Chapter 10
Introduction to Digital Signal Processing

This section will not be a comprehensive coverage of Digital Signal Processing that a student would learn as part of their electrical engineering curriculum, which often comes as a two-course sequence. The first course is a "Signals and Systems" course that would include the *Continuous Time Fourier Transform* (**CTFT**). Since computers are heavily involved in signal processing, the theory has been extended to signals that have been sampled in time. This would be the topic of the second course called Digital Signal Processing.

Although this section will be math heavy (yes, there is calculus involved), there will still be useful takeaways for those that use these signal processing algorithms but do not have the math background. I will highlight areas where you need to be careful and why certain choices are often made when using these algorithms.

10.1 Sampling

To process a signal with a computer, we first need to create a representation of this signal. As we will show in Sect. 10.2, we can represent signals using simple sinusoidal functions. However, this will only work correctly if we ensure that we have sampled the signal properly. These are the steps for sampling correctly, which is how we implement the **Sampling Theorem**.

Step 1: Determine the *maximum frequency* $f_{\max}$ contained in the continuous signal we wish to represent in a computer. If we do not know what this maximum frequency is, which is typically the case, then we need to limit the frequencies to a known $f_{\max}$ by *low pass filtering* the signal to remove all frequencies above $f_{\max}$. This filter is known as an anti-aliasing filter and this filtering needs to be performed *before* sampling. If there is no analog anti-aliasing circuitry before the analog-to-digital converter where the sampling occurs, then the sampled signal is suspect.

Step 2: Sample the continuous signal that has been conditioned to have no frequencies greater than $f_{\max}$ with a sampling rate f_s that is *greater* than twice $f_{\max}$

Sampling Theorem

$$\boxed{f_s > 2f_{\max}}$$

The reason we must sample greater than twice the maximum frequency is due to the periodicity of the sine and cosine functions. This identity is listed below for the cosine function.

$$\cos(\theta) = \cos(\theta + 2\pi) \tag{10.1}$$

What this means in practical terms is that if we do not ensure that the sampling theorem has been followed, then there will be high frequencies masquerading as low frequencies, which is known as *aliasing*. To illustrate this, let us create two signals where signal one is comprised of frequency f_1, which will be less than one-half f_s and thus properly sampled. We will create a second signal that will be comprised of frequency f_2, and this frequency will violate the sampling theorem. For convenience, we will create f_2 that is some multiple of the sampling rate higher than f_1:

$$f_2 = f_1 + kf_s \tag{10.2}$$

If we take the first signal with frequency f_1

$$x(t) = \cos(2\pi f_1 t + \phi) \tag{10.3}$$

and sample it at the sampling period $T_s = 1/f_s$, it becomes

$$\begin{aligned} x[n] &= x(nT_s) \\ &= \cos(2\pi f_1 nT_s + \phi) \end{aligned} \tag{10.4}$$

Now take the second signal with frequency f_2

$$y(t) = \cos(2\pi f_2 t + \phi) \tag{10.5}$$

and sample it at the same sampling rate:

$$\begin{aligned} y[n] &= y(nT_s) \\ &= \cos(2\pi f_2 nT_s + \phi) \\ &= \cos(2\pi(f_1 + kf_s)nT_s + \phi) \\ &= \cos(2\pi f_1 nT_s + 2\pi k f_s nT_s + \phi) \\ &= \cos(2\pi f_1 nT_s + 2\pi kn + \phi) \\ &= \cos(2\pi f_1 nT_s + \phi) \\ &= x[n] \end{aligned} \tag{10.6}$$

Thus, this higher frequency signal with frequency f_2 ends up looking identical to the signal with frequency f_1 after sampling, i.e., Eq. 10.4 equals Eq. 10.6. In a similar manner, all high-frequency terms greater than $f_{\max}$ would end up masquerading as lower frequencies less than $f_s/2$ when sampled at f_s, which is known as **aliasing**. Since we do not want this to happen, we need to ensure that the sampling theorem has been followed.

The illustration of sampling theorem in the frequency domain is shown in Fig. 10.1. The spectrum of the signal gets replicated in the frequency domain in multiples of $2\pi f_s$ and only one replica ($k = 1$) is shown in the figure. The sampling frequency f_s controls the spacing between the replicas, and for no overlap to occur, we can see from the figure that the following inequality needs to hold:

$$\begin{aligned} & 2\pi f_{\max} < 2\pi(f_s - f_{\max}) && \text{Radian Frequency} \\ \Longrightarrow\ & f_{\max} < f_s - f_{\max} && \text{Cyclic Frequency (Hz)} \\ \Longrightarrow\ & 2f_{\max} < f_s && \text{Sampling Theorem} \end{aligned}$$

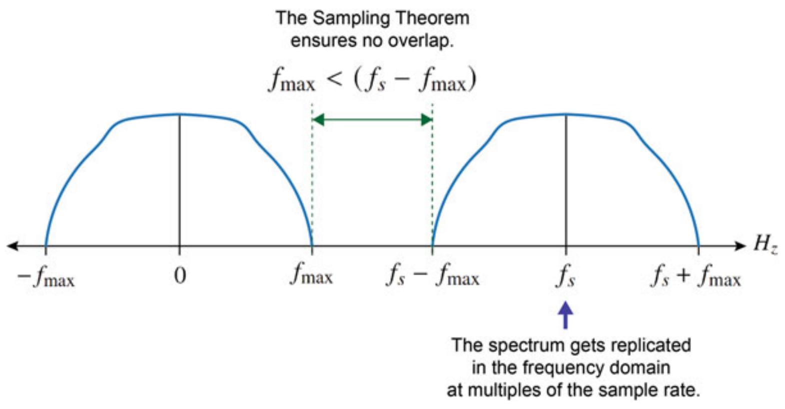
The Sampling Theorem ensures no overlap.
$f_{\max} < (f_s - f_{\max})$
H_z
$-f_{\max}$
0
$f_{\max}$
$f_s - f_{\max}$
f_s
$f_s + f_{\max}$
The spectrum gets replicated in the frequency domain at multiples of the sample rate.

Fig. 10.1: Illustration of the sampling theorem in the frequency domain. The spectrum of the sampled signal gets replicated at multiples of the sample rate f_s (i.e., kf_s and only $k = 1$ is shown in the figure). The sampling theorem $f_s > 2f_{\max}$ ensures that there will be no spectral overlap, i.e., aliasing

10.2 Fourier Series

In signal processing, one of the fundamental ideas is that we can represent signals such as speech by simple sinusoids and it does not matter if the speech signal is acoustic where people are talking to each other across a room or if the speech signal is converted to a digital representation and people are talking to each other across the country using their cell phones. We can represent any signal just by adding the appropriate number of sines and cosines together, each with their own amplitude, frequency, and phase shift. This is known as a Fourier series, which has the following mathematical form:

$$s(t) = \frac{a_0}{2} + \sum_{k=1}^{N} \left(a_k \cos\left(\tfrac{2\pi}{T}kt\right) + b_k \sin\left(\tfrac{2\pi}{T}kt\right)\right) \tag{10.7}$$

This is typically written in a complex exponential form using Euler's formula:

$$s(t) = \sum_{k=-N}^{N} c_k e^{j\frac{2\pi}{T}kt} \tag{10.8}$$

since

$$e^{j\theta} = \cos\theta + j\sin\theta \tag{10.9}$$

Let us show how this works with the following fairly complicated arbitrary piecewise waveform that has a period of $T = 5$ seconds. The waveform has three segments given by the following function and is shown in Fig. 10.2.

$$s(t) = \begin{cases} 3\sin\left(2\pi\frac{1}{6}t\right) + \frac{1}{2}\sin(2\pi 8t) & 0 \le t < 3 \\ e^{(\ln(1)-\ln(4))t+(4\ln(4)-3\ln(1))} & 3 \le t < 4 \\ 3 & 4 \le t < 5 \end{cases} \tag{10.10}$$

Let us determine the Fourier series coefficients c_k that will allow us to reconstruct this waveform using a summation of sinusoids. The Fourier coefficients are defined as

$$c_k = \begin{cases} \frac{1}{T}\int_0^T s(t)dt & k = 0\,(\text{DC term}) \\ \frac{2}{T}\int_0^T s(t)e^{-j\frac{2\pi}{T}kt}dt & k > 0 \end{cases} \tag{10.11}$$

We will use the fact that integration is a linear operator, which means we can break the waveform into separate segments and integrate each segment separately and integrate the terms within each segment separately. The first segment is the time interval $0 \le t < 3$ and there are two terms (two sinusoids with different frequencies

