Table of Contents

..... 1
Part 2 1
Part 3 2
Part 4 3
Part 5 5
Part 6 5
Part 7 7
km.m 8
pointclouds.m 9
pointrings.m 10
im2rgb.m 10
rgb2im.m 11

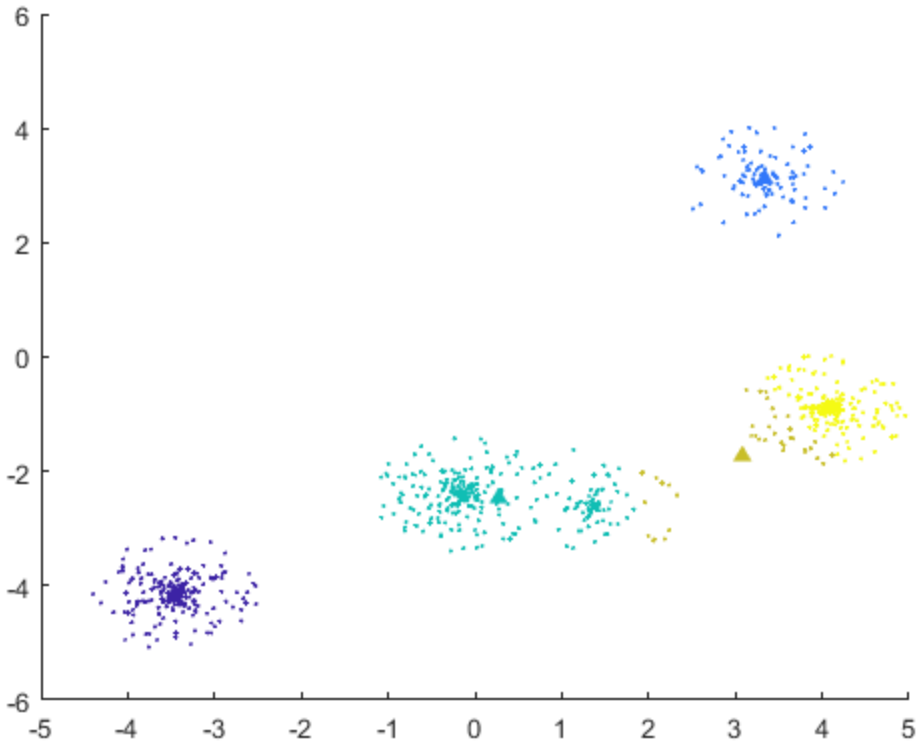

```
clear; clc;
```

Part 2

Examine the data generated by pointclouds.m. In particular, make sure to look at a scatter plot of the data for example by using scatter(X(1,:),X(2,:),1,Y). Finally, use the k-means function to classify the data generated by pointclouds.m into five clusters.
****

We plot the points in different colors corresponding to their labels. We also plot the means as triangles of the same color. Sorry yellow is hard to see, I don't know how to change it in Matlab.

```
X = pointclouds();
k = 5;
[mu, labels] = km(X, k);

figure()
hold on
scatter(X(1,:), X(2,:),1, labels);
scatter(mu(1,:), mu(2,:),[], 1:k, '^', 'filled');
hold off
```

