REPORT

Zajęcia: Windowing

Teacher: prof. dr hab. Vasyl Martsenyuk

Lab 2

Date: 01.03.2024

Topic: " Windowing"

Variant: 13

Agnieszka Białecka

Informatyka II stopień,

stacjonarne,

2 semestr,

Gr.1a

1. Problem statement

The objective is to be able the results of different type of windowing the signals.

2. Theoretical introduction

Windowing signals is a technique used in signal processing to limit the duration of a signal or to emphasize specific parts of it. It involves multiplying a signal by a window function, which is typically a mathematical function that has a value of 1 within a certain range (window) and decreases towards zero outside of this range.

The purpose of windowing signals is to reduce spectral leakage, which occurs when the frequency content of a signal spreads out into neighboring frequency bins due to the finite length of the signal. By applying a window function, the signal is tapered at the edges, reducing the contribution of the signal outside of the window and minimizing spectral leakage.

Common window functions include:

- the rectangular window - $w[k] \ = \ rect_N[k]$ takes all samples with equal weight into account
- Triangular window - have window length *2N − 1*, the triangular window can be expressed as the convolution of two rectangular windows $w[k] = rect_N[k] * rect_N[k]$
- Hann window - $w[k] = \frac{1}{2}(1 - cos(2\pi\frac{k}{N}))$ is a smooth window whose first and last value is zero. It features a fast decay of the side lobes
- Hamming window - $w[k] = 0.54 - 0.46cos(2\pi\frac{k}{N})$ is a smooth window function whose first and last value is not zero
- Blackman window - $w\ [k] = 0.42 - 0.5cos(2\pi\ \frac{k}{N}) + 0.08cos(4\pi\ \frac{k}{N})$ features a rapid decay of side lobes at the cost of a wide main lobe and low frequency selectivity

Each window function has its own characteristics and trade-offs in terms of main lobe width, side lobe level, and frequency resolution, and the choice of window function depends on the specific requirements of the signal processing application.

3. Input data (Variant)

This report was created with base of variant 13:

f_1	*500*
f_2	*500.25*
f_3	*499.75*

$\lvert x[k]\rvert_{max}$	*4*
f_s	*800*
N	*1800*

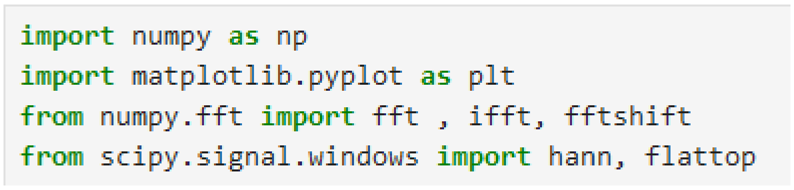
GitHub repository:

https://github.com/Delisolara/AaDEC

4. Course of actions

4.1. Importing Libraries

The first step was to upload the libraries that were used during the implementation of the code.

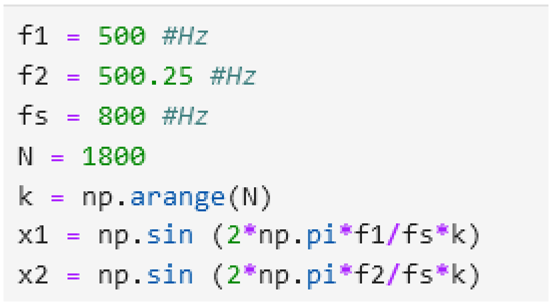


```python
import numpy as np
import matplotlib.pyplot as plt
from numpy.fft import fft , ifft, fftshift
from scipy.signal.windows import hann, flattop
```

Picture 1. Uploaded libraries

4.2. Generating Signals

The next step was to generate two signals of $f_1 = 500Hz$, $f_2 = 500.25Hz$ and amplitude $\lvert x[k]\rvert_{max} = 1$ for the sampling frequency $f_s = 800Hz$ in the range of $0 \leq k < N = 1800$.