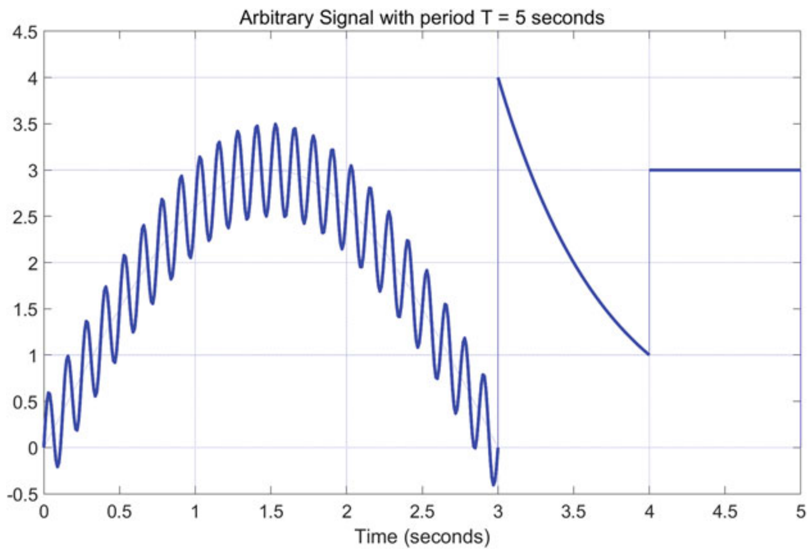

Fig. 10.2: An arbitrary signal given by Eq. 10.10

and amplitudes), so we will integrate each term separately over this interval. Thus the first term $3\sin\left(2\pi\frac{1}{6}t\right)$ in segment 1 has the DC coefficient:

$$c_0 = \frac{3}{5}\int_0^3 3\sin\left(\frac{\pi}{3}t\right)dt = \frac{3}{5}\left(\frac{-3}{\pi}\cos\left(\frac{\pi}{3}t\right)\Big|_0^3\right) \tag{10.12}$$

$$= \frac{-9}{5\pi}\left(\cos\left(\frac{3\pi}{3}\right) - \cos(0)\right) = \frac{-9}{5\pi}(-1-1) \tag{10.13}$$

$$= \frac{18}{5\pi} \tag{10.14}$$

Knowing that $\int e^{at}\sin(bt)dt = \frac{e^{at}}{a^2+b^2}\left[a\sin(bt) - b\cos(bt)\right]$, we get the rest of the coefficients for this first term:

$$c_k = \frac{6}{5}\int_0^3 3\sin\left(\frac{\pi}{3}t\right)e^{-j\frac{2\pi}{5}kt}\,dt \tag{10.15}$$

$$= \frac{6}{5}\left(\frac{1}{\frac{\pi^2}{9} + \left(\frac{-j2\pi k}{5}\right)^2}\right)\left[\left(\frac{-j2\pi k}{5}\sin\left(\frac{\pi}{3}t\right) - \frac{\pi}{3}\cos\left(\frac{\pi}{3}t\right)\right)e^{\frac{-j2\pi kt}{5}}\right]_0^3 \tag{10.16}$$

$$= \frac{6}{5}\left(\frac{1}{\frac{\pi^2}{9} - \frac{4\pi^2k^2}{25}}\right)\left[\frac{\pi}{3}e^{\frac{-j6\pi k}{5}} + \frac{\pi}{3}\right] \tag{10.17}$$

$$= \frac{2}{\frac{5\pi}{9} - \frac{4\pi^2k^2}{5}}\left(e^{\frac{-j6\pi k}{5}} + 1\right) \tag{10.18}$$

$$= \frac{90\left(1 + e^{-j\frac{6}{5}\pi k}\right)}{\pi\left(25 - 36k^2\right)} \tag{10.19}$$

Performing a similar integration for the second term $\frac{1}{2}\sin(2\pi 8t)$, we get the Fourier coefficients of

$$c_0 = 0 \tag{10.20}$$

$$c_k = \frac{20\left(1 - e^{-j\frac{6}{5}\pi k}\right)}{\pi\left(1600 - k^2\right)} \tag{10.21}$$

We are not done yet, because if we use this in Matlab, we will end up getting NANs (not a number) for the case when $k = 40$ because this results in $c_{40} = \frac{0}{0}$. For this case, we apply L'Hospital's rule:

$$\lim_{k\to 40} \frac{\frac{\partial}{\partial k}\left(20 - 20e^{-j\frac{6}{5}\pi k}\right)}{\frac{\partial}{\partial k}\left(1600\pi - \pi k^2\right)} = \lim_{k\to 40} \frac{24j\pi e^{-j\frac{6}{5}\pi k}}{-2\pi k} = \frac{-3j}{10} \tag{10.22}$$

Thus the coefficients that will give us the waveform in segment 1 ($0 \le t < 3$) are

$$c_k = \begin{cases} \frac{18}{5\pi} & k = 0 \text{ (DC term)} \\ \frac{90\left(1 + e^{-j\frac{6}{5}\pi k}\right)}{\pi\left(25 - 36k^2\right)} + \frac{20\left(1 - e^{-j\frac{6}{5}\pi k}\right)}{\pi\left(1600 - k^2\right)} & k > 0, k \ne 40 \\ \frac{90\left(1 + e^{-j\frac{6}{5}\pi k}\right)}{\pi\left(25 - 36k^2\right)} + \frac{-3j}{10} & k = 40 \end{cases} \tag{10.23}$$

We can check these coefficients by generating the waveform (blue curve) for segment 1 as shown in Fig. 10.3 where we use $N = 100$ coefficients and plot on top of the target waveform (green). Note that this curve is zero for the other two segments.

The waveform for segment 2 ($3 \le t < 4$) has the form $s(t) = e^{at+b}$, where a and b are chosen so that $s(t) = 4$ at $t = 3$ and $s(t) = 1$ at $t = 4$. This gives $a = \ln(1) - \ln(4)$ and $b = 4\ln(4) - 3\ln(1)$. The Fourier coefficients for segment 2 are calculated using Eq. 10.11 and are found to be

$$c_k = \begin{cases} \frac{e^{4a+b} - e^{3a+b}}{5a} & k = 0 \text{ (DC term)} \\ \frac{2}{5a - j2\pi k}\left(e^{4a+b-j\frac{8}{5}\pi k} - e^{3a+b-j\frac{6}{5}\pi k}\right) & k > 0 \end{cases} \tag{10.24}$$

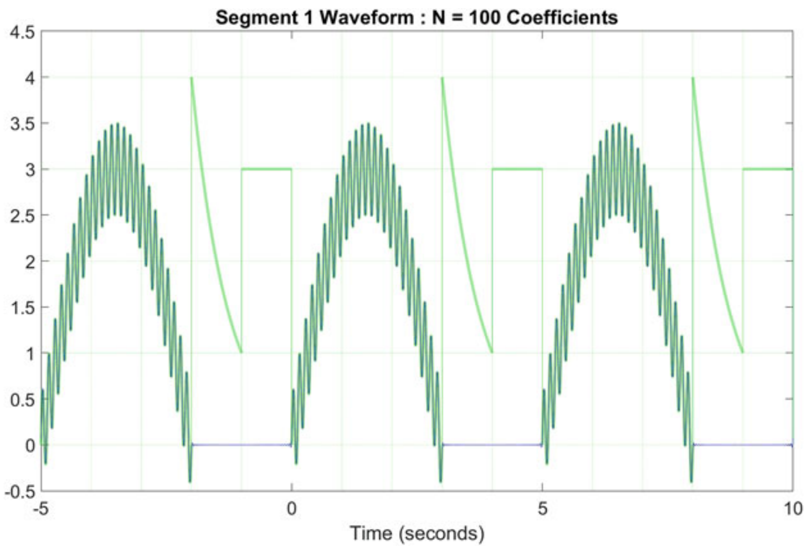

Fig. 10.3: Checking the coefficients for segment 1 ($0 \le t < 3$). The Fourier series is linear, so we can treat each segment independently

The waveform for segment 3 ($4 \le t < 5$) has a constant value of 3. This results in the following Fourier coefficients for segment 3:

$$c_k = \begin{cases} \dfrac{3}{5} & k = 0 \,(\text{DC term}) \\ \dfrac{-6}{j2\pi k}\left(1 - e^{-j\frac{8}{5}\pi k}\right) & k > 0 \end{cases} \tag{10.25}$$

The final waveform uses all the coefficients from all the segments and is the sum of the waveforms for each segment. This reconstruction is shown in Fig. 10.4.