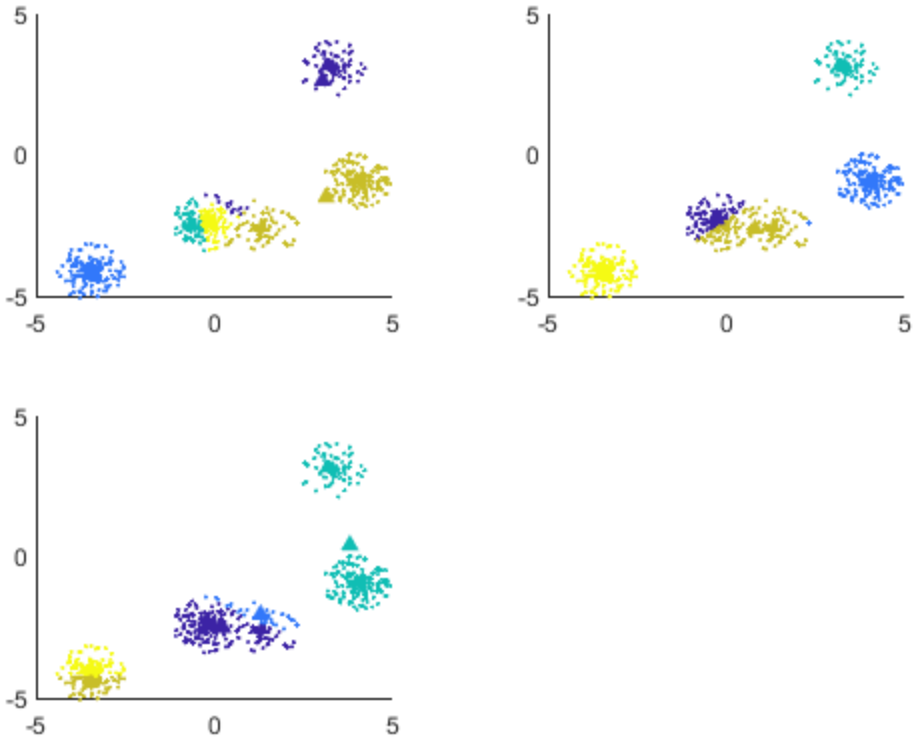


Part 3

Generate and turn in the classification result of your code for three separate runs of the k-means algorithm that lead to different results. This should be in the form of three scatter plots with the class assignments visually distinguished.
****

It works different each time. I.e it finds different means because of the random initialization.

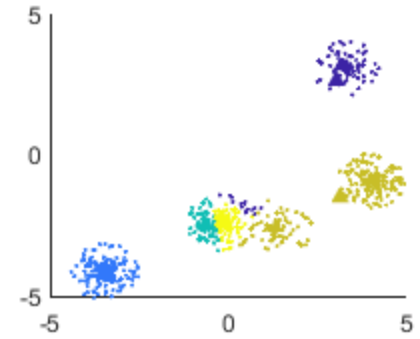
```
iter = 3;
figure()
tiledlayout(iter, 1);
axis equal;
for i = 1:iter
    [mu, labels] = km(X, k);

    nexttile;
    hold on
    scatter(X(1,:), X(2,:),1, labels);
    scatter(mu(1,:), mu(2,:),[], 1:k, '^', 'filled');
    hold off
end
```

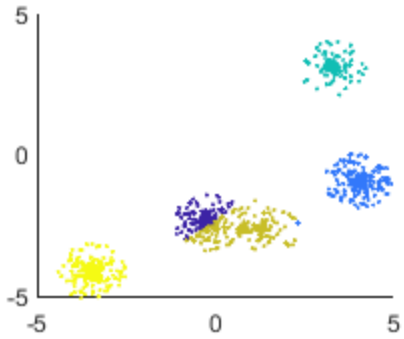


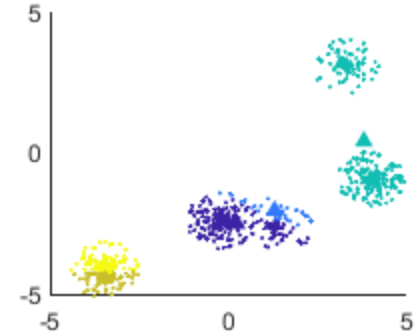
Part 4

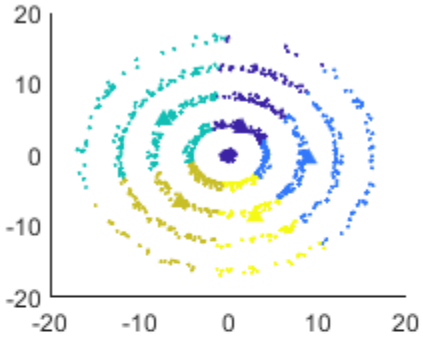
Do the same for the pointrings.m dataset, again producing three distinct classification result figures.
****

