Chapter 1

Digital filters

One of the main applications for the digital Fourier transform (DFT) is digital filtering, i.e. reducing unwanted frequency components or boosting the desired ones. We all have seen it in the form of an equalizer in a music player, which allows one to adjust the loudness of low frequency components (bass) relative to the middle and the high frequency compoenents (treble) of the sound spectrum. It used to be done with analog electronics components filters, but these days, with the proliferation of micro controllers, it is often done digitally via the DFT.

In this chapter, we will learn some lore associated with filters and how to implement them.

1.1 Nyquist frequency and the minimal sampling rate

Before we can filter any data, we should acquire it. The main question is what the sampling rate ($f_s = 1/\Delta t$) of the data acquisition should be.

Recall the discussion in section 15.2, where we showed that the highest observable frequency in the DFT spectrum is $\approx f_s/2$. This brings us to

Nyquist-Shannon sampling criteria

If our signal has the highest frequency f_{max} then we need to sample it with

$$f_s > 2f_{max} \tag{1.1}$$

Pay attention to the $>$ relationship!

The Nyquist-Shannon sampling criteria is quite often used in reverse form: one cannot acquire frequencies in the signal above the Nyquist frequency $f_{Nq} = f_s/2$.

This criteria is not very constructive. How would we know what the highest frequency of the signal is? Sometimes we know it from the physical limitations of our apparatus. If we don't know highest frequency, we should sample the signal with some sampling frequency. If at the high frequency end of the spectrum, the strength of components drops to zero then we have sampled fast enough, so we might even try to reduce the sampling frequency. Otherwise, we are *undersampling*. We must increase the sampling frequency until we see the high end of the spectrum asymptotically small.

Words of wisdom

Choosing the sampling frequency is the most important part of the data acquisition. No amount of post-processing will be able to recover or restore the signal which was acquired with a wrong sampling frequency.

1.1.1 Undersampling and aliasing

We will see how an undersampled signal might look in examples shown below. In this section, we will sample the signal described by the following equation

$$y(t) = \sin(2\pi 10t) \tag{1.2}$$

i.e. this is the sinusoidal signal with the signal frequency $f_{signal} = 10$ Hz.

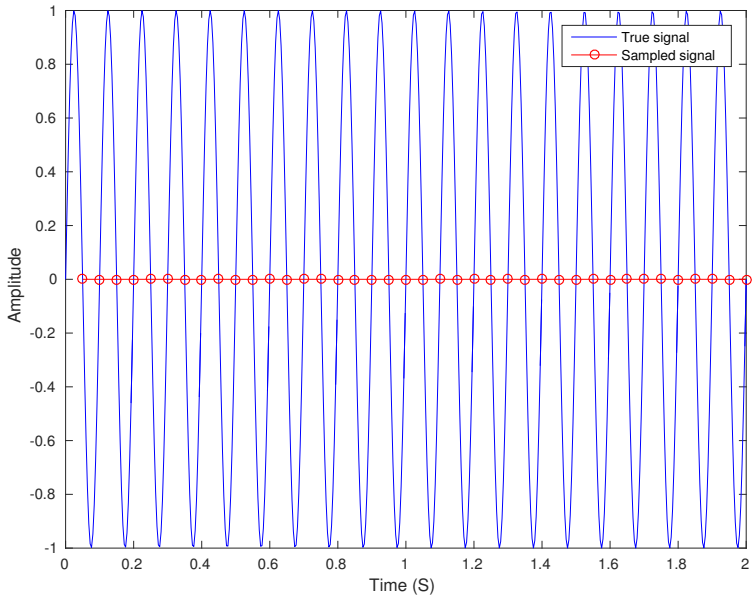

Figure 1.1: The signal described by eq. (1.2) (lines) and the undersampled signal acquired with $f_s = 2f_{signal}$ (circles and lines).

At first, we sample our signal with $f_s = 2f_{signal}$. As we can see in fig. 1.1 the sampled signal appears as a straight line, even though it evident that the underlying signal is sinusoidal. We clearly used the wrong sampling frequency. This example emphasizes the $>$ relationship in eq. (1.1). Note that the lines connecting the sampling points are just to guide an eye. The DFT has no notion of the signal values in between the sampled points.