The Matlab files to plot this waveform are listed in Table 10.1 and can be downloaded to create the waveform with a different number of coefficients. In Fig. 10.4 and in the subplot with $N = 500$ coefficients, one can see overshoots and ringing still happening at the discontinuities. This is known as the Gibbs phenomenon [1]. To get rid of the overshoots and ringing, one has to include a large number of coefficients, which tells us that discontinuities and sharp corners contain very high frequencies.

The function *sumexp.m* in Table 10.1 is done with one line of Matlab code as shown in Listing 10.1 and illustrates vectorized Matlab code in contrast to using slower *for* loops.

```
s=real(C(:)'*(exp(1j*2*pi*f(:)*[0:(1/fs):dur])));
```

Listing 10.1: Matlab: Summation of complex exponentials

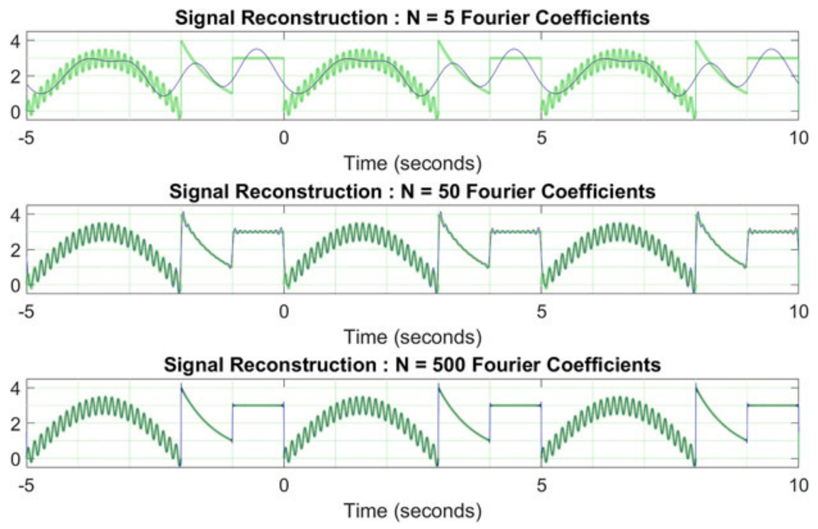

Fig. 10.4: Signal reconstruction using different numbers of Fourier coefficients ($N = 5, 50, 500$). As N increases, the signal converges to the "true" solution. The signal is periodic with period $T_0 = 5$ seconds and three periods are plotted in each case

Table 10.1: Matlab files used to plot arbitrary waveform in Fig. 10.4

File	Description	Link
fourier_series_Nvalues.m	Script that created Fig. 10.4	Click for file
fourier_series_waveform.m	Waveform function	Click for file
sumexp.m	Complex exponential summation	Click for file
fourier_series_target_template.m	Script to plot true waveform	Click for file

To interpret this line of code, we start with creating a row vector `[0:(1/fs):dur]` of time values that has a matrix dimension of $1 \times N_t$, where N_t is the number of samples (sample rate times time). We then take the vector of harmonic frequencies f and force it to be a column vector `f(:)` with dimension $N_f \times 1$, where N_f is the number of frequencies. Thus we do not care if f comes into the function as a row or column vector since we force it to be a column vector by using `f(:)`. We then multiple the column vector of frequencies by the row vector of time samples:

$$\underset{N_f \times 1}{f(:)} \; * \; \underset{1 \times N_t}{\text{[0:(1/fs):dur]}} = \underset{N_f \times N_t}{\text{f(:)*[0:(1/fs):dur]}} \tag{10.26}$$

This results in an outer product matrix $N_f \times N_t$ containing all combinations of frequencies and times. This matrix gets multiplied by the complex term $j * 2 * \pi$ and is the argument to Matlab's exp() function that results in a matrix of complex values of size $N_f \times N_t$.

We then take the vector of complex Fourier coefficients C that could be passed into the function as either a row or a column vector. It has the same number of elements as frequencies since this gives the amplitude and phase shift for each frequency (harmonic). To force it to be a row vector, we first force it to be a column vector $C(:)$ and then take its transpose $C(:)'$. We then have the product:

$$\underset{1\times N_f}{C(:)'} \;*\; \underset{N_f\times N_t}{\text{exp(1j*2*pi*f(:)*[0:(1/fs):dur])}} = \underset{1\times N_t}{\text{C(:)'*exp(1j*2*pi*f(:)*[0:(1/fs):dur])}} \tag{10.27}$$

which sums over all frequencies and we are left with the waveform as a function of time only. Finally, we use *real()* since we are only interested in the real part of the signal.

10.3 Geometric Interpretation of the Fourier Transform

If you came across the following equation for the *Fourier Transform* (**FT**), infrequently called the *Continuous Time Continuous Frequency Fourier Transform* (**CTCFFT**), could you picture in your head what this equation is doing?

$$\hat{s}(f) = \mathcal{F}\{s(t)\} = \int_{-\infty}^{\infty} s(t)e^{-j2\pi ft}\,dt \tag{10.28}$$

Let us break this apart, but first let us define what the variables are. The variable f stands for frequency with units of hertz (Hz) or cycles per second. The variable t stands for time with units of seconds. The grouping $2\pi f$ stands for angular frequency, often replaced by ω, and has units of radians per second. The variable j stands for the complex number $\sqrt{-1}$ (sometimes you will see the letter i instead of j).

The first step in breaking this apart is to use **Euler's formula**

$$e^{j\theta} = \cos(\theta) + j\sin(\theta) \tag{10.29}$$

which has the following geometric interpretation as seen in Fig. 10.5. The value $e^{j\theta}$ is a point on the unit circle that lives in the complex plane. The position of the point is determined by the angle θ and has the x coordinate value on the real axis of $\cos\theta$ and the y coordinate value on the imaginary axis of $\sin\theta$. We interpret the unit vector (radius one) that connects the origin with point $e^{j\theta}$ as being projected to both the real and imaginary axes using basic trigonometry. Note the circle is called the **Unit Circle** since it has a radius of 1 and is the range of $e^{j\theta}$. Thus Euler's formula maps all real values of θ to the unit circle and a point specified by θ has a projection to the real and imaginary axes. The real axis projection has the length of $\cos(\theta)$ and the imaginary axis projection has the length of $\sin(\theta)$.

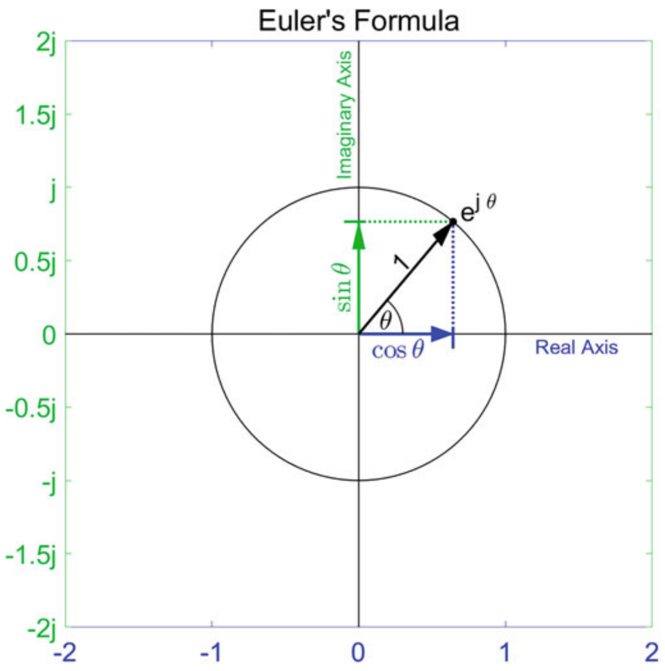

Fig. 10.5: Geometric interpretation of Euler's formula. We interpret the unit vector (radius one) that connects the origin with point $e^{j\theta}$ as being projected to both the real and imaginary axes using basic trigonometry

We can then interpret the term $s(t)e^{-j2\pi ft}$ as a vector with a time varying radius of $s(t)$ that is being projected to both the real and imaginary axes, while the Euler angle is also changing in time with the value of $\theta = 2\pi ft$. This causes the vector (whose radius is changing in time) to spin around the origin in the complex plane and this spinning vector is called a phasor. As we did for the unit vector in Euler's formula, we interpret the signal $s(t)$ as being projected to both the real and imaginary axes as illustrated in Fig. 10.6 where the projected signal length on the real axis is $s(t)\cos(2\pi ft)$ and the projected signal length on the imaginary axis is $s(t)\sin(2\pi ft)$.

