Graph Signal Processing
Computer Homework 2 (Report)

Mohammad Parsa Dini - std id: 400101204

December 13, 2024

Problem 1

A. First of all, this is the adjacency matrix of graph G:

$$W = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

. and also here's the graph:

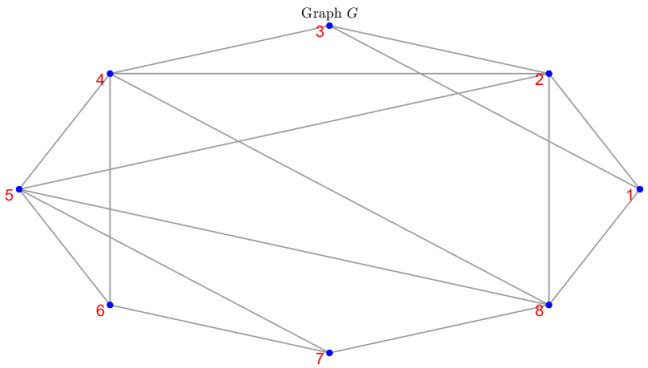

Figure 1: The graph G

B. Here is ho we define the signal $x = 2u_1 + u_2$:

```
G = gsp_compute_fourier_basis(G);      % computing the fourier basis
U12 = G.U(:, 1:2);                     % the 1st and 2nd components of U
x = 2*U12(:,1) + U12(:,2);             % defining the signal x
x_n = awgn(x, 10);                     % adding white gaussian noise
snr_n = db(snr(x_n, x));               % getting the snr
```

The signals are depicted down below are the signals x and its noisy version x_n in time domain:

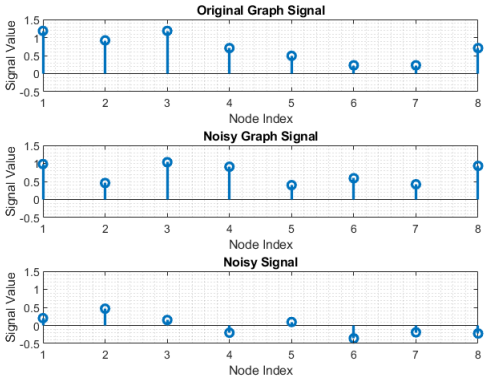

Figure 2: The graph signals x , x_n time representation with respect to shift operators W_N and L_N

C. The signals are depicted down below are the Fourier representations of signals x and its noisy version x_n:

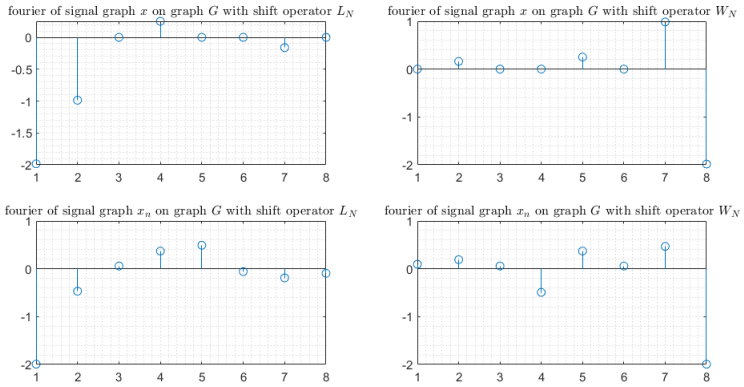

Figure 3: The graph signals x , x_n fourier representation with respect to shift operators W_N and L_N

D. Now that we have $W_N = D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$ and also $L_N = D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}$ as shift operators, we assign $h_{\text{filter},L_N} = \text{diag}([1,1,0,0,0,0,0,0])$ and $h_{\text{filter},W_N} = \text{diag}([0,0,0,0,0,0,1,1])$ as the filter for the signals based on normalized laplacian and normalized weight matrix, respectively. They are designed as such that they choose the first two components of low-frequent components. These components are the first two components of Laplacian matrix whereas the last two components of weight matrix.

