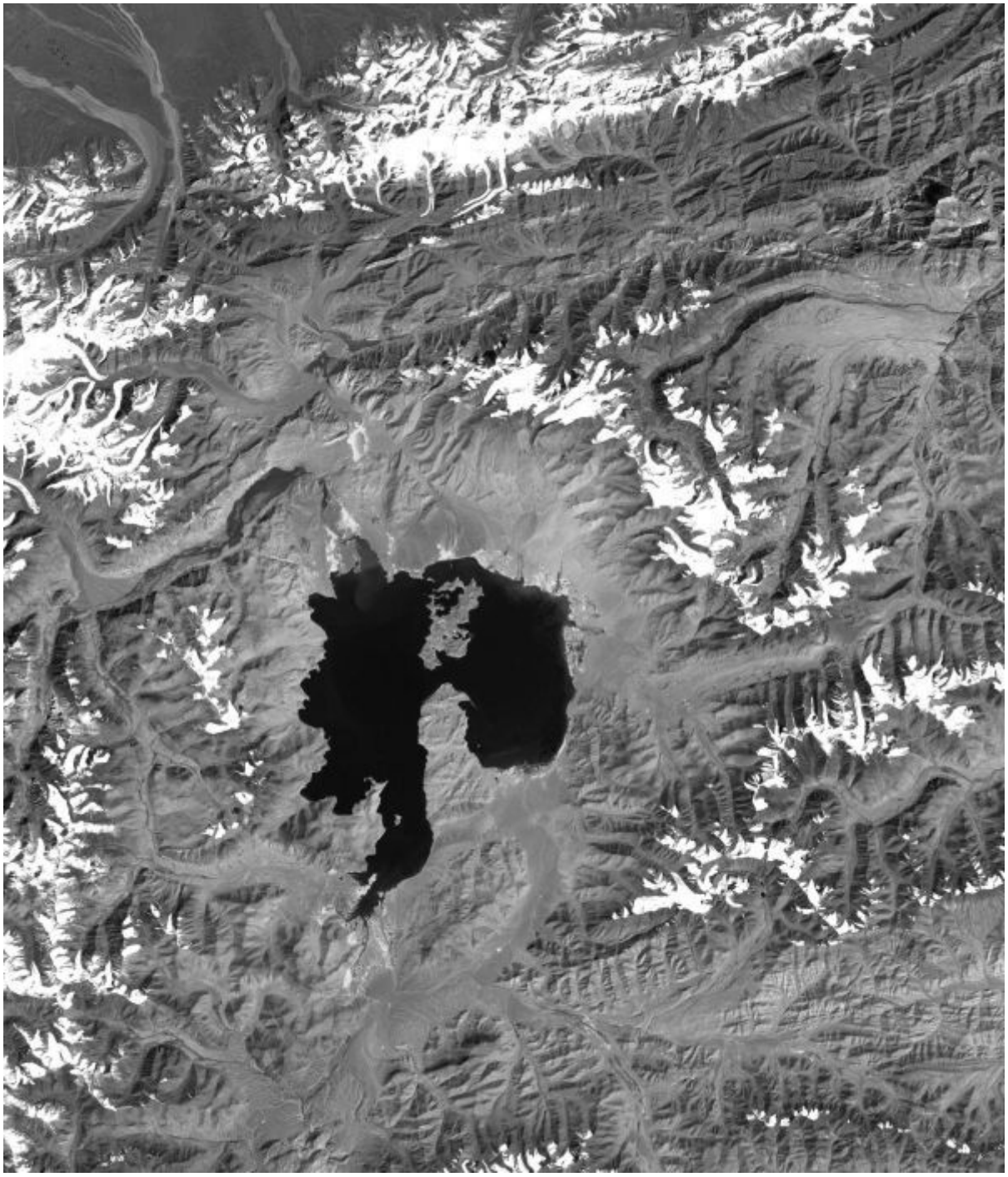
```
16
17 -   min=1;
18 -  for i=1:nColors
19 -      if cluster_center(i)<cluster_center(min)
20 -          min = i;
21 -      end
22 -  end
23
24 -   class = zeros(nrows*ncols, 1);
25
26 -  for i=1:nrows*ncols
27 -      if cluster_idx(i,min)>0.7 && cluster_idx(i,min)<0.8
28 -          class(i) = 1;
29 -      else
30 -          class(i) = 3;
```

```
23
24 -    class = zeros(nrows*ncols, 1);
25
26 -    for i=1:nrows*ncols
27 -        if cluster_idx(i,min)>0.7 && cluster_idx(i,min)<0.8
28 -            class(i) = 1;
29 -        else
30 -            class(i) = 3;
31 -        end
32 -    end
33
34 -    cluster_idx = class;
35
36 -    pixel_labels = reshape(cluster_idx,nrows,ncols);
37
```

After we get the indices from clustering we process the cluster centres and calculate the minimum.
For simplicity we make all the indexes with value greater than 0.7 and less than 0.8 as 1 and rest all as 3.
We then plot the image using pixel_label function.

Output

Original Image



After Clustering

FURTHER IMPROVEMENTS-

- For further improving the quality of extraction and getting a perfect outline of water we can use inbuilt image processing functions for boundary recognition and then highlighting the area inside the boundary.

Drawbacks of Fuzzy C-Means

- Need to define c, the number of clusters.
- Need to determine membership cutoff value and the fuzzy set rules.

- Converges to a local minimum.
- Clusters are sensitive to initial assignment of centroids.

- Fuzzy c-means is not a deterministic algorithm.

ANALYSIS FOR K MEANS AND FUZZY C MEANS CLUSTERING:

RUNNING TIME ANALYSIS-