Aristotle University of Thessaloniki

Computer Science Department

Second assignment in NDM-06-01

Wavelets, Nonlinear Dynamics, Graph Signal Processing

Alexandros Korkos
alexkork@csd.auth.gr
3870

Thessaloniki, June 30, 2023

1 Wavelets

1.1 Description

Using concepts of WT you are asked to remove the blinking activity from the EEG-signal in Data_for_WT_Task > EEG_Blinking.mat.

1.2 Solution

Execute: Wavelet > wavelets.m

This solution is based on the findings of [1].

At first, we assume that the noise on the given signal is non (we will see further down were this is important for our solution). For this particular solution, wavelet functions (WF) Daubechies (db2, db6, db8) and Meyer (dmey) are used during the wavelet transform for noise removal. RMSE difference was calculated to measure the effectiveness of the noise removal using these wavelets.

The denoising process is quite simple. We take the Raw EEG Signal, apply Wavelet denoisingg and finally we have the EEG after denoising.

The wavelet denoising, is broken down to:

- Input of EEG Signal
- Wavelet Decomposition
- Threshold method
- Wavelet reconstruction
- EEG after Denoising

RMS difference was calculated for each of the WFs (db2, db6, db8, and dmey) as show in table 1.

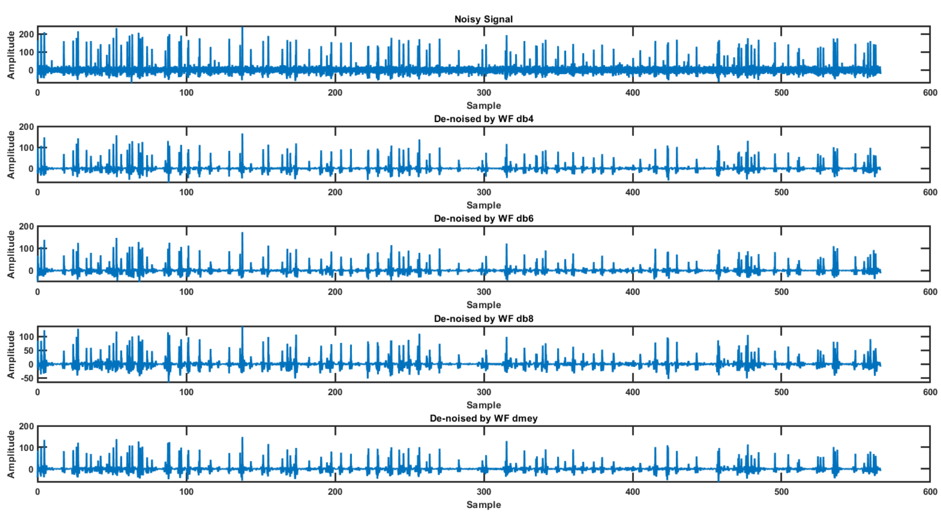

Figure 1: Raw EEG signal and denoised verions of that

Table 1: RMSE values for each method

	db4	db6	db8	dmey
RMSE	13.4	14.48	15.17	13.93

All four methods can effectively remove noise from EEG signals. Examining the RMSE values, we can see that all methods are of the same efficiency.

