Lab 2.Windowing

Abstract

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

1. Importing Libraries

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

2. Generating Signals

Generate two sine signals of $f_1 = 200$ Hz and $f_2 = 200.25$ Hz and amplitude $|x[k]|_{\max} = 1$ for the sampling frequency $f_s = 800$ Hz in the range of $0 \leq k < N = 1600$.

```
f1 = 200  # Hz
f2 = 200.25  # Hz
fs = 800  # Hz
N = 1600
k = np.arange(N)
x1 = np.sin(2*np.pi*f1/fs*k)
x2 = np.sin(2*np.pi*f2/fs*k)
```

3. Generating Windows

Generate - a rectangular window, - a Hann window and - a flat top window with the same lengths as the sine signals. Note: we analyze signals, so we use ‘sym=False‘ (periodic window) rather than ‘sym=True‘ (symmetric window, used for FIR filter design). Plot the obtained window signals over k.

```
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)
```

4. DFT spectra using FFT algorithm

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 from ‘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)
```

4.1. "Normalized" level of DFT within the interval

Plot the **normalized** level of the DFT spectra in between 175 Hz and 225 Hz and -50 and 0 dB.

Note that we are dealing with analysis of sine signals, so a convenient **normalization** should be applied for the shown level. This can be achieved by making the result independent from the chosen DFT length N. Furthermore, considering negative and positive frequency bins, multiplying with 2 yields normalization to sine signal amplitudes. Since the frequency bin for 0

Hz and (if N is even) for $f_s/2$ exists only once, multiplication with 2 is not required for these bins.

4.1.1. Preparations for solution

It is meaningful to define a function that returns the level of DFT in term of sine signal normalization.

Furthermore, the DFT frequency vector should be set up.

```
# 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
```

The proposed handling is independent of N odd/even and returns the whole DFT spectrum. Since we normalized for physical sine frequencies, only the part from 0 Hz to fs/2 is valid. So, make sure that spectrum returned from fft2db is only plotted up to fs/2.

4.1.2. Solution

```
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(175, 225)
plt.ylim(-60, 0)
plt.xticks(np.arange(175, 230, 5))
plt.yticks(np.arange(-60, 10, 10))
plt.legend()
```

```python
#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 case hann')
plt.plot(f, fft2db(X2whann), 'C3o-', ms=3, label='worst case hann')
plt.xlim(175, 225)
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)

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(175, 225)
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)
```

4.2. Window DTFT spectra normalized to their mainlobe maximum

Plot the level of the window DTFT spectra normalized to their mainlobe maximum for $-\pi \leq \Omega \leq \pi$ and -120 dB to 0 dB. Use zero-padding or the formulas for interpolation towards the DTFT to achieve a sufficiently high resolution of the spectra. To inspect the mainlobe in detail spectra might be plotted within the range $-\pi/100 \leq \Omega \leq \pi/100$ as well.

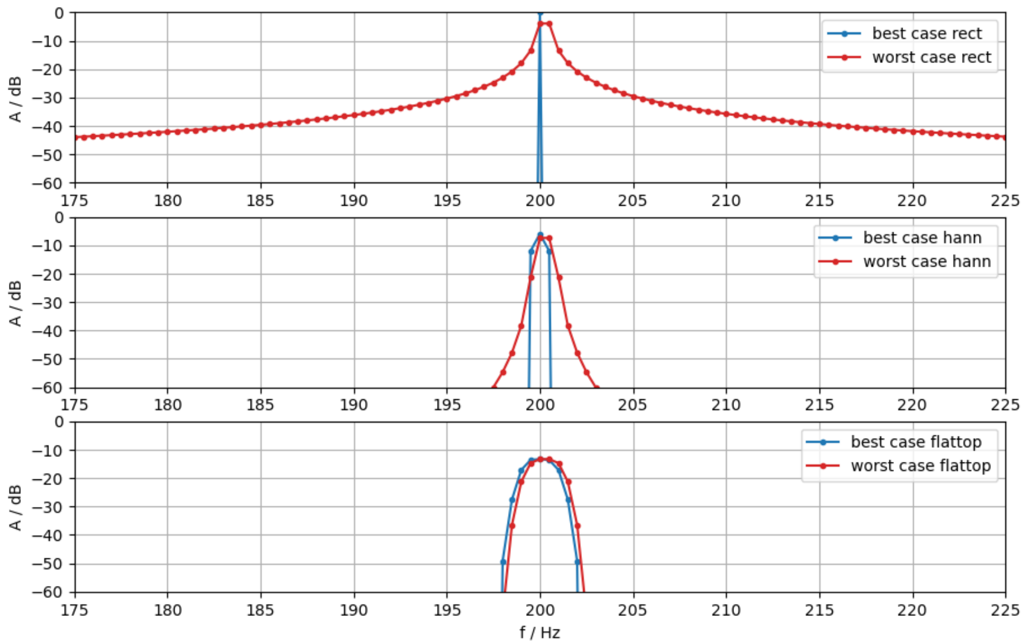

Figure 1: DFT spectra using FFT algorithm

4.2.1. Preparations for solution

It is again meaningful to define a function that returns the quasi-DTFT and the evaluated digital frequencies. Here is a proposal using zeropadding (to obtain DTFT-like frequency resolution) and fftshift (to bring mainlobe into the middle of numpy array), which then requires Ω from $-\pi$ to π.

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

```
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
                                                   #sidelobe
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 into mainlobe

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

5. Tasks

Generate three sine signals of given f_1, f_2, and f_3 and amplitude $|x[k]|_{\max}$ for the sampling frequency f_s in the range of $0 \leq k < N$.