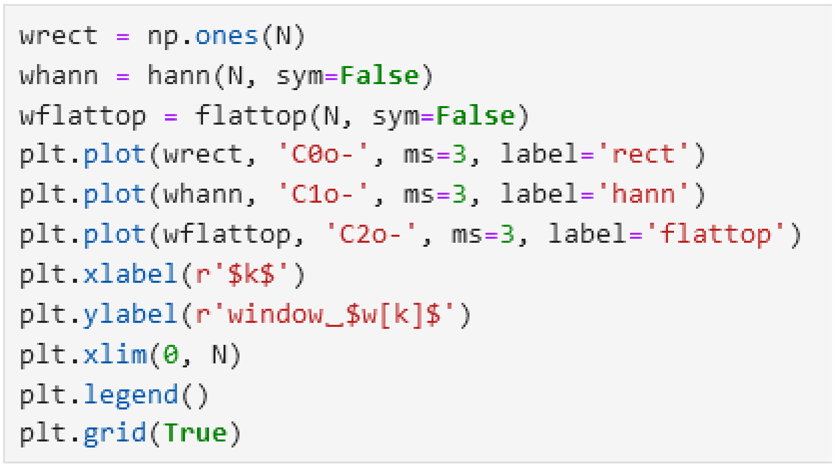


```python
f1 = 500 #Hz
f2 = 500.25 #Hz
fs = 800 #Hz
N = 1800
k = np.arange(N)
x1 = np.sin (2*np.pi*f1/fs*k)
x2 = np.sin (2*np.pi*f2/fs*k)
```