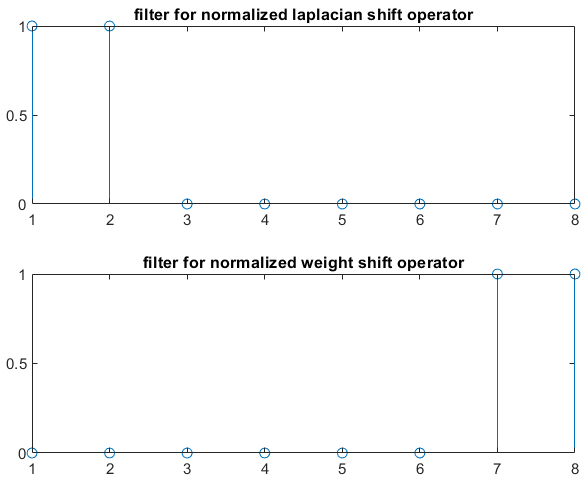

Figure 4: The filters h_{filter,L_N} and h_{filter,W_N}

E. Here is the code for getting the basis based on the shift operator, getting its fourier transform and then apply the filter and the apply an inverse fourier transform:

```
[U_lap, e_lap] = eig(L_N);
Lambda1 = diag(e_lap);
Gn.L = L_N;
Gn = gsp_compute_fourier_basis(Gn);

W_N = D^(-1/2) * W * D^(-1/2);
H = gsp_graph(W_N);
H = gsp_compute_fourier_basis(H);
[U_wn, e_wn] = eig(W_N);
Lambda2 = diag(e_wn);

Filter_H_L = [1 1 0 0 0 0 0 0];
x_n_hat_L = (U_lap)' *x_n;
y_hat_L = diag(Filter_H_L)* x_n_hat_L;
y_L = (U_lap)* y_hat_L;

Filter_H_W = [0 0 0 0 0 0 1 1];
x_n_hat_W = (U_wn)' *x_n;
y_hat_W = diag(Filter_H_W)* x_n_hat_W;
y_W = (U_wn)* y_hat_W;
```

And the results of filtered signals are depicted down below:

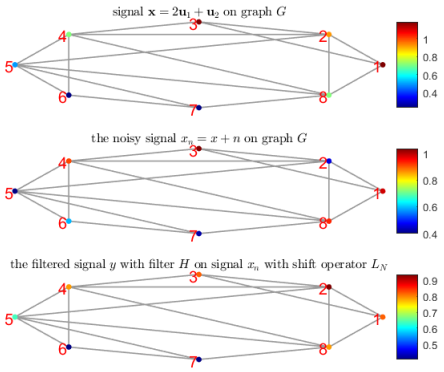

Figure 5: The graph signal x_n after being filtered by h_{filter,L_N}

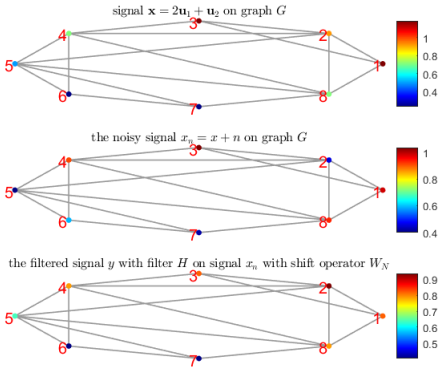

Figure 6: The graph signal x_n after being filtered by h_{filter,W_N}

F. We calculate the SNR of both filters as below (It seems both have the same performance over retrieving the signal x):

```
1
2  noise_component_L = y_L - x;
3  filtered_noise_power_L = sum(noise_component_L.^2) /
       length(noise_component_L);
4  snr_filtered_L = 10 * log10(signal_power / filtered_noise_power_L);
5
6  noise_component_W = y_W - x;
7  filtered_noise_power_W = sum(noise_component_W.^2) /
       length(noise_component_W);
8  snr_filtered_W = 10 * log10(signal_power / filtered_noise_power_W);
9
10 if snr_filtered_W > snr_filtered_L
11     disp('The $W_N$ matrix outperforms with respect to $L_N$');
12 elseif snr_filtered_W == snr_filtered_L
13     disp('Both matrices have the same performance');
14 else
15     disp('The $L_N$ matrix outperforms with respect to $W_N$');
16 end
17 disp(snr_filtered_L);
18 disp(snr_filtered_W);
```

```
-------
>> Both matrices have the same performance
>>    11.4358

>>    11.4358
```

G. We consider the MSE metric for the error in order to get both signals close to each other:

$$\begin{aligned}\|x_{\text{ideal}} - x_{\text{fir}}\|_2 &= \|\hat{x}_{\text{ideal}} - \hat{x}_{\text{fir}}\|_2 \\ &= \|\hat{x}\|_2 \cdot \|h_{\text{ideal}} - h_{\text{fir}}\|_2 \\ &\implies h_{\text{fir,opt}} = \arg\min_{h\in\text{FIR}} \|h_{\text{ideal}} - h\|_2\end{aligned}$$

Let the filter be as $h_{\text{FIR}} = h_0 + h_1\lambda + h_2\lambda^2 = \begin{pmatrix}\lambda^0 & \lambda^1 & \lambda^2\end{pmatrix}\begin{pmatrix}h_0\\h_1\\h_2\end{pmatrix} = \Lambda_3^T h_3$.

Solving the optimization implies:

$$h_{opt} = \arg\min_{h\in FIR} |h_{ideal} - h|^2 = \Lambda_3 \arg\min_x |h_{ideal} -_L ambda_3 x|^2 = \Lambda_3\Lambda_3^\dagger h_{ideal}$$

which suggests: $(h_0, h_1, h_2) = \Lambda_3^\dagger h_{ideal}$. According to the above relations, we can find an FIR filter of arbitrary length that minimizes the distance to the ideal filter. In Figure 6, we have shown this filter with length 3 for this case.

Here is the code for generating the result. Firstly, we choose only three components and afterwards we use the pseudo-inverse of matrix to calculate the best FIR filter with three taps. Using this code we came with the best coefficients as:

$$h_{\text{filter-3},L} = [1.14, -1.35, 0.37, 0, 0, 0, 0, 0]$$

$$h_{\text{filter-3},W} = [0, 0, 0, 0, 0, 0.15, 0.61, 0.37]$$

```
n = 3;
S123_L = Lambda1 .^ (0:n-1);
S123_Wn =  Lambda2 .^ (0:n-1);
h_L = pinv(S123_L) * Filter_H_L';
FIR_L = S123_L * h_L;
h_Wn = pinv(S123_Wn) * Filter_H_W';
FIR_Wn = S123_Wn * h_Wn;
----------
>> h_L =

      1.1476
     -1.3650
      0.3743

>> h_Wn =

      0.1568
      0.6164
      0.3743
```