Now, let us expand the Fourier Transform definition using Euler's formula:

$$\hat{s}(f) = \int_{-\infty}^{\infty} s(t)e^{-j2\pi ft}\,dt \tag{10.30}$$

$$= \int_{-\infty}^{\infty} s(t)[\cos(2\pi ft) + j\sin(2\pi ft)]\,dt \tag{10.31}$$

$$= \int_{-\infty}^{\infty} s(t)\cos(2\pi ft)\,dt + j\int_{-\infty}^{\infty} s(t)\sin(2\pi ft)\,dt \tag{10.32}$$

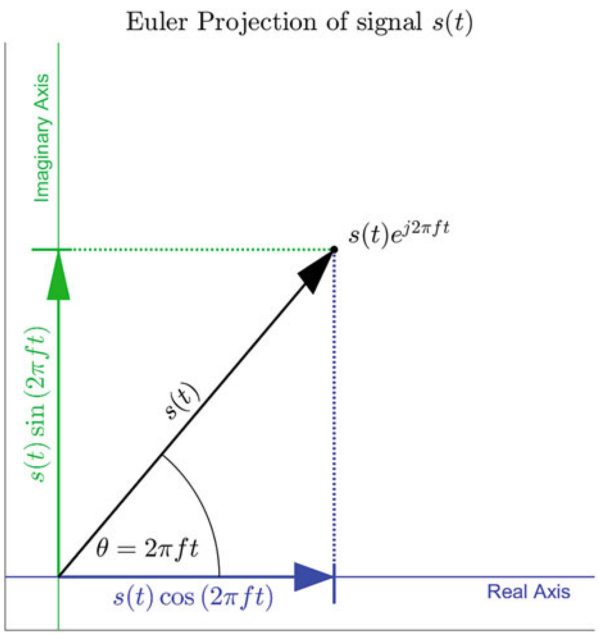

Fig. 10.6: Euler's formula projects the signal onto the real and imaginary axes. The signal $s(t)$ controls the vector length that varies in time along with the projection angle. The example shown here assumes that both $s(t)$ and cos() are positive. However, both terms can be both positive and negative and this will change the quadrant that the projection occurs in

Let us focus on the term $s(t)\cos(2\pi ft)$, which is the signal's projection to the real axis. For illustration purposes, we will use the speech signal shown as the blue line in Fig. 10.7. This will be the signal $s(t)$ being projected to the real axis by the cosine with time varying angle $\theta = 2\pi ft$ where $f = 112.51$ Hz (chosen to match the pitch of the speech signal well). This cosine with $f = 112.51$ Hz is shaded in a light green color that is behind the blue speech signal.

The product $s(t)\cos(2\pi ft)$, which is the projection to the real axis, is plotted in panel A of Fig. 10.8 as a function of time. The product $s(t)\sin(2\pi ft)$, which is the projection to the imaginary axis, is plotted in panel B as a function of time. The positive part of the product (as a function of time) is colored in blue and the negative part of the product is colored in red.

The first term in the Fourier integral

$$\int_{-\infty}^{\infty} s(t)\cos(2\pi ft)\,dt \tag{10.33}$$

is just the area under the curve in panel A of Fig. 10.8 where the positive areas (blue) add and the negative areas (red) subtract. In Matlab, this is just the summation

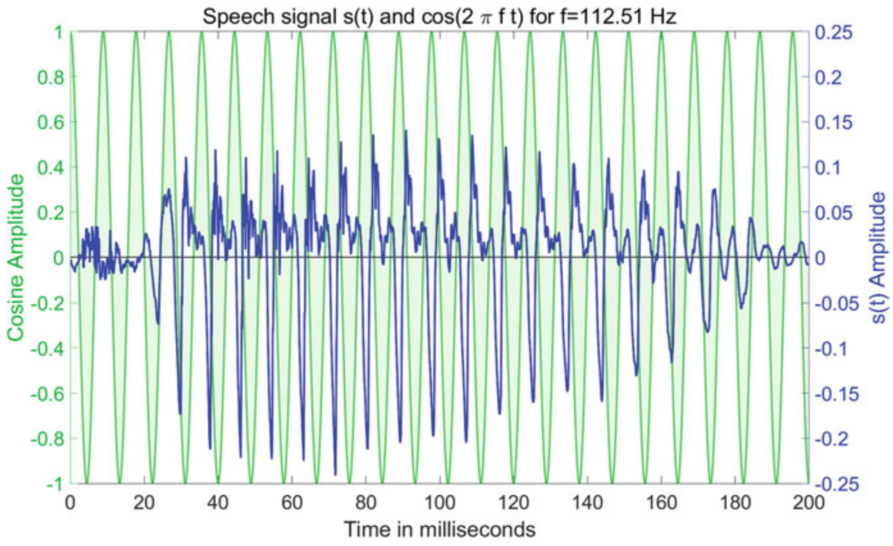

Fig. 10.7: Speech signal $s(t)$ used in Euler projection is the curve in blue. The cosine used for the projection is shaded in green. The cosine frequency of 112.51 Hz was chosen since it had the greatest Fourier Transform magnitude as can be seen in Fig. 10.10. The cosine frequency aligns well with the pitch frequency of the speech signal where most of the speech signal energy is contained

of the product vector $s(t)$.* $\cos(2\pi f t)$ over the interval shown, which results in the value -28.57. The signal goes significantly negative during the positive cycle of $\cos(2\pi 112.5t)$, so the area becomes significantly negative. This means that the signal matches this frequency well (the value would be positive if the negative part of the speech signal aligned with the negative cycle of the cosine). Similarly, the area for the imaginary axis project has a value of -7.88. Thus the Fourier Transform at the frequency $f = 112.5$ is $\hat{s}(f) = \hat{s}(112.5) = -28.57 - j7.88$, which has a magnitude of $m = \sqrt{(-28.57)^2 + (-7.88)^2} = 29.64$. This is the point of greatest magnitude marked in Fig. 10.10. Thus, when the signal oscillates at the same frequency as the cosine, the resulting area of the summation will be large (and can be either positive or negative), which tells us that there is a lot of signal energy at this frequency. Using both the sine and cosine tells us the phase shift of the signal relative to the sine and cosine functions, which is why the result is a complex number.

In contrast, an example of a frequency that does not match the speech signal well ($f = 636$ Hz) is shown in Fig. 10.9. At this projection frequency, the signal summations are close to zero (magnitude = 0.08), which is the low magnitude marked in Fig. 10.10. Notice that the positive and negative areas are similar and nearly symmetric. The cosine is going positive and negative, while the signal is not, which causes the product to be symmetric about zero. This tells us that the signal does not match this frequency well and thus does not have much signal energy at this frequency.

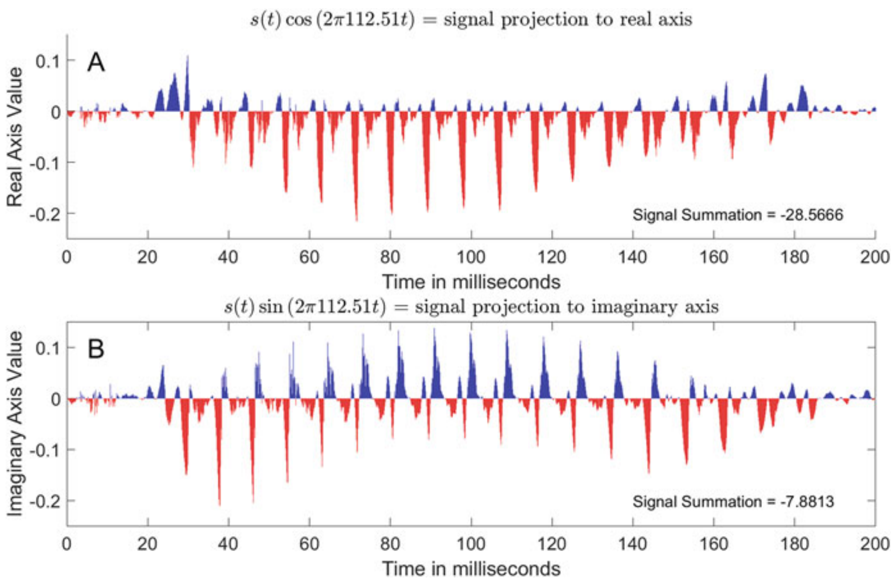

Fig. 10.8: In panel A, the speech signal $s(t)$ is projected to the real axis and the real axis value is plotted as a function of time. In panel B, the speech signal $s(t)$ is projected to the imaginary axis and the imaginary axis value is plotted as a function of time

