MATLAB report

Tuukka Nurminen AOT 803

2018

Table of contents

Executive Summary 2
Lecture 1. 3
Lecture 2. 3
Lecture 3. 3
Lecture 4. 4
Lecture 5. 5
Lecture 6. 7
Lecture 7. 12
Lecture 8. 16
Lecture 9. 17
Lecture 10. 19
Lecture 11 20
Lecture 12 25
Lecture 13 29
Lecture 14 33
Lecture 15. 38
References 39

Executive Summary

In this course we studied basics of MATLAB. Focus of the course was mostly on data handling. Course subjects varied from basic graph plotting to neural networks. While some of the studied subjects were quite difficult to understand the pacing of the course was excellent. Lectures were good at displaying how deep you can go with the functions and used toolboxes. In all the course gave a good overall picture of using MATLAB as a data analysing tool.

Lecture 1.

I this lecture we got instructions for installing MATLAB.

Lecture 2.

In this lecture we studied the very basic functionalities of MATLAB like how to form variables and how to save them for later use.

Lecture 3.

In this lecture we studied the basics of graphics in MATLAB. Result from my experimenting can be seen below.

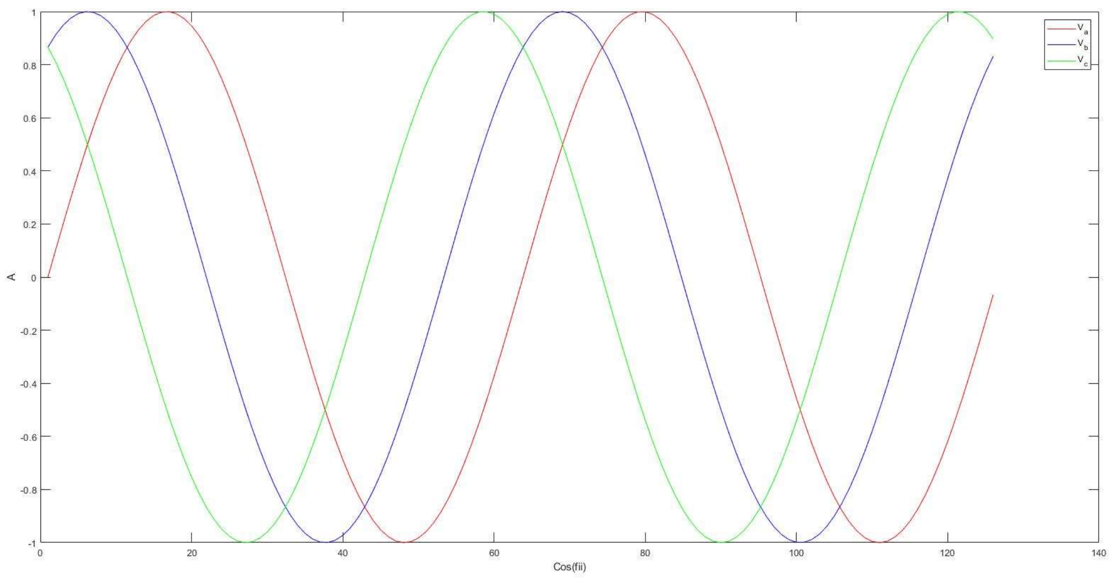


Figure 1. Basic form of three-phase current

Lecture 4.

In this lecture we studied more advanced ways to form graphs in MATLAB. For example, we used different formats for the graphs and we used function *subplot* which creates different areas for plotting in single window. All the results can be seen below.

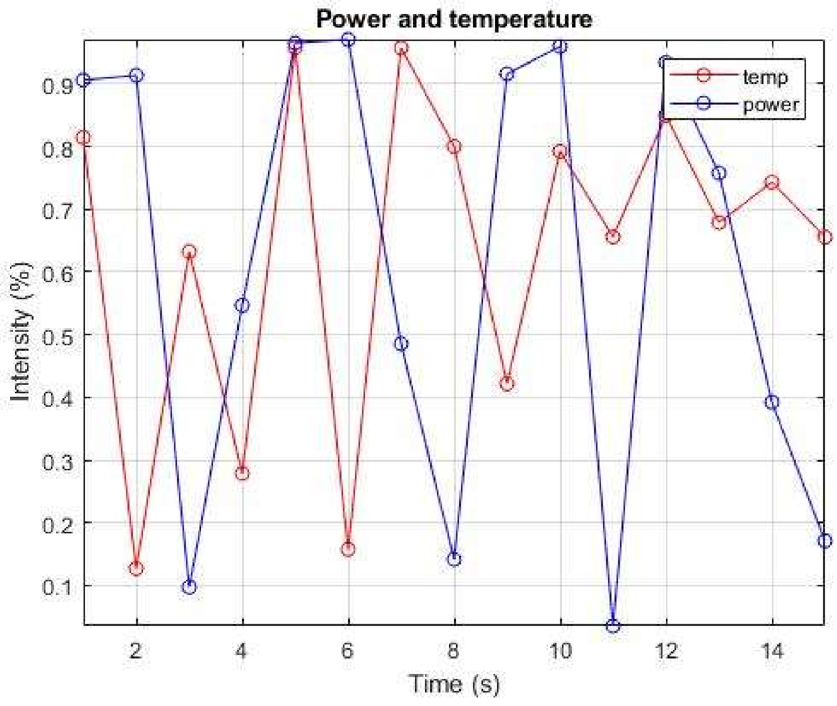


Figure 2. Data plotted with circles representing datapoints and lines connecting them

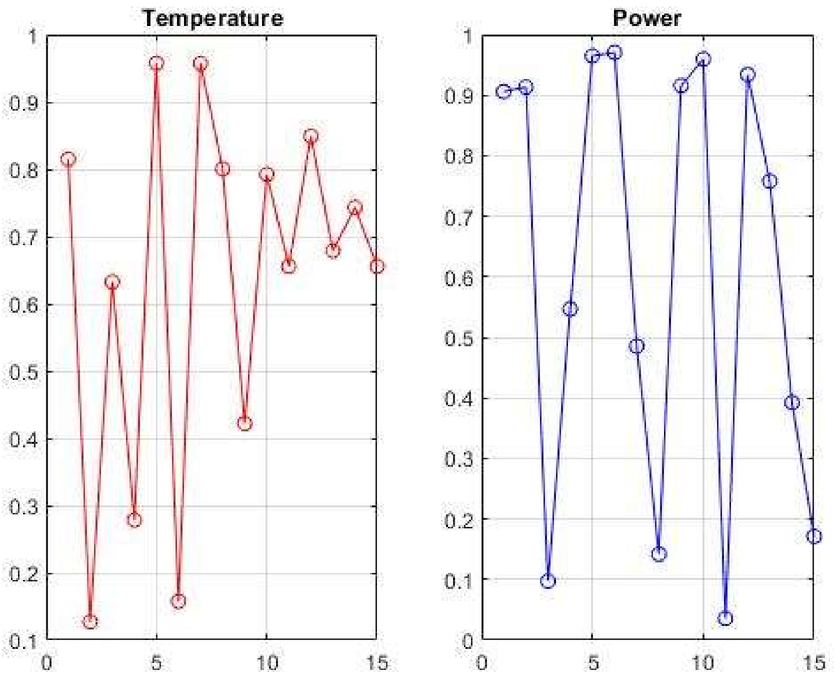


Figure 3. Same graphs separated to two subplots

We also studied how to use plot tools and perform basic fitting for the graphs, result can be seen below.

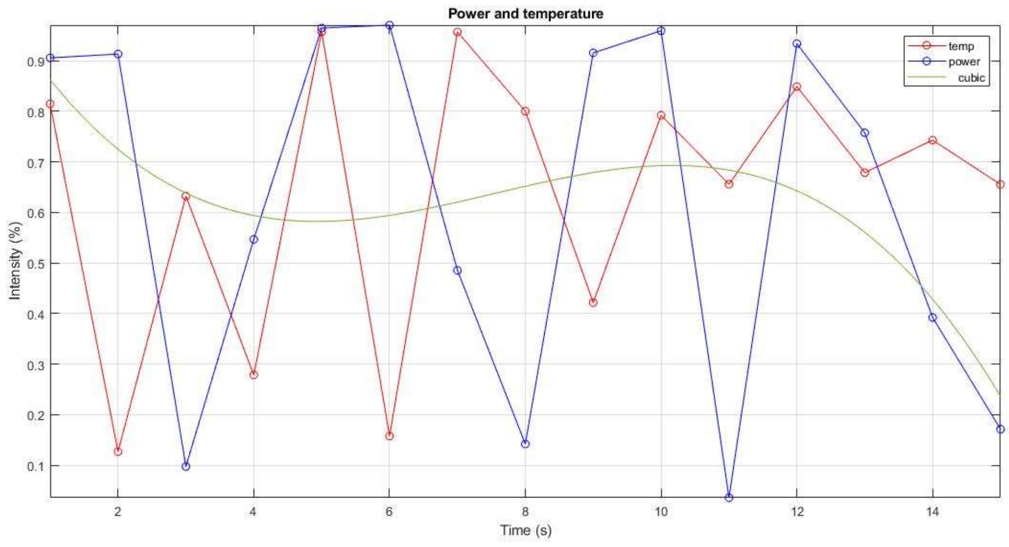


Figure 4. Data plotted with cubic fitting for the power datapoints

Lecture 5.

In this lecture we used curve fitting tool box result can be seen below.

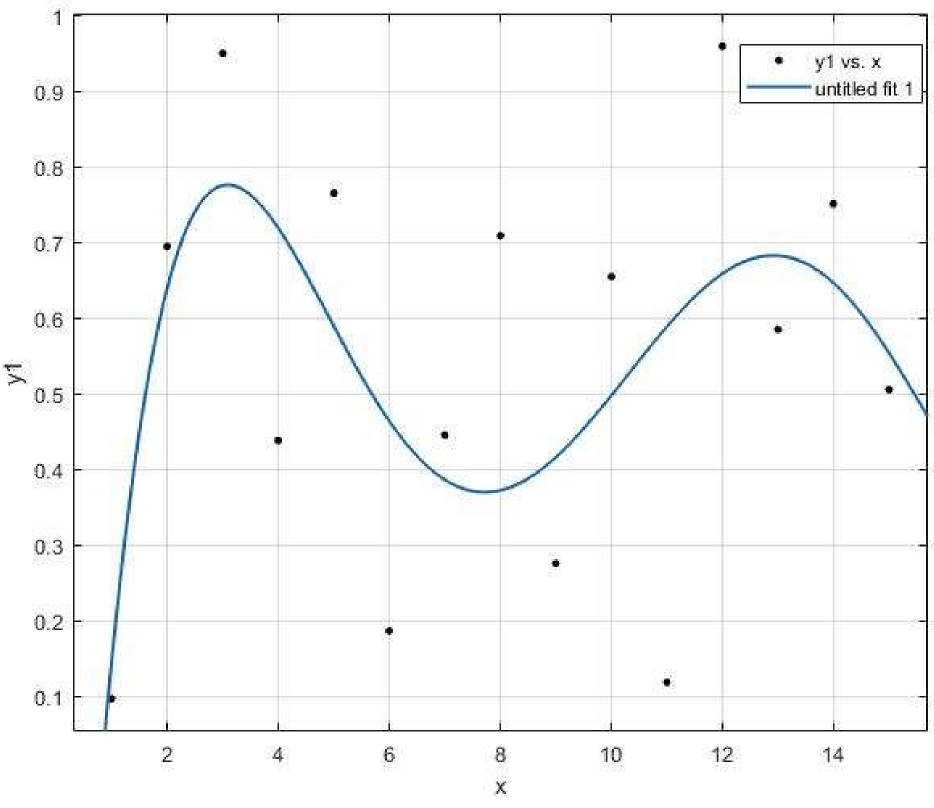


Figure 5. Data points with 5th polynomial curve fitted

We also used function *fit* which we used to fit curve into datapoints utilizing smoothing splines. Result can be seen below.

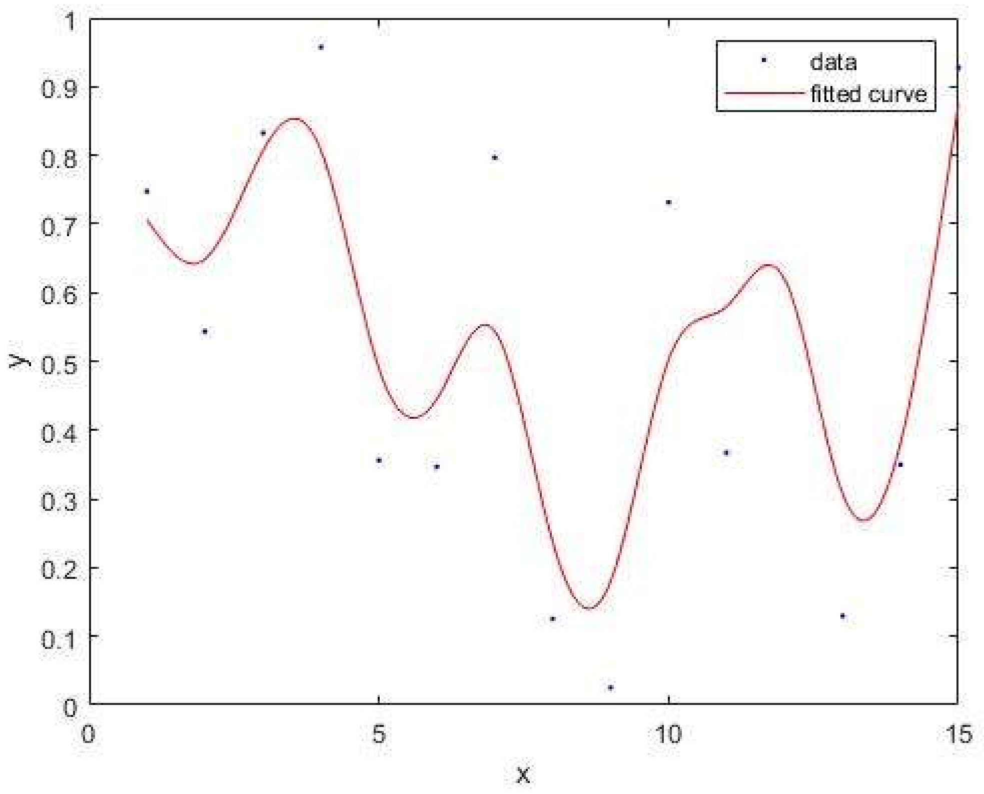
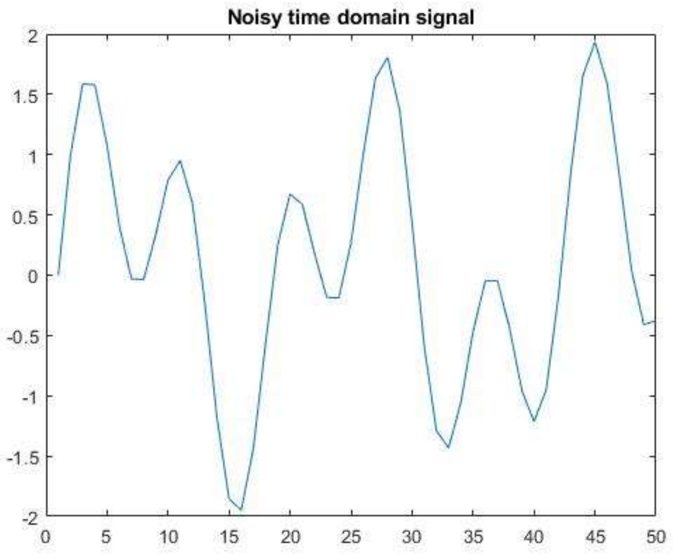


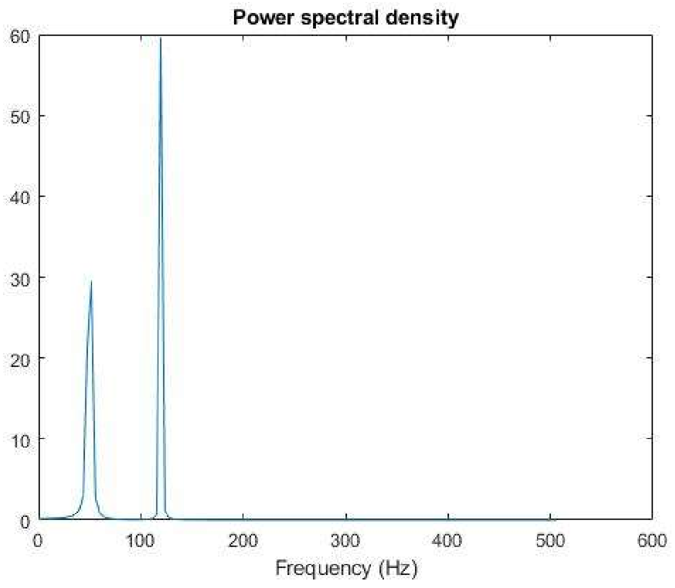
Figure 6. Datapoints with curve fitted into them using smoothing splines

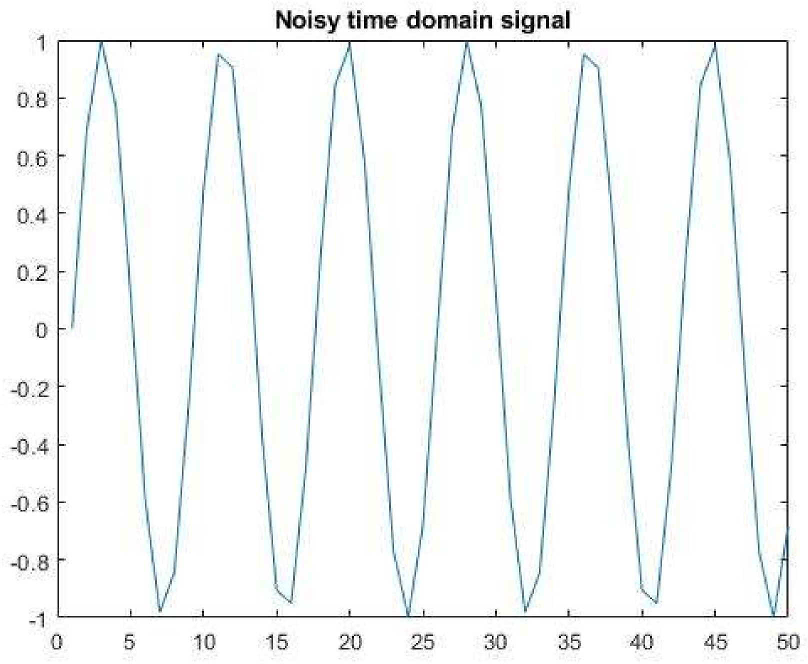
As we can see in the last picture the curve fit a lot better to the datapoints but it's mostly due to usage of more advanced fitting algorithm. Also, because in the cftool we can also determine the smoothing factor of the curve and thus we have more control over the fitting.

