Communications Project

FFT Windowing Functions

Homework Results

Autor: Richard Grünert

18.4.2023

1 Introduction

The DFT/FFT is acting on discrete sequences which usually come from some form of finite-time real world measurement. Even if the sample rate is suffient enough as to not cause aliasing, signal frequency content that lies in between DFT frequency bins (i.e. not exactly at a multiple of $\Delta f = \frac{f_{\text{sample}}}{N}$) will cause so-called *spectral leakage*. This means that the energy will be spread to the adjacent bins of the actual signal frequency.

This can be understood by thinking of a sinusoidal signal being cut out by a rectangular window that is a sequence of all ones inside the sampling interval and zeros outside. This cutting-out happens by multiplication, meaning that in frequency domain, it will be the convolution of the signal's spectrum with the rectangular spectrum which is a sinc function. Now, the FFT will sample this continuous spectrum in steps of $\Delta f = \frac{f_{\text{sample}}}{N}$. If the input frequency is not a multiple of this then the values around the nearest multiple (as well as further ones) will be those of the window's sinc function. Figure 1 illustrates this.

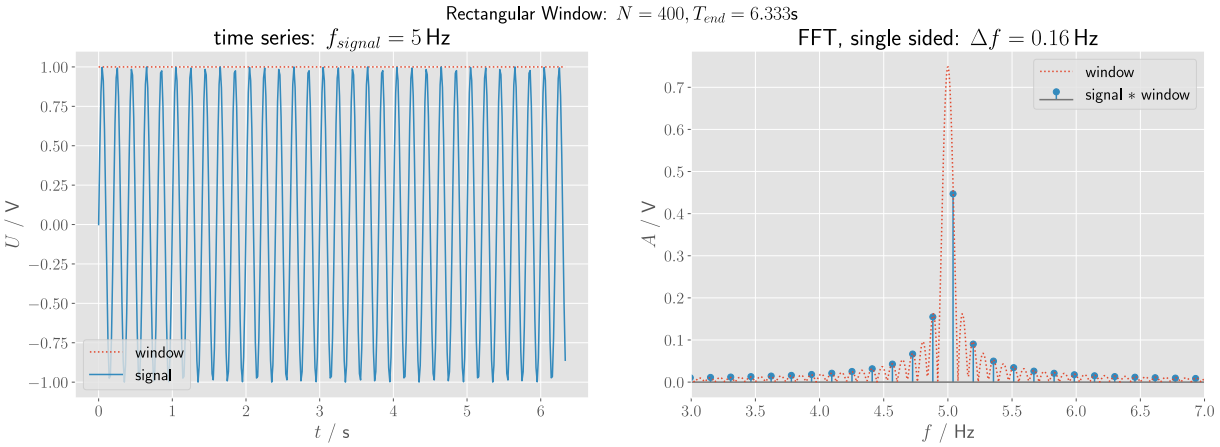

Figure 1: Visualization of DFT spectral leakage. Note the discontinuity from the last to the first sample in the time series.

One obvious way to reduce the leakage is to increase the sample interval (enlargen the rectangular window) which in turn reduces the width of the convoluting sinc function as well as the frequency domain resolution. Another way is to change the shape of the window by applying a windowing function to the time domain sequence. This aims to bring the samples at the beginning and end of the sequence towards a common value, thereby reducing any discontinuity (i.e. you can periodically continue the sequence without any "jumps"). [1]

The following will investigate the influence of different windowing functions using python with `scipy`, `numpy`, and `matplotlib`. The full python script can be found in appendix A.