Picture 2. Implemented code

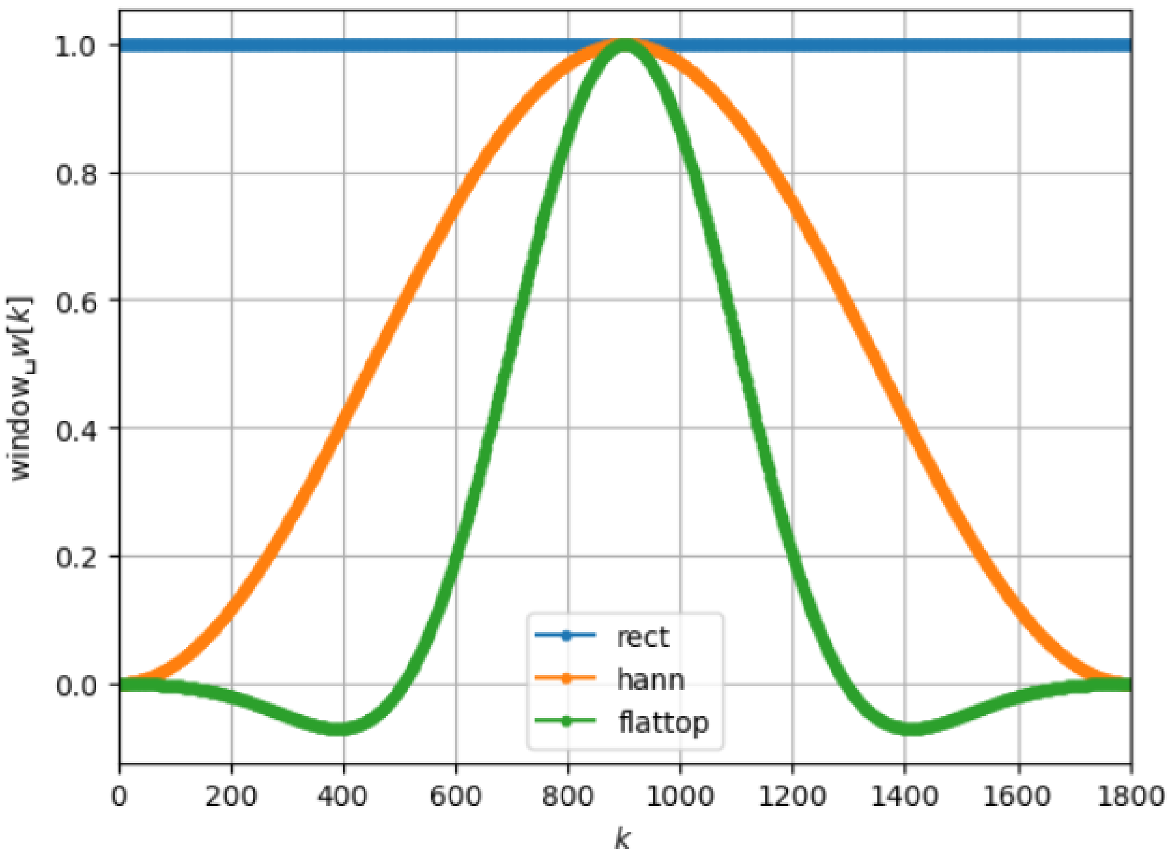
4.3. Generating Windows

Another step was the implementation of rectangular window, a Hann window and a flat top window with the same lengths as the sine signals.



```
wrect = np.ones(N)
whann = hann(N, sym=False)
wflattop = flattop(N, sym=False)
plt.plot(wrect, 'C0o-', ms=3, label='rect')
plt.plot(whann, 'C1o-', ms=3, label='hann')
plt.plot(wflattop, 'C2o-', ms=3, label='flattop')
plt.xlabel(r'$k$')
plt.ylabel(r'window_$w[k]$')
plt.xlim(0, N)
plt.legend()
plt.grid(True)
```

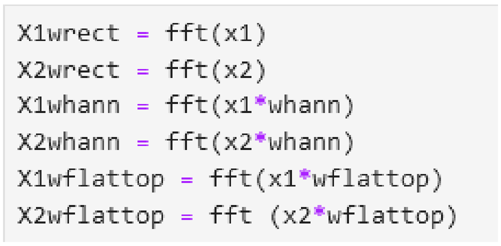
Picture 3. Implemented code



Picture 4. The result

4.4. DFT spectra using FFT algorithm

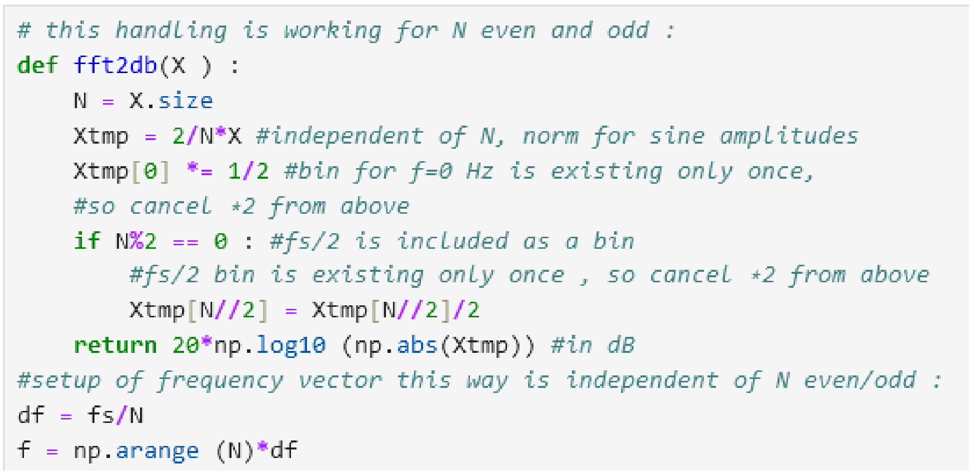
The next step was to window both sine signals *x1* and *x2* with the three windows and calculate the corresponding DFT spectra using FFT algorithm either from 'numpy.fft' or 'scipy.fft' package.



```
X1wrect = fft(x1)
X2wrect = fft(x2)
X1whann = fft(x1*whann)
X2whann = fft(x2*whann)
X1wflattop = fft(x1*wflattop)
X2wflattop = fft (x2*wflattop)
```

Picture 5. Implemented code

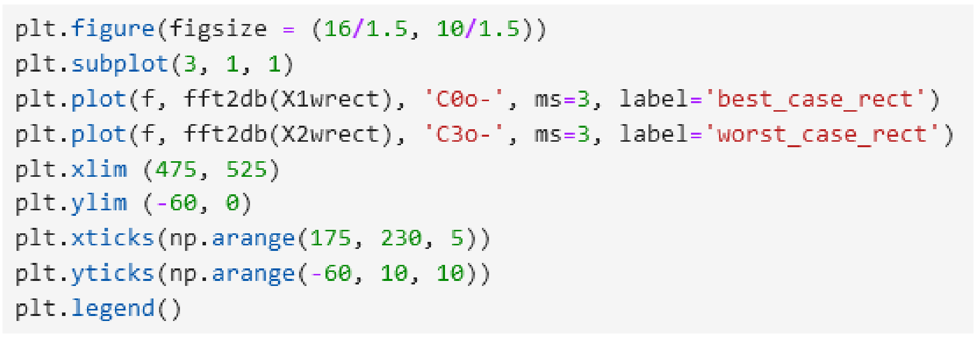
Since we were dealing with the analysis of sine signals, a convenient normalization was applied. This was achieved by making the result independent of the chosen DFT length N. Furthermore, considering both negative and positive frequency bins, multiplying by 2 yields normalization to sine signal amplitudes. Since the frequency bin for 0Hz and (if N is even) for fs/2 exists only once, multiplication by 2 is not required for these bins.



```
# this handling is working for N even and odd :
def fft2db(X ) :
    N = X.size
    Xtmp = 2/N*X #independent of N, norm for sine amplitudes
    Xtmp[0] *= 1/2 #bin for f=0 Hz is existing only once,
    #so cancel *2 from above
    if N%2 == 0 : #fs/2 is included as a bin
        #fs/2 bin is existing only once , so cancel *2 from above
        Xtmp[N//2] = Xtmp[N//2]/2
    return 20*np.log10 (np.abs(Xtmp)) #in dB
#setup of frequency vector this way is independent of N even/odd :
df = fs/N
f = np.arange (N)*df
```