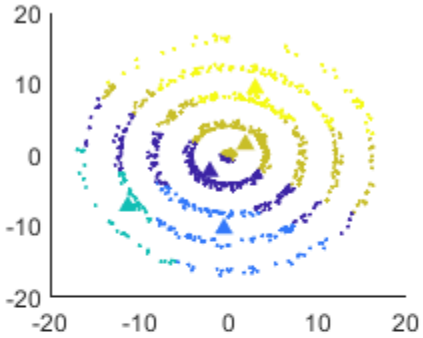
```
X = pointrings();

iter = 3;
figure()
tiledlayout(iter, 1);
axis equal;
for i = 1:iter
    [mu, labels] = km(X, k);

    nexttile;
    hold on
    scatter(X(1,:), X(2,:),1, labels);
    scatter(mu(1,:), mu(2,:),[], 1:k, '^', 'filled');
    hold off
end
```

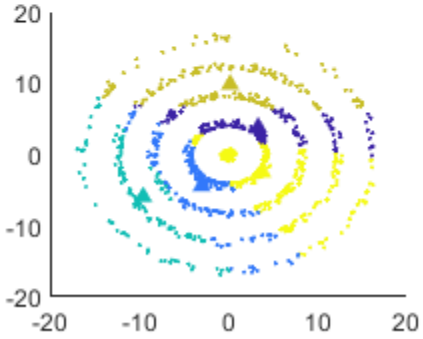

5
0
-5
-5
0
5


5
0
-5
-5
0
5


5
0
-5
-5
0
5


20
10
0
-10
-20
-20
-10
0
10
20


20
10
0
-10
-20
-20
-10
0
10
20


20
10
0
-10
-20
-20
-10
0
10
20

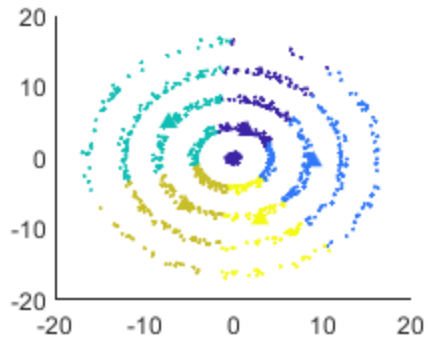
Part 5

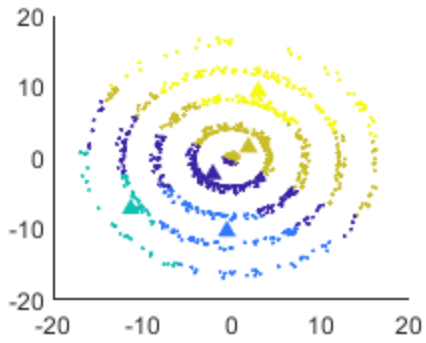
Which dataset do you believe k-means performed a better job clustering, on average? Why do you believe this is the case?
****
I think it did better on the point clouds dataset. For the rings, we probably want to cluster each ring into a group, but k means divides up the rings like a pie. While it had a tendancy not to get the actual clusters correct each time for pointclouds, it still did a pretty good job and typically made some clusters similarly. While on the rings it performs very different and creates clusters that don't really reflect the physical characteristics.