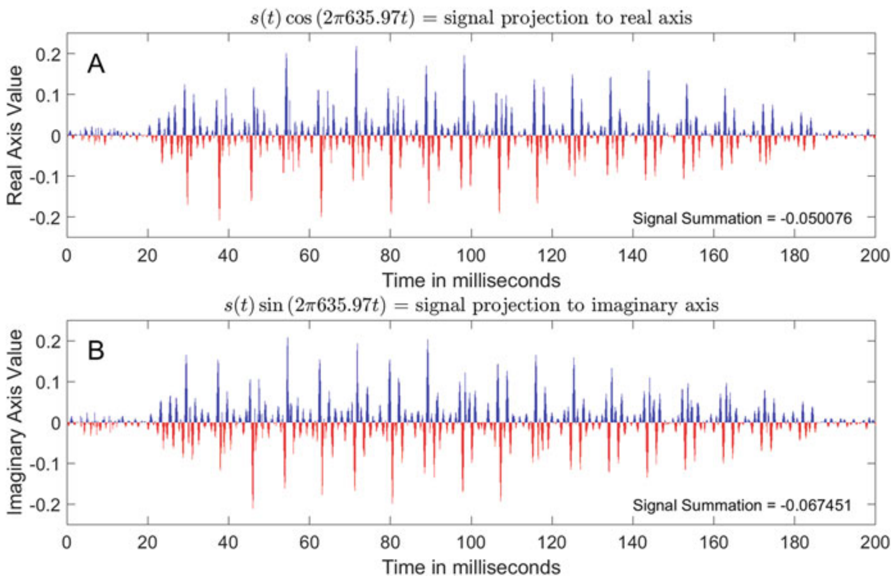

Fig. 10.9: Example where the frequency does not match the signal well. The positive area (blue) in this case is similar in area to the negative area (red)

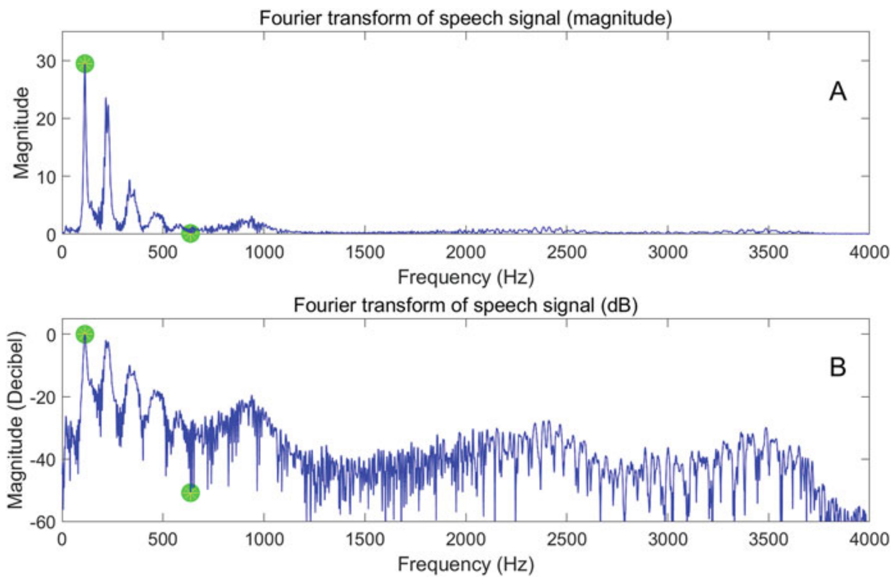

Fig. 10.10: Fourier transform of the speech signal from Fig. 10.7. Panel A shows the magnitude of the Fourier transform as a function of frequency. Plot B plots the normalized magnitude of the Fourier transform in $dB = 20\log_{10}(m/\max(m))$. The largest magnitude point in both A and B panels is at the frequency ($f = 112.51$ Hz), which is used in Figs. 10.7 and 10.8. The low magnitude point in both panels is at the frequency ($f = 636.97$ Hz), which is used in Fig. 10.9. Both these points are marked by green circles

10.4 The Fast Fourier Transform (FFT)

There are many reasons to use the Fourier Transform such as examining the frequency content of a signal or transforming convolution performed in the time domain to a much simpler multiply operation in the frequency domain. However we will ignore much of this mathematical infrastructure since there are many books devoted to this topic ([2, 3]). Rather, we will turn our attention to some practical considerations when using the FFT. We will use as our example a speech signal that has been sampled in time by an *analog-to-digital converter* (**ADC**).

Our mathematical starting point is the Fourier Transform (Eq. 10.28) that we restate here:

$$\hat{s}(f) = \mathcal{F}\{s(t)\} = \int_{-\infty}^{\infty} s(t)e^{-j2\pi ft}\,dt$$

Notice that the limits of integration are from $-\infty$ to $+\infty$. This is fine when we integrate continuous functions. However, when we use computers to compute the Fourier Transform, we immediately break this assumption since we do not have the time, memory, or patience to deal with infinities. Can we still use the Fourier Transform definition when we immediately modify the limits of integration? The general answer is that we cannot assume that it will give us valid answers. This would be like using an instrument outside of the manufacture's working specifications. Thus we need to know what this deviation from the definition is doing to our analysis. Modifying the limits of integration gets us into *windowing*, which is covered in Sect. 10.5.1 Windowing (page 178).

There are additional modifications that are made to the Fourier Transform as given in Eq. 10.28 to allow computers to implement the transform. These are:

Mod 1: Changing continuous time to discrete time. This is known as the *Discrete Time Continuous Frequency* (**DTCF**) **Fourier Transform** and is covered in Sect. 10.4.1 The Discrete Time Continuous Frequency Fourier Transform (page 173). This is useful when you want to determine the frequency content of a signal over an arbitrary frequency range (typically a small range) with arbitrary high precision.

Mod 2: Changing continuous frequency to discrete frequency (with discrete time). This is known as the *Discrete Time Discrete Frequency* (**DTDF**) Fourier Transform and is usually called the *Discrete Fourier Transform* (**DFT**), which is covered in Sect. 10.4.2 The Discrete Fourier Transform (page 174).

Mod 3: Speeding up the DFT, which is called the *Fast Fourier Transform* (**FFT**), which is covered in Sect. 10.4.3 FFT (page 176). This is the most commonly used form of the Fourier Transform when using computers.

10.4.1 The Discrete Time Continuous Frequency Fourier Transform

The first step toward using the Fourier Transform with computers is to use samples of a signal. This means that the time steps are discrete where they have been sampled at a particular time interval or sample rate. This changes the integral of the Fourier Transform definition to a summation and the signal samples are denoted $s[n]$. The Fourier Transform that uses discrete time samples is given as

$$\hat{s}(f) = \mathcal{F}_{dtcf}\{s[n]\} = \sum_{n=-\infty}^{\infty} s[n]e^{-j2\pi fn}$$

where $n \in \mathbb{Z}, f \in \mathbb{R}, \frac{-f_s}{2} \leq f \leq \frac{f_s}{2}$.

Notice that the limits of the summation are from $-\infty$ to $+\infty$, so we have not dealt with windowing yet (see Sect. 10.5.1 Windowing (page 178)). In this definition, we can use any real frequency value in the domain $\frac{-f_s}{2} \leq f \leq \frac{f_s}{2}$, which corresponds to the principal values of the sine/cosine functions ($-\pi \le \omega \le \pi$) where $\omega = 2\pi f$ and f_s is the sampling frequency.