In the following example, we sample with $f_s = 1.1f_{signal}$, i.e. our sampling frequency does not satisfy the Nyquist-Shannon criteria. As we can see in fig. 1.2, the sampled signal does not reproduce the underlying signal. Moreover, the undersampled signal appears as a signal with a lower frequency. This phenomenon is called *aliasing* or *ghosting*. It arises due to the periodicity of the DFT spectrum, as high frequency components require the existence of the c_M components, where $M > N$ but we recall that $C_{-n} = C_{N-n}$, see eq. (15.17). Thus $c_M = c_m$ where $m = M - N \times l$ and l is an integer. In other words, if the signal is undersampled, i.e. the sampling frequency does not satisfy section 1.1 above, and a high frequency component appears as a low frequency one. Consequently, in the undersampled spectrum, we see the ghost frequency components

$$f_{ghost} = |f_{signal} - l \times f_s| \tag{1.3}$$

For the case depicted in fig. 1.2, we see the appearance of the signal with $f_{ghost} = 0.1$ Hz or period of 1 seconds[1].

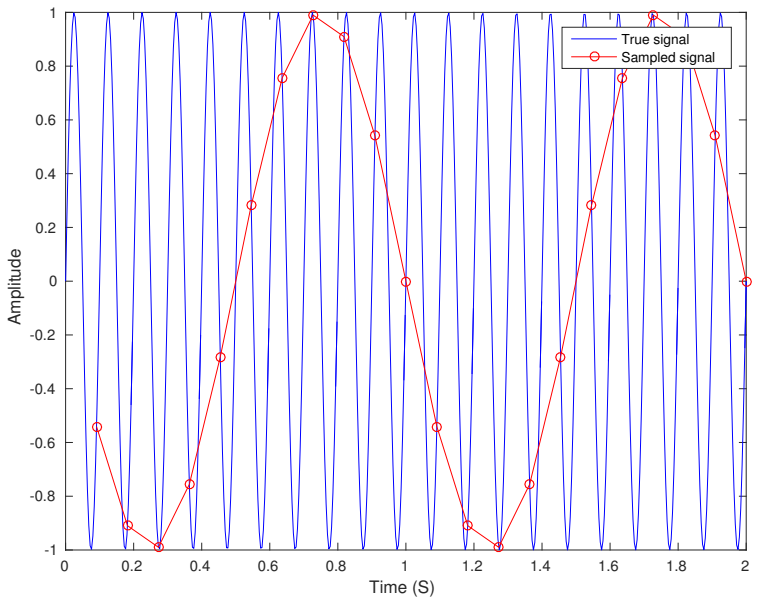

Figure 1.2: The signal described by eq. (1.2) (lines) and the undersampled signal acquired with $f_s = 1.1f_{signal}$ (circles and lines).

[1] If you ever use a digital oscilloscope be aware of the aliasing phenomenon. If you choose your acquisition rate wrongly, you will see nonexistent signals.

In some cases, the aliasing makes even stranger looking sampled signals which do not resemble the underlying signal at all. This is illustrated in fig. 1.3, where the $f_s = 1.93 f_{signal}$.

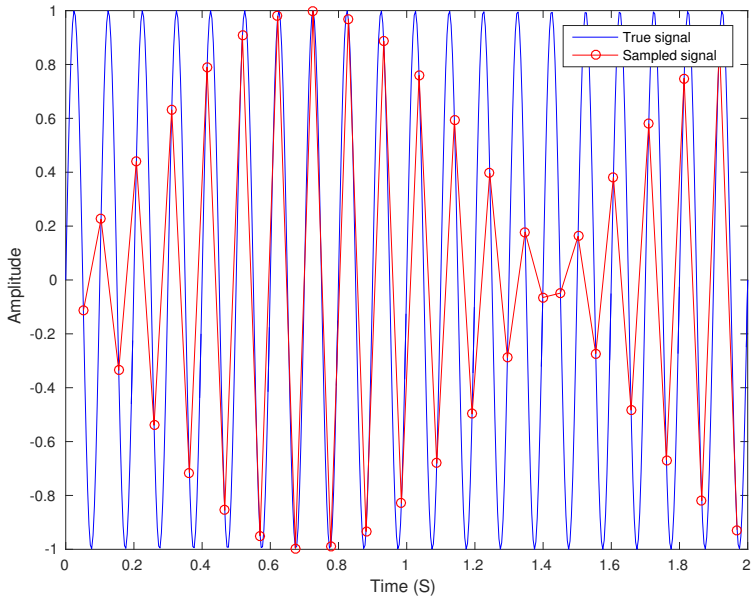

Figure 1.3: The signal described by eq. (1.2) (lines) and the undersampled signal acquired with $f_s = 1.93 f_{signal}$ (circles and lines).