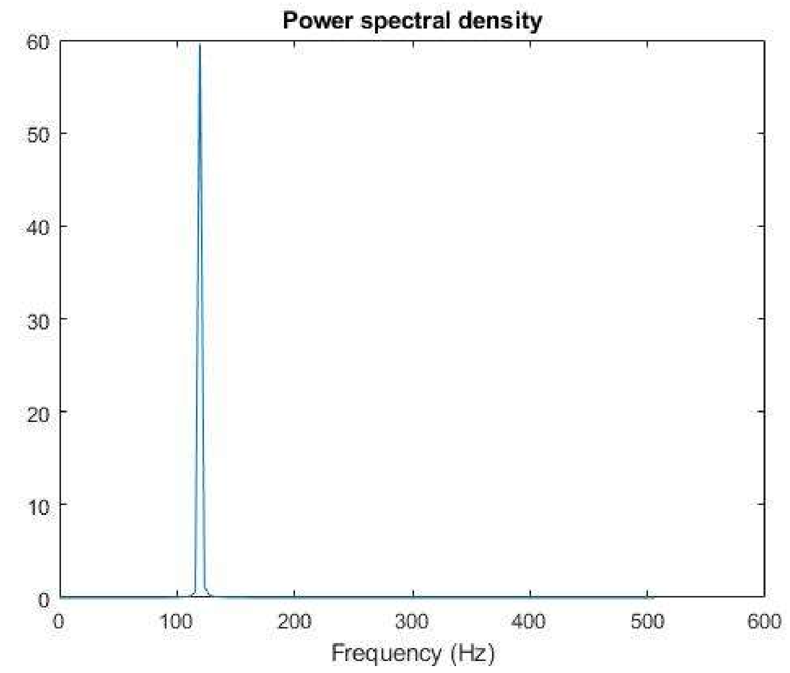
Lecture 6.

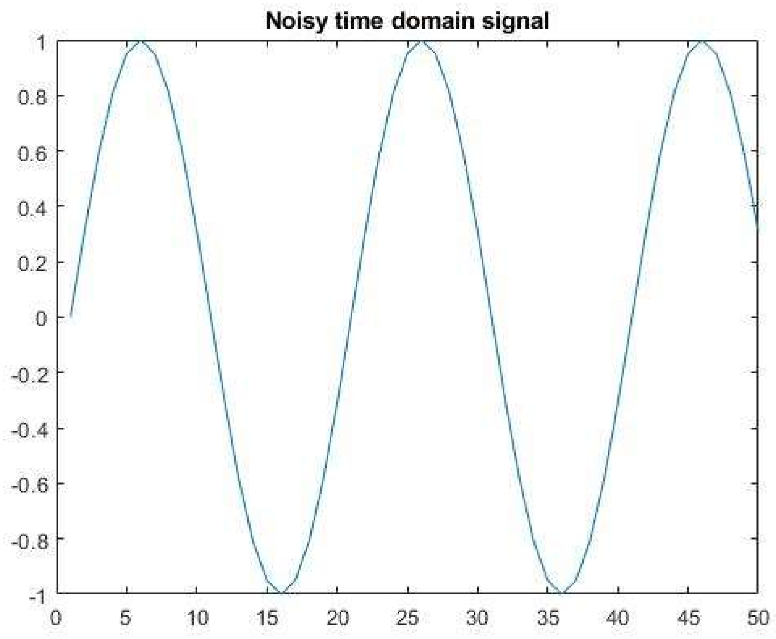
In this lecture we studied functions *fft* and *SNR*. Both functions are utilized when we want to analyse signal with noise.

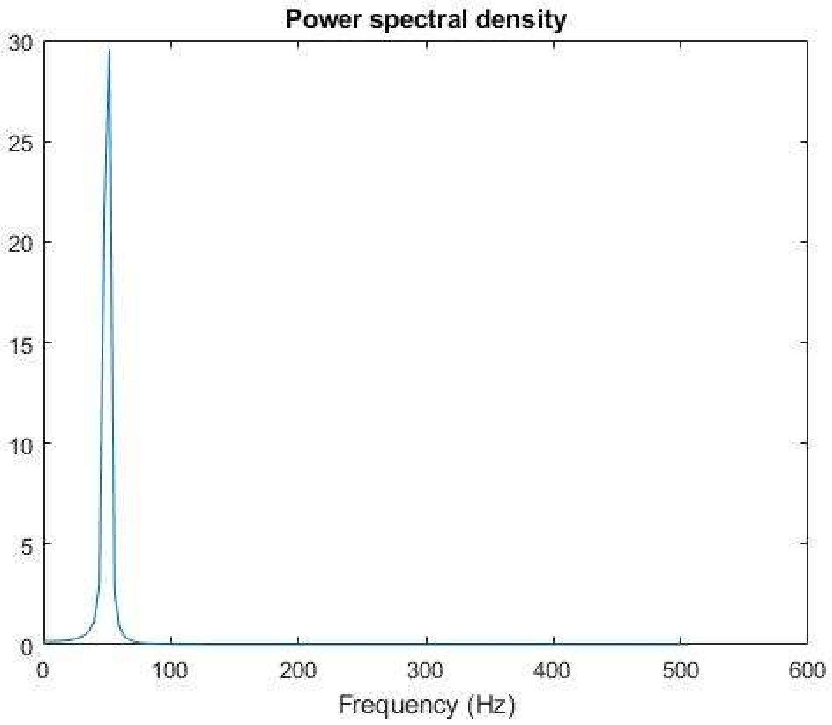





Noisy time domain signal
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
5
10
15
20
25
30
35
40
45
50

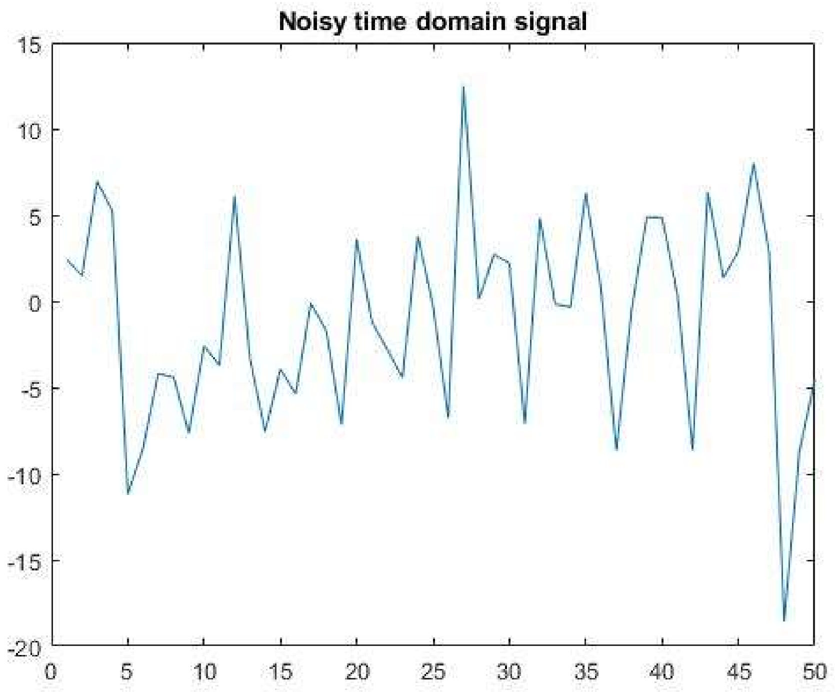

Power spectral density
60
50
40
30
20
10
0
0
100
200
300
400
500
600
Frequency (Hz)

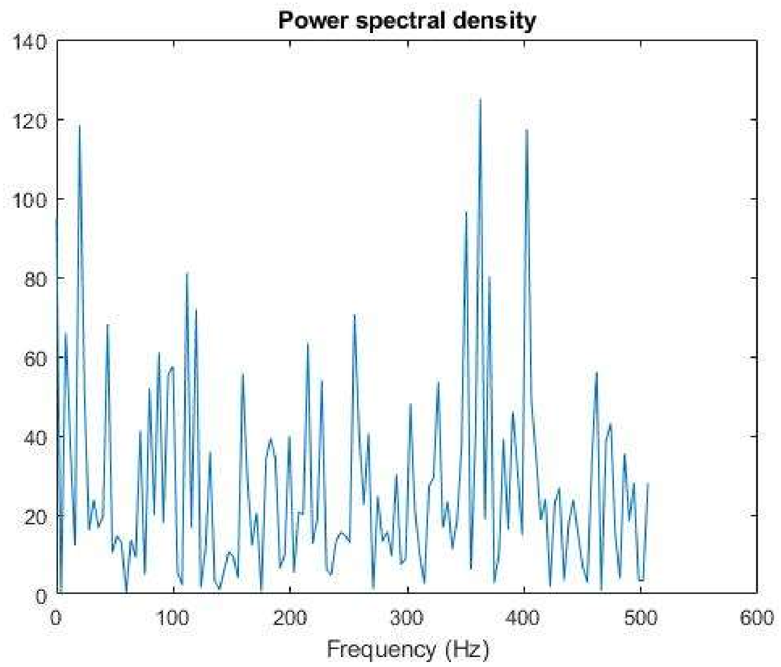




Figures 7.-12. Different signals and effect of fft on them.

As we can see from the graphs the fft picks from the signal amplitude spikes. We can also see that the amplitude of the spike changes with the period of the incoming signal. For the snr we added some additional noise to see with what noise ratio fft spikes become indistinguishable. Results can be seen below.





Figures 13.-14. Same signal as above but with additional noise

Snr here was -13,8395 and we can clearly see that the spikes are very hard to point out of the spectral density graph. More experiments show that when snr reaches value of ~-10 it mixes up the filtering. We also studied the effect of snr on square wave. Results can be seen below.

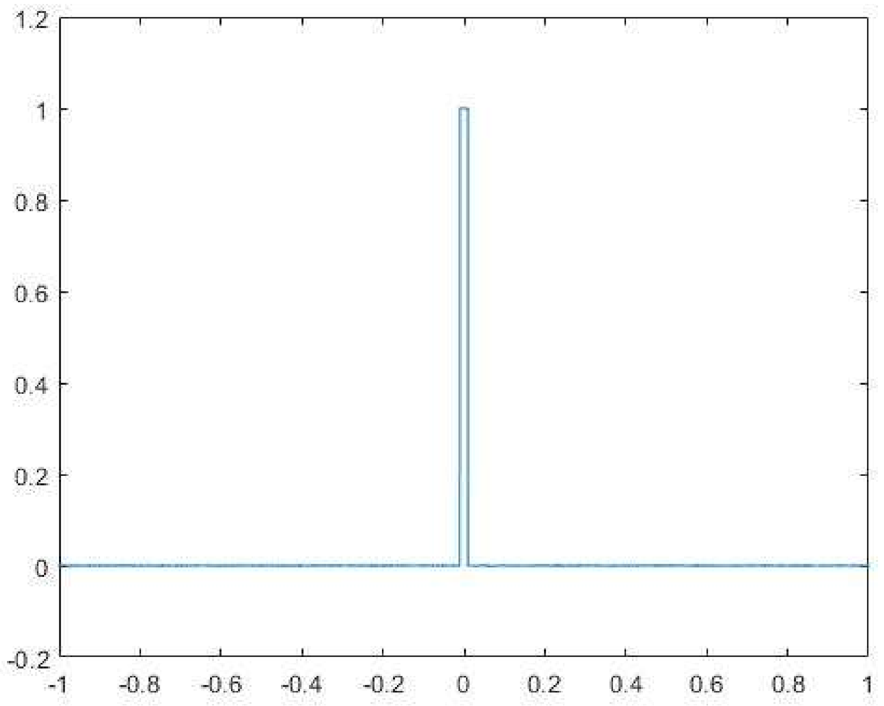


Figure 15. Square wave with very little noise and snr value of 39,9954

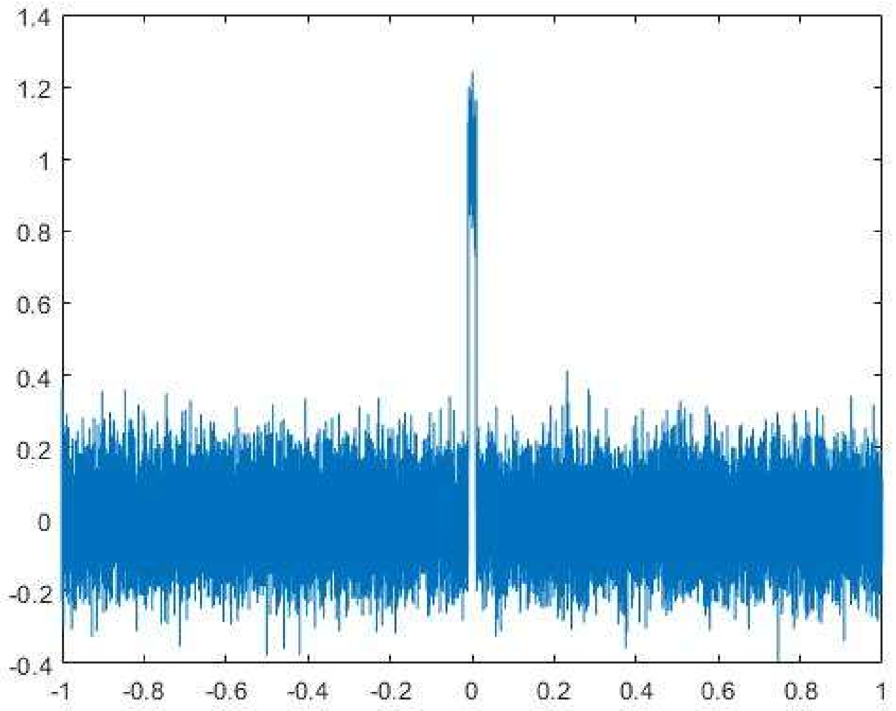
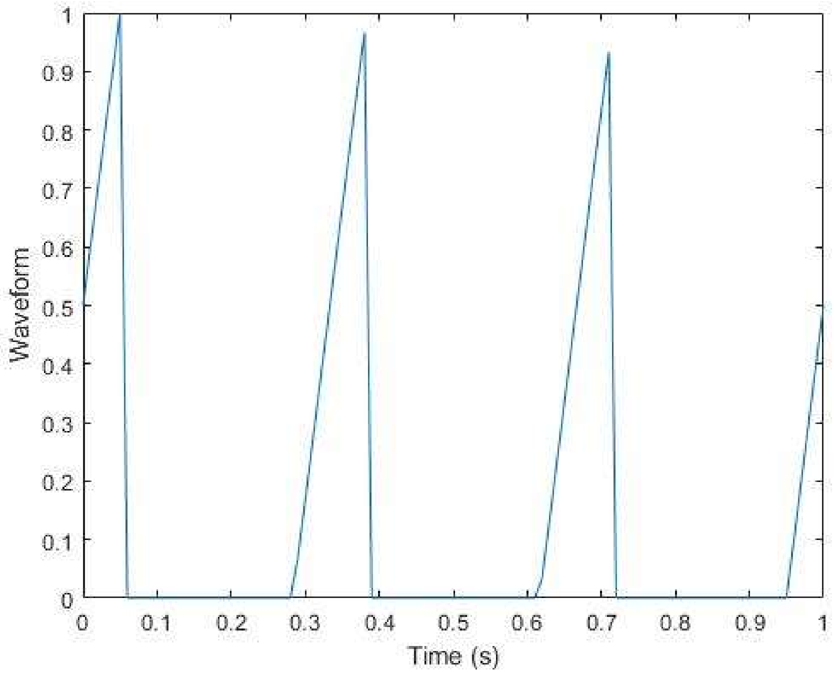


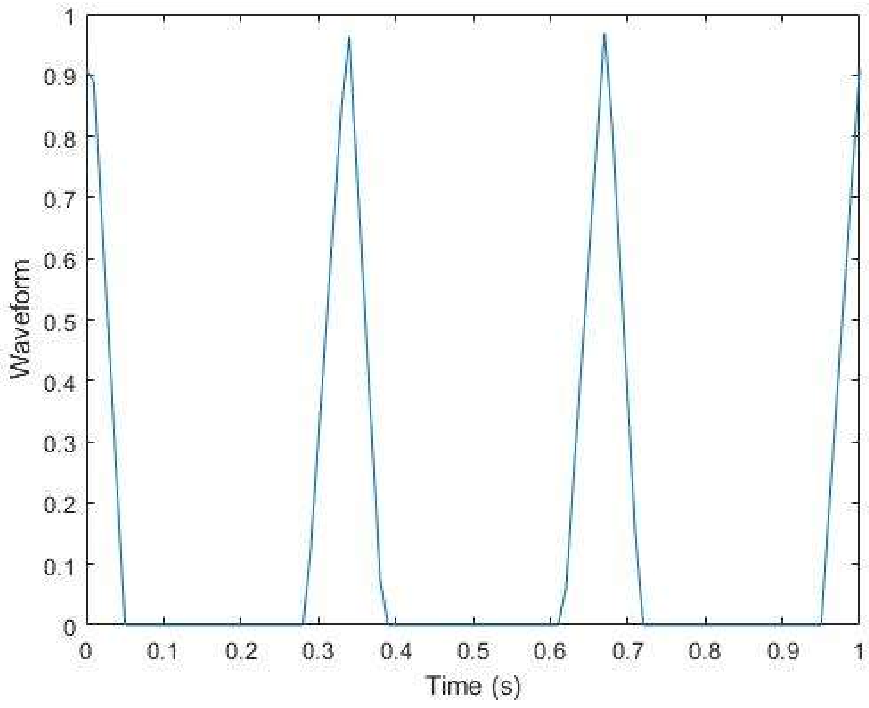
Figure 16. Square wave with a lot of noise and snr value of -0,0046

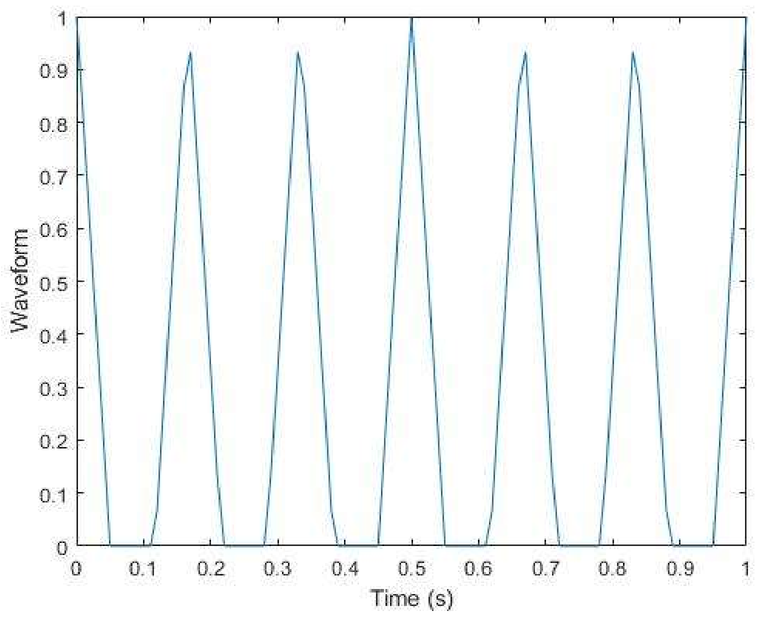
From the graphs we can see that when snr value is or is close to negative values the signal has a lot of noise in it, but the noise is not so loud, and we can distinguish the original signal pulse.