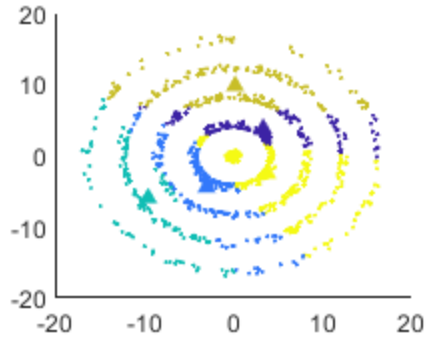
Part 6

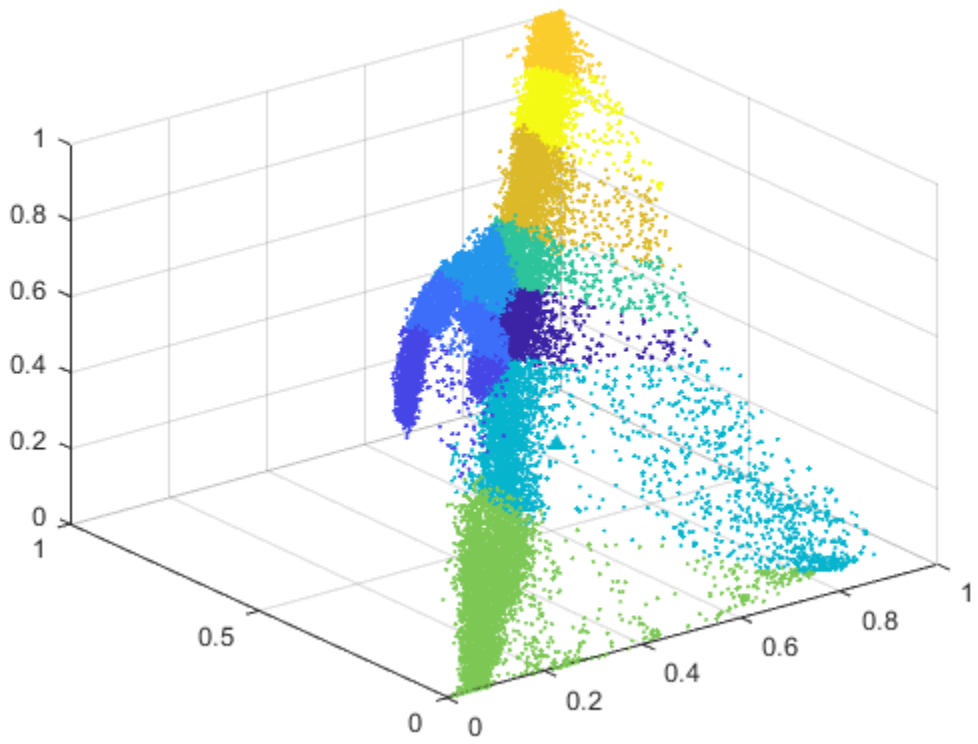
Use the function im2rgb.m to read the pixel RGB values of either of the provided images (or pick an image of your own): [X,I,dims] = im2rgb('plane_small.png') Perform a k-means clustering of these color values (X) in R3 using ten clusters.
****
Wow! This is fun. I like the 3D plot. It is completely impossible to read on the PDF in a meaningful way, but it is really interesting in the interactive one to rotate around. All the points seem to mostly lie on a plane, which is pretty interesting in itself.

```
[X,I,dims] = im2rgb('plane_small.png');
k = 10;
[mu, labels] = km(X, k);

figure()

scatter3(X(1,:), X(2,:), X(3,:), 1, labels);
hold on
scatter3(mu(1,:), mu(2,:), mu(3,:), [], 1:k, '^', 'filled');
hold off
```