Words of wisdom

If you cannot sample fast enough, build a low-pass electronics filter which removes fast frequency components larger than obtainable $f_s/2$. Otherwise, the digitized signal will be contaminated by ghost signals.

1.2 DFT filters

The material outlined in the previous section about the importance of the proper sampling frequency is strictly speaking outside the scope of this book. It is in the domain of data acquisition and instrumental science. Nevertheless, it is always a good idea to apply sanity checks to the data and see what could have gone wrong before you start analyzing the data.

For now, we assume that someone gave us the data and we have to work with what we received. As we discussed at the beginning of the chapter, the job of a digital filter is to somehow modify the spectrum of the signal, i.e. to boost or suppress certain frequencies, and then reconstruct the filtered signal.

The recipe is the following:

- Calculate the DFT (use MATLAB's `fft`) of the signal;
- Have a look at the spectrum and decide which frequencies are to be modified;
- Apply a filter which adjusts the amplitudes of the frequencies we are interested in;
  - For signals belonging to the real numbers domain: if you attenuate the component with the frequency f by g_f, you need to attenuate the component at $-f$ by g_f^*. Otherwise, the inverse Fourier transform, i.e. the reconstructed signal, will have non zero imaginary part.
- Calculate inverse DFT (use MATLAB's `ifft`) of the filtered spectrum;
- Repeat if needed.

Mathematical representation of the digital filtering

$$y_{filtered}(t) = \mathcal{F}^{-1}\left[\mathcal{F}(y(t)) \times G(f)\right] = \mathcal{F}^{-1}\left[Y(f) \times G(f)\right] \tag{1.4}$$

where

$$G(f) = Y_{filtered}(f)/Y(f) \tag{1.5}$$

is the frequency dependent *gain* function[2]. The $G(f)$ controls how much we change the corresponding spectral component.

The eq. (1.4) looks quite intimidating, but we will show below that the filtering is very easy. Simultaneously, we will learn some standard filters and their descriptions.

1.2.1 Low-pass filter

In all following examples, we will work with the following signal

$$y(t) = 10\sin(2\pi f_1 t) + 2\sin(2\pi f_2 t) + 1\sin(2\pi f_3 t) \tag{1.6}$$

where $f_1 = 0.02$ Hz is the slow component, $f_2 = 10f_1$ Hz, and $f_3 = 0.9$ Hz is the fast component. This signal is depicted in the left part of fig. 1.4. The spectrum of this signal sampled with $f_s = 2$ Hz on the interval of 100 seconds, is shown in the right part of fig. 1.4. As expected, it consists of three strong frequency components corresponding to f_1, f_2, and f_3.

The *low-pass filter* is the filter which strongly suppresses or attenuates high frequency components of the spectrum, while keeping low frequency components mostly intact. We specifically focus on the brick wall low-pass filter described by the following gain equation

$$G(f) = \begin{cases} 1, |f| \leq f_{\text{cutoff}} \\ 0, |f| > f_{\text{cutoff}} \end{cases} \tag{1.7}$$

[2]In spite of the name, it is quite often less than unity.