Plot: [1] 1. the "normalized" level of the DFT spectra. 2. the window DTFT spectra normalized to their mainlobe maximum. The intervals for f, Ω, and amplitudes should be chosen by yourself for the best interpretation purposes.

Interpret the results of the figures obtained regarding the best and worst case for the different windows. Why do the results for the signals with frequencies f_1 and f_2 differ? [2]

Variants

Reports in the form:

[1] similarly to Fig.1, Fig. 2

[2] use the lecture for help

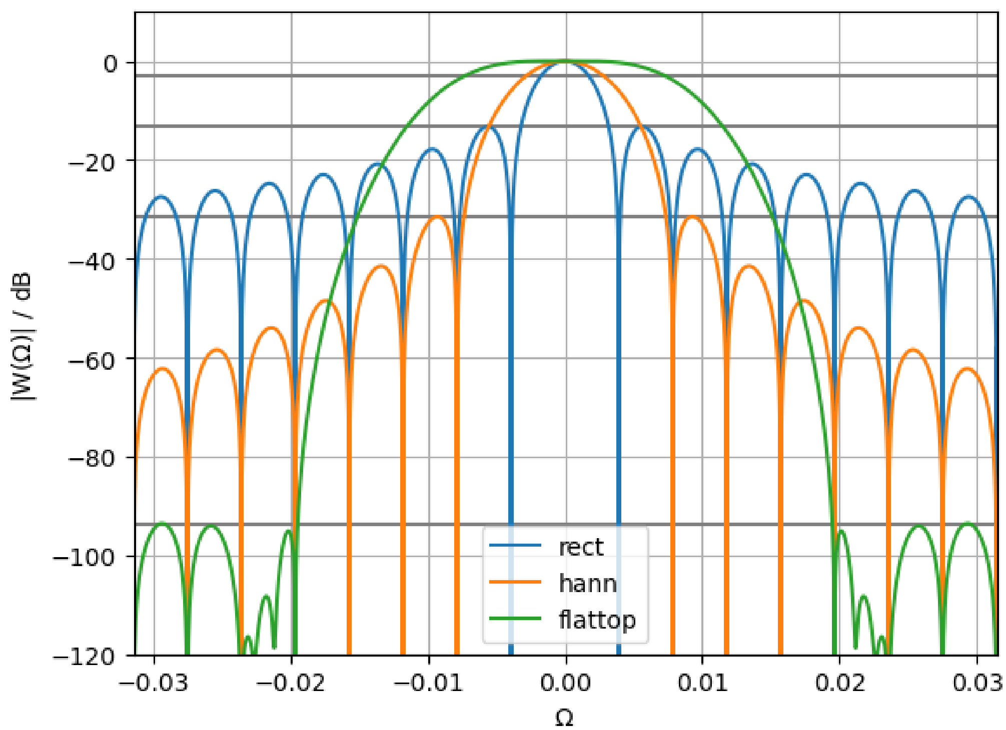

Figure 2: Window DTFT spectra normalized to their mainlobe maximum

No	f_1	f_2	f_3	$\lvert x[k]\rvert_{\max}$	f_s	N
1	300	300.25	299.75	2	400	2000
2	400	400.25	399.75	2	600	3000
3	500	500.25	499.75	2	800	1800
4	600	600.25	599.75	2	500	2000
5	300	300.25	299.75	2	400	2000
6	600	600.25	599.75	3	800	2000
7	400	400.25	399.75	3	600	3000
8	500	500.25	499.75	3	800	1800
9	600	600.25	599.75	3	500	2000
10	300	300.25	299.75	3	400	2000
11	200	200.25	199.75	4	400	2000
12	400	400.25	399.75	4	600	3000
13	500	500.25	499.75	4	800	1800
14	600	600.25	599.75	4	500	2000
15	500	500.25	499.75	4	800	2000

Table 1: Variants

1. Report (file .pdf)

2. file .ipynb

3. pdf-export the file .ipynb

upload to the remote repozitorium (e.g. Github) and link save in the report. Upload the report to eLearning.ubb.edu.pl.

References

References

[pandasUG] Pandas User's Guide `https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html`

[DA2016] Data Analysis with Python and pandas using Jupyter Notebook `https://dev.socrata.com/blog/2016/02/01/pandas-and-jupyter-notebook.html`

[MIT] https://ocw.mit.edu/courses/res-6-008-digital-signal-processing-spring-2011/pages/study-materials/