Picture 6. Implemented code

The proposed handling is independent of *N* odd/even and returns the whole DFT spectrum. Since normalization was applied for physical sine frequencies, only the part from 0 Hz to fs/2 is valid.



```
plt.figure(figsize = (16/1.5, 10/1.5))
plt.subplot(3, 1, 1)
plt.plot(f, fft2db(X1wrect), 'C0o-', ms=3, label='best_case_rect')
plt.plot(f, fft2db(X2wrect), 'C3o-', ms=3, label='worst_case_rect')
plt.xlim (475, 525)
plt.ylim (-60, 0)
plt.xticks(np.arange(175, 230, 5))
plt.yticks(np.arange(-60, 10, 10))
plt.legend()
```

Picture 7. Implemented code

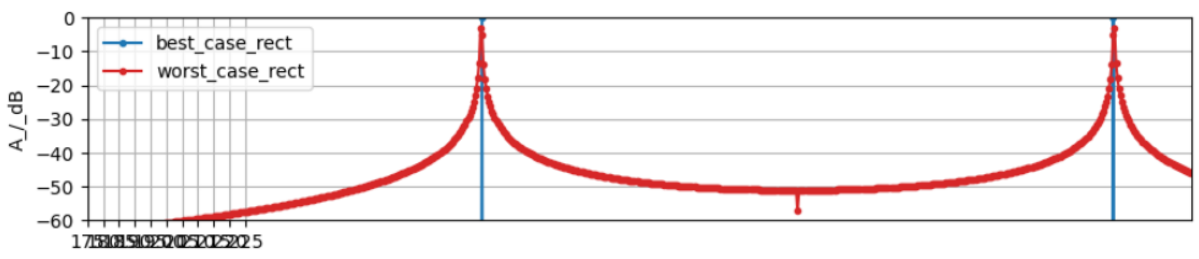
```
#plt.xlabel('f/Hz')
plt.ylabel('A_/_dB')
plt.grid(True)
plt.subplot(3,1,2)
plt.plot(f, fft2db(X1whann), 'C0o-', ms=3,label='best_casehann')
plt.plot(f, fft2db(X2whann), 'C3o-', ms=3, label='worst_case_hann')
plt.xlim(475, 525)
plt.ylim(-60, 0)
plt.xticks(np.arange(175, 230, 5))
plt.yticks(np.arange(-60, 10, 10))
plt.legend()
```