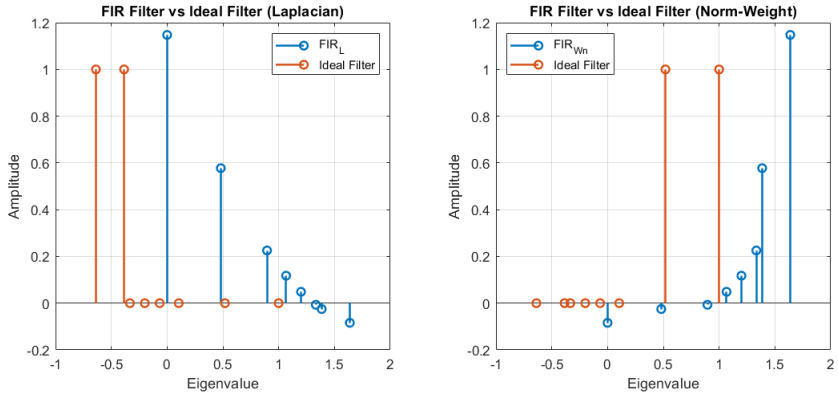

Figure 7: FIR filters' coeffs with three taps and also the ideal one

H. The original signal, the noisy signal, filtered with the Laplacian filter, and filtered with the normalized weight filter. As seen, the SNR in Figures 5, 6 is much better than 7, as expected, with the increase in the FIR filter length, but this difference becomes smaller and smaller.
$SNR_{default} = 7.635 \rightarrow SNR_{FIR,L_N} = 13.48 \rightarrow SNR_{FIR,Wn} = 12.3$

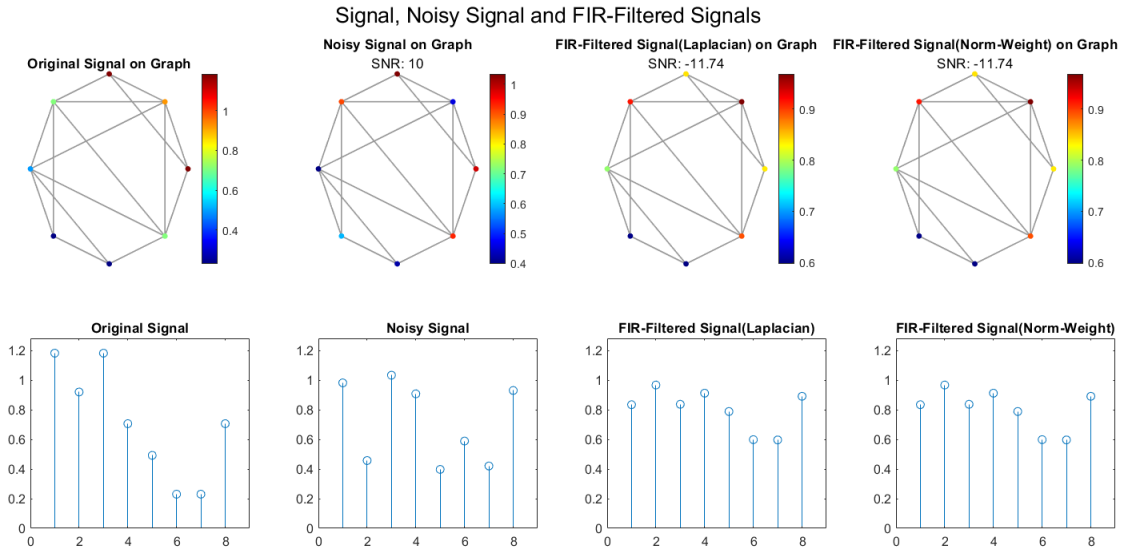

Figure 8

And the code is put down here:

```
x_FIR_L = U_lap * (FIR_L .* x_n_hat_L);
x_FIR_Wn = U_wn * (FIR_Wn .* x_n_hat_W);

snr_FIR_L = db(snr(x_FIR_L, x));
snr_FIR_Wn = db(snr(x_FIR_Wn, x));
```

Problem 2

A. Here is the code for generating the SBM Graph with three clusters:

```
N = 150;
p = 8 * log(N) / N;
q = log(N) / N;
k = 3;
dim = k;

% Generate SBM graph
seed = 20;
rng(seed)
sig = randi([0, k - 1], [N, 1]);
sig = 2 * rescale(sig) - 1;
P = (sig == sig') * p + (sig ~= sig') * q;
A = ones(N) - (rand(N) > P);
A = triu(A);
A = abs(A - A');

% Create graph and compute Fourier basis
G = gsp_graph(A);
G = gsp_create_laplacian(G);
G = gsp_compute_fourier_basis(G);

% Choose subset of Fourier basis for visualization
U23 = G.U;
G.coords = U23(:, 2:max(3, dim));
```