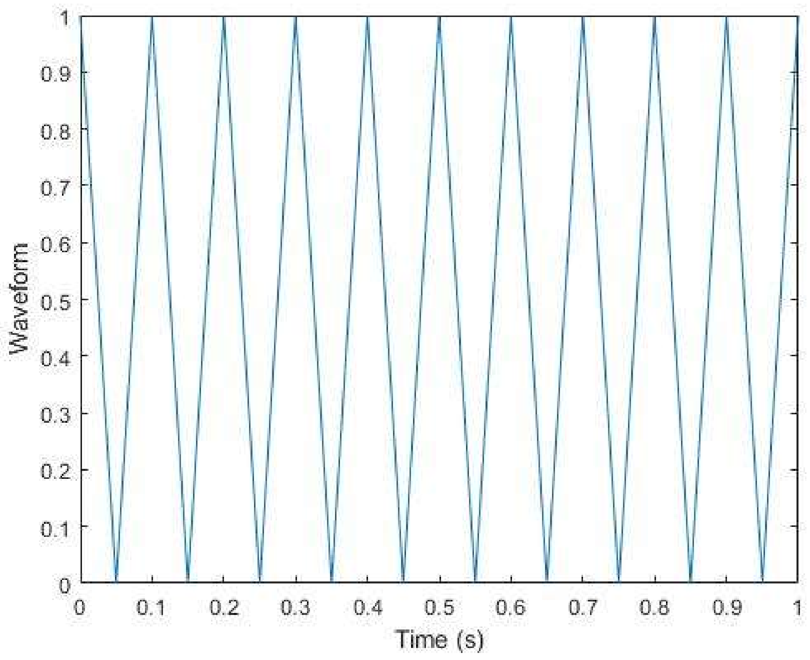
Lecture 7.

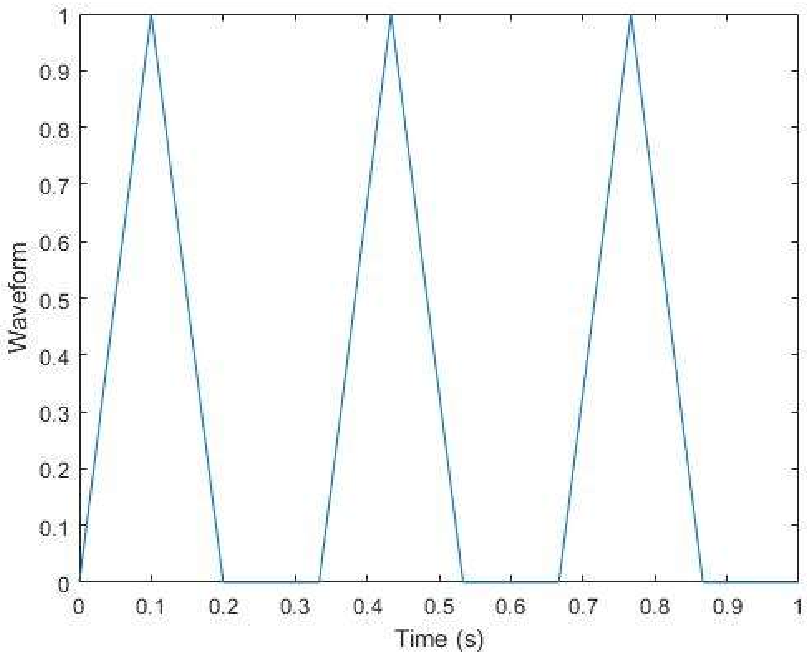
In this lecture we studied different ways to generate signals in MATLAB. First, we studied function *pulstran* which creates signal displayed below we also studied how different parameters change the output form. Results can be seen below.





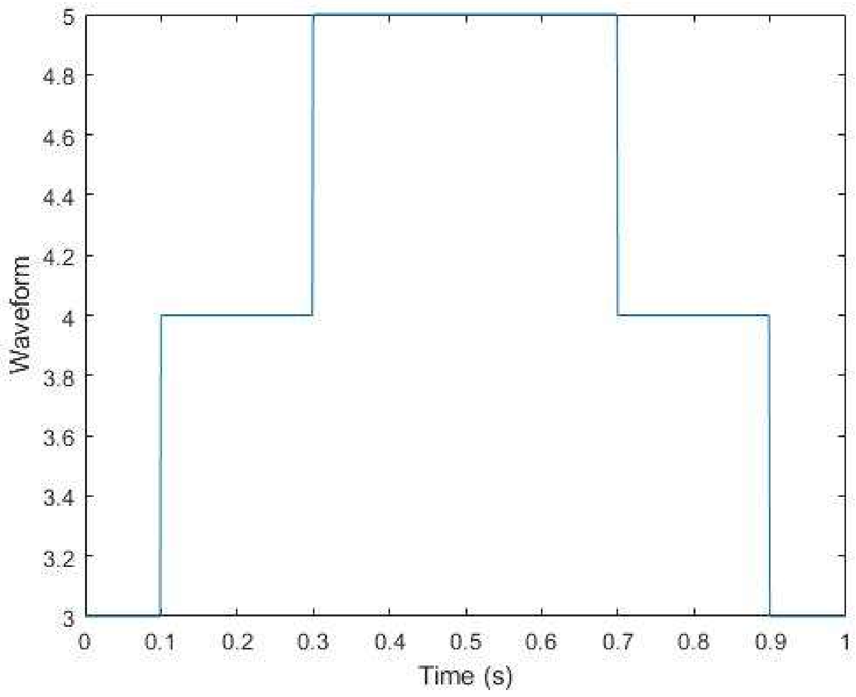

Waveform
Time (s)

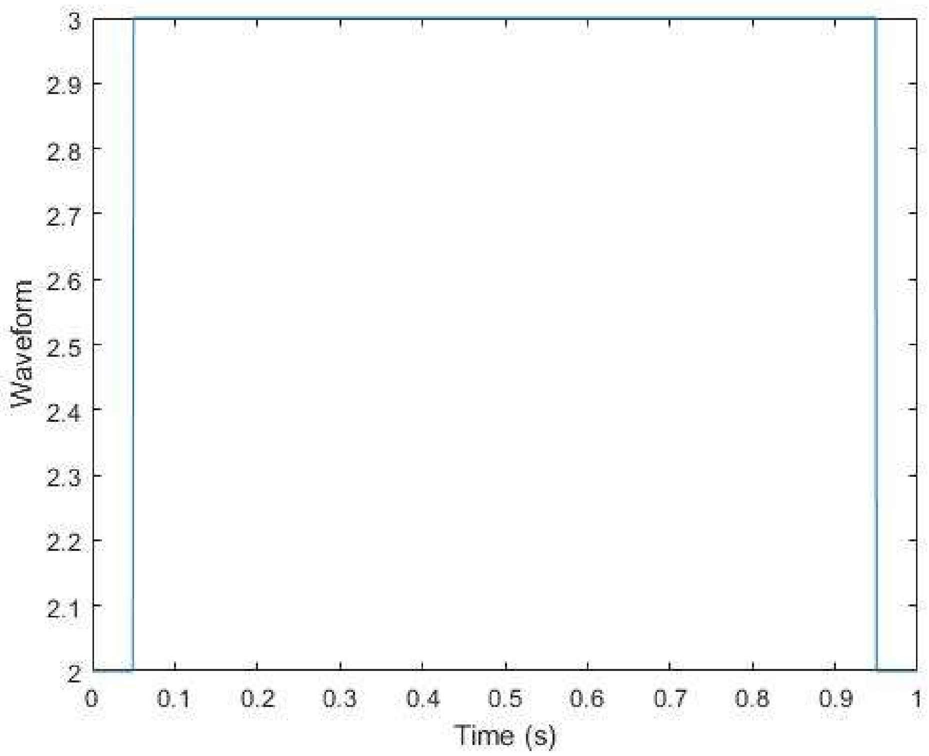

Waveform
Time (s)



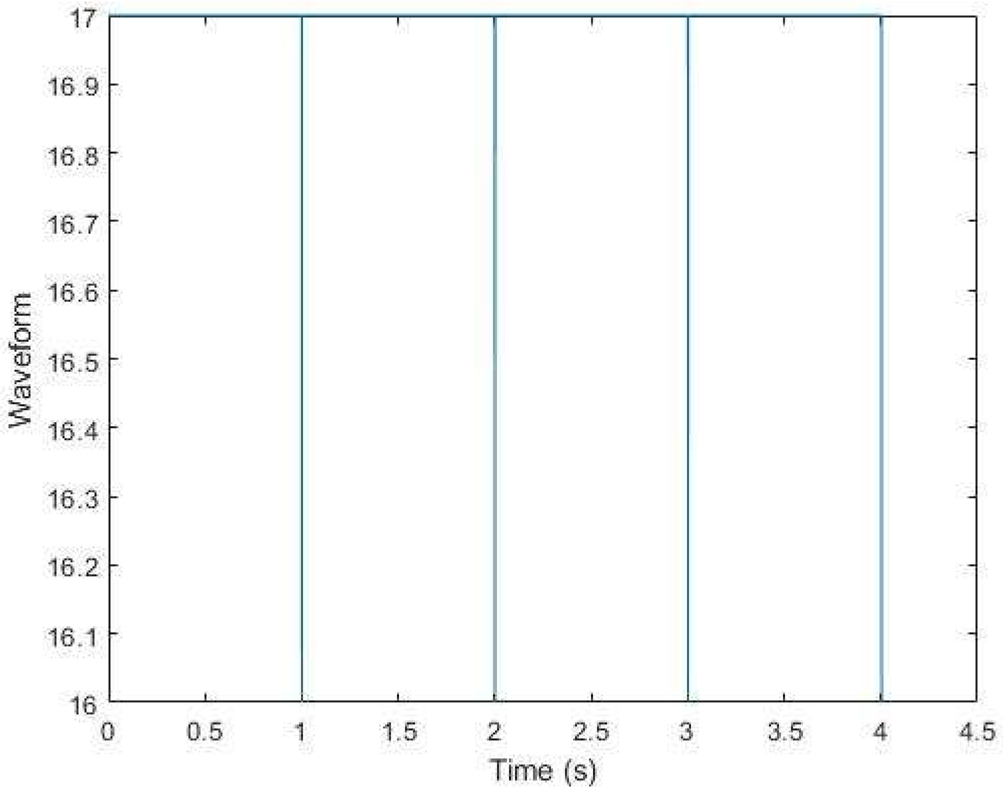
Figures 17.-21. Outputs of pulstrans function with different parameters

We can see that pulstrans is easily customisable for our needs. We studied pulstrans further by changing the form of the graph to rectangular wave. Results can be seen below.

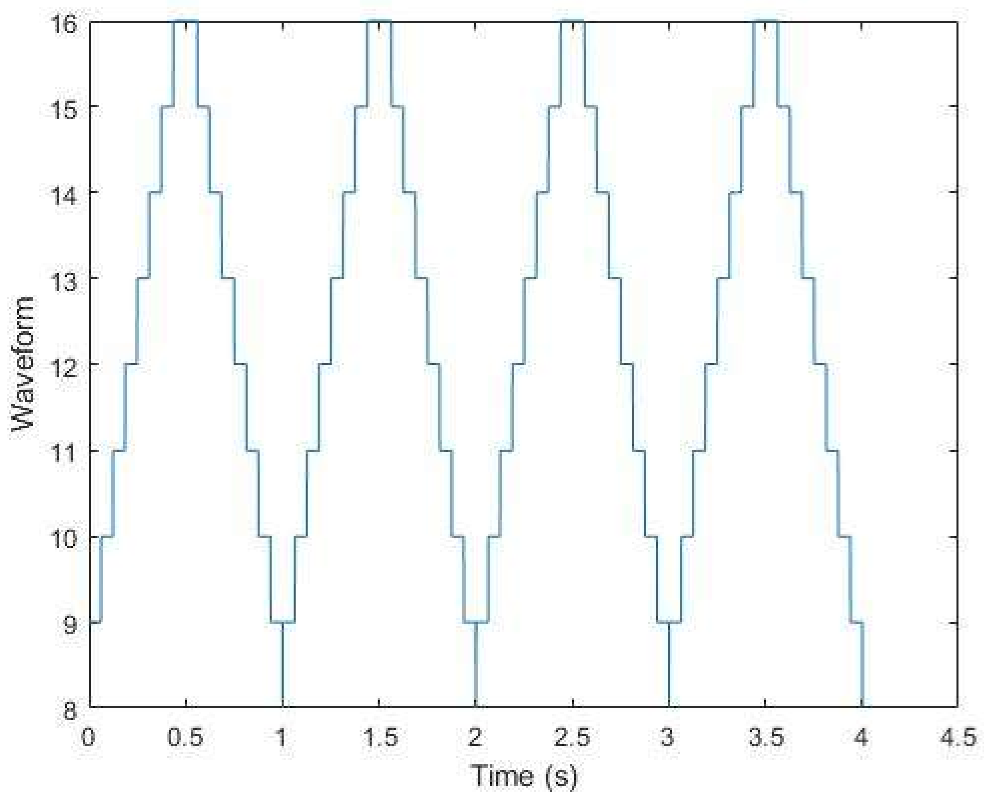




Waveform
Time (s)



Waveform
Time (s)



Figures 22.-25. Pulstrans with square wave argument and different parameters

Same as before we can easily change the output to fit for our needs. We can also change the argument of the pulstrans to *gauspuls, sawtooth* or *square* all of these gives the output different form. In all we can conclude that pulstrans is very versatile function and we can use it to generate variety of waveforms and easily modify them to fit our needs.

Lecture 8.

Lecture was about importing data from Excel. First, we did this by taking the data from excel manually and got figure. Figure is in a form that is often required for science publications.

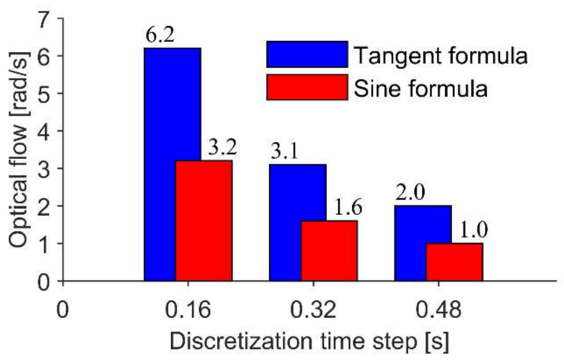


Figure 26. Manually extracted data plotted as bar graph

Second, we used MATLAB code to get the information from the excel. Result can be seen below.

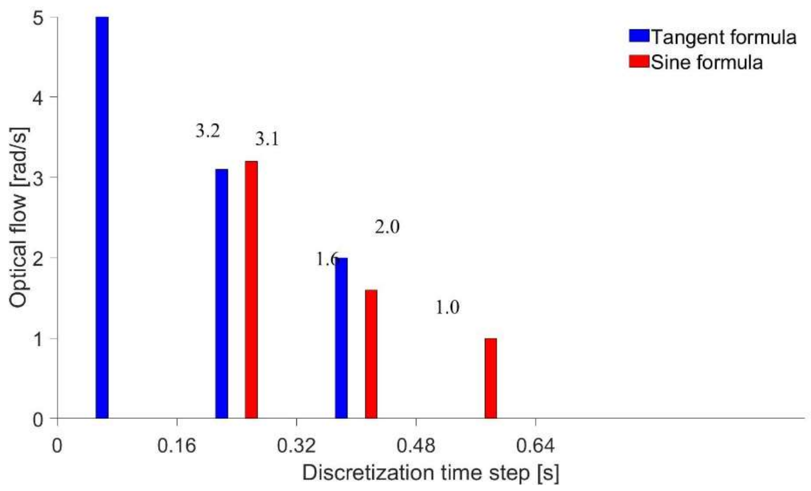
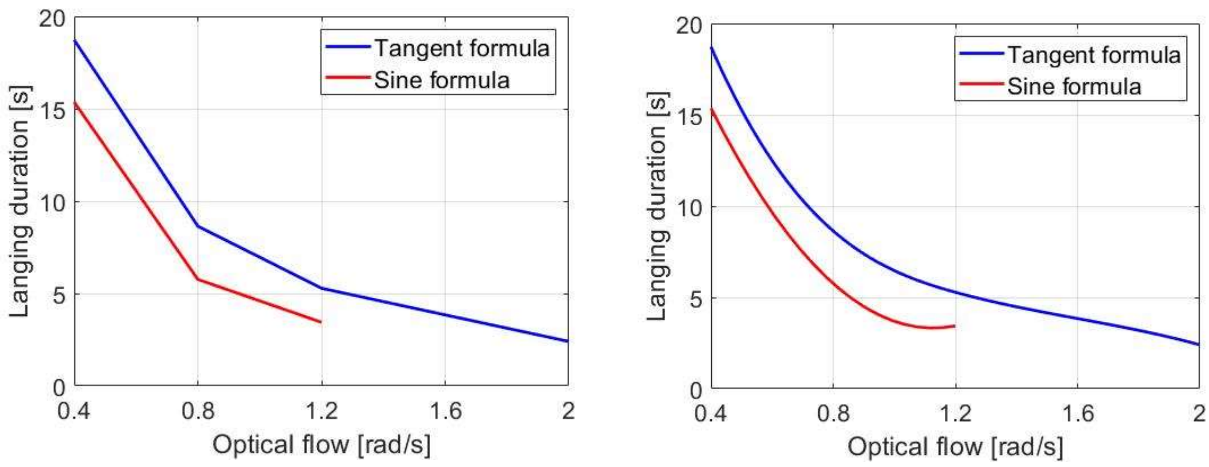


Figure 27. Result graph of the coded data gathering

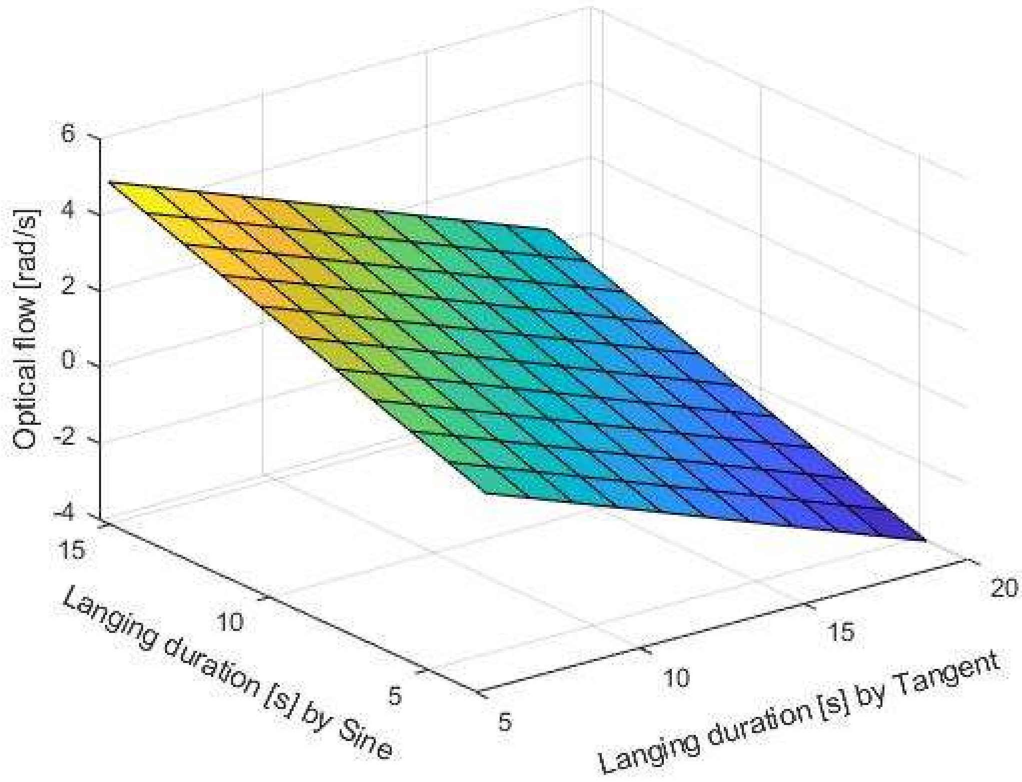
Lecture 9.