The implementation following is written in MATLAB. We are using the `wdenoise` function, assuming that the noise is non white (`NoiseEstimate="LevelDependent"`, else `LevelInDependent`). For the noised method other values can be used as well producing the same results.

```
load EEG_Blinking.mat

y = eeg1;
Ts = 1/Fs;
t = 1:numel(y);
time = Ts*t;

% Wavelet De-Noising

db4 = wdenoise(y,8,'Wavelet','db4', DenoisingMethod="UniversalThreshold",ThresholdRule="soft", NoiseEstimate="LevelDependent");
db6 = wdenoise(y,8,'Wavelet','db6', DenoisingMethod="UniversalThreshold", ThresholdRule="soft", NoiseEstimate="LevelDependent");
db8 = wdenoise(y,8,'Wavelet','db8', DenoisingMethod="UniversalThreshold",ThresholdRule="soft", NoiseEstimate="LevelDependent");
dmey = wdenoise(y,8,'Wavelet','dmey', DenoisingMethod="UniversalThreshold", ThresholdRule="soft", NoiseEstimate="LevelDependent");

% RMS Difference Calculation

rms_db4 = rmse(db4, y)
rms_db6 = rmse(db6, y)
rms_db8 = rmse(db8, y)
rms_dmey = rmse(dmey, y)

% Plot

figure(1);
subplot(5,1,1); plot(time, y); title('Noisy Signal'); xlabel("Sample"); ylabel("Amplitude");
subplot(5,1,2); plot(time, db4); title('De-noised by WF db4'); xlabel("Sample"); ylabel("Amplitude");
subplot(5,1,3); plot(time, db6); title('De-noised by WF db6'); xlabel("Sample"); ylabel("Amplitude");
subplot(5,1,4); plot(time, db8); title('De-noised by WF db8'); xlabel("Sample"); ylabel("Amplitude");
subplot(5,1,5); plot(time, dmey); title('De-noised by WF dmey'); xlabel("Sample"); ylabel("Amplitude");
```

Listing 1: Wavelet denoising in MATLAB

2 Nonlinear Dynamics

2.1 Description

Following the steps described in `NonLinearDynamics` demonstration script and using the multi-channel EEG data from previous demos, you may provide a NonLinear Dynamics description of the brain activity as a function of brain-rhythm (i.e. band) and sensor-position.

2.2 Solution

Execute: `Non Linear Dynamics` › non_linear_dynamics.m

At first, the Non Linear Dynamics description as a function of brain-rhythms is going to be presented.

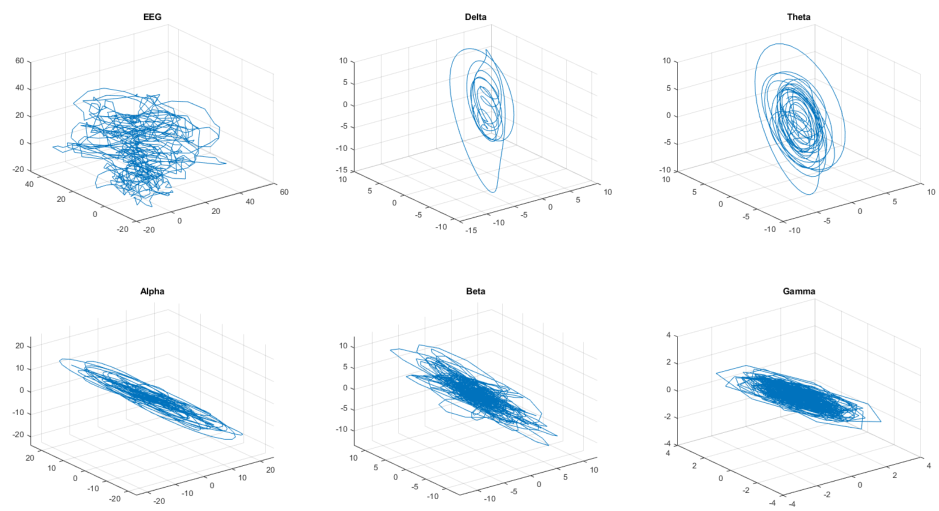

Figure 2: Dynamical trajectories

We have to mention that, A smaller approximate entropy means more predictable the signal and a smaller correlation dimension means smaller level of chaotic complexity. This values are shown in the table below.

Table 2: Approximate Entropy & Correlation Dimension

Band	Approximate Entropy	Correlation Dimension
Delta	0.6110	1.8858
Theta	0.7932	2.8042
Alpha	0.5077	2.8134
Beta	1.0227	2.9320
Gamma	0.9862	2.7197

The code 2 for the above statements, is seen below.

```
load EEG_data.mat
A = [];
B = [];
fdata = [];
bands = [1 4 ; 4 8 ; 8 13 ; 13 30 ; 30 45];

for i = 1:5
    [A(i, :), B(i, :)] = butter(3, bands(i, :)/(Fs/2));
    fdata(:, :, i) = filtfilt(A(i, :), B(i, :),data')';
end

sensor = 4;

% Plot