Picture 8. Implemented code – continuation

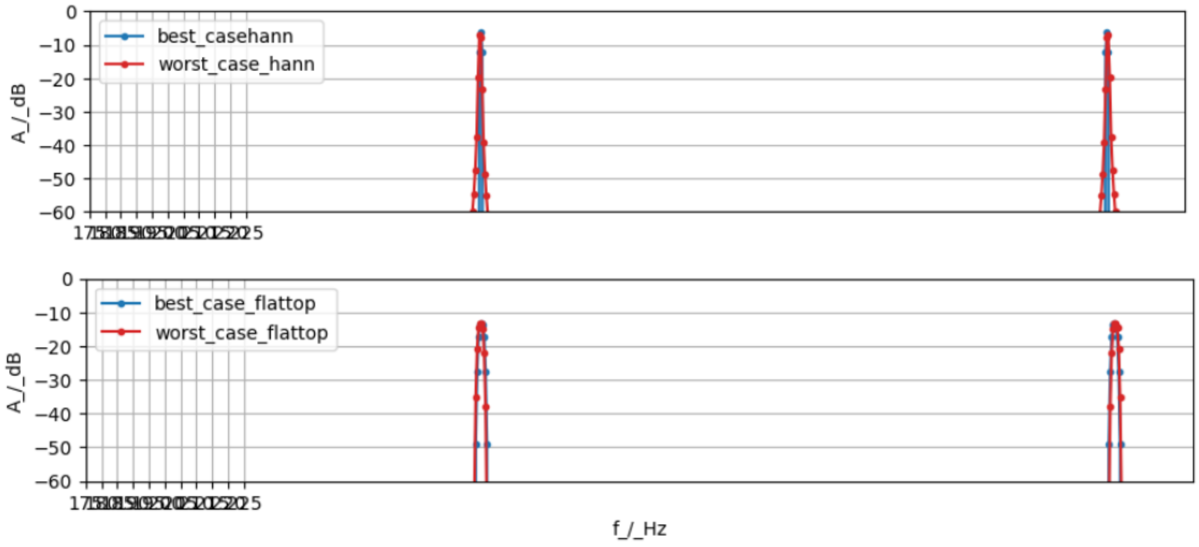
```
#plt.xlabel('f/Hz')
plt.ylabel('A_/_dB')
plt.grid(True)
plt.subplot(3, 1, 3)
plt.plot(f, fft2db(X1wflattop), 'C0o-', ms=3, label='best_case_flattop')
plt.plot(f, fft2db(X2wflattop), 'C3o-', ms=3, label='worst_case_flattop')
plt.xlim(475, 525)
plt.ylim(-60, 0)
plt.xticks(np.arange(175, 230, 5))
plt.yticks(np.arange(-60, 10, 10))
plt.legend()
plt.xlabel('f_/_Hz')
plt.ylabel('A_/_dB')
plt.grid(True)
```

Picture 9. Implemented code – continuation

Next, a level plot was created for the window DTFT spectra, normalized to their main lobe maximum for -π ≤ Ω ≤ π and ranging from -120dB to 0dB. To achieve a sufficiently high resolution of the spectra, zero padding or interpolation formulas towards DTFT had to be used.