We practiced programming linear- and 3D graphs using MATLAB. First, we used MATLAB code to smooth curve first using function *linspace* which generates linearly spaced vectors. Second, we used for smoothing function called spline which is a mathematical way to interpolate curve into set of data. Results for smoothing can be seen below.

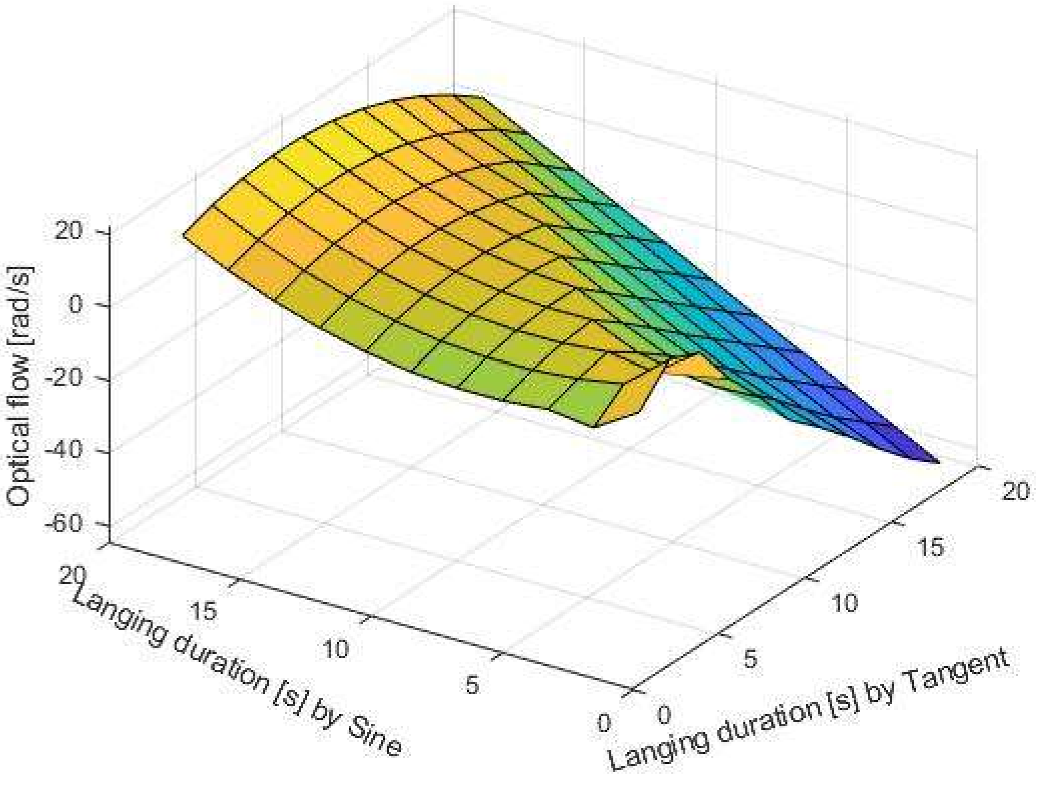


Figures 28.-29. Effect of smoothing in a graph

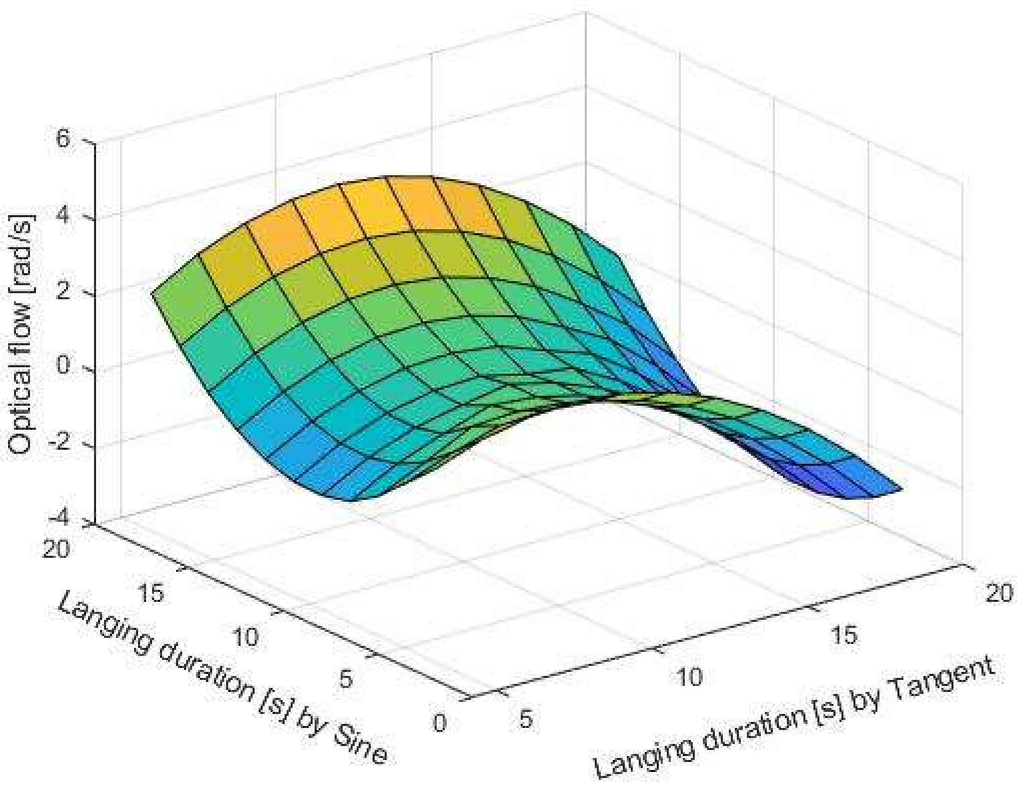
For 3D -graphs we used function *meshgrid* to plot out the data. Result can be seen below.



6
4
2
0
-2
-4
Optical flow [rad/s]
15
10
5
Langing duration [s] by Sine
5
10
15
20
Langing duration [s] by Tangent



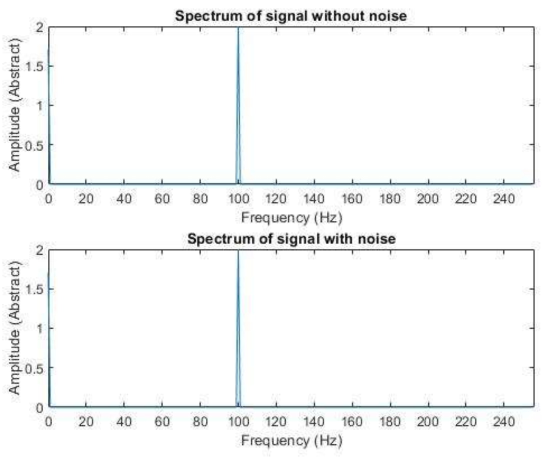
20
0
-20
-40
-60
Optical flow [rad/s]
20
15
10
5
0
Langing duration [s] by Sine
0
5
10
15
20
Langing duration [s] by Tangent

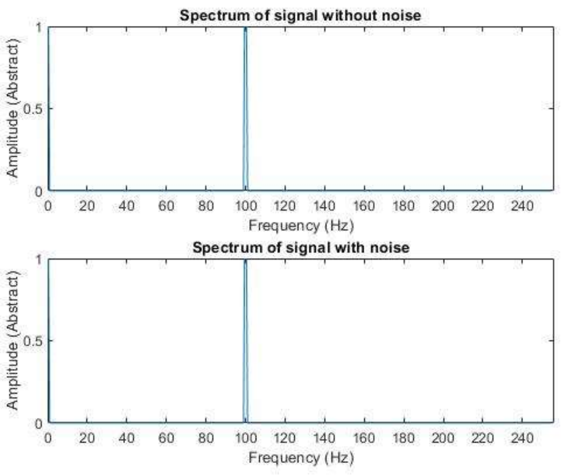


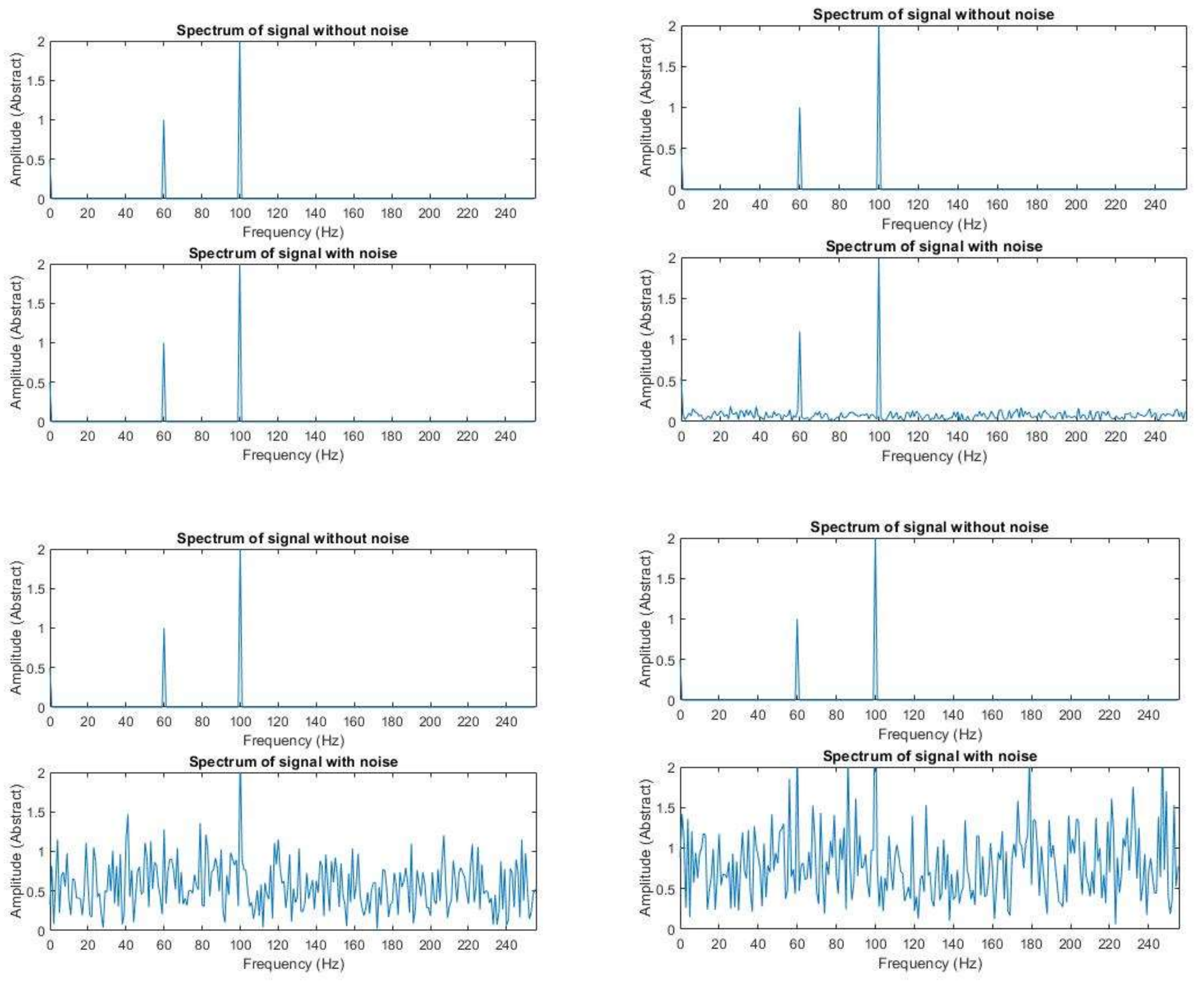
Figures 30.-32. Smoothing effect on a 3D graph.

Lecture 10.

In this lecture we studied the effects of fft (Discrete Fourier transform) -function. We also used functions like *sind* which means sine function with degreases, *randn* which creates random (gaussian) numbers and *subplot* creates subsection into one figure. The results were depicted as a sinusoidal signal and same signal with noise in it.







Figures 33.-39. Effect of fft on different signals

From the pictures we can see that the more there is noise the more help we can get out of FFT. For example, in the last picture we can't even identify the peaks and in the second last picture it's getting worse. The run section and breakpoint functionalities allow us to debug the MATLAB code more efficiently. Also, it gives more defined structure for your MATLAB code. In bigger programs dividing the code into sections gives you ability to run only the parts you want to show without needing to run the whole program.

Lecture 11

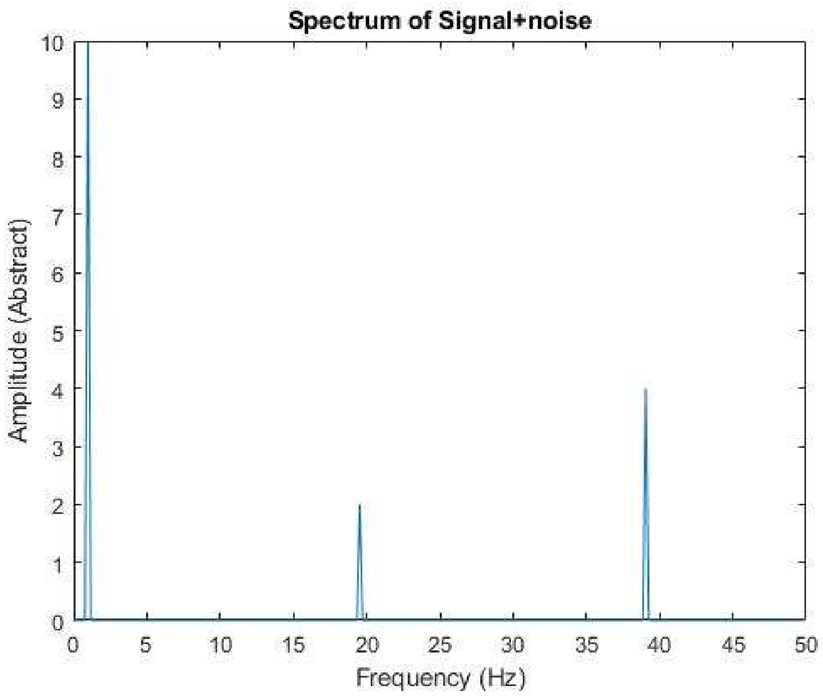
A digital filter is characterized by its transfer function H. In common it is:

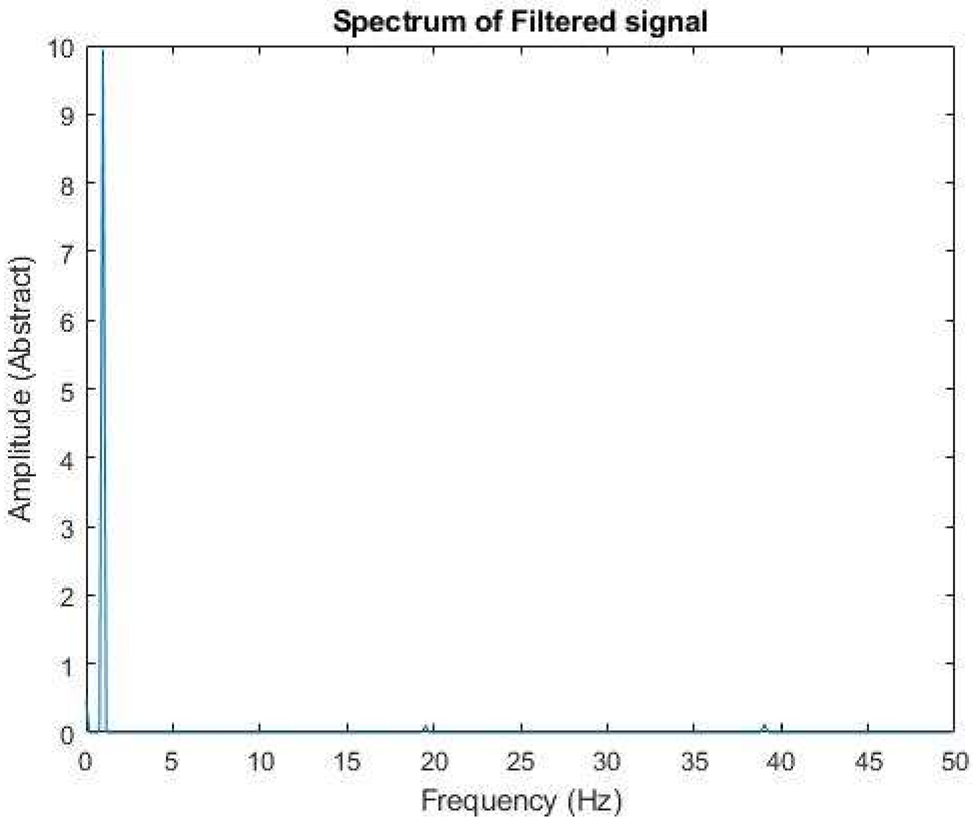
$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 + \sum_{k=1}^{N} a_k z^{-k}} = \frac{B(z)}{A(z)},$$