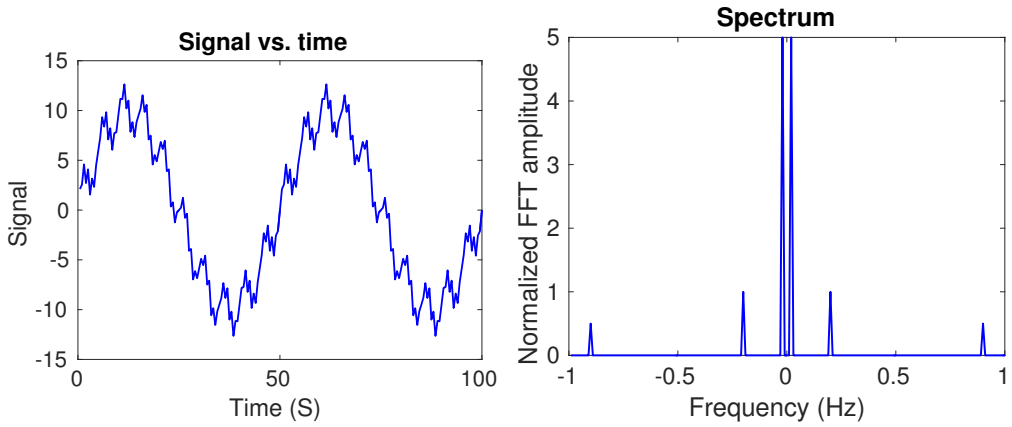

Figure 1.4: The signal described by eq. (1.6) (on the left) and its spectrum (on the right).

The gain function is usually complex, so it is often shown in the Bode plot representation where we plot the absolute value (magnitude) of the gain versus frequency in the upper sub plot and the complex angle (or phase) of the gain versus frequency in the lower sub plot. For this filter, it is shown in the right part of fig. 1.5. The name 'brick wall' comes from the very sharp transitions of the filter gain near the cutoff frequency f_{cutoff} which is equal 0.24 Hz in this case.

We obtain the filtered `y` signal with the following code

```
freq=fourier_frequencies(SampleRate, N);
G=ones(N,1); G( abs(freq) > Fcutoff, 1)= 0;
y_filtered  =  ifft(  fft( y ) .* G  )
```

As you can see, it is very simple. The filter strength is calculated and assigned at the second line of the above code, and the filter application is done at the last line. The indispensable function `fourier_frequencies` connects a particular index in the DFT spectrum to its frequency (we discussed its listing 15.5 in section 15.5).

The filtered spectrum is shown in the center part of fig. 1.5. As expected, the spectral component with the frequency f_3 is now zero, since it lies beyond the cutoff frequency. We now have only f_1 and f_2 in the spectrum. Consequently, the filtered signal does not have a high frequency (f_3) component, as it shown in the comparison of the filtered and raw signals in the right part of fig. 1.5. The filtered signal is much smoother, i.e. missing the high frequency components. Thus, application of the low-pass filter is sometimes referred as *smoothing* or *denoising*[3].

[3]'Denoising' is actually the misnomer since the useful signal can be located at high frequencies as well.

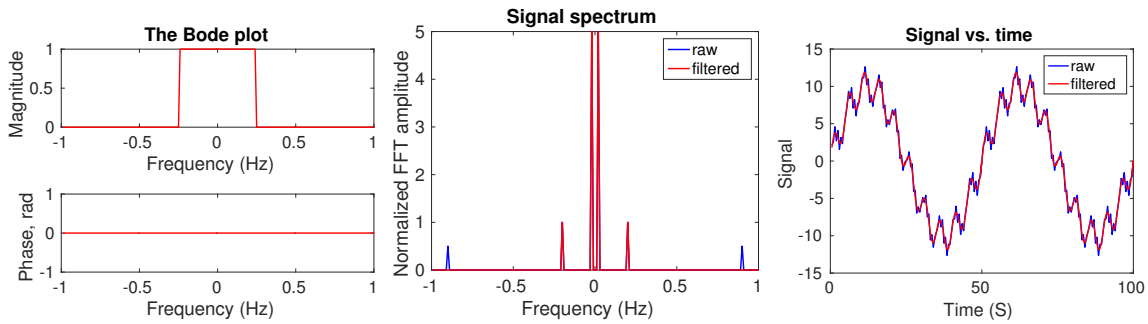

Figure 1.5: The Bode plot of the brick wall low-pass filter (left). The comparison of the filtered and unfiltered spectra (center) and signals (right).