```

```
figure(1);
subplot(2,3,1); plot(data(sensor,1:10*Fs)); title("EEG");
subplot(2,3,2); plot(fdata(sensor,1:10*Fs, 1)); title("Delta");
subplot(2,3,3); plot(fdata(sensor,1:10*Fs, 2)); title("Theta");
subplot(2,3,4); plot(fdata(sensor,1:10*Fs, 3)); title("Alpha");
subplot(2,3,5); plot(fdata(sensor,1:10*Fs, 4)); title("Beta");
subplot(2,3,6); plot(fdata(sensor,1:10*Fs, 5)); title("Gamma");

% EEG bands as a dynamical trajectory

[XX, eLAG, eDIM] = phaseSpaceReconstruction(data(sensor,1:10*Fs));
[XX1, eLAG1, eDIM1] = phaseSpaceReconstruction(fdata(sensor,1:10*Fs, 1));
[XX2, eLAG2, eDIM2] = phaseSpaceReconstruction(fdata(sensor,1:10*Fs, 2));
[XX3, eLAG3, eDIM3] = phaseSpaceReconstruction(fdata(sensor,1:10*Fs, 3));
[XX4, eLAG4, eDIM4] = phaseSpaceReconstruction(fdata(sensor,1:10*Fs, 4));
[XX5, eLAG5, eDIM5] = phaseSpaceReconstruction(fdata(sensor,1:10*Fs, 5));

aE1 = approximateEntropy((fdata(sensor,1:10*Fs, 1)),eLAG1,eDIM1)
aE2 = approximateEntropy((fdata(sensor,1:10*Fs, 2)),eLAG2,eDIM2)
aE3 = approximateEntropy((fdata(sensor,1:10*Fs, 3)),eLAG3,eDIM3)
aE4 = approximateEntropy((fdata(sensor,1:10*Fs, 4)),eLAG4,eDIM4)
aE5 = approximateEntropy((fdata(sensor,1:10*Fs, 5)),eLAG5,eDIM5)

cDim1 = correlationDimension((fdata(sensor,1:10*Fs, 1)),eLAG1,eDIM1)
cDim2 = correlationDimension((fdata(sensor,1:10*Fs, 2)),eLAG2,eDIM2)
cDim3 = correlationDimension((fdata(sensor,1:10*Fs, 3)),eLAG3,eDIM3)
cDim4 = correlationDimension((fdata(sensor,1:10*Fs, 4)),eLAG4,eDIM4)
cDim5 = correlationDimension((fdata(sensor,1:10*Fs, 5)),eLAG5,eDIM5)

% Plot

figure(2);
subplot(2,3,1);plot3(XX(1:1000,1),XX(1:1000,2),XX(1:1000,3));grid; title("EEG");
subplot(2,3,2);plot3(XX1(1:1000,1),XX1(1:1000,2),XX1(1:1000,3));grid; title("Delta");
subplot(2,3,3);plot3(XX2(1:1000,1),XX2(1:1000,2),XX2(1:1000,3));grid; title("Theta");
subplot(2,3,4);plot3(XX3(1:1000,1),XX3(1:1000,2),XX3(1:1000,3));grid; title("Alpha");
subplot(2,3,5);plot3(XX4(1:1000,1),XX4(1:1000,2),XX4(1:1000,3));grid; title("Beta");
subplot(2,3,6);plot3(XX5(1:1000,1),XX5(1:1000,2),XX5(1:1000,3));grid; title("Gamma");
```

Listing 2: Non Linear Dynamics description as a function of brain-rhythms in MATLAB

The second part of the exercise, demanded to provide a Non Linear Dynamics description as a function of sensor. Because the number of sensors is quite big, the Approximate Entropy & Correlation Dimension is not reported in this file and only in the MATLAB script 3.