The visualized graph is depicted down below:

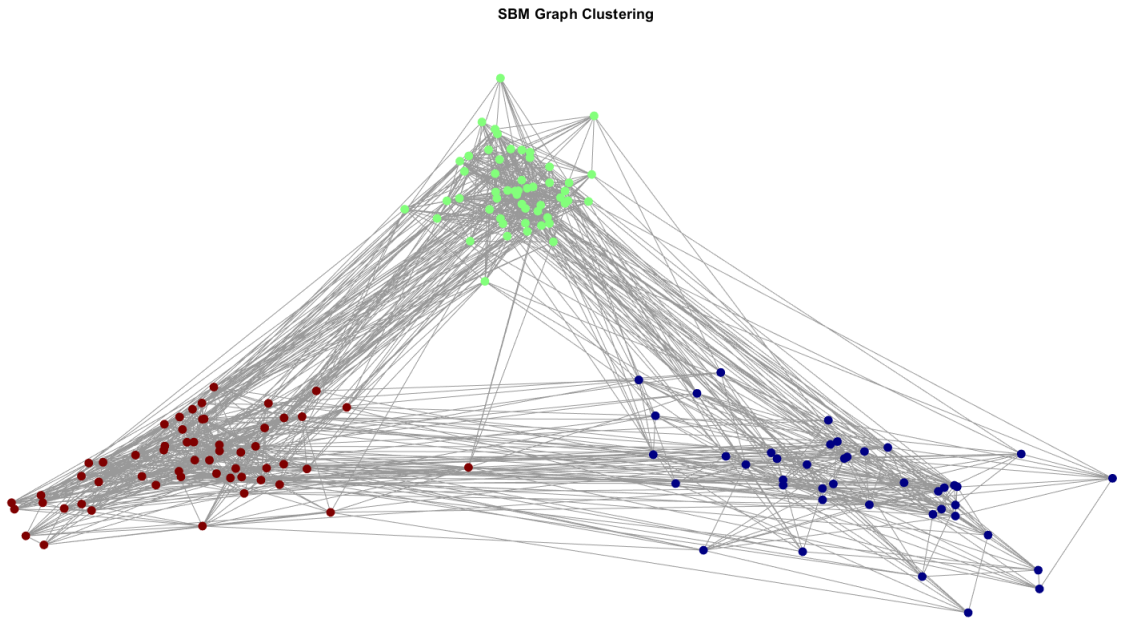

Figure 9: The SBM graph with three clusters

B. Now, using the graph we constructed in the previous section, we generate t smooth signals using an FIR filter of degree r. Our filter is defined as $H = (1 - \alpha L_G)^{-r}$, and we pass t white noise signals through this filter to obtain the smooth signals.
As shown in Figure 10, these smooth signals can perform good classification on their own and provide insight into the groupings. Now, we will attempt to use these signals to perform grouping and discover the graph structure.

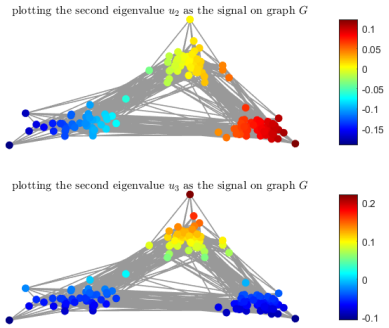

Figure 10: The eigenvalue signals u_2, u_3 on SBM graph

And this is the code for generating smooth signals:

```
T = 1000;
r = 31;
X_smooth = gen_smooth(G, T, r);

figure();
gsp_plot_signal(G, X_smooth(:,1), param);
title('Smooth signal on graph');

function x_smooth = gen_smooth_filtered(G, t, r)
    x = randn(G.N, t);
    d_max = max(G.d);
    alpha = 1 / (2 * d_max);
    H = (eye(G.N) - alpha * G.L) ^ (r-1);
    x_smooth = H * x;
end
```