20
10
0
-10
-20
-20
-10
0
10
20


20
10
0
-10
-20
-20
-10
0
10
20


20
10
0
-10
-20
-20
-10
0
10
20

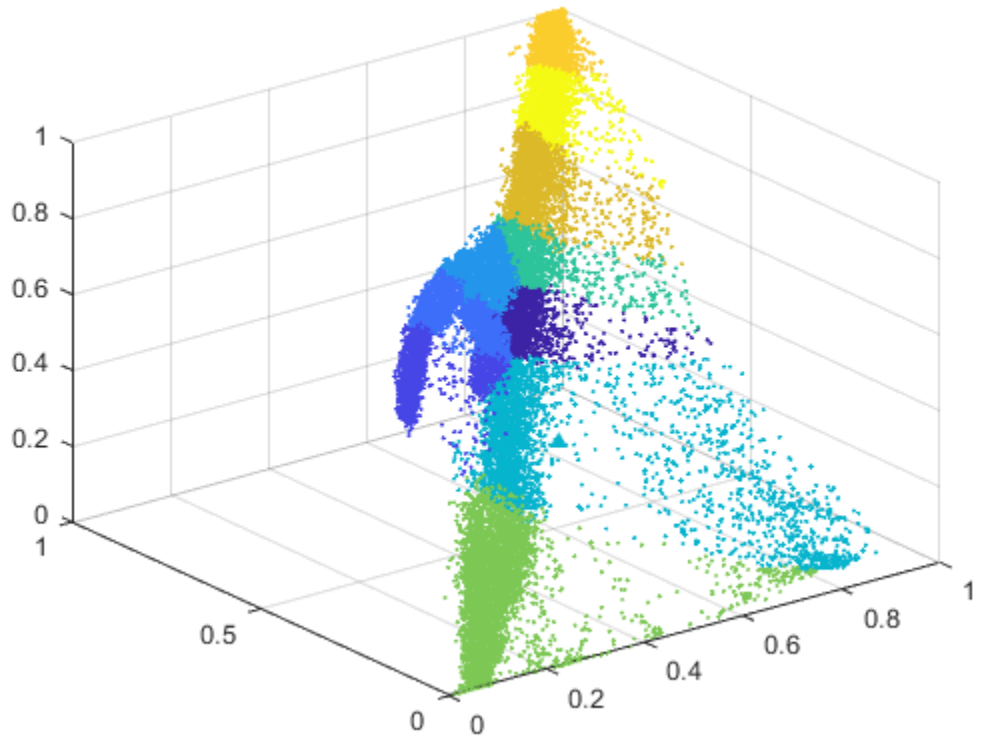

1
0.8
0.6
0.4
0.2
0
1
0.5
0
0
0.2
0.4
0.6
0.8
1

Part 7

Generate and turn in the image produced by setting each pixel color to the centroid of the class it belongs to. This can be accomplished with the command: imshow(rgb2im(mu(:,labels), dims)) where mu contains the learned centroid values, labels contains the learned class assignments, and dims is the image dimensions from im2rgb. Show the original image, for comparison: imshow(I)
****

WOW! Look at that! It did such a good job. I really did not expect that. The clustered image is still definitely recognizeable! That is super neat. I thought part 6 was cool, but this is definitely more cool. It is very impressive how it is able to preserve the objects together. Like the body of the planes are both one cluster, and the tail fin, and even the NASA lettering.

```
figure()
tiledlayout(2,1);
nexttile;
imshow(rgb2im(mu(:,labels), dims));
nexttile;
imshow(I);
```



km.m

```
%% K-Means
%  Separate data points into K clusters with no other information.
% Inputs:
%  X - D-by-N matrix of N points in D dimensions.
%  K - Integer number of clusters to detect.
% Outputs:
%  mu - D-by-K matrix with the learned cluster centroids.
%  labels - Length N vector with integer (1, 2, ..., K) class
 assignments.

function [mu, labels] = km(X, K)
    [D, N] = size(X);
    mu = X(:, randperm(N, K));  % init k random centroids from set
    labels = ones(1, N);

    run = true;
    while run
        % E step
        next_assignment = ones(1, N);
        for i = 1:N  % for all datapoints
            dist = vecnorm(mu - X(:,i));  % get dist to centers
            [~, idx] = min(dist);  % get index of closest center
```

```
            next_assignment(i) = idx; % assign point i to cluster
        end

        % M step
        for j = 1:K  % for all clusters
            assigned = next_assignment == j; % logical of all points
 assigned to cluster j
            s = sum(X(:, assigned), 2);
            mu(:, j) = s / size(X(:, assigned), 2);
        end

        run = all(next_assignment == labels);
        labels = next_assignment;

    end
end
```