This form is useful when you want to examine the frequency content of a signal within a specific frequency range and with arbitrary precision (limited by machine precision and data types used). This form was used to find the local frequency maximum and local frequency minimum with high precision (double precision) in Figs. 10.7, 10.8, 10.9 and 10.10. (click here for the source file of dtcfft.m)

10.4.2 The Discrete Fourier Transform

We converted time to discrete sample times in Sect. 10.4.1 The Discrete Time Continuous Frequency Fourier Transform (page 173) and we now need to convert the transform to discrete frequencies, so we can easily deal with them using computers. To do this, we sample the frequencies around the unit circle in the complex plane with uniform spacing. The full circle has 2π radians and we divide this into N intervals that gives us N frequencies (normalized). This gives the Discrete Fourier Transform:

$$\hat{s}[k] = \text{DFT}\{s[n]\} = \sum_{n=0}^{N-1} s[n]e^{-j\frac{2\pi}{N}kn} \quad \text{where} \quad k = 0, 1, 2, \ldots, N-1$$

The index n in the signal $s[n]$ is understood to represent samples in the signal that are spread apart in time by T_s seconds, where $Fs = 1/T_s$ is the sample rate in Hz. Furthermore, it is assumed that the maximum frequency content in the signal before sampling was less than $Fs/2$, which is why you always see anti-aliasing filters before analog-to-digital (ADC) converters (if you do not see them in the system, the hardware and signal are suspect). The index k in the spectrum $\hat{s}[k]$ represents the normalized radian frequency $\hat{\omega} = \frac{2\pi}{N}k$ (normalized to 2π), which means that regardless of FFT length N, the frequency sampling (evaluation) is done once around the unit circle. We can also normalize frequency f with respect to the sampling rate F_s, thus $\hat{\omega} = 2\pi\frac{f}{F_s}$. This means that $\frac{2\pi}{N}k = 2\pi\frac{f}{F_s}$ or $f = k\frac{F_s}{N}$, which is how you convert the DFT index k to frequency. **Note:** The DFT index k starts at zero, which is different from the Matlab FFT indexing that starts at one (see example in Table 10.3). A further wrinkle is that normalized radian frequencies $\pi < \hat{\omega} <= 2\pi$ (frequencies on the bottom half of the unit circle) actually represent negative radian frequencies since the principle argument for sinusoids must be in the interval $-\pi < \theta <= \pi$. Thus the normalized radian frequencies in the interval $\pi < \hat{\omega} <= 2\pi$ effectively have 2π subtracted from them resulting in negative frequencies.

In practice, the DFT is not used much because there is a much faster algorithm called the *Fast* Fourier Transform, or FFT (see Sect. 10.4.3). The computational cost for the DFT is $O(N^2)$ in contrast to the FFT's computational cost of $O(N \log N)$, which is significantly faster for larger FFT sizes as can be seen in Table 10.2.

Table 10.2: Computational cost of FFT vs DFT

Power of 2	FFT size	FFT	FFT	FFT	DFT	DFT	DFT	FFT
		Real ×'s	Real +'s		Real ×'s	Real +'s		Speedup
v	$N = 2^v$	$4N(\log_2(N) - 1)$	$2N \log_2(N)$	Real ops	$4N^2$	$4N^2 - 2N$	Real ops	
3	8	64	48	112	256	240	496	4.4
4	16	192	128	320	1024	992	2016	6.3
5	32	512	320	832	4096	4032	8128	9.8
6	64	1280	768	2048	16,384	16,256	32,640	15.9
7	128	3072	1792	4864	65,536	65,280	130,816	26.9
8	256	7168	4096	11,264	262,144	261,632	523,776	46.5
9	512	16,384	9216	25,600	1,048,576	1,047,552	2,096,128	81.9
10	1024	36,864	20,480	57,344	4,194,304	4,192,256	8,386,560	146.3
11	2048	81,920	45,056	126,976	16,777,216	16,773,120	33,550,336	264.2
12	4096	180,224	98,304	278,528	67,108,864	67,100,672	134,209,536	481.9
13	8192	393,216	212,992	606,208	268,435,456	268,419,072	536,854,528	885.6
14	16,384	851,968	458,752	1,310,720	1,073,741,824	1,073,709,056	2,147,450,880	1638.4
15	32,768	1,835,008	983,040	2,818,048	4,294,967,296	4,294,901,760	8,589,869,056	3048.2
16	65,536	3,932,160	2,097,152	6,029,312	17,179,869,184	17,179,738,112	34,359,607,296	5698.8

10.4.3 FFT

We will not get into the derivation of the FFT since there are good books on the subject ([4, 5]). Rather, we will look at how to use and interpret the FFT. The signal that we will examine is shown in Fig. 10.11 where the waveform is shown in the top plot. This signal (sampled at $F_s = 2000$ Hz) is zero for 0.5 seconds, a 10 Hz signal for 1 second, zero for 0.5 seconds, 100 Hz for 1 second, and then zero for 0.5 seconds. This results in a signal with 7002 samples. Since the FFT needs an input length that is a power of 2, we take an FFT of length $N = 8192$ where we add zeros to the end of the signal to make a signal with 8192 samples (known as zero padding). The output of the FFT is a vector of 8192 complex values, which is hard to plot. Since we are interested in the frequency content of the signal, we plot the magnitude, which is the middle plot of Fig. 10.11 where it has been plotted in decibels (i.e., Matlab command: m1 = 20*log10(abs(f1));) and where the peak dB value has been set to zero (i.e., Matlab command: m1 = m1-max(m1);). Setting the peak dB value to zero is typically done since we are usually more interested in the relative magnitudes of the frequencies in the signal than their absolute magnitudes. As an example in audio, we are typically more interested in how the audio sounds (harmonics, etc.) rather than how loud it is when it comes to viewing the audio spectrum.

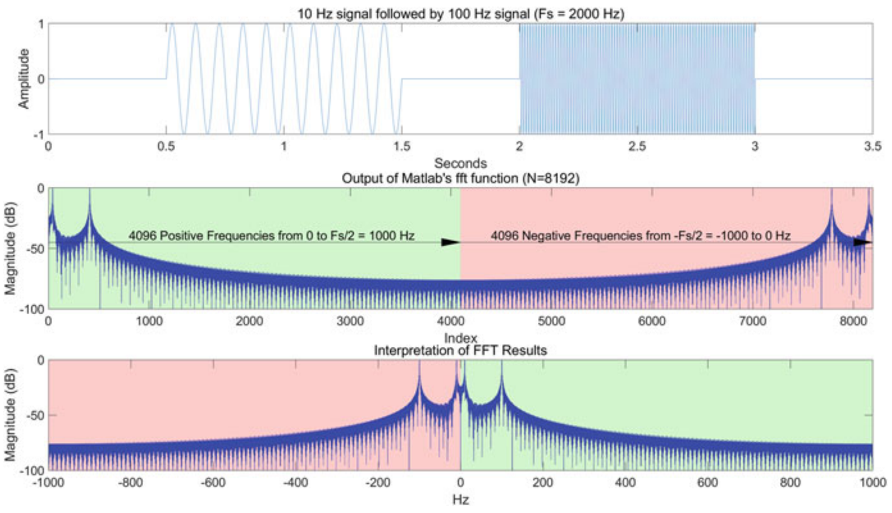

Fig. 10.11: Top figure: signal with 10 and 100 Hz components sample at $Fs = 2000$ Hz; middle figure: Output of Matlab's fft() function that puts the negative frequencies in the last half of the output vector; bottom figure: spectrum as expected with frequencies ordered on the real axis

One aspect to notice about the FFT result that is shown in the middle plot of Fig. 10.11 is the symmetry about the midpoint (DFT index k = 4096 or Matlab's

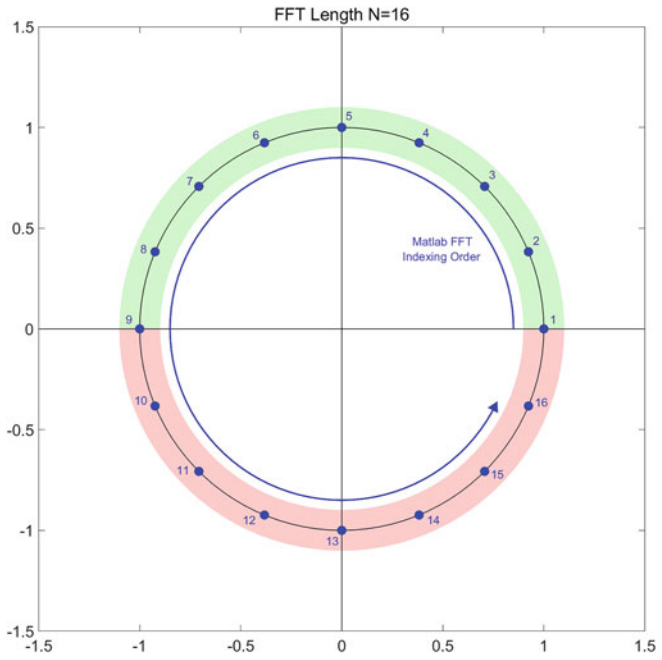

Fig. 10.12: Matlab's FFT indexing

index i = 4097), assuming that the FFT was taken of a signal with real (not complex) values. What can be confusing is that all the FFT values past the midpoint are associated with negative frequencies and you would expect to see the plot shown at the bottom of Fig. 10.11 with negative frequencies ordered as expected on the abscissa axis. Thus just plotting the FFT result will place the negative frequencies on the right side of the plot (red section) past the positive frequencies (green section). The reason this occurs is because the FFT normalized frequencies are evaluated around the unit circle from 0 to 2π as shown in Fig. 10.12 and the frequencies in the range $\pi < \hat{\omega} <= 2\pi$ (bottom half of the circle) are converted by the sinusoid functions to $-\pi < \hat{\omega} <= 0$ by effectively subtracting 2π due to the principle arguments of sinusoids being $-\pi <= \hat{\omega} <= \pi$. The associated DFT indexing and frequencies (assuming Fs = 2000) are listed in Table 10.3 for a 16-point FFT.