1.2.2 High-pass filter

The *high-pass filter* is the exact opposite of the low-pass filter, i.e. it attenuates low frequency components and leaves intact high frequency components. For example, the brick wall high-pass filter can be describe by the following equation

$$G(f) = \begin{cases} 0, |f| \leq f_{\text{cutoff}} \\ 1, |f| > f_{\text{cutoff}} \end{cases} \tag{1.8}$$

The Bode plot of this filter is shown in the left part of fig. 1.6. The MATLAB implementation of the brick wall high-pass filter is shown in the code below

```
freq=fourier_frequencies(SampleRate, N);
G=ones(N,1); G( abs(freq) < Fcutoff, 1)= 0;
y_filtered  =  ifft(  fft( y ) .* G  )
```

The filtered spectrum, missing the low frequency component f_1, is shown in the center part of fig. 1.6. As we can see in the right part of fig. 1.6, the high-pass filter gets rid of the slow envelope from the raw signal.

1.2.3 Band-pass and band-stop filters

The *band-pass filter* allows only frequency components within a certain bandwidth (f_{bw}) in the vicinity of the specified central frequency (f_c) to pass, while all other frequencies are strongly attenuated. For example, the brick wall band-pass filter is described by the following equation

$$G(f) = \begin{cases} 1, ||f| - f_c| \leq \frac{f_{bw}}{2} \\ 0, ||f| - f_c| > \frac{f_{bw}}{2} \end{cases} \tag{1.9}$$

It can be implemented by the following MATLAB's code

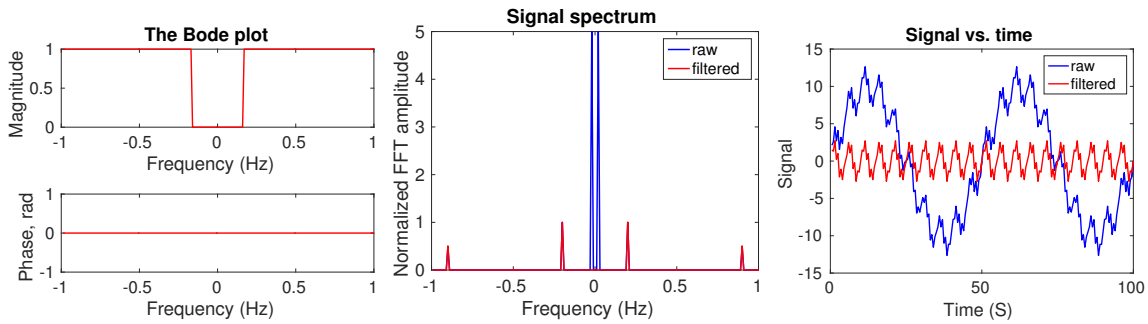

Figure 1.6: The Bode plot of the brick wall high-pass filter (left). The comparison of the filtered and unfiltered spectra (center) and signals (right).

```
freq=fourier_frequencies(SampleRate, N);
G=ones(N,1); G( abs(abs(freq)-Fcenter) > BW/2, 1)=0;
y_filtered  =  ifft(  fft( y ) .* G  )
```

The *band-stop* (or *band-cut*) filter is the exact opposite of the band-pass filter. The brick wall band-stop filter described by the following equation

$$G(f) = \begin{cases} 0, ||f| - f_c| \leq \frac{f_{bw}}{2} \\ 1, ||f| - f_c| > \frac{f_{bw}}{2} \end{cases} \tag{1.10}$$

The MATLAB's implementation is shown below

```
freq=fourier_frequencies(SampleRate, N);
G=zeros(N,1); G( abs(abs(freq)-Fcenter) > BW/2, 1)=1;
y_filtered  =  ifft(  fft( y ) .* G  )
```

The Bode plot of the filter with $f_c = f_2 = 0.2$ Hz and $f_{bw} = 0.08$ Hz is shown in the left part of fig. 1.7. The band-stop filter with above parameters will remove f_2 from our signal spectrum as it shown in the center part of fig. 1.7. The filtered signal now looks like slow envelope with the frequency f_1 with the high frequency (f_3) "fuss" on top of it, as it is shown in the right part of fig. 1.7.

1.3 Filter's artifacts

The ease of the implementation of the brick wall filters comes at a price: they often produce ring down artifacts which were not present in the original raw signal.