2 Signal under Test

The number of points has been chosen to be $N = 600$. With a total time window of $T = 2\,\text{s}$ this results in a sampling frequency of $f_\text{s} = \frac{N}{T} = 300\,\text{Hz}$ and an FFT frequency step size of $\Delta f = \frac{1}{T} = 0.5\,\text{Hz}$

The signal used in each case is a sum of three sinusoids:

$$u(t) = 1\,\text{V} \cdot \sin\left(2\pi f_0 \cdot t\right) + 1\,\text{V}\sin\left(2\pi f_1 \cdot t\right) + 1\,\text{V}\sin\left(2\pi f_2 \cdot t\right)$$

with

$$f_0 = 1.5\,\text{Hz}$$
$$f_1 = 4.25\,\text{Hz}$$
$$f_2 = 6.1\,\text{Hz}$$

The frequencies have been chosen such that one lies exactly on a multiple of Δf (f_0), one exactly half way between two multiples (f_1), and one lies close to a multiple (f_2).

Figure 2 shows the signal in time and its fft.

3 Windowing Results

3.1 Rectangular

In the time domain, there is a clear discontinuity between start and end of the time series and as expected, the spectral leakage is present in the spectrum (figure 2). This spectrum will be used as a reference. The figures of the following window types will show their differences towards this reference window in their spectra for comparison.

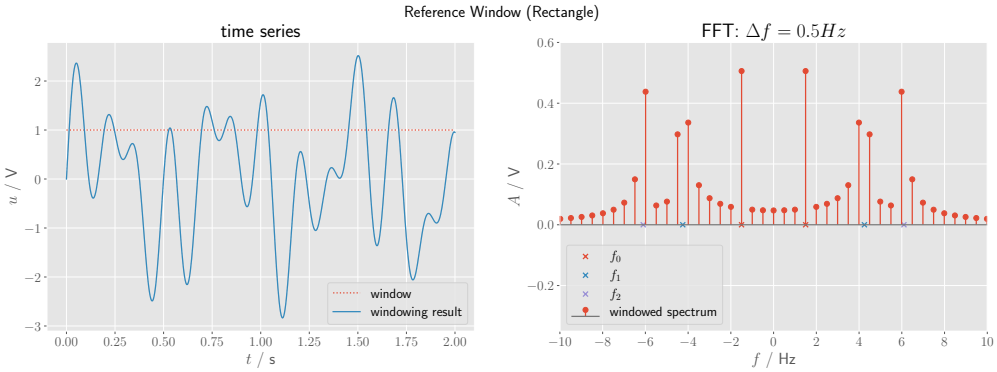

Figure 2: Time series and FFT with rectangular window (default).

As for the signal frequencies, f_0 is exactly at a multiple of Δf and does not seem to produce any leakage. f_1's spectral content is approximately evenly split between the two adjacent bins. f_2's split is in favour of the closer frequency bin.

3.2 Bartlett

The Bartlett window is a form of triangular window where the beginning and end points are zero. [3] A triangular pulse/window also is of sinc-shape like the rectangular window, however it falls off more rapidly with frequency [2][4], which makes it a potentially better candidate of a windowing function.

Figure 4 shows the effective reduction of the leakage around the signal frequencies. The positive differences, however, indicate an increase in leakage, especially around f_0. This will be the case for all window types discussed here.

3.3 Hanning

The Hanning window has a shape that results in a time series quite similar to the Bartlett window (figure 5). The difference is best seen in the comparison of the window spectra as seen in figure 3. The Bartlett window falls off rapidly with frequency but shows quite high side lobes which means it might be better suited for features that are closer in frequency. This behaviour can also be seen in figures 5 and 4, e.g. at the samples around f_2.

3.4 Hamming

The hamming window shows a main lobe that is less wide than that of the hanning window but it does not drop off as fast for the higher side lobes. [1] This is also to an extend visible in figure 6. So, for frequency bins closer to the signal frequency, there will be less leakage than with the hanning window but it is the opposite for bins further away.

