3D_OBJECTS Dataset k=10

```
clc
clear
```

Load the file

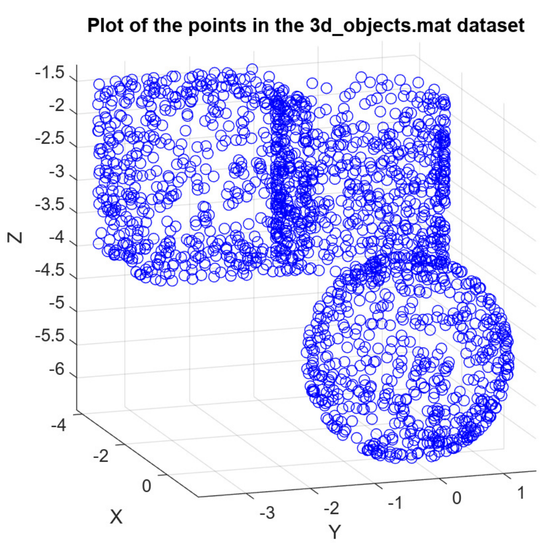
```
objects_mat = load('3d_objects.mat');

% Display the structure of the file
disp(objects_mat);
```

```
X: [1800x3 double]
```

```
% Extract the matrix of points
X = objects_mat.X;
```

Plot the points

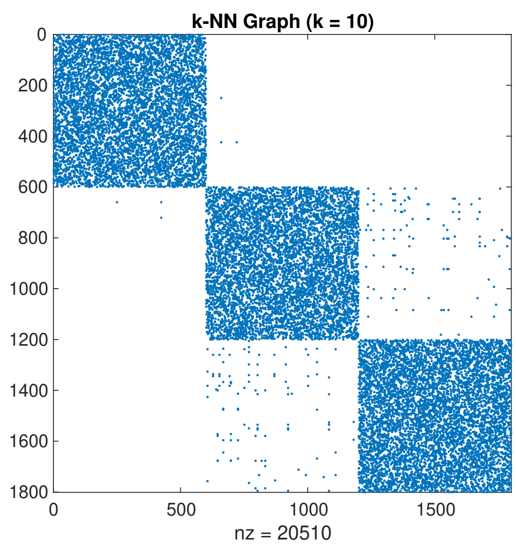
```
figure;
scatter3(X(:,1), X(:,2), X(:,3), 'b','c');
xlabel('X');
ylabel('Y');
zlabel('Z');
title('Plot of the points in the 3d\_objects.mat dataset');
grid on;
axis equal;
```



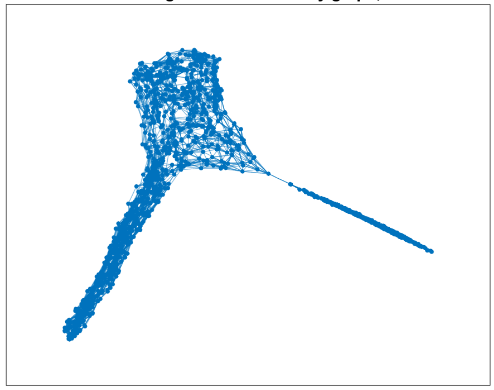
1.Similarity matrix and adjacency matrix

Three independent simulations of the script were carried out, with k set to 10, 20 and 40, respectively. The results presented in this script were obtained with a value of k fixed at 10. To analyse other values of k, it is simply necessary to change the value of k in the source code and re-run the script.

```
k_values = [10, 20, 40];
k = 10;

% Construct the k-nearest neighborhood similarity graph and its adjacency
% matrix W
W = knn_graph(X, k);

% Visualize the graph using its similarity matrix
figure;
spy(W);
title(['k-NN Graph (k = ', num2str(k), ')']);
```




```
% Store W as a sparse matrix
W = sparse(W);

% Visualize the graph G corresponding to the adjacency matrix W
G = graph(W);
figure;
plot(G);
title(['k-nearest neighborhood similarity graph, k = ', num2str(k)]);
```