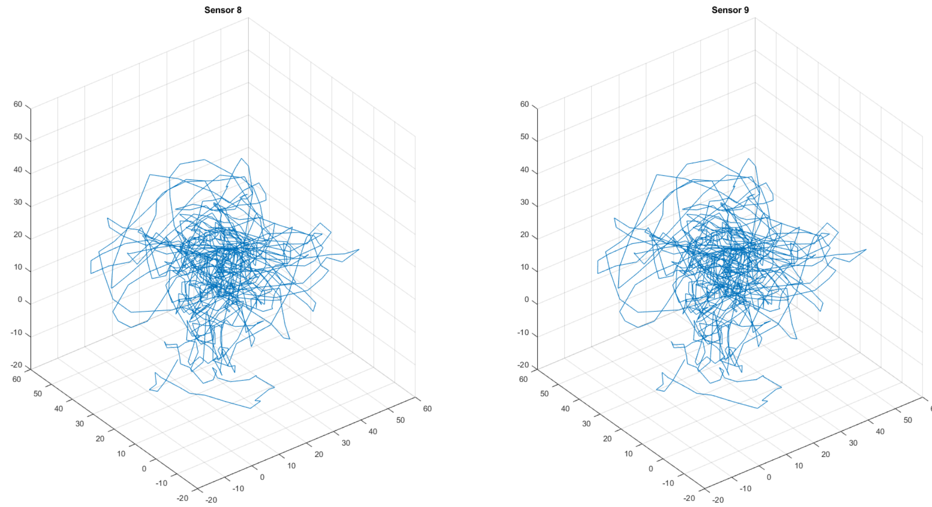

Figure 3: Dynamical trajectories

As said, the number of sensors is high so, all values are calculated in a loop and accordingly plotted and shown.

```
clear;
load EEG_data.mat

k = 1;

for i = 1:5
    figure(i)
    for j = 1 : 2
        if i == 1 && j == 1
            [XX, eLAG, eDIM] = phaseSpaceReconstruction(data(2, 1:10*Fs));
            subplot(1,2,j); plot3(XX(1:1000,1),XX(1:1000,2),XX(1:1000,3)); grid; title("EEG");
        else
            [XX, eLAG, eDIM] = phaseSpaceReconstruction(data(i, 1:10 * Fs));
            subplot(1,2,j); plot3(XX(1:1000,1),XX(1:1000,2),XX(1:1000,3)); grid; title("Sensor" + " " + num2str(k));
            aE = approximateEntropy((data(k,1:10*Fs)),eLAG,eDIM)
            cDim = correlationDimension((data(k,1:10*Fs)),eLAG,eDIM)
            k = k + 1;
        end
    end
end
```

Listing 3: Non Linear Dynamics description as a function of sensors in MATLAB

3 Graph Signal Processing

Execute: `Graph Signal Processing` › graph_signal_processing.m

3.1 Compute the Graph Fourier Transform of a graph signal

3.1.1 Description

Compute the Graph Fourier Transform of a graph signal. Write a function in MATLAB that takes as an input a graph shift $S \in \mathbb{R}^{NxN}$ and a signal $x \in \mathbb{R}^N$ defined on the graph and generates as output $\tilde{x} \in \mathbb{R}^N$, the graph Fourier representation of x. Make sure to order the eigenvectors of S in increasing order of absolute value of the associated eigenvalues.

3.1.2 Solution

```
function x_tilde = graphFourierTransform(S, x)
    % Calculate the eigenvectors and eigenvalues of the graph shift matrix
    [V, D] = eig(S);

    % Sort the eigenvalues and eigenvectors in ascending order
    [~, index] = sort(diag(D));
    V = V(:, index);

    % Compute the graph Fourier representation x'
    x_tilde = V' * x;
end
```

Listing 4: Graph Fourier Transform in MATLAB

3.2 Understanding the data

3.2.1 Description

Load the file `graph_sp_data.mat`. You will see the adjacency matrix of a graph $A \in \mathbb{R}^{50x50}$, and four graph signal called $x_i \in \mathbb{R}^{50}, i = 1, 2, 3$ and $y \in \mathbb{R}^{50}$. How many connected components does the graph have? Plot signals x_1, x_2 and x_3. Can you tell which one 'varies faster' in the graph domain?