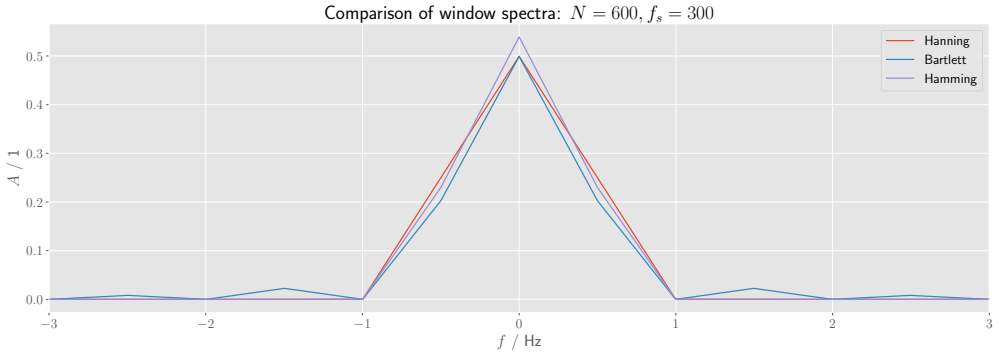

Figure 3: Comparison of the spectra of the used windows.

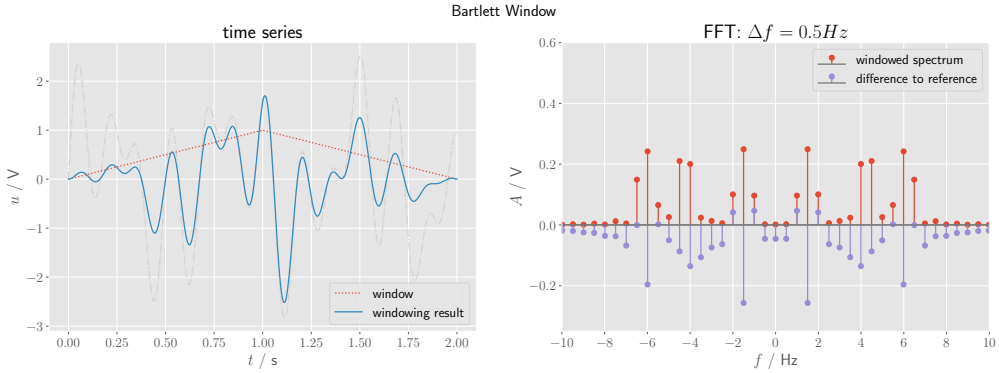

Figure 4: Time series and FFT after applying the Bartlett window.

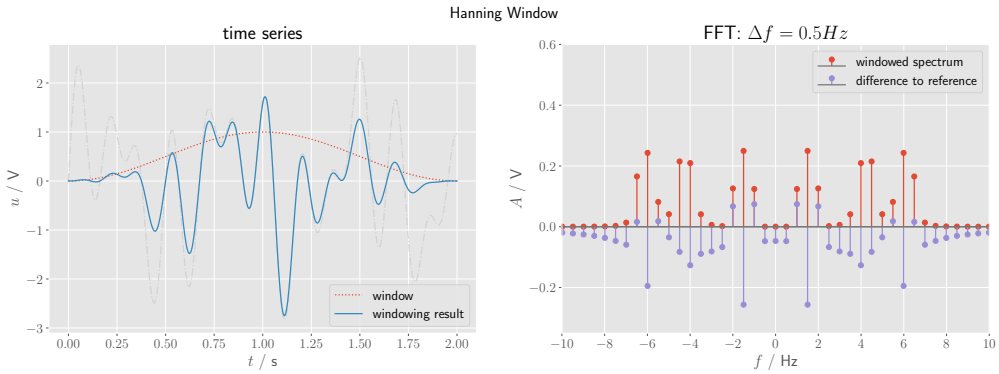

Figure 5: Time series and FFT after applying the Hann window.