pointclouds.m

```
%% generate POINTS from the CLOUDS data set
%%  Produces a collection of points in 2D (five clusters).
%% Inputs:
%%  None
%% Outputs:
%%  X - 2-by-N matrix with N points in 2D (the columns).

function [X] = pointclouds()
    N = 1000;
    K = 5;
    SPACING = 9;
    RADIUS = 1;
    rng(1000);

    X = zeros(2,N);
    Y = zeros(N,1)';

    centers = SPACING*(rand(K,2) - 0.5);
    for i = 1:size(X,2)
        theta = rand()*2*3.1415926;
        radius = RADIUS*rand();
        radius = radius*radius;

        class = round((K-1)*((i-1)/(N-1)) + 1);
        X(:,i) = centers(class,:) + radius*[cos(theta),sin(theta)];
        Y(i) = class;
    end
    rng('shuffle')
end
```

pointrings.m

```
%% generate POINTS from the RINGS data set
%%  Produces a collection of points in 2D (five rings).
%% Inputs:
%%  None
%% Outputs:
%%  X - 2-by-N matrix with N points in 2D (the columns).

function [X] = pointrings()
    N = 1000;
    K = 5;
    SPACING = 4;
    SCATTER = 1;
    rng(12345);

    X = zeros(N,2);
    Y = zeros(N,1)';

    for i = 1:size(X,1)
        theta = rand()*2*3.1415926;
        radius = SCATTER*rand();
        radius = radius*radius;

        class = round((K-1)*((i-1)/(N-1)) + 1);
        X(i,:) = (SPACING*(class-1) + radius)*[cos(theta),sin(theta)];
        Y(i) = class;
    end
    X = X';
    rng('shuffle')
end
```

im2rgb.m

```
%% load and convert an IMage file 2 a sequence of RGB triples
%%  Load an image as a sequence of RGB triples.
%% Inputs:
%%  ImageFile - A string with the path to (name of) an image file to
 load.
%% Outputs:
%%  X - 3-by-(dims(0)*dims(1)) matrix with an RGB triple in each
 column.
%%  I - dims(0)-by-dims(1)-by-dims(2) matrix of integers with the raw
 image data.
%%  dims - A triple of numbers with the height, width, and number of
 channels (3 for RGB).

function [X,I,dims] = im2rgb(ImageFile)
    I = imread(ImageFile);
```

```
    W = size(I,2);
    H = size(I,1);
    XI = cast(reshape(I,[W*H,3]),'double')/255.0;
    dims = size(I);
    X = XI';
end
```

rgb2im.m

```
%% RGB colors 2 IMage conversion
%%  Converts a sequence of RGB colors representing an image with the
 given
%%   dimensions into a MATLAB image.
%% Inputs:
%%  X - A 3-by-(dims(0)*dims(1)) matrix where each column represents
 the RGB
%%    values of a pixel in the image.
%%  dims - Three integers representing the size of the image to
 create.
%% Outputs:
%%  I - A dims(0)-by-dims(1)-by-3 matrix of integers representing the
 RGB image.

function [I] = rgb2im(X,dims)
 if (numel(dims) ~= 3), error('rgb2im requires dims to be three
 integer values.'); end
 if (dims(1)*dims(2) ~= size(X,2)), error('rgb2im number of RGB
 triples does not match image size.'); end
 if (size(X,1) ~= 3), error('rgb2im must have X with three rows (RGB
 values).'); end
    I = cast(255.0*reshape(X',[dims(1),dims(2),3]),'uint8');
end
```

Published with MATLAB® R2021a