3.2.2 Solution

In this graph we have 1 connected component, calculated using the function below.

```
function numComponents = countConnectedComponentsLaplacian(A)
    % Compute the Laplacian matrix
    L = diag(sum(A, 2)) - A;

    % Calculate the eigenvalues of the Laplacian matrix
    eigenvalues = eig(L);

    % Count the number of eigenvalues that are close to zero
    numComponents = sum(abs(eigenvalues) < 1e-10);
end
```

Listing 5: Calculate connected components on a graph in MATLAB

3.3 Finding the frequency representation of signals

3.3.1 Description

For the remainder of the lab practice, we define as graph-shift $S = L$ the Laplacian of the loaded graph with adjacency matrix A. Using your function in Section 3.1, plot $\tilde{x_1}, \tilde{x_2}, \tilde{x_3}$, i.e. the frequency representations of x_1, x_2, and x_3, respectively. By looking at the plots, can you tell which signal varies slower and which one varies faster in the graph domain?

3.3.2 Solution

```
D = diag(sum(A, 2));     % Degree matrix
L = D - A;               % Laplacian matrix

% Set the graph shift matrix S as the Laplacian matrix L
S = L;
```

Listing 6: Frequency representation of signals

3.4 Quantifying the variation of signals

3.4.1 Description

Use total variation through the Laplacian quadratic form in to quantify the variation of a signal in a graph. Do these results confirm your intuition from Section 3.3?

3.4.2 Solution

$$TV_G(\mathbf{x}) = \mathbf{x}^{\mathbf{T}}\mathbf{L}\mathbf{x}$$

produced by

```
tvd1 = x1' * L * x1
tvd2 = x2' * L * x2
tvd3 = x3' * L * x3
```

Listing 7: Quantifying the variation of signals

and resulting

Table 3: The variation of signals

Signal	x1	x2	x3
TV_G	241.7993	$1.3856e+03$	$8.7613e+03$

3.5 Compute the inverse Graph Fourier Transform of a graph signal

3.5.1 Description

Write a function in MATLAB that takes as an input a graph shift $S \in \mathbb{R}^{NxN}$ and the frequency coefficients $\tilde{x} \in \mathbb{R}^N$ of a graph signal and outputs $x \in \mathbb{R}^N$, the original signal. Make sure to order the eigenvectors of S in increasing order of absolute value of the associated eigenvalues.

3.5.2 Solution

```
    function x = inverseGraphFourierTransform(S, xs)
        % Compute the eigenvalues and eigenvectors of the graph shift
        [V, D] = eig(S);
        eigenvalues = diag(D);

        % Sort the eigenvectors based on the absolute values of the eigenvalues
        [~, idx] = sort(abs(eigenvalues));
        V_sorted = V(:, idx);

        % Compute the inverse graph Fourier transform
        x = V_sorted * xs;
```

```
    end

%%% main %%%

% Compute the inverse Graph Fourier Transform of a graph signal

xx1 = inverseGraphFourierTransform(S, gft1);
xx2 = inverseGraphFourierTransform(S, gft2);
xx3 = inverseGraphFourierTransform(S, gft3);

rmse(x1, xx1)
rmse(x2, xx2)
rmse(x3, xx3)
```

Listing 8: Inverse Graph Fourier Transform

To be sure that the signal is the same, RMSE of each signal were calculated:

$$RMSE(x_1, x_1^{'}) = 3.2255e - 15 \approx 0$$
$$RMSE(x_2, x_2^{'}) = 5.6771e - 15 \approx 0$$
$$RMSE(x_3, x_3^{'}) = 6.1051e - 15 \approx 0$$

Those results, mean that the original signal was successfully computed.

3.6 Reconstruction and Parseval's theorem

3.6.1 Description

If you need to compress x_1 by only keeping $K = 5$ frequency coefficients, which ones would you keep? Perform the reconstruction and plot the original signal and the reconstructed one. Also, compute the energy of the error. Can you compute the energy of the reconstruction error without actually performing the reconstruction? What quality of the GFT allows you to do this?

3.6.2 Solution

3.7 Denoising a graph signal

3.7.1 Description

Assume that graph signal y loaded from the file `graph_sp_data.mat` is in fact composed of a graph signal z of bandwidth 3 contaminated with white noise. Your objective is to recover z by keeping the correct frequency coefficients. Plot $\tilde{y}, y$ and your recovered signal z.