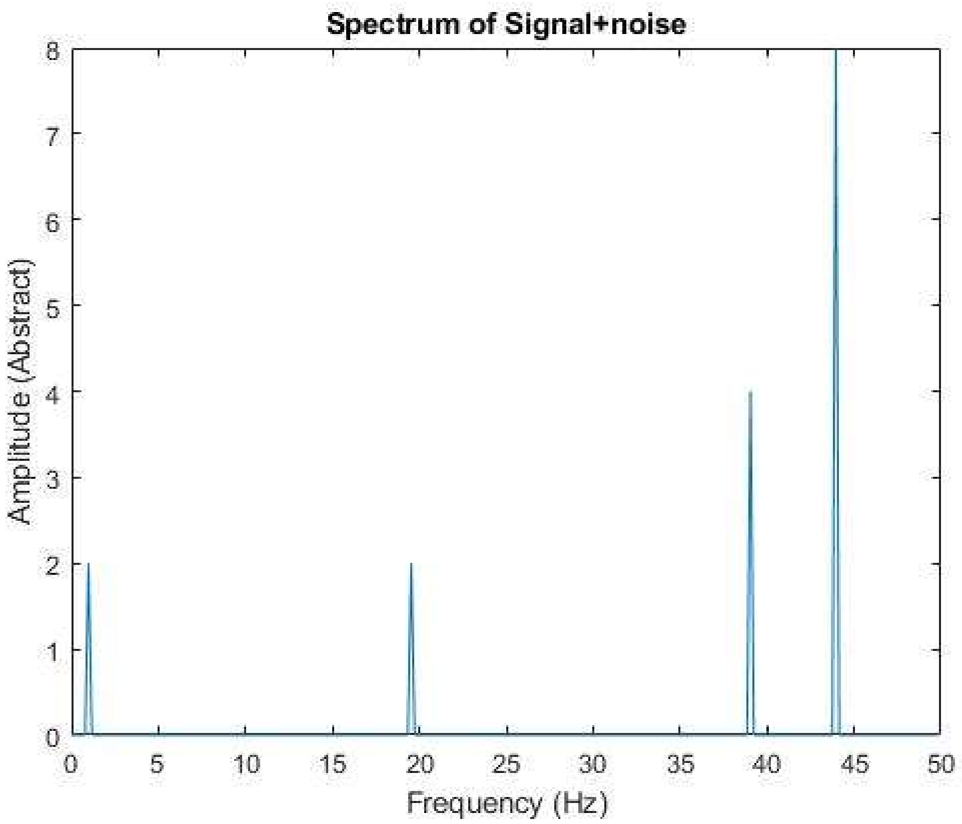
where X is the input and Y is the output. MATLAB has a wide tool for Digital Filter Design. But we will learn a simple filtering by 1-D digital filter, embedded in MATLAB. It has the simple transfer function:

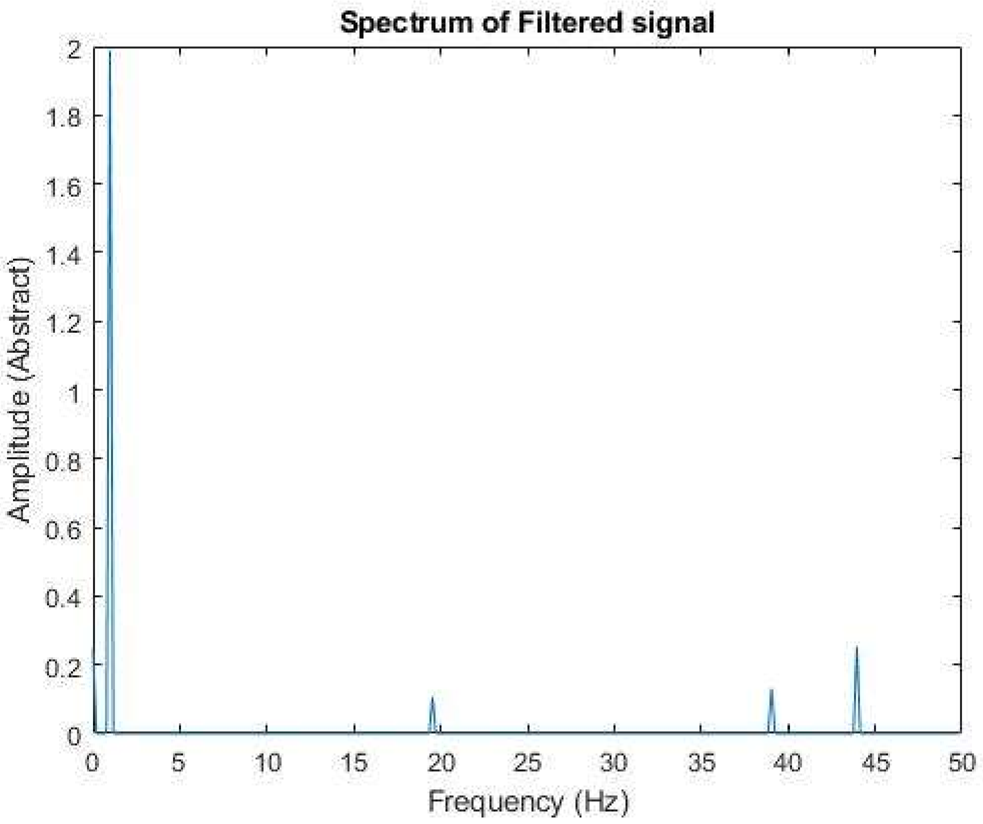
$$H(z) = \frac{b(1)}{a(1) + a(2)z^{-1}}:$$

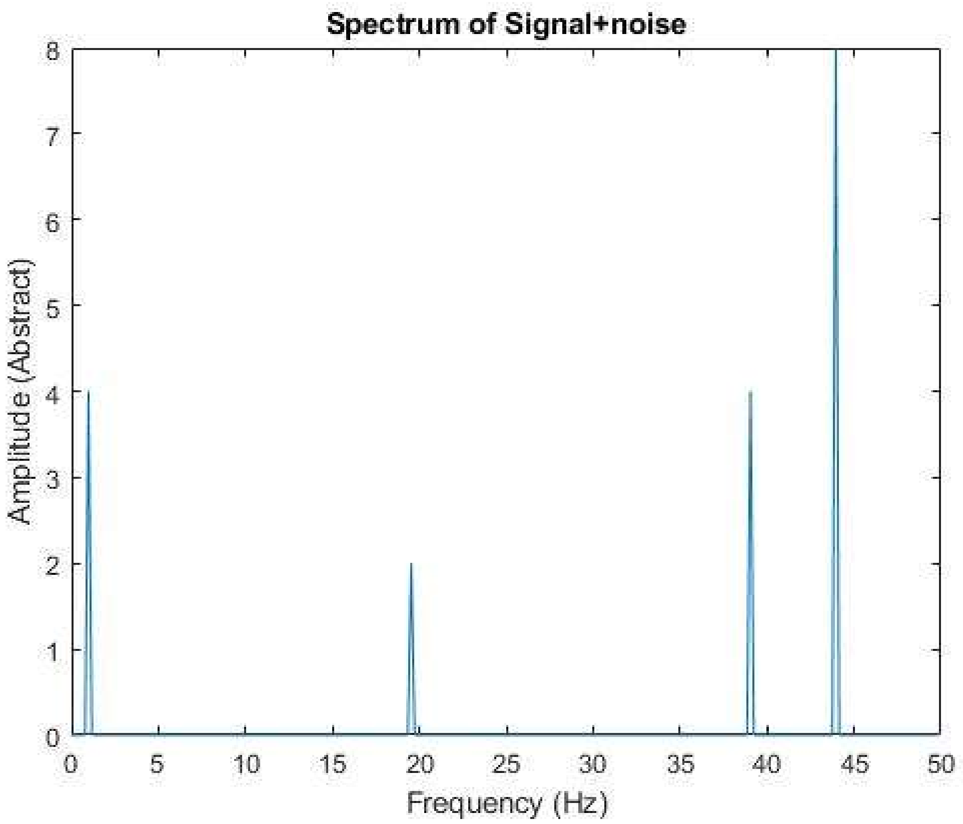
In MATLAB we use function *filter* which uses function mentioned before. *Filter* function has parameters a and b. The parameter b determines the amplitude of the filtered signal and is a single value. The parameter a is a row-vector of two values. Different effects of filters can be seen in graphs below.

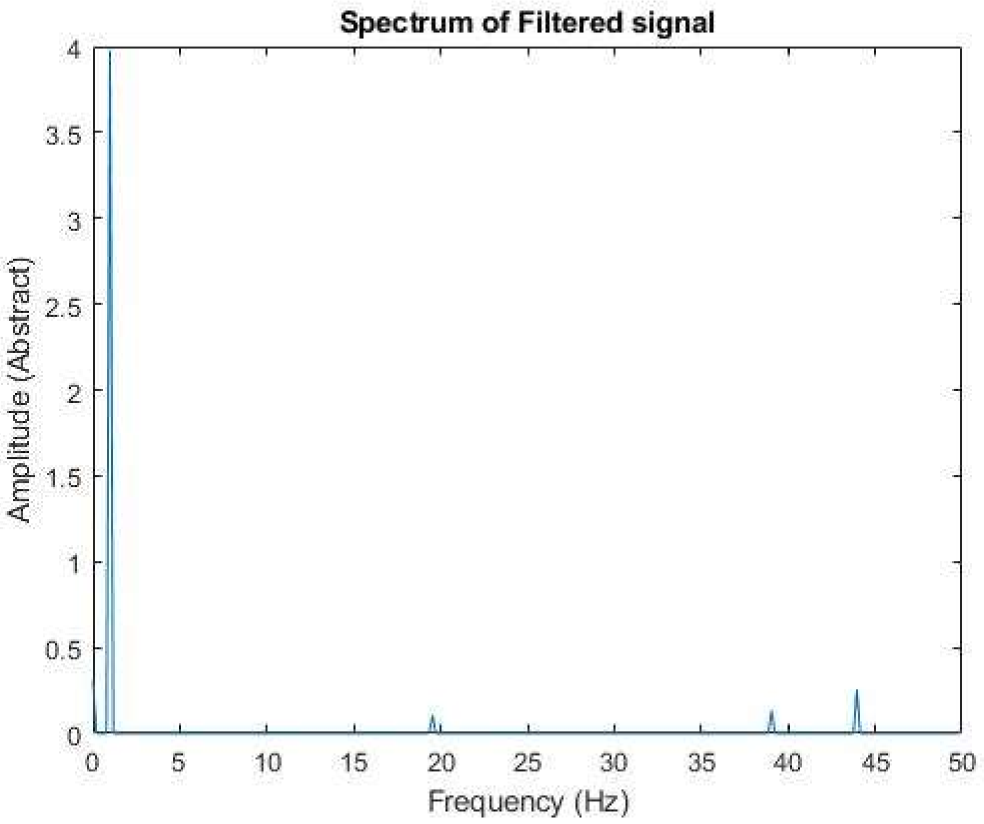


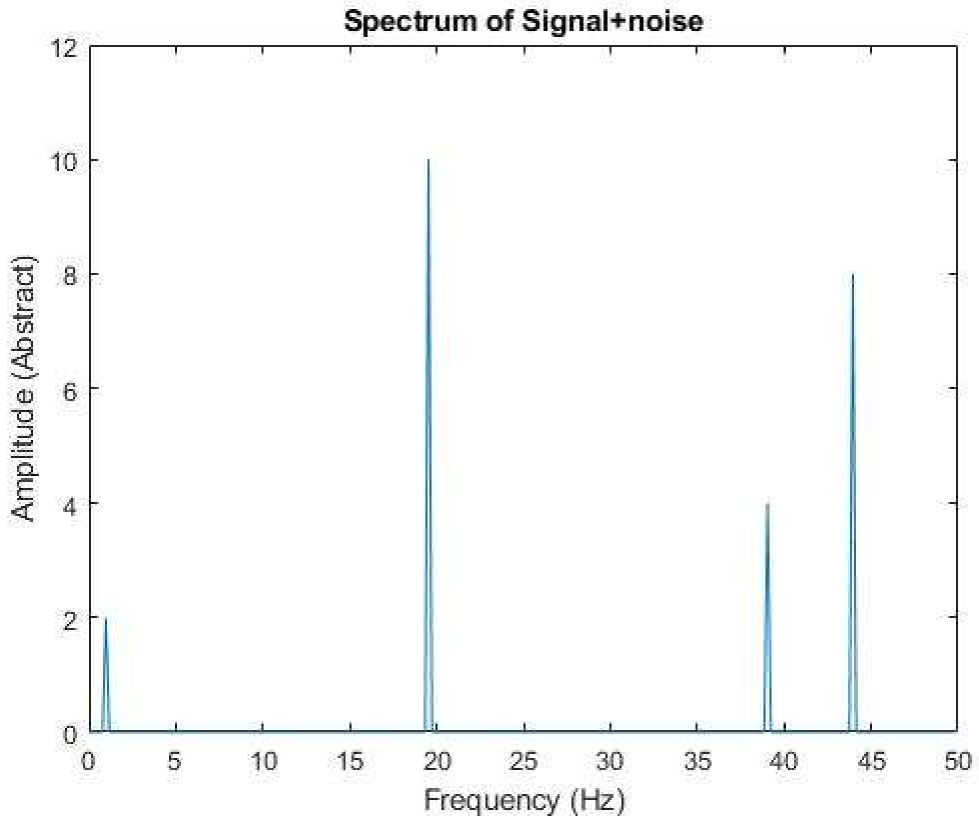


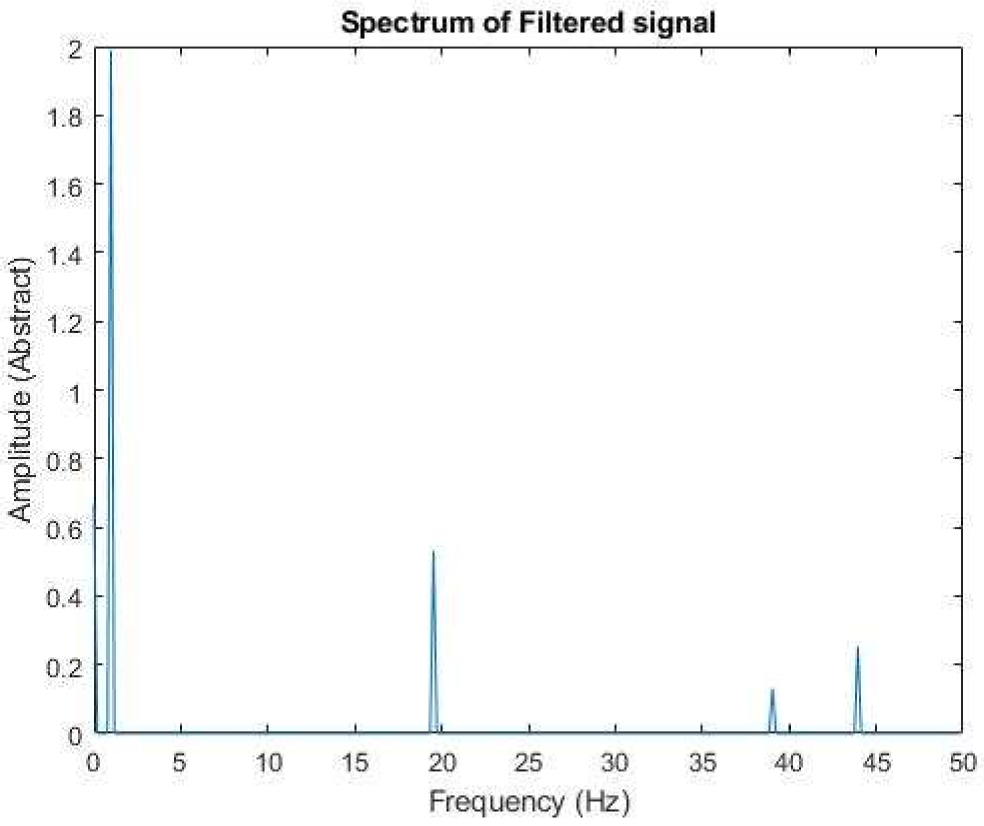

Spectrum of Signal+noise
Amplitude (Abstract)
Frequency (Hz)
0
1
2
3
4
5
6
7
8
0
5
10
15
20
25
30
35
40
45
50


Spectrum of Filtered signal
Amplitude (Abstract)
Frequency (Hz)
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
0
5
10
15
20
25
30
35
40
45
50


Spectrum of Signal+noise
Amplitude (Abstract)
Frequency (Hz)
0
1
2
3
4
5
6
7
8
0
5
10
15
20
25
30
35
40
45
50


Spectrum of Filtered signal
Amplitude (Abstract)
Frequency (Hz)
0
0.5
1
1.5
2
2.5
3
3.5
4
0
5
10
15
20
25
30
35
40
45
50



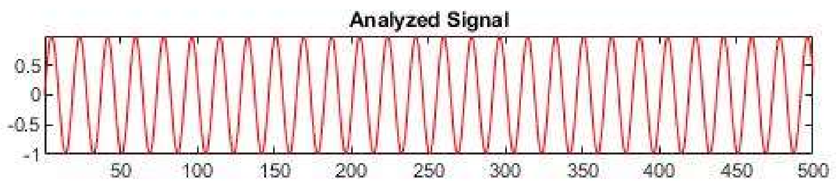


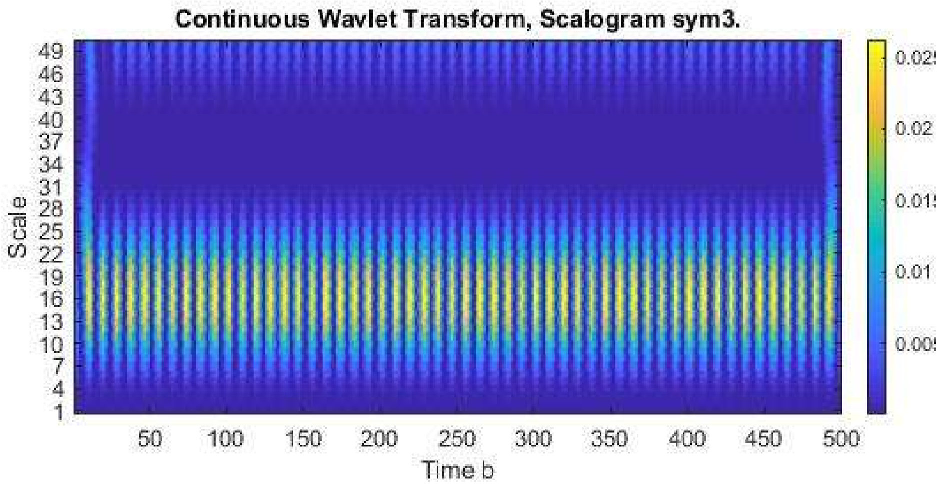
Figures 40.-47. Effect of build filter on different signals

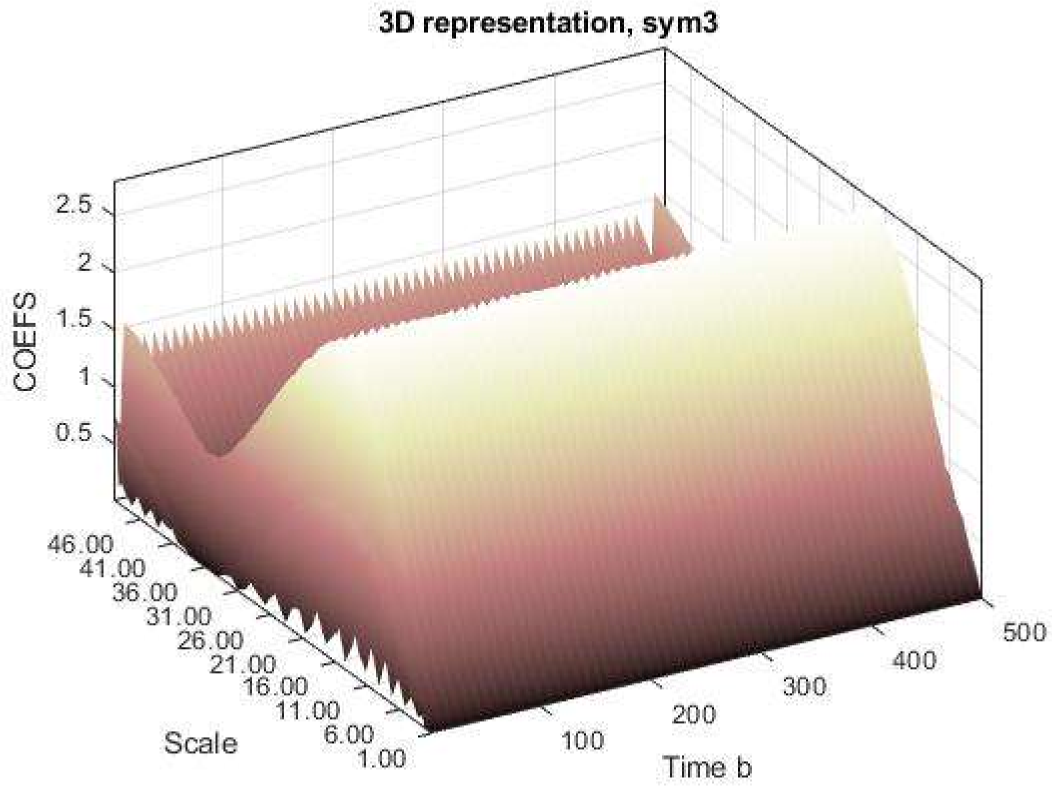
We can clearly see that the used filter amplifies lower frequencies and filters heavily higher frequensies.