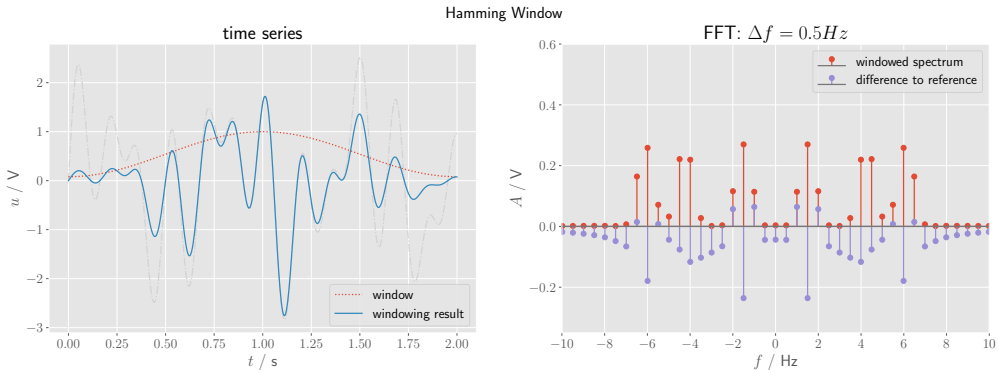

Figure 6: Time series and spectrum after applying the Hamming window.

4 Conclusion

The difference between these three window types is not large in this example. However, all have in common that they reduce the spectral leakage of the rectangular window's spectral side lobes.

In each case, the amplitudes of the harmonic frequencies of interest were reduced as well. Each window function introduced a reduction by a factor of approximately 0.5. Also, for frequencies that lie exactly on a multiple of Δf, windowing actually increases the leakage. This means that in practice, the reduction of the main lobe in relation to the side lobes of the windows may have to be evaluated depending on the signal of interest.

References

Literature

[1] Richard G. Lyons. *Understanding Digital Signal Processing*. Ed. by Patty Donovan. 2nd ed. ISBN 0-13-108989-7. Prentice Hall Professional Technical Reference, 2004.

[2] Adolf J. Schwab and Wolfgang Kürner. *Elektromagnetische Verträglichkeit*. 5th ed. ISBN 978-3-540-42004-0. Springer Berlin Heidelberg New York, 2007.

[3] The SciPy community. *scipy.signal.windows.bartlett*. https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.windows.bartlett.html. Accessed: 2023-04-17.

[4] Helmut Ulrich and Stephan Ulrich. *Laplace-Transformation, Diskrete Fourier-Transformation und z-Transformation*. 11th ed. ISBN 978-3-658-31877-2. Springer Vieweg, 2023.

Software Used

[1] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[2] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[3] Python Core Team. *Python: A dynamic, open source programming language*. Python version 3.7. Python Software Foundation. 2019. URL: https://www.python.org/.