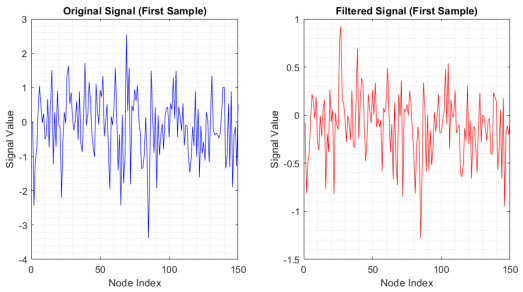

Figure 11: The time domain on another sight

This is another visualization, this time on the graph itself:

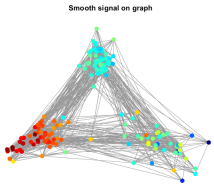

Figure 12: $r = 31$

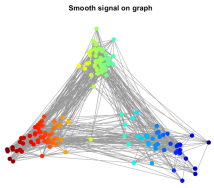

Figure 13: $r = 200$

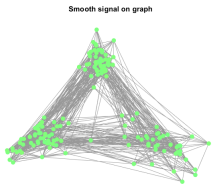

Figure 14: $r = 800$

Figure 15: The filtered smooth signal on graph G with $r = 31$, $r = 200$, and $r = 800$, respectively.

The image above shows that, however the graph was random, but we could cluster nodes to some extent. As we can see the value of the filtered signal is roughly the same in each class which shows we were successful!
However, increasing T was helpful up to a point, afterwards as you can see, it was destructive and resulted in a uniform filtered signal.

C. Random signals x are generated through a random process and are white Gaussian. The **spectral density** of this signal is as follows:

$$\Sigma_x = I_n = U\sigma^2 U^H = U\text{diag}(1_n)U^H$$

$$\Sigma_y = U\text{diag}(1_n \cdot |h|^2)U^H = U\text{diag}(|h|^2)U^H = H^2$$

$$HL = LH$$

As a result, the eigenvectors of the covariance matrix are the same as the eigenvectors of the Laplacian matrix. Using this, estimation can be performed. However, we must note that the order of eigenvalues in H is the reverse of their order in L, and consequently, the order of the signal eigenvectors is also reversed.

After estimating the eigenvectors, the usual steps of **spectral clustering** are carried out.

Here are the the results:

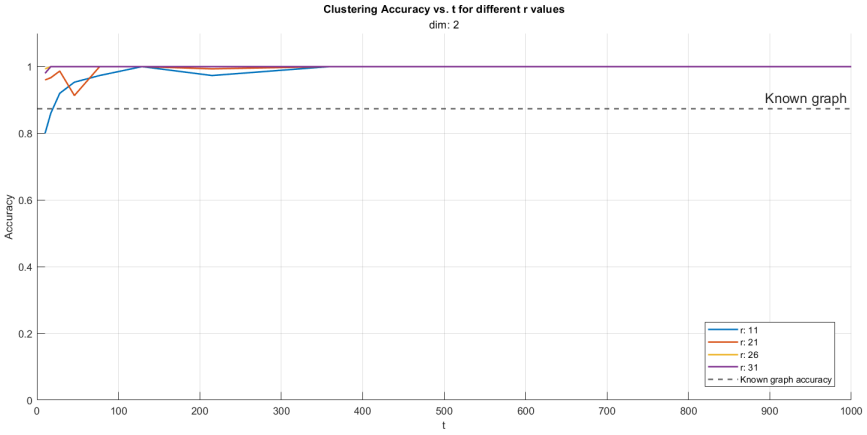

Figure 16: The time domain on another sight

As evident from Figure 16, this grouping is performed with very high accuracy.

Using the first two eigenvalues is sufficient. In general, we found that for k-class classification, using the first $k-1$ eigenvectors for embedding is enough.