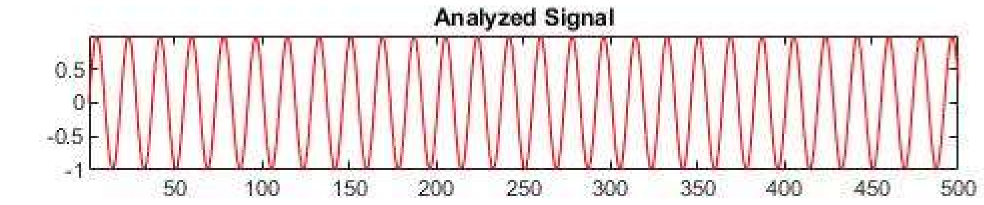
Lecture 12

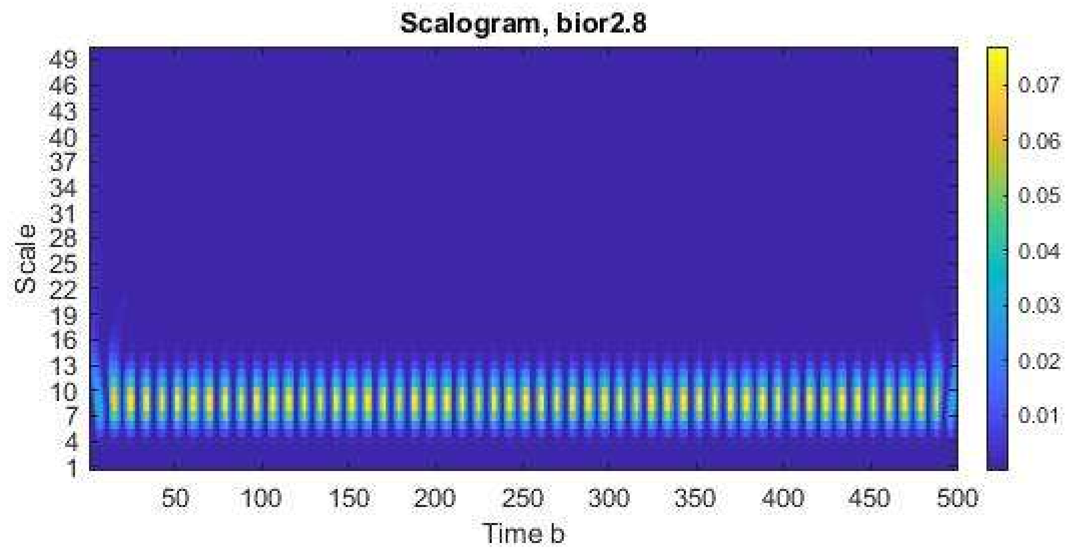
In lesson 12 we studied function *CWT* which utilizes wavelets in analysing signal. Compared to *FFT CWT* displays the time dependency of the frequencies instead of just the amplitudes. Results for depicting sound signals with wavelets can be seen below.

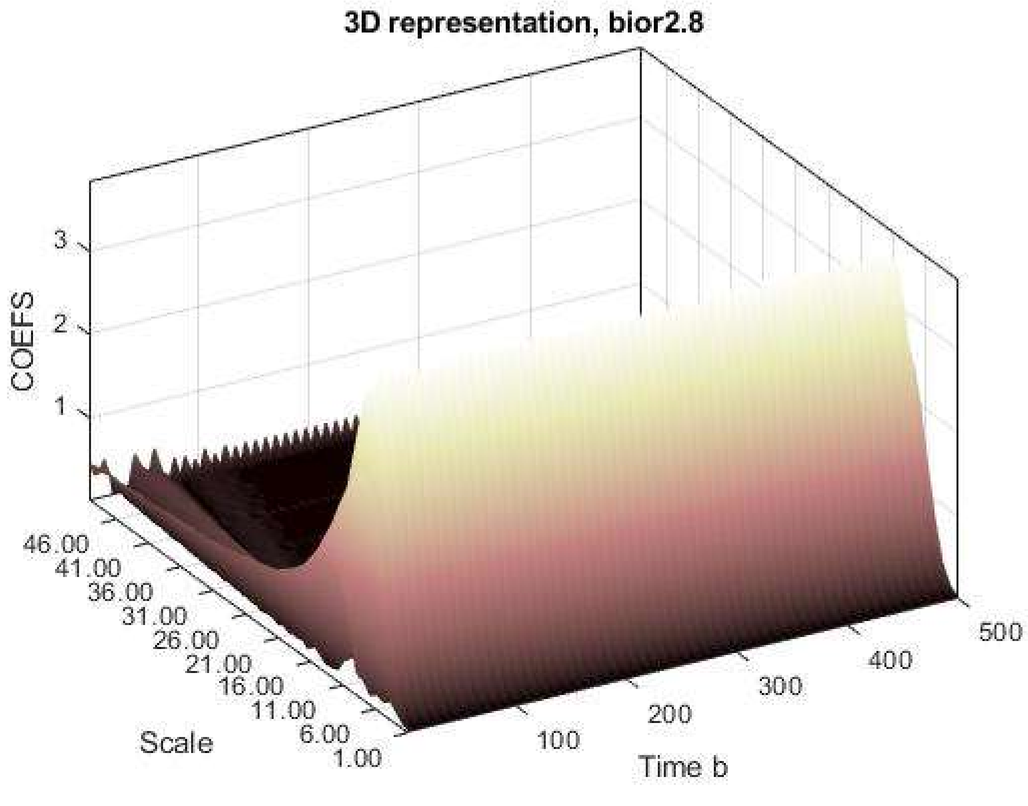


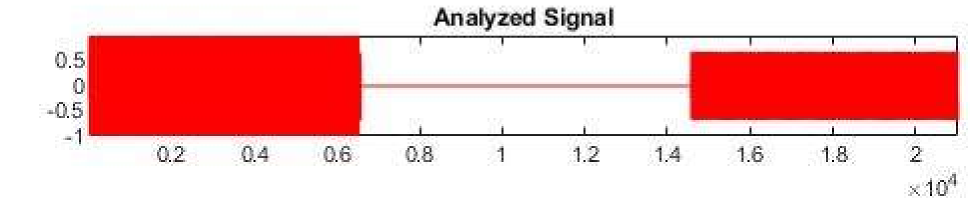


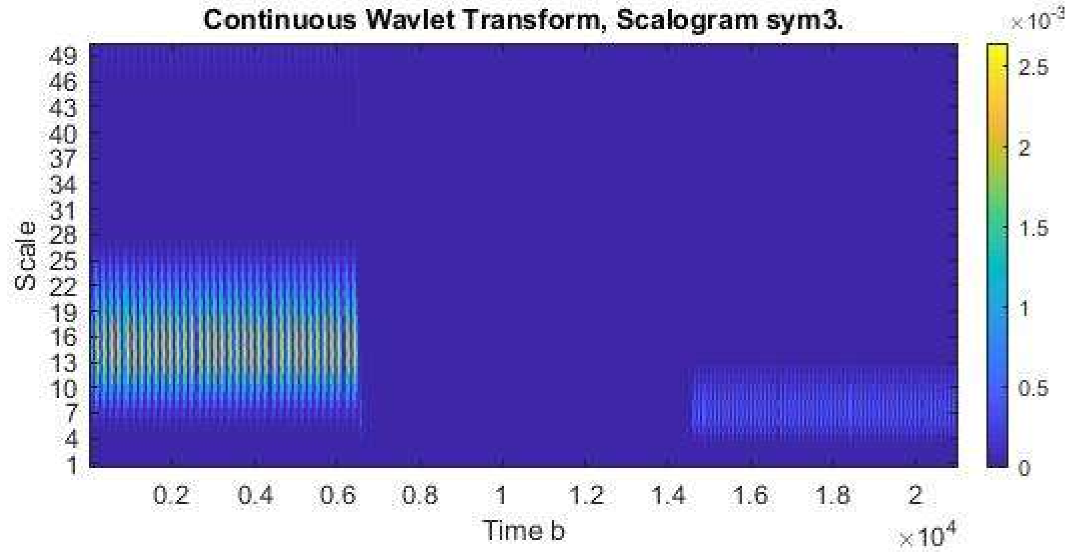


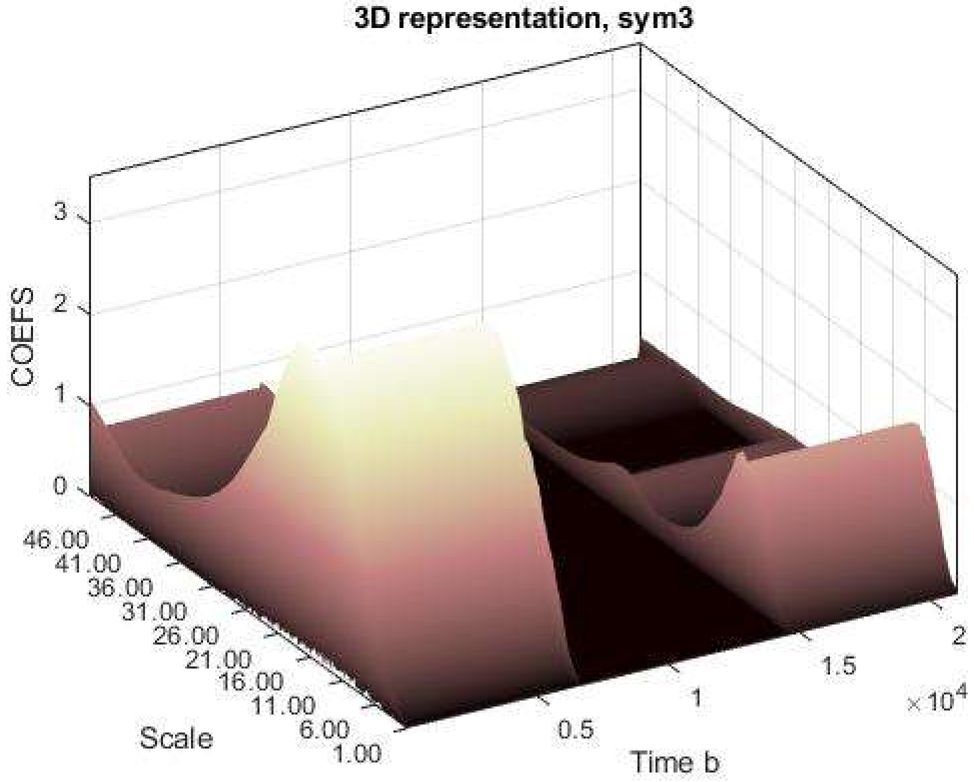

Analyzed Signal
0.5
0
-0.5
-1
50
100
150
200
250
300
350
400
450
500

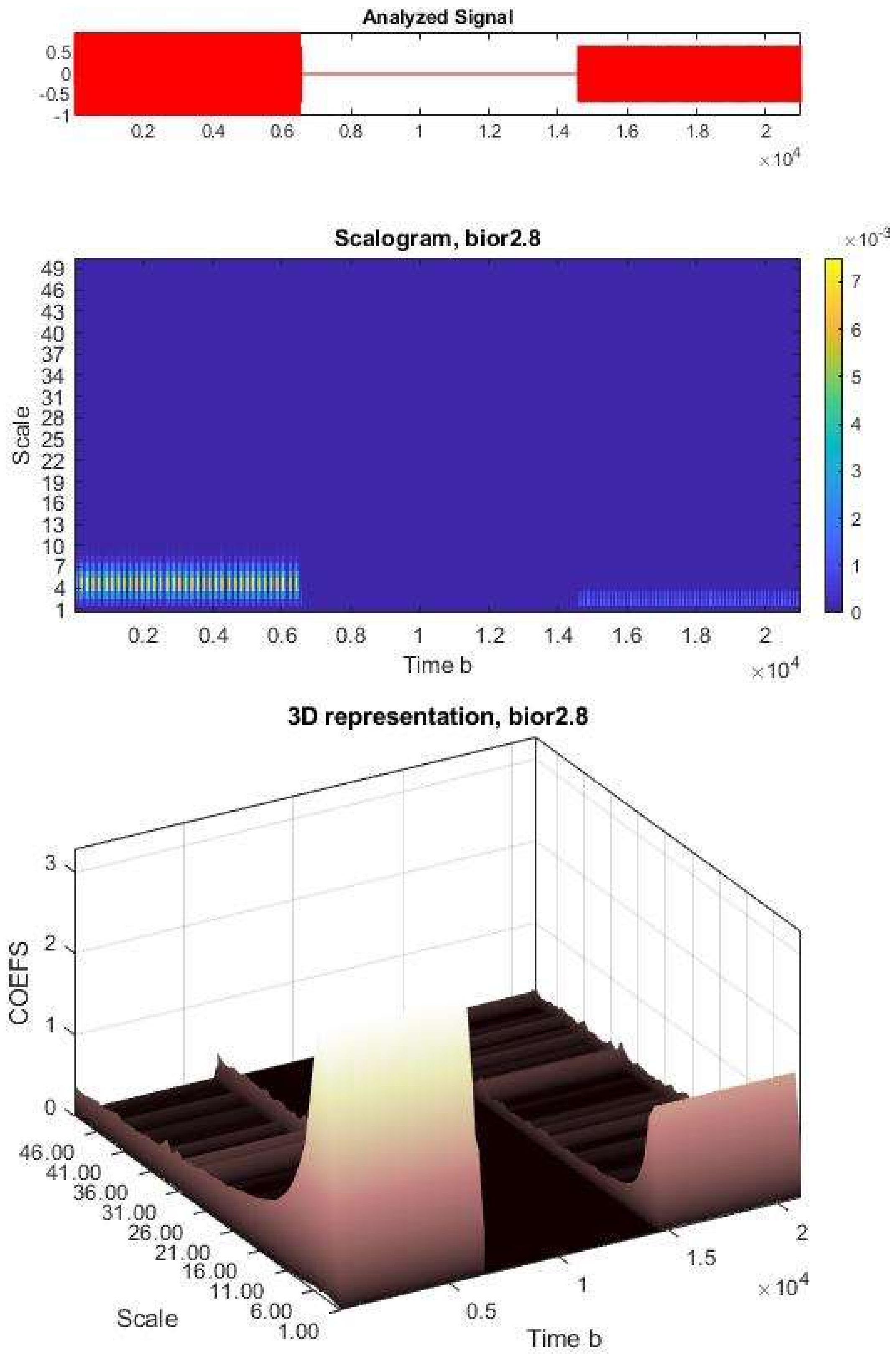

Scalogram, bior2.8
Scale
49
46
43
40
37
34
31
28
25
22
19
16
13
10
7
4
1
Time b
0.07
0.06
0.05
0.04
0.03
0.02
0.01


3D representation, bior2.8
COEFS
3
2
1
Scale
46.00
41.00
36.00
31.00
26.00
21.00
16.00
11.00
6.00
1.00
Time b
100
200
300
400
500


Analyzed Signal
0.5
0
-0.5
-1
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
$\times10^4$


Continuous Wavlet Transform, Scalogram sym3.
Scale
49
46
43
40
37
34
31
28
25
22
19
16
13
10
7
4
1
Time b
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
$\times10^4$
$\times10^{-3}$
2.5
2
1.5
1
0.5
0


3D representation, sym3
COEFS
3
2
1
0
Scale
46.00
41.00
36.00
31.00
26.00
21.00
16.00
11.00
6.00
1.00
Time b
0.5
1
1.5
2
$\times10^4$