Let's have a look at the signal and its spectrum correspondingly depicted on the left and right of fig. 1.8. When we apply the brick wall low-pass filter with $f_{\text{cutoff}} = 0.24$ Hz (see its

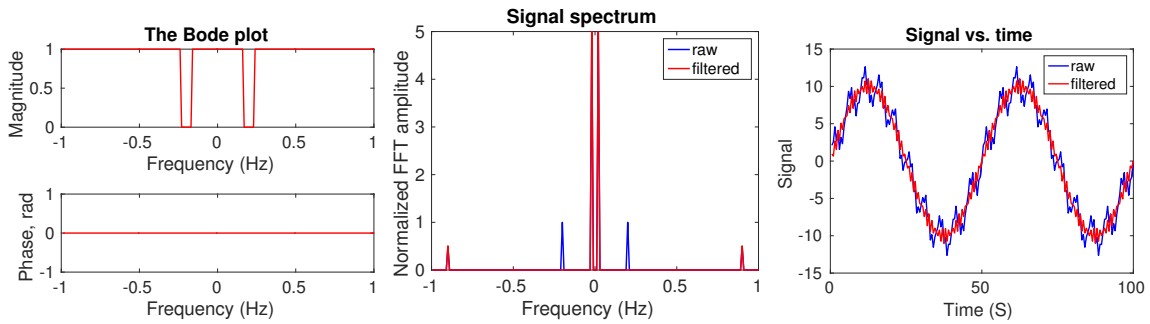

Figure 1.7: The Bode plot of the brick wall band-stop filter (left). The comparison of the filtered and unfiltered spectra (center) and signals (right).

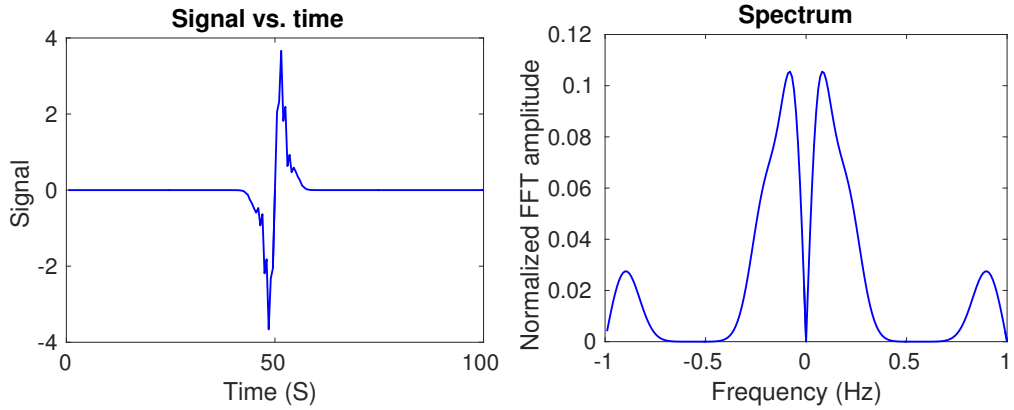

Figure 1.8: The sample signal (on the left) and its spectrum (on the right).

Bode plot in the left part of fig. 1.9), we obtain the filtered spectrum shown in the center part of fig. 1.9. The significant discontinuity at 0.24 Hz produces a large ring-down-like disturbance on the filtered signal, as we can see in the right part of fig. 1.9. The easiest way to avoid it is to use a filter without discontinuities in the spectrum. For example, we can construct the smooth low-pass gain function according to the following equation

$$G(f) = \left| \frac{1}{1 + i(f/f_{\text{cutoff}})} \right| \tag{1.11}$$

with the Bode plot depicted in the right part of fig. 1.10.

This filter is weaker than its brick wall counterpart. Thus, the high frequency components are not as strongly suppressed (see the center part of fig. 1.10). Nevertheless, we successfully removed high frequencies from the raw signal without introducing artifacts (see the right part of fig. 1.10).