Due to the symmetry as seen in the middle plot of Fig. 10.11 when taking the FFT of real signals, typically only the first half of the FFT vector is plotted since it contains the positive frequencies. **Note:** If you are performing frequency domain processing of a real signal that involves taking the inverse FFT and you modify a positive frequency value by modifying either the magnitude or the phase, you also need to modify the associated negative frequency in the same manner, i.e., if you modify a Matlab FFT value at index i (DFT index k=i-1), you also need to modify the Matlab FFT value at index $j = N - i + 2$ (DFT index $j = N - k + 1$), where N is the FFT length.

Table 10.3: $N = 16$-point FFT indexing translations ($Fs = 2000$)

Matlab FFT index $i = 1 : N_{\text{FFT}}$	DFT index $k = i - 1$	Frequency (Hz) $f = k\frac{Fs}{N_{\text{FFT}}} = k\frac{2000}{16}$	Matlab conjugate frequency index $i_{\text{conj}} = N_{\text{FFT}} - k + 1 = N_{\text{FFT}} - i + 2$
1	0	0 (DC)	
2	1	125	16
3	2	250	15
4	3	375	14
5	4	500	13
6	5	625	12
7	6	750	11
8	7	875	10
9	8	1000 (Nyquist)	
10	9	−875	8
11	10	−750	7
12	11	−625	6
13	12	−500	5
14	13	−375	4
15	14	−250	3
16	15	−125	2

10.5 Practical Considerations When Using the FFT

10.5.1 Windowing

In the real-time analysis and synthesis FPGA example system in Sect. 3.1, one can see a "window" being applied in the Simulink model in Fig. 3.7 before being sent to the FFT engine. What window should be applied here? What would happen if you eliminated this windowing step? There are several reasons for performing this step.

To answer these questions, we first need to go back to the original definition of the Fourier transform, which we show here again:

$$\hat{s}(f) = \mathcal{F}\{s(t)\} = \int_{-\infty}^{\infty} s(t)e^{-j2\pi ft}\, dt \tag{10.34}$$

Notice the limits of integration in this definition. When we use computers, we do not have the time to start at time $t = -\infty$ and then wait until $t = +\infty$. Even if we could, we would not have the memory to be able to store a signal this long. So what happens if we have a signal that only lasts from time $t = t_1$ to time $t = t_2$ and is zero outside this time interval? Let us rewrite this as

$$\hat{s}(f) = \int_{-\infty}^{\infty} s(t)e^{-j2\pi ft}\,dt \tag{10.35}$$

$$= \int_{-\infty}^{t_1} s(t)e^{-j2\pi ft}\,dt + \int_{t_1}^{t_2} s(t)e^{-j2\pi ft}\,dt + \int_{t_2}^{\infty} s(t)e^{-j2\pi ft}\,dt \tag{10.36}$$

$$= 0 + \int_{t_1}^{t_2} s(t)e^{-j2\pi ft}\,dt + 0 \tag{10.37}$$

$$= \int_{t_1}^{t_2} s(t)e^{-j2\pi ft}\,dt \tag{10.38}$$

However, we should not take this approach since we are getting away from using the definition of the Fourier Transform. So instead of playing with the integration limits, let us leave them alone but modify our signal instead. To do this, let us define a new signal $w(t)$ as

$$w(t) = \begin{cases} 1 & \text{if } t_1 \leqslant t \leqslant t_2 \\ 0 & \text{otherwise} \end{cases} \tag{10.39}$$

and we can rewrite the Fourier Transform where we multiple by this windowing function to time limit our signal and not have to mess with the limits of integration. The signal is now zero outside the time interval $t_1 \leqslant t \leqslant t_2$.

$$\hat{s}(f) = \int_{-\infty}^{\infty} s(t)w(t)e^{-j2\pi ft}\,dt \tag{10.40}$$

This means that when we take the FFT of a signal using a computer, we are ***always*** applying a window function, even if we do not think we are. **Note: If you do not explicitly apply a window to your signal, you are in effect using a *rectangular window* as defined in Eq. 10.40**.

By using a finite signal, which we have to do when using computers, we have essentially applied a rectangular window to an infinite signal. So what does this do to our FFT results? The mathematical answer is the following where we generalize to any window function $w(t)$. If $\hat{s}(f)$ is the Fourier Transform of $s(t)$ as seen in Eq. 10.34 and $\hat{w}(f)$ is the Fourier Transform of $w(t)$, we take the inverse Fourier Transform as defined by

$$s(t) = \mathcal{F}^{-1}\{\hat{s}(f)\} = \frac{1}{2\pi}\int_{-\infty}^{\infty} \hat{s}(f)e^{j2\pi ft}\,df \tag{10.41}$$

Then the Fourier Transform of the windowed signal is

$$\mathcal{F}\{s(t)w(t)\} = \int_{-\infty}^{\infty} [s(t)w(t)]\, e^{-j2\pi f t}\, dt \tag{10.42}$$

$$= \int_{-\infty}^{\infty} \left[\frac{1}{2\pi}\int_{-\infty}^{\infty} \hat{s}(\zeta)e^{j2\pi\zeta t}\, d\zeta\right] w(t)e^{-j2\pi f t}\, dt \tag{10.43}$$

$$= \frac{1}{2\pi}\int_{-\infty}^{\infty} \hat{s}(\zeta)\Big[\int_{-\infty}^{\infty} w(t)e^{-j2\pi f t}e^{j2\pi\zeta t}\, dt\Big]\, d\zeta \tag{10.44}$$

$$= \frac{1}{2\pi}\int_{-\infty}^{\infty} \hat{s}(\zeta)\Big[\int_{-\infty}^{\infty} w(t)e^{-j2\pi(f-\zeta)t}\, dt\Big]\, d\zeta \tag{10.45}$$

$$= \frac{1}{2\pi}\int_{-\infty}^{\infty} \hat{s}(\zeta)\hat{w}(f-\zeta)\, d\zeta \tag{10.46}$$

$$= \frac{1}{2\pi}[\hat{s}(f) * \hat{w}(f)] \tag{10.47}$$

Thus multiplication of the signal and the window function in the time domain is equivalent to **convolution** in the frequency domain.

What does this mean in practice? Before we answer this, we first need to know what the Fourier transform of our rectangular window is. For convenience, we will define $t_1 = -\frac{T}{2}$ and $t_2 = -\frac{T}{2}$ for our rectangular window in Eq. 10.39.

$$\mathcal{F}\{w(t)\} = \hat{w}(f) = \int_{-\infty}^{\infty} w(t)e^{-j2\pi f t}\, dt \tag{10.48}$$

$$= \int_{-\frac{T}{2}}^{\frac{T}{2}} e^{-j2\pi f t}\, dt \tag{10.49}$$

$$= \frac{1}{-j2\pi f}\left[e^{-j2\pi f t}\Big|_{-\frac{T}{2}}^{\frac{T}{2}}\right] \tag{10.50}$$

$$= \frac{1}{-j2\pi f}\left[e^{-j\pi f T} - e^{j\pi f T}\right] \tag{10.51}$$

$$= \frac{T}{\pi f T}\left[\frac{e^{j\pi f T} - e^{-j\pi f T}}{2j}\right] \tag{10.52}$$

$$= \frac{T}{\pi f T}\sin(\pi f T) \tag{10.53}$$

$$= T\operatorname{sinc}(fT) \tag{10.54}$$

We can use the Matlab sinc() function to plot this function, which we do in Fig. 10.13 in the right column for two different values of time width T. The associated rectangular windows are shown on the left side and the associated Fourier transform on the right side. Note that the longer rectangular window in time (bottom left) results in narrower sinc function in frequency (bottom right). This means that to get a better frequency resolution in the frequency domain, a longer window of the signal in time needs to be taken.