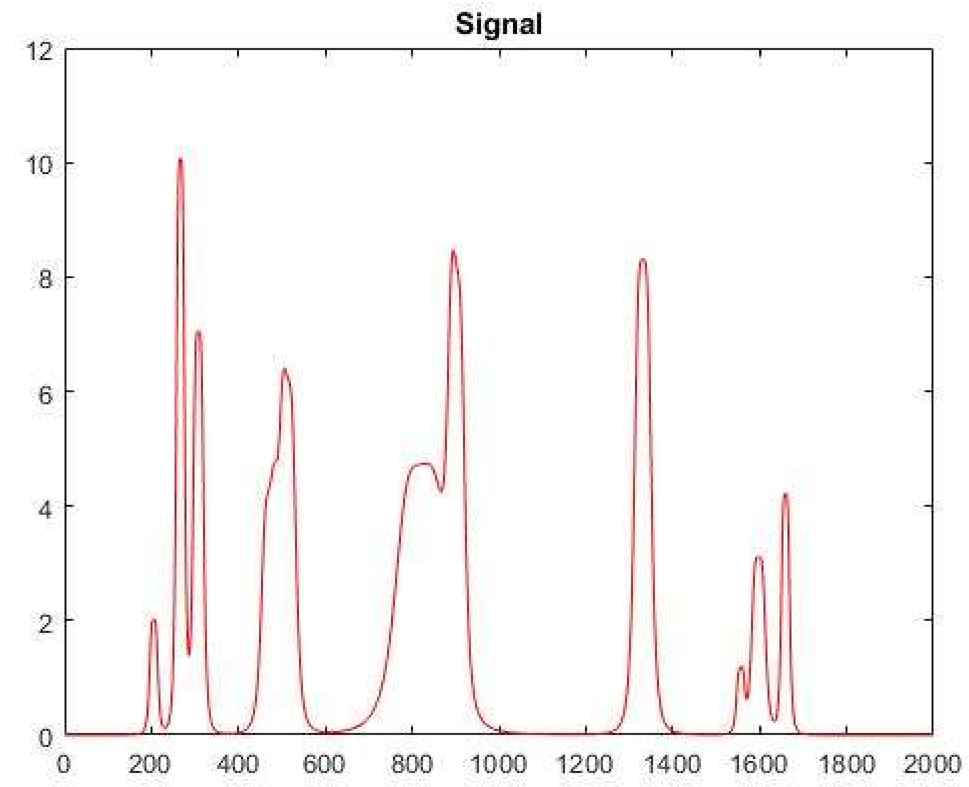


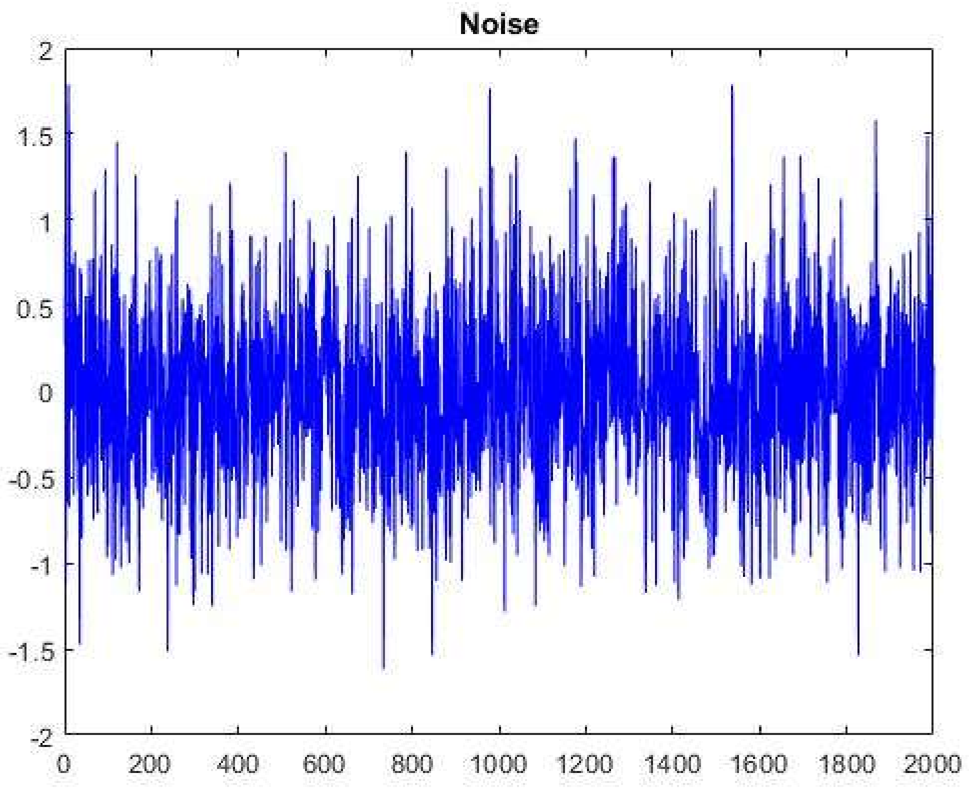
Figures 48.-55. Two different signals plotted with wavelets and for each signal two methods were used

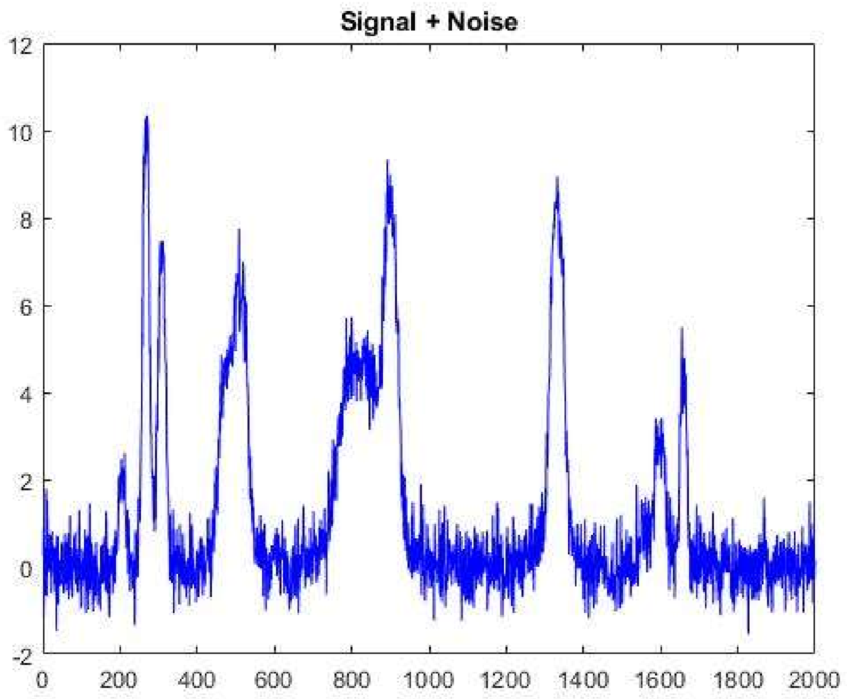
We can conclude that wavelets are good way to plot out signal when we also want to study its time dependencies.

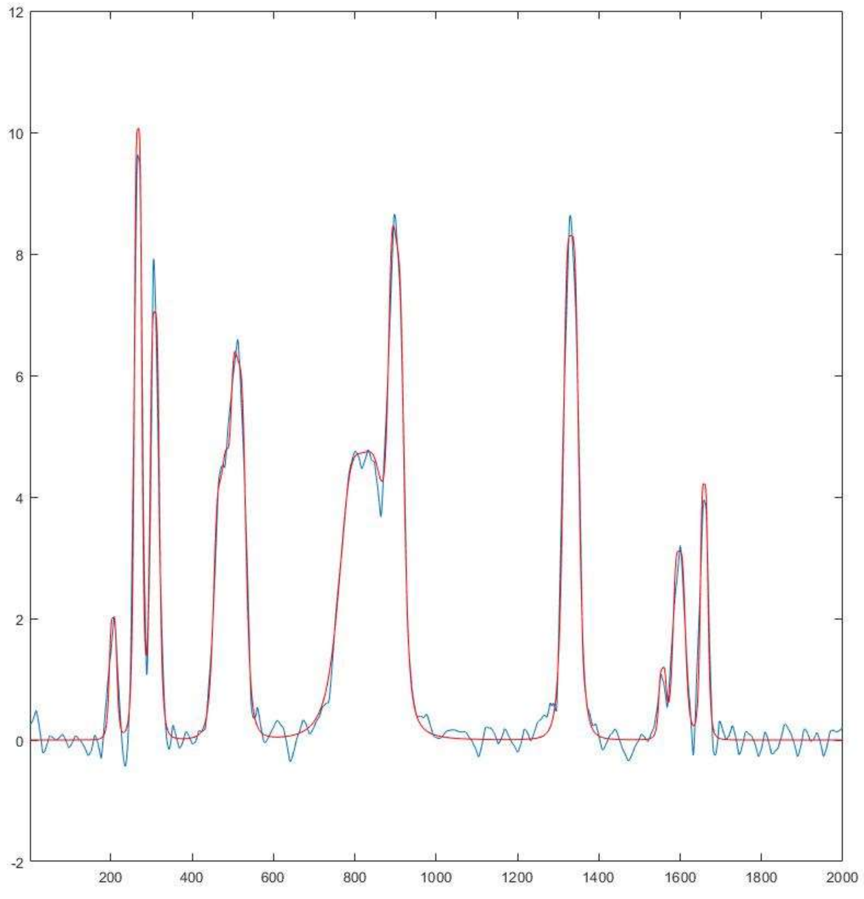
Lecture 13

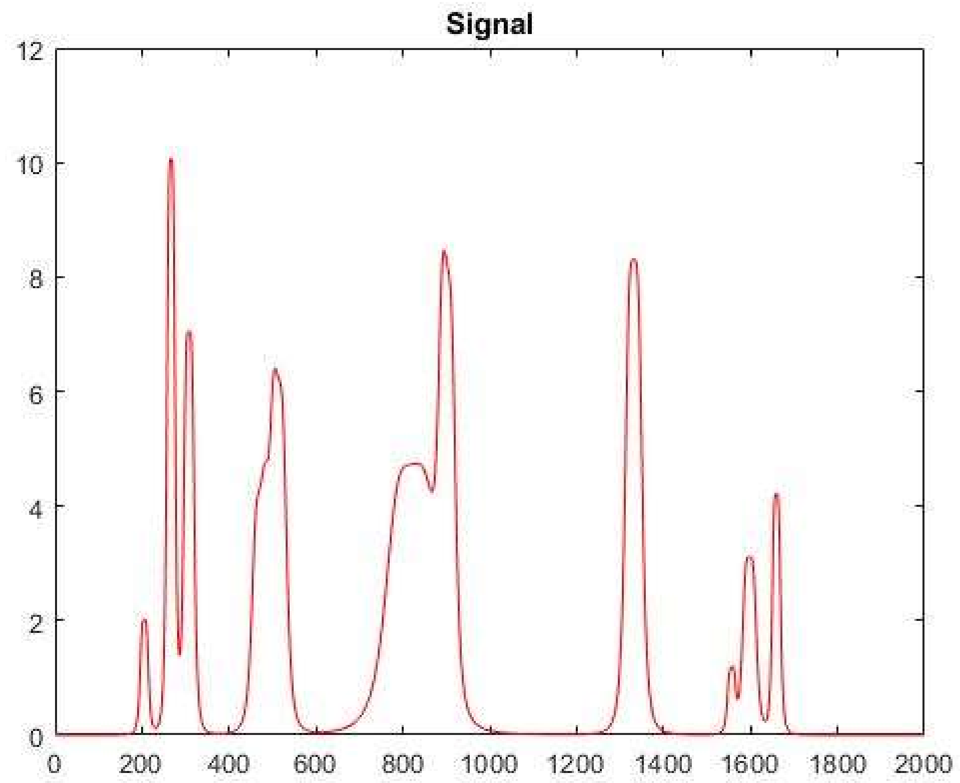
In this lecture we studied *DWT* and its application in denoising signal. Our task was to Increase the noise magnitude so that the DWT reconstruction became unrecognisable this was achieved with noise magnitude of 10.5. Results can be seen below.

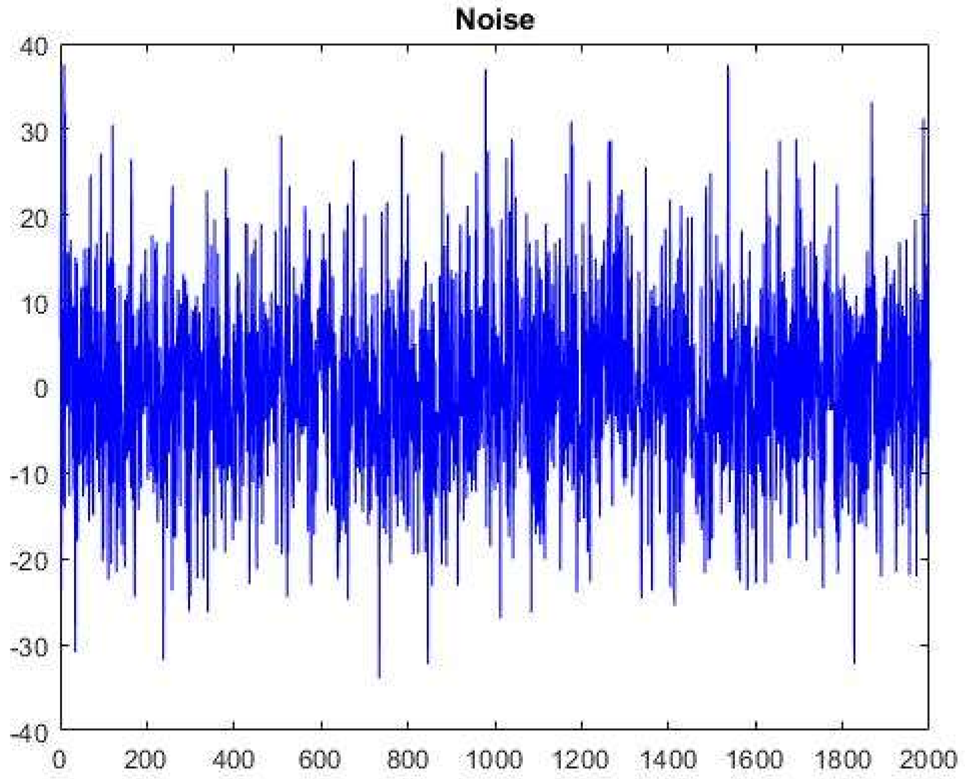


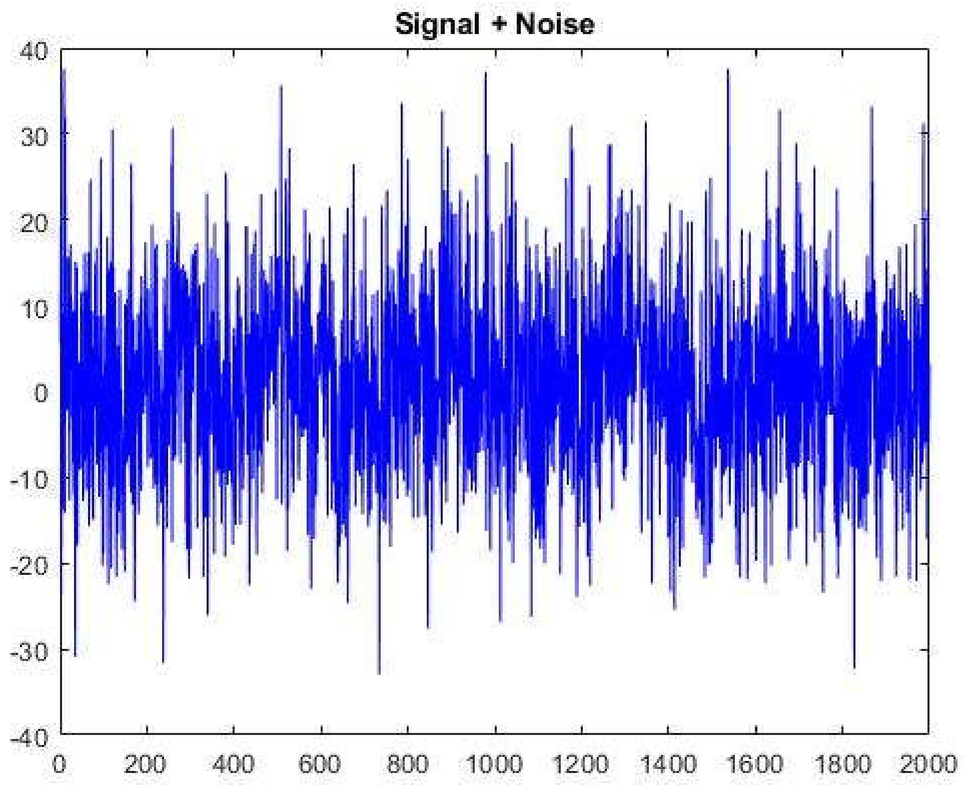


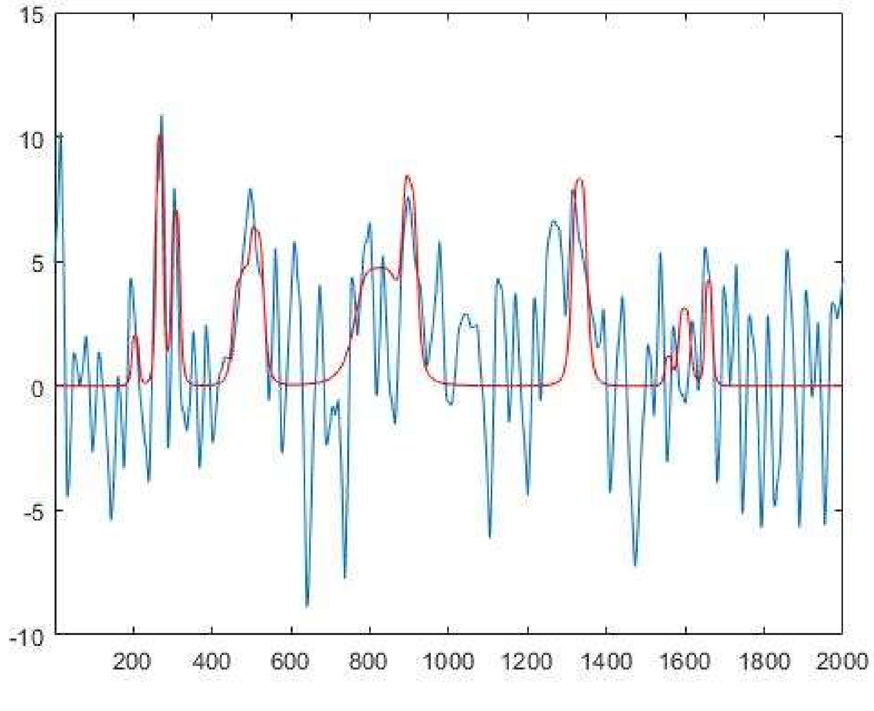

Signal + Noise
12
10
8
6
4
2
0
-2
0
200
400
600
800
1000
1200
1400
1600
1800
2000


12
10
8
6
4
2
0
-2
200
400
600
800
1000
1200
1400
1600
1800
2000


Signal
0
2
4
6
8
10
12
0
200
400
600
800
1000
1200
1400
1600
1800
2000


Noise
40
30
20
10
0
-10
-20
-30
-40
0
200
400
600
800
1000
1200
1400
1600
1800
2000



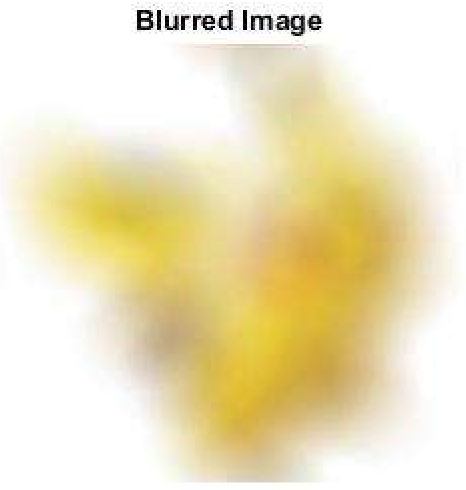


Figures 56.-63. Effect of noise magnitude

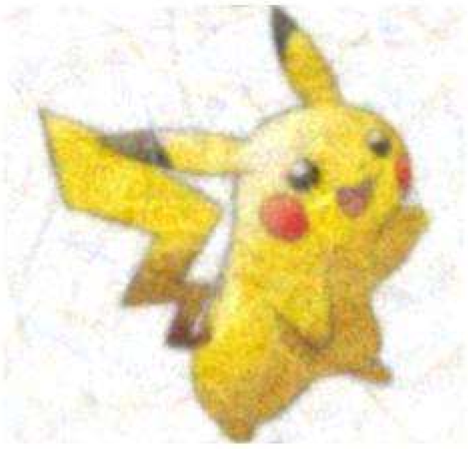
From these graphs we can see that the sym4 algorithm used here stops working with strong noise. But still noise requires a lot of amplification to affect on the reconstruction and still we can conclude that the pikes on the original signal are still reconstructed nicely while the areas between is where the reconstruction fails the most.