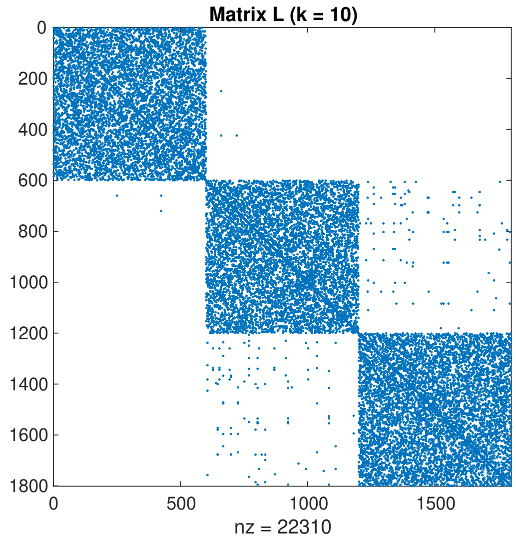


2.Construct the degree matrix D and the Laplacian matrix L

```
N = size(W, 1);

% Inizialize the degree matrix D
D = zeros(N, N);

% The degree of each point is given by the sum of the elements of each row in W
D = diag(sum(W, 2));

% Save D in a sparse format
D = sparse(D);

% Compute the Laplacian matrix L
L = D - W;

% Compute the normalized Laplacian matrix L(sym)
D_inv_sqrt = diag(1 ./ sqrt(diag(D)));
L = D_inv_sqrt * L * D_inv_sqrt;
L = sparse(L);

if issparse(L) % issparse(L)==1 means that L is stored in a sparse format
    disp("The matrix L is stored in a sparse format")
end
```

```
The matrix L is stored in a sparse format
```

```
% Plot the Laplacian matrix L
figure;
spy(L);
title(['Matrix L (k = ', num2str(k), ')']);
```



3.Compute the number of connected components of the similarity graph

```
% The points with the same number belong to the same connected component
bins = conncomp(G);

% Number of connected components
num_components = max(bins);

% Display the result
disp(['Number of connected components: ', num2str(num_components)]);
```

```
Number of connected components: 1
```

4 - 5. Compute eigenvalues and eigenvectors

This script implements two methods for computing the M smallest eigenvalues and their corresponding eigenvectors: the deflation method and the inverse power method. Currently, the script is configured to use the deflation method. In this case, we are applying the methods to the normalized symmetric Laplacian matrix:

$$L_{sym} := D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$$

So, the first eigenvalue and eigenvector are computed using the inverse power method.

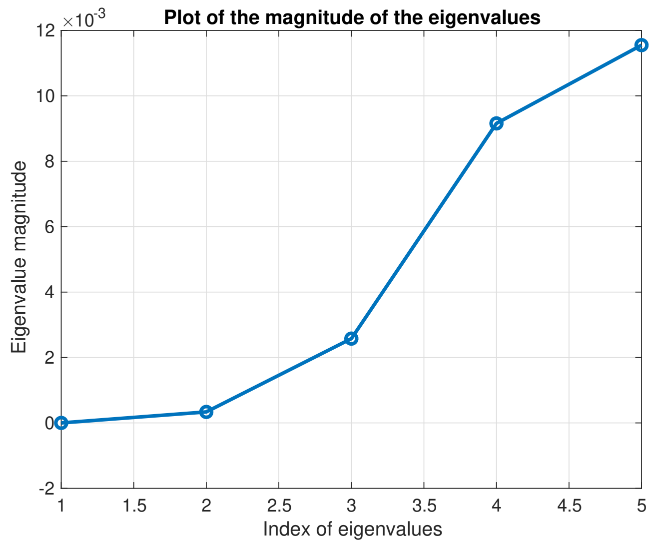
```
% Set a number M of values to be computed (later it will be changed)
M = 5;

% Inizialize the eigenvalues vector and the eigenvectors matrix
eigvalues = zeros(M, 1);
eigvects = zeros(N, M);

% Choose the vector v that will be used for the inverse power method
v = 0.5 * ones(N, 1);
v(1:2:N) = -0.5;
```

```
% Max iterations in the power method
maxIter = 1000;
% Relative tolerance
relTol = 1e-10;

% A known fact from theory is that L is semi pos def and has at least one
% eigenvalue = 0 and that the vector of all ones is a corresponding
% eigvalues(1) = 0;
% eigvects(:, 1) = ones(N,1)/ norm(ones(N,1));

% Or use the inverse power method to compute them (if L_sym, use this)
[eigvalues(1), eigvects(:, 1)] = invpower_method(L, v(1:end), maxIter, relTol);

% Compute the reamining eigenvetors and eigenvalues
[eigvalues, eigvects, residualnorms] =
    deflation_method(L, v, eigvects, eigvalues,M, maxIter, relTol);

% Check how good the approximation is by comparing with eigs function of
% Matlab
[mat_eigvects, mat_eigs] = eigs(L, M, 'smallestabs');

norm(eigvalues - diag(mat_eigs))
```

```
ans = 1.5138e-12
```

Now, find the actual number M of eigenvalues that will be used for the clustering algorithm

```
% Plot the computed eigenvalues
x = 1:M;

figure;
plot(x, eigvalues, '-o', 'LineWidth', 2);
xlabel('Index of eigenvalues');
ylabel('Eigenvalue magnitude');
title('Plot of the magnitude of the eigenvalues');
grid on;
```