[4] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

A Python Code

```python
# Homework I Comsys Project
# Author: R. Grünert, 2023

import numpy as np
import scipy
from numpy.fft import fft, fftshift
from matplotlib import pyplot as plt

# uncomment these for LaTeX integration
plt.style.use('ggplot')
plt.rcParams['text.usetex'] = True
plt.rcParams['figure.figsize'] = 20, 6.18
plt.rcParams['font.size'] = 16


class FFTWindow:
    def __init__(self, size):
        self.size = size
        self.values = self.calc_values()
        self.name = "none"

    def apply_to(self, sequence):
        return self.values * sequence

    def calc_values(self):
        return []


class RectangleWindow(FFTWindow):
    def __init__(self, size):
        super().__init__(size)
        self.name = "Rectangle"

    def calc_values(self):
        return np.ones(self.size)


class HanningWindow(FFTWindow):
    def __init__(self, size):
        super().__init__(size)
        self.name = "Hanning"

    def calc_values(self):
        return np.hanning(self.size)


class HammingWindow(FFTWindow):
    def __init__(self, size):
        super().__init__(size)
        self.name = "Hamming"

    def calc_values(self):
        return np.hamming(self.size)


class BartlettWindow(FFTWindow):
```

```
    def __init__(self, size):
        super().__init__(size)
        self.name = "Bartlett"

    def calc_values(self):
        return np.bartlett(self.size)


def get_spectrum(signal, ts):
    N = len(signal)
    spectrum = fft(signal) / N
    freq = np.fft.fftfreq(N, ts)
    freq = fftshift(freq)
    spectrum = fftshift(spectrum)
    return (freq, spectrum)


def plot_signal_and_window_with_spectrum(signal, window, t, spectrum, freq):
    fig, (ax1, ax2) = plt.subplots(1, 2)
    #fig.tight_layout()
    ax1.plot(t, signal, linestyle="-.", color="lightgray")
    # interestingly, you can put window without the .values reference
    # and it will work. Not sure why but i'll take it
    ax1.plot(t, window, linestyle="dotted", label="window")
    ax1.plot(t, window.apply_to(signal), label="windowing result")
    ax2.stem(freq, np.abs(spectrum), label="windowed spectrum")
    ax2.set_xlim([-10, 10])
    ax2.set_ylim([-0.35, 0.6])
    ax1.set_ylabel("$u$ / V")
    ax1.set_xlabel("$t$ / s")
    ax2.set_ylabel("$A$ / V")
    ax2.set_xlabel("$f$ / Hz")
    ax1.legend()
    ax2.legend()
    ax2.set_xticks(np.arange(-10, 11, 2))

    ax1.set_title("time series")
    ax2.set_title("FFT: $\Delta f = " + str(round(freq[1]-freq[0], 2)) + " Hz$")

    return (fig, (ax1, ax2))


def my_savefig(fname):
    #plt.savefig(fname, bbox_inches='tight', format='pdf')
    plt.savefig(fname, format='pdf')


def signal_under_test(t):
    return\
        1.0 * np.sin(1.5 * 2.0 * np.pi * t) +\
        1.0 * np.sin(4.25 * 2.0 * np.pi * t) +\
        1.0 * np.sin(6.1 * 2.0 * np.pi * t)


def main():

    NUM_POINTS = 600
    t = np.linspace(0.0, 2, NUM_POINTS)
```

```
    Ts = t[1] - t[0]
    signal = signal_under_test(t)

    windows = [
        HanningWindow(NUM_POINTS),
        BartlettWindow(NUM_POINTS),
        HammingWindow(NUM_POINTS)
    ]

    # reference windowing
    ref_window = RectangleWindow(NUM_POINTS)
    freq, ref_spectrum = get_spectrum(ref_window.apply_to(signal), Ts)

    fig, (ax1, ax2) = plot_signal_and_window_with_spectrum(
        signal, ref_window, t, ref_spectrum, freq)

    sig_freqs = [1.5, 4.25, 6.1]
    i = 0
    for freq in sig_freqs:
        ax2.scatter([-freq, freq], [0, 0], marker='x', label="$f_{" + str(i) + "}$")
        i += 1
    ax2.legend()
    fig.suptitle("Reference Window (Rectangle)")
    my_savefig("outputs/Rectangle.pdf")

    # windowing plots
    for window in windows:

        freq, myspec = get_spectrum(window.apply_to(signal), Ts)

        diff = np.abs(myspec) - np.abs(ref_spectrum)

        fig, (ax1, ax2) = plot_signal_and_window_with_spectrum(
            signal, window, t, myspec, freq)

        ax2.stem(freq, diff, linefmt="C2-", label="difference to reference")

        ax2.legend()
        fig.suptitle(window.name + " Window")

        my_savefig("outputs/" + window.name + '.pdf')

    # plot window spectra
    plt.figure()
    for window in windows:
        freq, spec = get_spectrum(window.values, Ts)
        spec = np.abs(spec)
        plt.plot(freq, spec, label=window.name)
        plt.xlim([-3, 3])
    plt.title("Comparison of window spectra: $N=" + str(NUM_POINTS) + ", f_{s}=" +
    str(round(1/Ts)) + "$")
    plt.xlabel("$f$ / Hz")
    plt.ylabel("$A$ / 1")
    plt.legend()
    my_savefig("outputs/win_spectra.pdf")

    plt.show()

```

```
172
173 if __name__ == '__main__':
174     main()
```