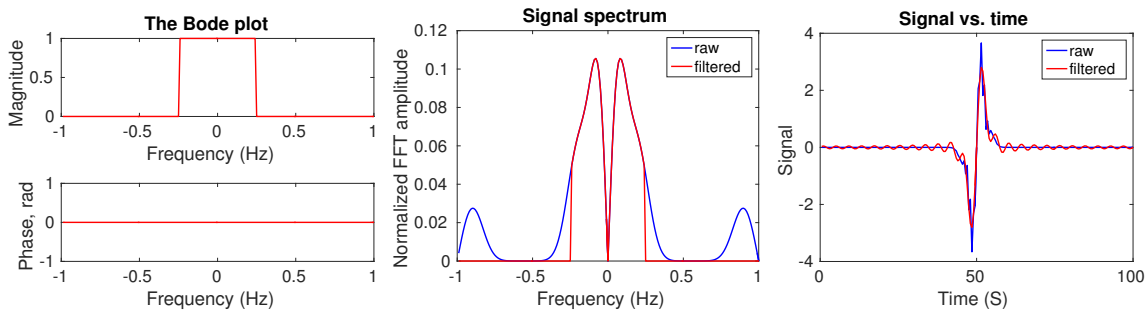

Figure 1.9: The Bode plot of the brick wall low-pass filter (left). The comparison of the filtered and unfiltered spectra (center) and signals (right).

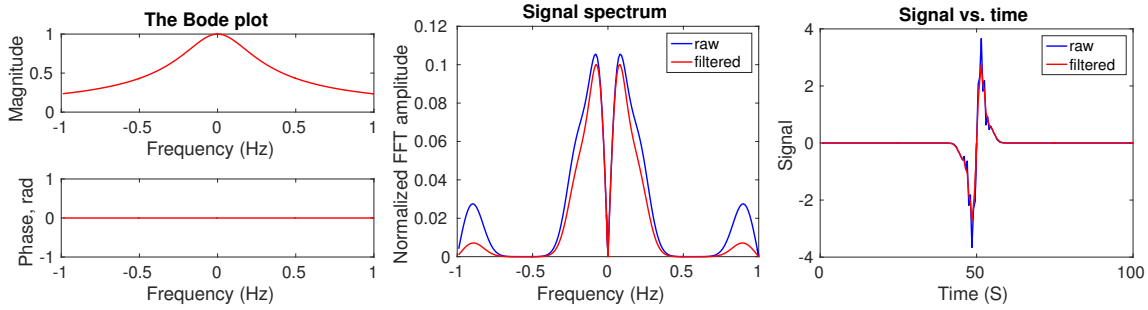

Figure 1.10: The Bode plot of the smooth low-pass filter (left). The comparison of the filtered and unfiltered spectra (center) and signals (right).

1.4 Windowing artifacts

The time domain discontinuities generate spurious frequency components in the DFT spectrum. Such discontinuities are often located at the beginning and the end of the acquisition period since this period is often arbitrarily chosen.

For example, even if we sample a pure cosine signal the ends of the acquired signals might not match, as it shown in the left part of fig. 1.11. We expect a spectrum with the single frequency matching the cosine one, but discontinuities generate nonexistent frequency components, as it shown in the right part of fig. 1.11. For example, we see non zero spectral components beyond 0.5 Hz frequency for the underlying cosine with 0.045 Hz frequency.

To avoid this, usually some sort of *windowing* function ($w(t)$) is applied and then the DFT calculated on $y(t) \times w(t)$. There are many window functions[4], but they all have a common property: they asymptotically approach zero at the beginning and the end of the acquisition time to remove discontinuity at the ends. For example, the Hann window coefficients are

[4] Some of the most popular are Hamming, Tukey, Cosine, Lanczos, Triangulars, Gaussians, Bartlett-Hann, Blackmans, Kaisers.

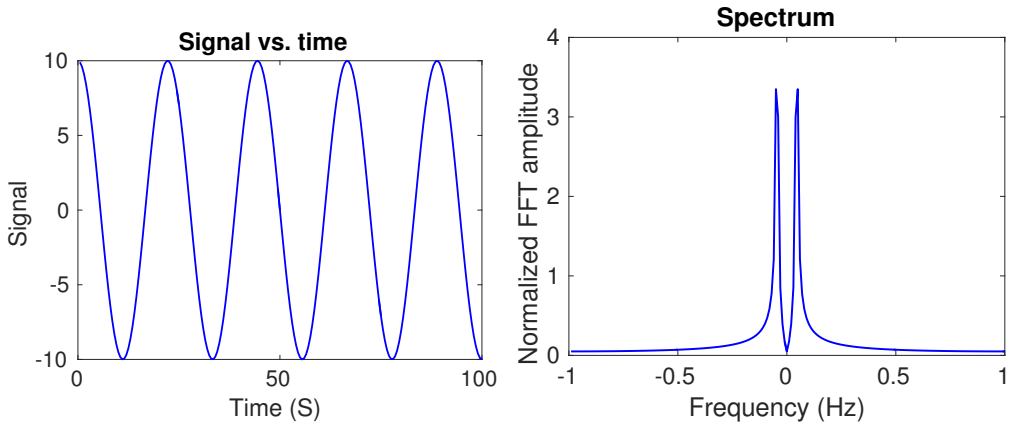

Figure 1.11: The cosine signal acquired for the time not matching it own period and its DFT spectrum.