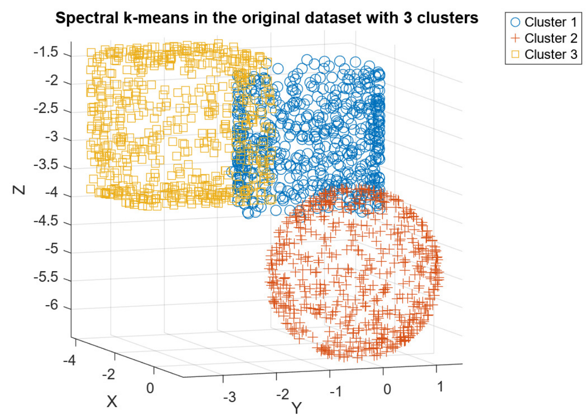
The suitable number of eigenvalues is 2 since eig3 is much larger than eig2 for k=10

```
M = 3;

% Define the matrix U that will be used for the spectral clustering
U = eigvects(:, 1:M);
```

6 - 7 - 8. Spectral clustering, k means

```
% Clusterize using k means and obtain the indices (and the centroids)
% inside the clusters of each point
[idx, C] = kmeans(U, M);

% Assing the original data to the corresponding clusters
A = cell(M, 1);

for i = 1:N
    % Find the cluster of y_i
    cluster_idx = idx(i);

    % Assing it to x_i
    A{cluster_idx} = [A{cluster_idx}; X(i, :)];
end

% Plot of the clusterized data in the original space
X_spect_clust = zeros(N, 4);
X_spect_clust(:, 1: 3) = X;
X_spect_clust(:, 4) = idx;

markers = ['o', '+', 's'];
figure;
```

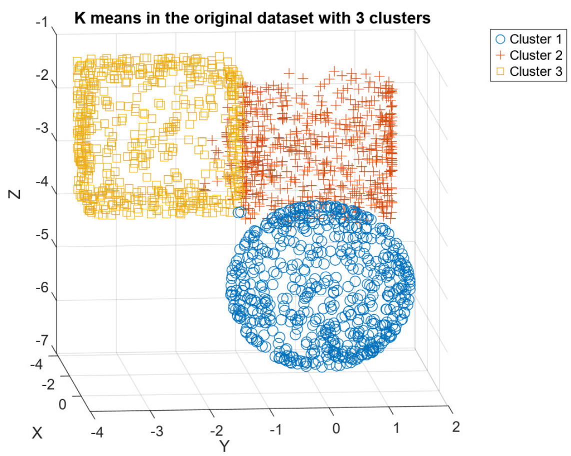
```
for k = 1:M
    cluster_points = X_spect_clust(X_spect_clust(:, 4) == k, :);
    scatter3(cluster_points(:,1), cluster_points(:,2), cluster_points(:,3), ...
        50, 'Marker', markers(k), 'DisplayName', ['Cluster ' num2str(k)]);
    hold on
end

title(['Spectral k-means in the original dataset with ', num2str(M), ' clusters']);
xlabel('X');
ylabel('Y');
zlabel('Z');
legend show;
grid on;
axis equal;
```



9.a K MEANS TO THE ORIGINAL DATA

```
k_value = 3;

% Clusterize the original data
[idx_k, C_k] = kmeans(X, k_value);

% Add the index to X_kmeans
X_kmeans = zeros(N, 4);
X_kmeans(:, 1: 3) = X;
X_kmeans(:, 4) = idx_k;


markers = ['o', '+', 's'];
figure;

for k = 1:M
    cluster_points_k = X_kmeans(X_kmeans(:, 4) == k, :);
    scatter3(cluster_points_k(:,1), cluster_points_k(:,2), cluster_points_k(:,3), ...
        50, 'Marker', markers(k), 'DisplayName', ['Cluster ' num2str(k)]);
    hold on
end
```

```
title(['K means in the original dataset with ', num2str(M), ' clusters']);
xlabel('X');
ylabel('Y');
zlabel('Z');
legend show;
grid on;
```



9.b DBSCAN TO THE ORIGINAL DATA

Apply the DBSCAN algorithm

```
% Neighborhood radius
epsilon = 0.5;
% Minimum points for a cluster
minPts = 15;

idx_d = dbscan(X, epsilon, minPts);

X_dbscan = zeros(N, 4);
X_dbscan(:, 1: 3) = X;
X_dbscan(:, 4) = idx_d;

% Only the different clusters (-1 are put together as outliers)
cluster_ids = unique(X_dbscan(:, 4));

figure;
colors = hsv(length(cluster_ids));
for k_idx = 1:length(cluster_ids)
    k = cluster_ids(k_idx);
    cluster_points_d = X_dbscan(X_dbscan(:, 4) == k, :);
        scatter3(cluster_points_d(:, 1), cluster_points_d(:, 2), cluster_points_d(:, 3), ...
        50, colors(k_idx, :), '*');
    hold on
end

title('DBSCAN in the original dataset');
```

```
xlabel('X');
ylabel('Y');
zlabel('Z');
legend show;  grid on;
hold off;
```