Lecture 14

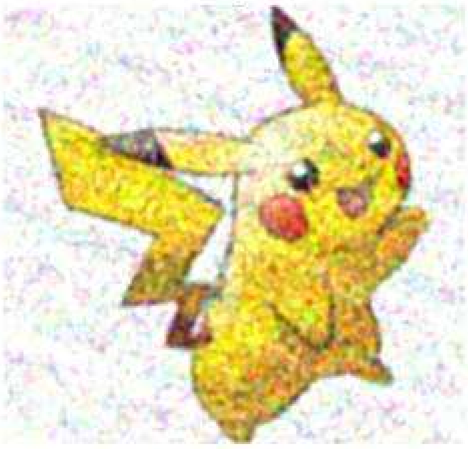
In this lesson we studied the image processing tool box and basic operation of neural network. We used any picture and we read it with function *imread* and added some noise to the picture. After adding the noise, we utilized different algorithms in order to remove the noise. We utilized algorithms like *Wiener, Regul, Blind* and *Lucy* results can be seen below.

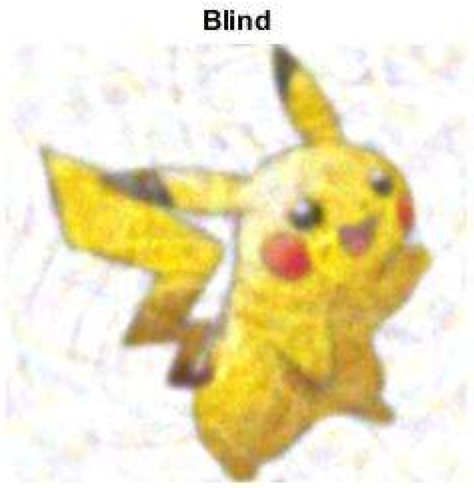


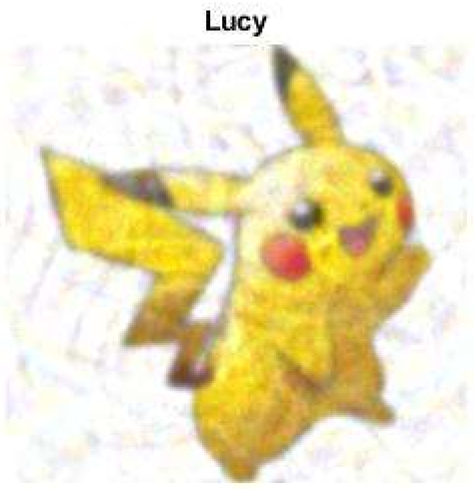
Wiener



Regul

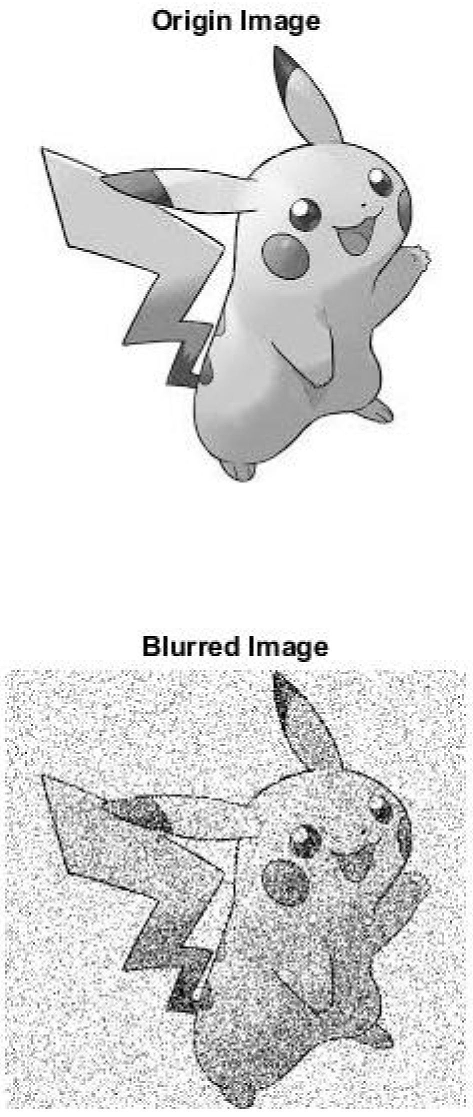


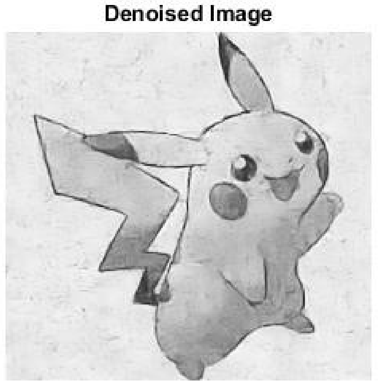




Pictures 1.-6. Original image and noised and denoised variants of the original

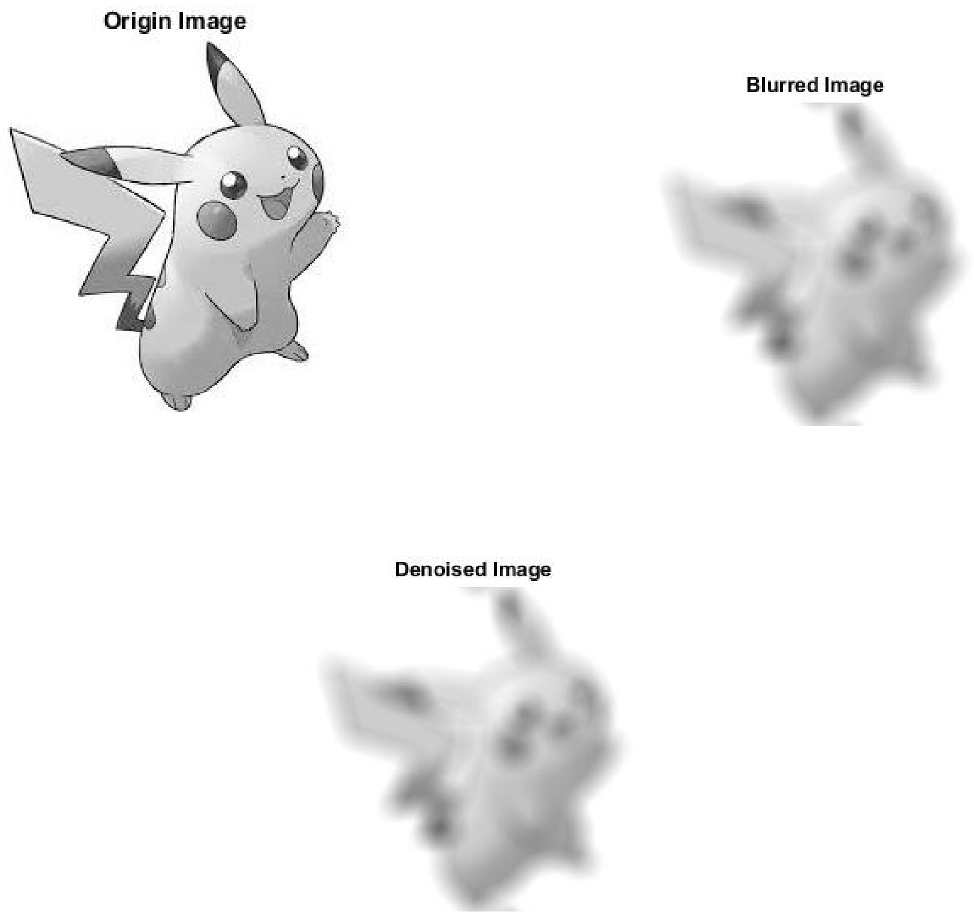
From these pictures we see that different denoising algorithms try different approaches for the blurred image. Overall clearest result was achieved with regul while it also included some colour defects. On average all the methods used were successful in denoising the image. For second task we used the *Neural Network Toolbox* and experimented how it denoises the blurred image. Result can be seen below.





Pictures 7.-9. Image denoised by using neural network (pretrained)

As we can see the result achieved by neural network is good some noise can be seen in the last picture but not so much that it would affect the visibility of the picture. Lastly, we utilized neural network for denoising blurred image. Results can be seen below.



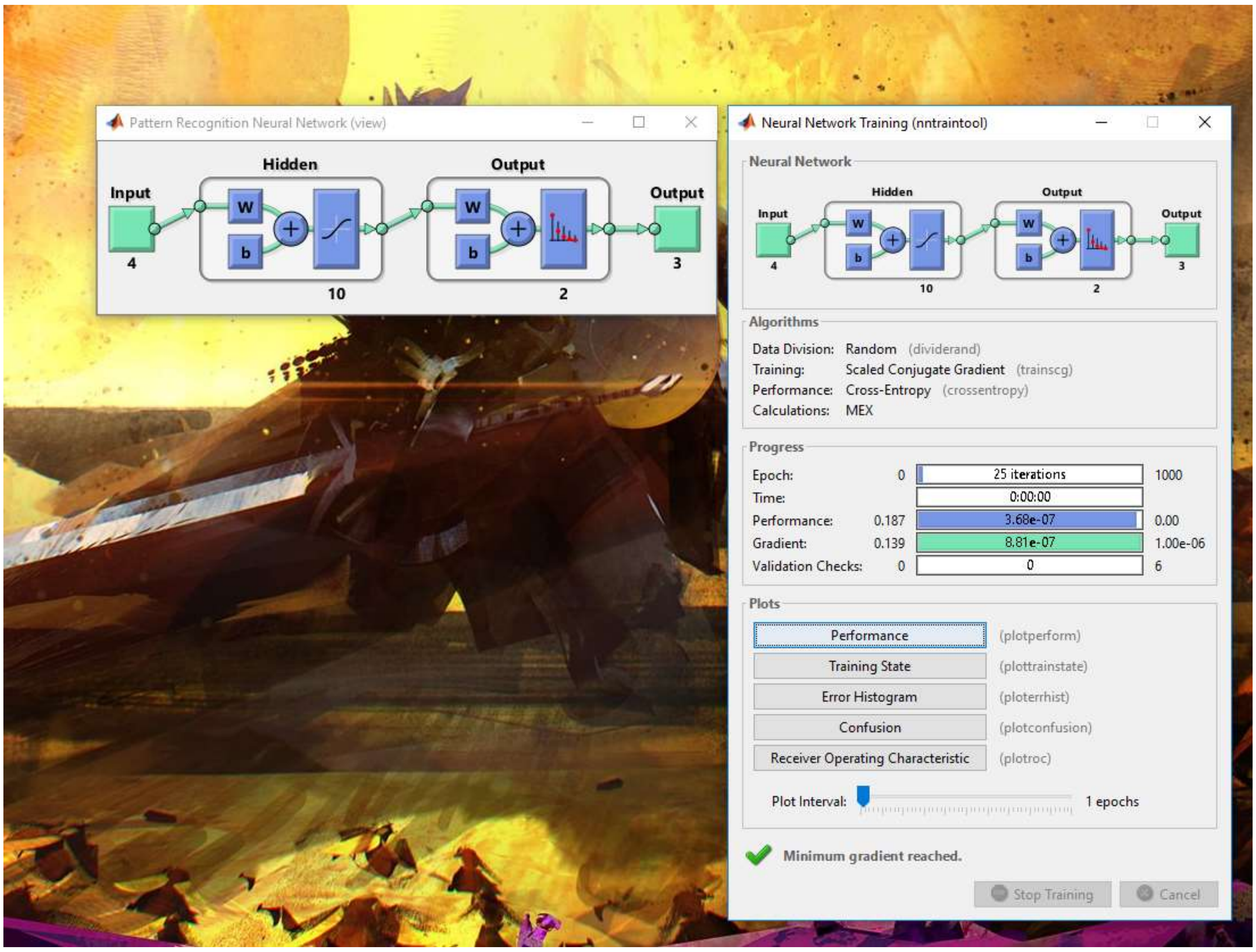
Pictures 10.-12. Amtempt on restoring blurred image

As we can clearly see the neural network didn't have almost any effect on the picture and from that we can conclude that this type of restoring is not suitable for blurred images and we should use methods used before.

Lecture 15.

On this lecture we studied neural networks. We tried to see with what dataset we get different result from the pattern recognition algorithm. With a lot of cutting of he data we get very low success rate for the algorithm.

```
x(:,1:50)=[];
x(:,11:40)=[];
x(:,21:60)=[];
t(:,1:50)=[];
t(:,11:40)=[];
t(:,21:60)=[];
```



Data Division: Random (dividerand)
Training: Scaled Conjugate Gradient (trainscg)
Performance: Cross-Entropy (crossentropy)
Calculations: MEX

Epoch: 0 25 iterations 1000
Time: 0:00:00
Performance: 0.187 3.68e-07 0.00
Gradient: 0.139 8.81e-07 1.00e-06
Validation Checks: 0 0 6

(plotperform) (plottrainstate) (ploterrhist) (plotconfusion) (plotroc)

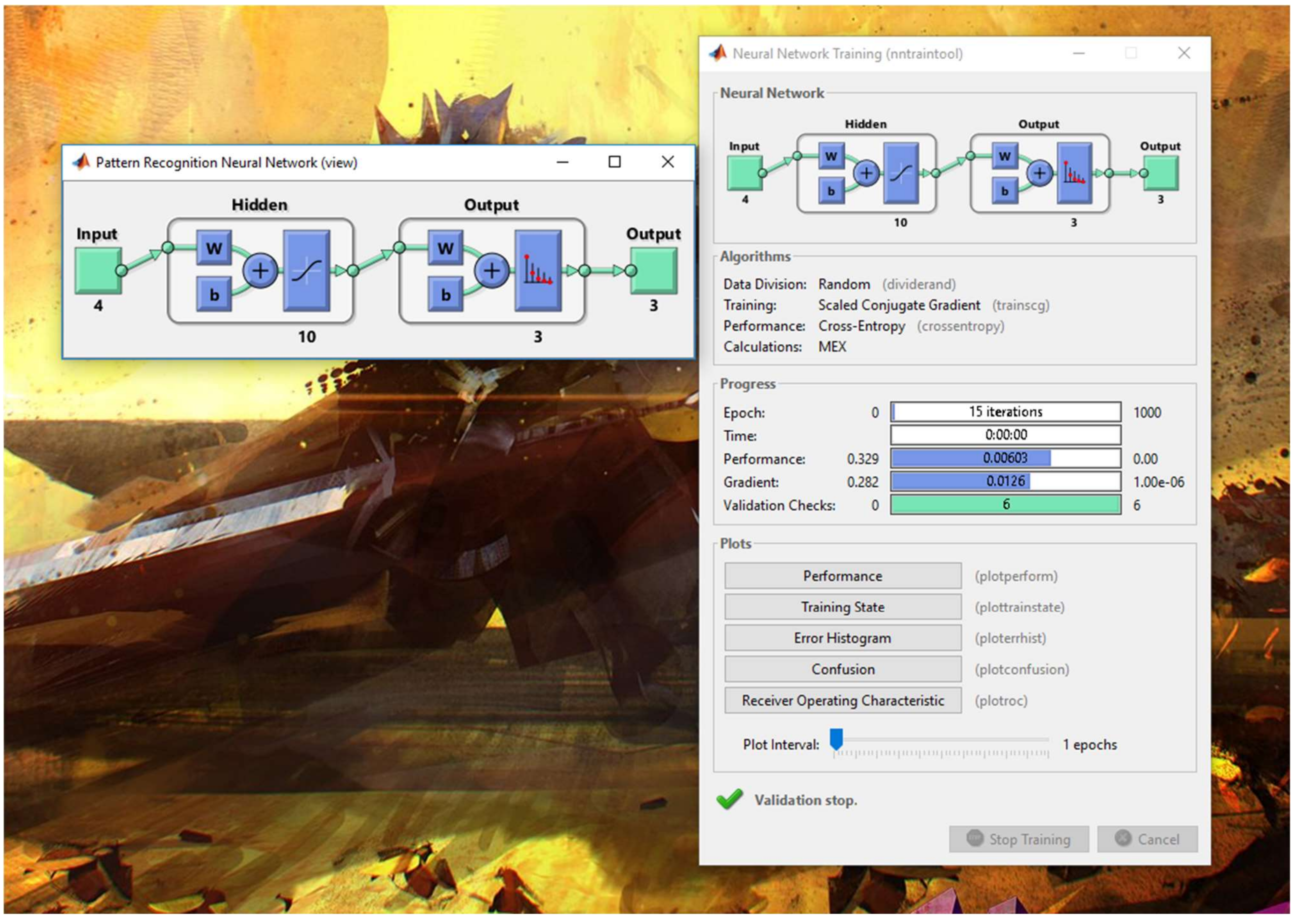
Plot Interval: 1 epochs

Minimum gradient reached.

Picture 13. Stats for large data cuts

With these cuts I got either 0 or 1 successful validation checks (or if you run it enough times you can get full six validation checks). With a lot less, cuts we cut 100% success rate for the neural network. Used cuts can be seen below.

```
x(:,1:8)=[];
x(:,11:17)=[];
x(:,21:32)=[];
t(:,1:8)=[];
t(:,11:17)=[];
t(:,21:32)=[];
```



Neural Network Training (nntraintool)

Neural Network

Algorithms

Data Division: Random (dividerand)
Training: Scaled Conjugate Gradient (trainscg)
Performance: Cross-Entropy (crossentropy)
Calculations: MEX

Progress

Epoch:	0	15 iterations	1000
Time:		0:00:00	
Performance:	0.329	0.00603	0.00
Gradient:	0.282	0.0126	1.00e-06
Validation Checks:	0	6	6

Plots

Performance (plotperform)
Training State (plottrainstate)
Error Histogram (ploterrhist)
Confusion (plotconfusion)
Receiver Operating Characteristic (plotroc)

Plot Interval: 1 epochs

Validation stop.

We achieve varying results when we vary the cuts within previously mentioned lines. Neural networking can be utilized in many applications. It’s a good way to recognise items from data which you would need to do by hand otherwise. Usefulness of neural network is basically only limited by the computing power of the computer.

References

Math Works, basic tutorials for MATLAB

Khamukhin, Aleksandr Anatolievich. Lecture notes for course Introduction to Matlab.