given by the following equation

$$w_n = \frac{1}{2}\left[1 - \cos\left(2\pi \frac{n-1}{N-1}\right)\right] \tag{1.12}$$

Our cosine signal with applied Hann's window looks like signal shown in the left part of fig. 1.12. The resulting DFT spectrum shape better matches the single frequency spectrum of the original cosine, as we can see in the right part of fig. 1.12.

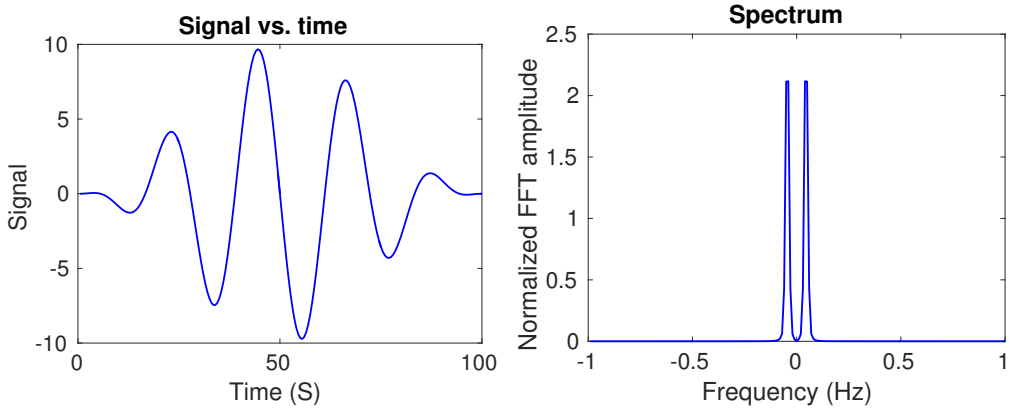

Figure 1.12: Signal and its spectrum after application of the Hann window.

We are calculating `fft(y.*w)`, i.e. spectrum of the modified function, so the DFT spectrum strength and shape should not match exactly the underlying signal. Nevertheless, a window function often drastically improves the fidelity of the spectrum, although it often reduces the spectral RBW which now is $\sim 1/T_{window} < T_{acq}$, here T_{window} is characteristic time where window function is large, and T_{acq} is the full acquisition time.

1.5 Self-Study

Problem 1: Download the wave file `'voice_record_with_a_tone.wav'`[5]. It is the audio file. If you play it, you will hear a very loud single-frequency tone, which masks the recorded voice. Apply an appropriate band-stop filter to hear the message. What is the message?

To obtain audio data, use the following commands. This assumes that the audio file is in the current folder of MATLAB.

```
[ydata, SampleRate]=audioread('voice_record_with_a_tone.wav', 'double');
% the following is needed if you want to save the filtered signal
info=audioinfo('voice_record_with_a_tone.wav');
NbitsPerSample=info.BitsPerSample;
```

After execution of the above, the `ydata` variable will hold the amplitudes of audio signal sampled equidistantly with the sampling rate stored in the `SampleRate` variable. Note that columns in `ydata` correspond to audio channels. So, there could be more than one. However, it sufficient to process only one channel for this problem.

Once, you filtered your data, it is a good idea to normalize it to 1. This will make it louder.

You can play audio data within MATLAB with the following command:

```
sound(y_filtered(:,1), SampleRate);
```

Alternatively, you can save it into a wave audio file with the following command:

```
audiowrite('voice_record_filtered.wav', y_filtered, SampleRate, ...
  'BitsPerSample', NbitsPerSample);
```

[5]The file is available at http://physics.wm.edu/programming_with_MATLAB_book/ch_digital_filters/data/voice_record_with_a_tone.wav