3.7.2 Solution

Examining the Graph Fourier Transform of the y signal, we see a spike in the 3 first coefficients. Which leads us, that the rest of the signal is some kind of white noise and so on we set to 0 all coefficients except for the first 3.

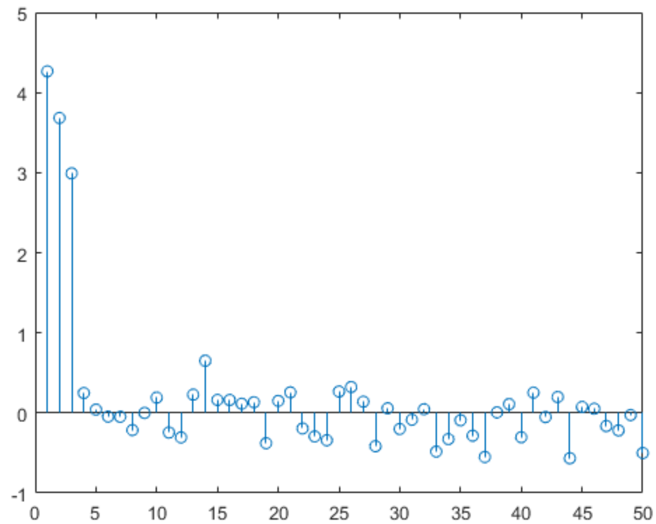

Figure 4: GFT of y

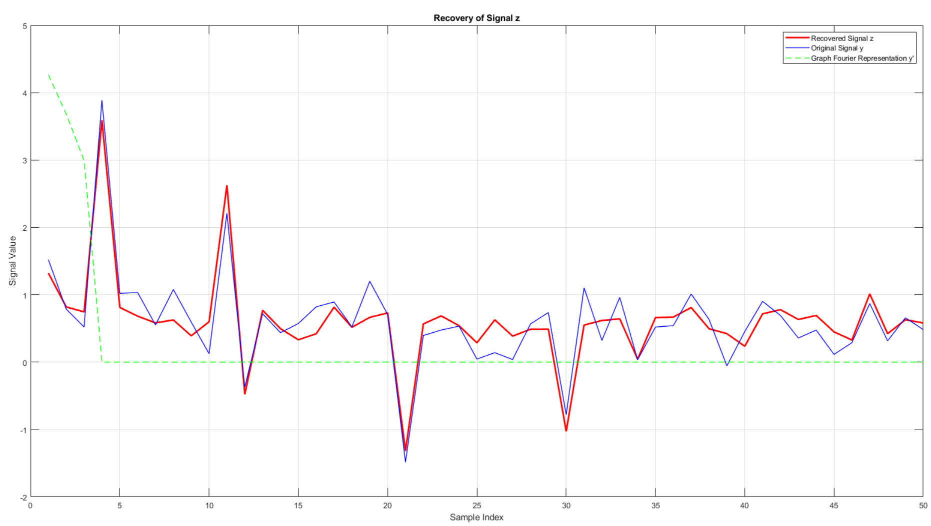

Figure 5: Signals y, y′ and z

```
y_gft = graphFourierTransform(S, y);

% Set all frequency coefficients except the first 3 to zero
y_gft(4:end) = 0;

% Recover the graph signal z by taking the inverse graph Fourier transform
z = inverseGraphFourierTransform(S, y_gft);

% Plotting the recovered signal z, original signal y, and contaminated signal y
figure;
plot(z, 'r', 'LineWidth', 2);
hold on;
```

```
plot(y, 'b', 'LineWidth', 1);
plot(y_prime, 'g--', 'LineWidth', 1);
xlabel('Sample Index');
ylabel('Signal Value');
title('Recovery of Signal z');
legend('Recovered Signal z', 'Original Signal y', 'Graph Fourier Representation y''');
grid on;
```

Listing 9: Denoising a graph signal in MATLAB

References

[1] K. Asaduzzaman, M. B. I. Reaz, F. Mohd-Yasin, K. S. Sim, and M. S. Hussain, “A study on discrete wavelet-based noise removal from eeg signals,” in *Advances in Computational Biology* (H. R. Arabnia, ed.), (New York, NY), pp. 593–599, Springer New York, 2010.