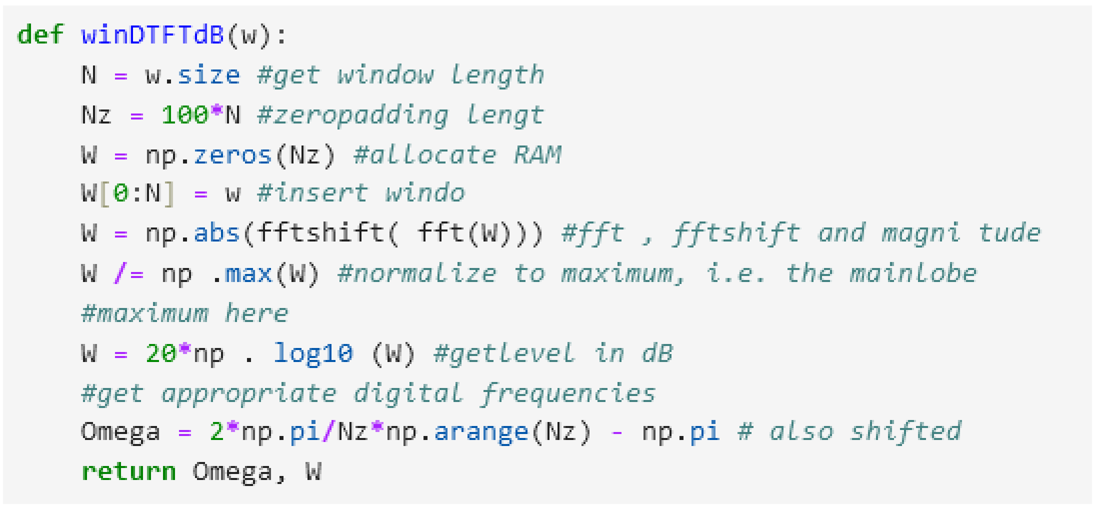


Picture 10. The result



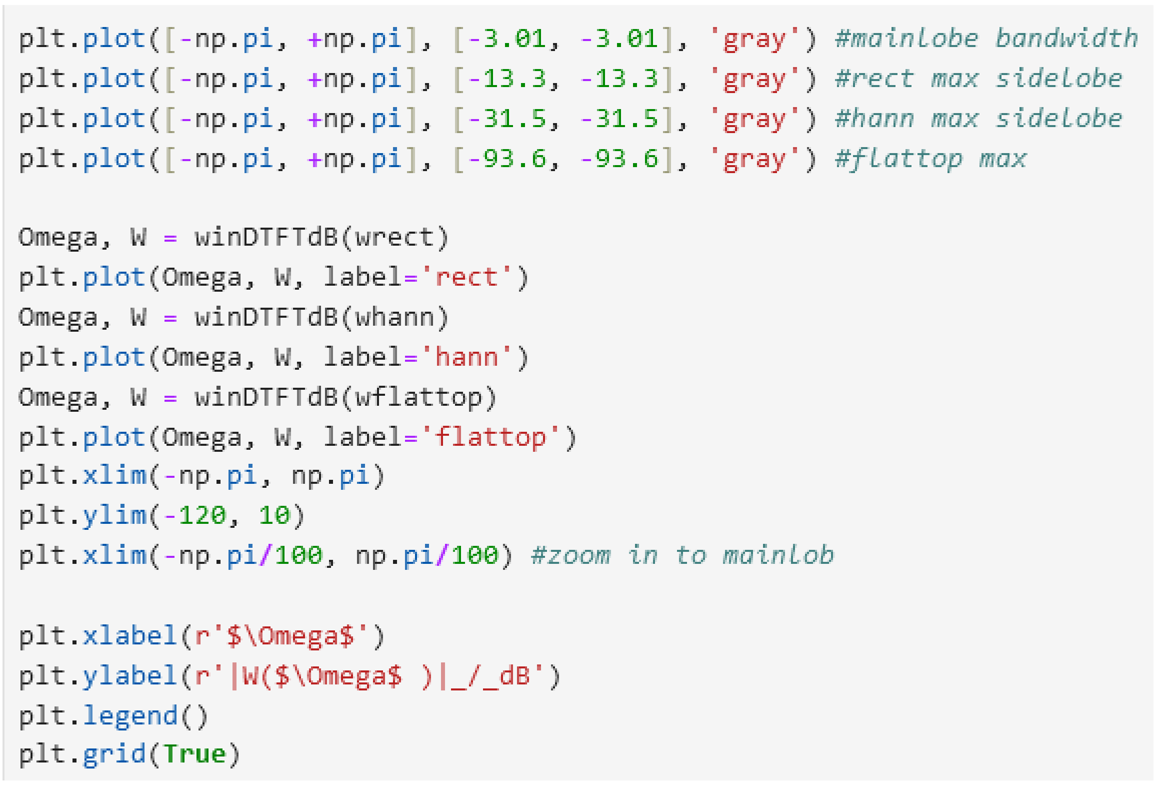
Picture 11. The result – continuation

Then a function returning quasi-DTFT was definedand estimated digital frequencies. This uses zeropadding to achieve DTFT-like frequency resolution and fftshift to bring the main lobe to the center of the numpy array.



```
def winDTFTdB(w):
    N = w.size #get window length
    Nz = 100*N #zeropadding lengt
    W = np.zeros(Nz) #allocate RAM
    W[0:N] = w #insert windo
    W = np.abs(fftshift( fft(W))) #fft , fftshift and magni tude
    W /= np .max(W) #normalize to maximum, i.e. the mainlobe
    #maximum here
    W = 20*np . log10 (W) #getlevel in dB
    #get appropriate digital frequencies
    Omega = 2*np.pi/Nz*np.arange(Nz) - np.pi # also shifted
    return Omega, W
```

Picture 12. Implemented code

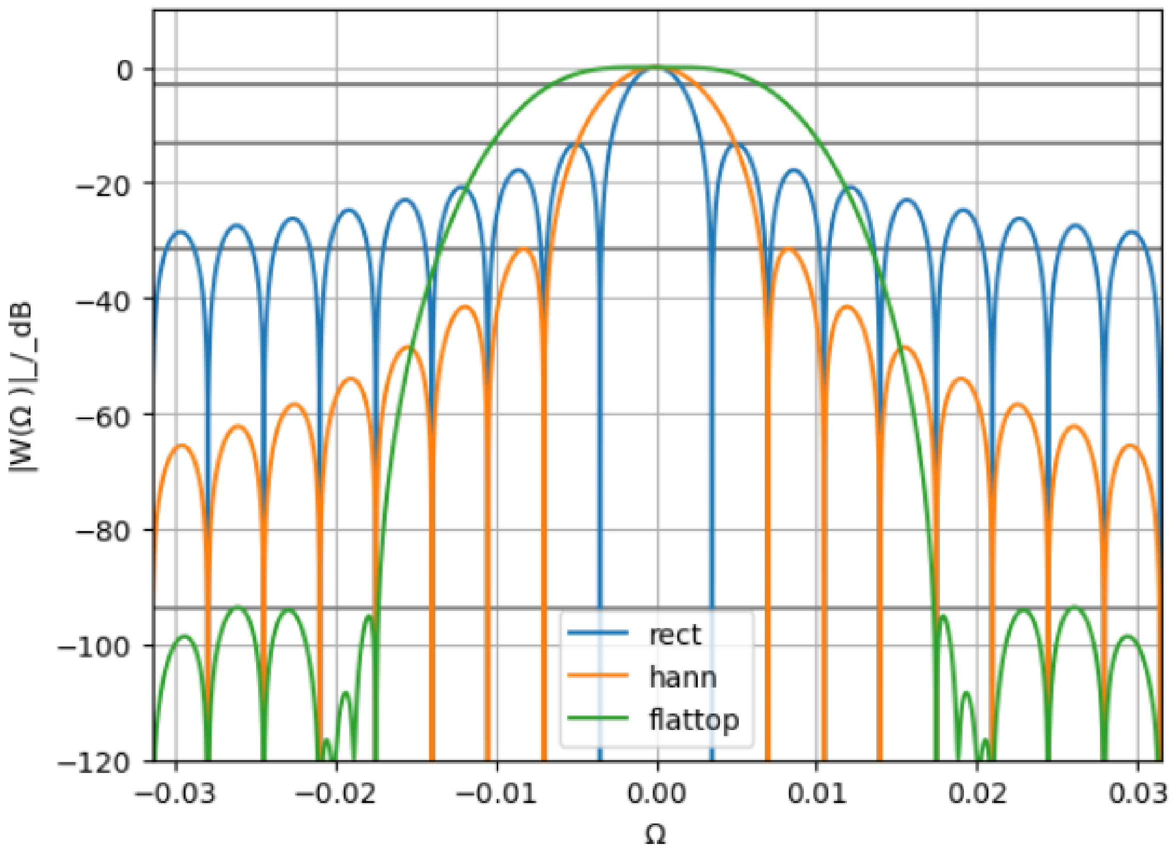


```
plt.plot([-np.pi, +np.pi], [-3.01, -3.01], 'gray') #mainlobe bandwidth
plt.plot([-np.pi, +np.pi], [-13.3, -13.3], 'gray') #rect max sidelobe
plt.plot([-np.pi, +np.pi], [-31.5, -31.5], 'gray') #hann max sidelobe
plt.plot([-np.pi, +np.pi], [-93.6, -93.6], 'gray') #flattop max

Omega, W = winDTFTdB(wrect)
plt.plot(Omega, W, label='rect')
Omega, W = winDTFTdB(whann)
plt.plot(Omega, W, label='hann')
Omega, W = winDTFTdB(wflattop)
plt.plot(Omega, W, label='flattop')
plt.xlim(-np.pi, np.pi)
plt.ylim(-120, 10)
plt.xlim(-np.pi/100, np.pi/100) #zoom in to mainlob

plt.xlabel(r'$\Omega$')
plt.ylabel(r'|W($\Omega$ )|_/_dB')
plt.legend()
plt.grid(True)
```

Picture 13. Implemented code



Picture 14. The result

5. Conclusions

Based on this lab we explored the application of various windowing techniques in signal processing, with the objective of analyzing their impact on signal spectra.

The course of actions involved generating signals and implementing different window functions. The signals were then windowed and their corresponding Discrete Fourier Transform (DFT) spectra calculated using the Fast Fourier Transform (FFT) algorithm. A normalization method was applied to ensure independence from the chosen DFT length, and the resulting spectra were visualized through level plots normalized to their main lobe maximum.