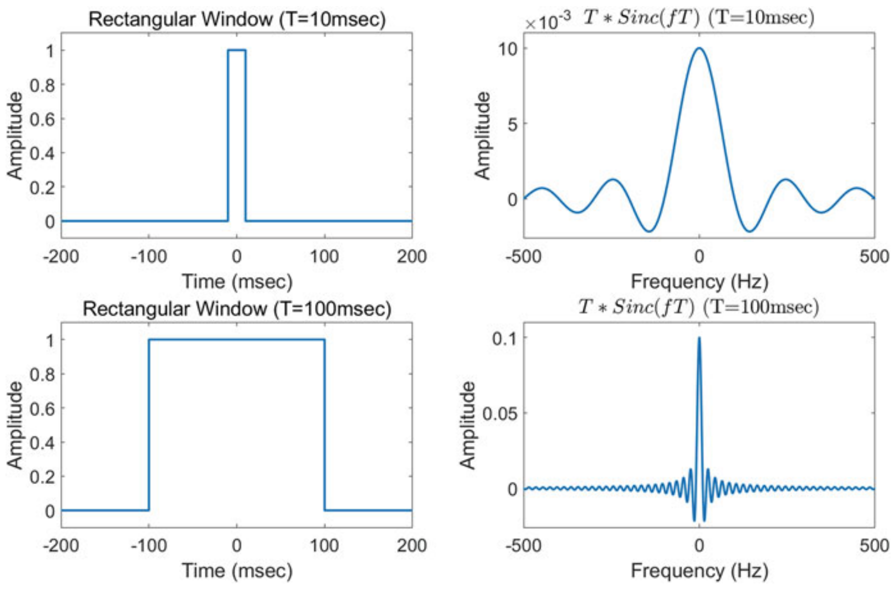

Fig. 10.13: The Fourier transform of a rectangular window is a sinc function. Longer windows in time result in narrower sinc functions in frequency

10.5.2 Window Length

In practice we can view the effect of the rectangular window (and other windows) as a blurring operation in the frequency domain where longer windows in time blur less in the frequency domain. Shorter windows blur more. To illustrate this, let us look at the spectrum of a cosine signal with frequency $f = 1000$ Hz where the Fourier transform has been compute of the cosine signal using rectangular windows of sizes [5 10 50 100 1000] msec as shown in Fig. 10.14. As we take longer rectangular windows of the cosine function in time, the resulting spectrum looks more and more like the line spectrum we expect for the cosine with $f = 1000$ Hz (shown as the green vertical line).

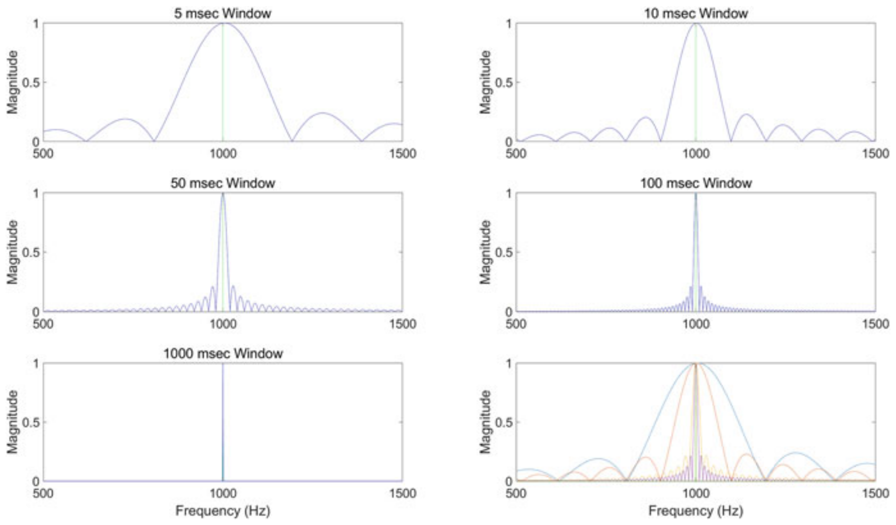

Fig. 10.14: Spectrum of a cosine signal with frequency $f = 1000$ Hz evaluated using windows of sizes [5 10 50 100 1000] msec ($Fs = 10{,}000$ Hz). The green vertical line is located at $f = 1000$ Hz. The last plot (bottom right) gives all the plots together for comparison purposes

Window length becomes important if we are trying to resolve frequencies that are close together. As an example, let us create a signal that is the sum of two cosines, one at $f1 = 1000$ Hz and the other at $f1 = 1050$ Hz. Now let us look at the resulting spectrum using windows of different lengths as shown in Fig. 10.15. With the 10 msec window, we do not even know that there are two frequencies. This is because the spectrum of the 10 msec window (shown in the upper right of Fig. 10.14) has been convolved with the two close frequencies and has blurred them together. As the windows get longer, the blurring from the associated sinc functions (convolution in the frequency domain) becomes less noticeable. Using a 1000 msec window results in hardly any blurring since the sinc function is very narrow relative to the separation (50 Hz) of the two frequencies.

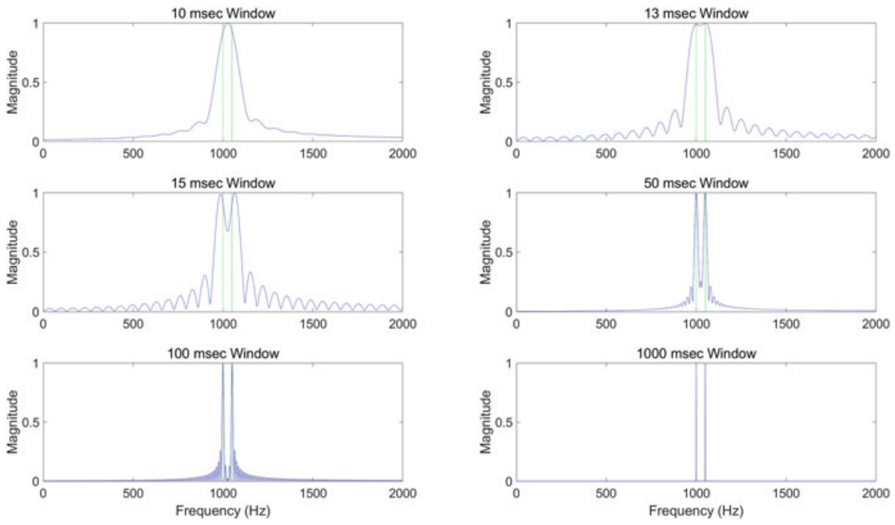

Fig. 10.15: Spectrum of a signal with two cosines with frequencies $f = 1000\,\text{Hz}$ and $f = 1050\,\text{Hz}$ evaluated using windows of sizes [10 13 15 50 100 1000] msec ($Fs = 10{,}000\,\text{Hz}$). If the window is too short (e.g., 10 msec window top left), the signal frequencies get blurred together and you cannot resolve them at all. The green vertical lines are located at $f = 1000\,\text{Hz}$ and $f = 1050\,\text{Hz}$

In summary, the longer the window is in time, the narrower the associated peak is in frequency. Thus if you are interested in good frequency resolution, use a longer window, which means using the FFT with more points. The trade-off is lower time resolution, since you do not know where the frequency occurs within a window, and longer latency for the FFT computation.

10.5.3 Window Edge Effects

If you look at the Fourier series example in Fig. 10.4, we determined what the Fourier series coefficients should be over the interval $0 \leq t \leq T = 5$ seconds. What happened outside of this interval? Since the sinusoids in the Fourier series are periodic, the signal represented by the Fourier coefficients becomes periodic with period T, as can be seen in the figure that shows three periods. Thus anytime we represent a signal in the frequency domain by taking the FFT, we have turned it into a periodic signal in time where it continuously repeats outside the window for all time. This is a subtle point since we typically expect the signal to remain the same when we extract a portion of it and do not realize that by extracting it (i.e., windowing) and taking the FFT, we have caused it to repeat continuously in time. This is because we are only paying attention to the time interval where we extracted the signal and do not consider what happened in time outside this interval. Keep this in mind since this needs to be combined with the following point. If a signal has discontinuities in it, similar to the discontinuities in the signal example in Fig. 10.4, it will take many harmonic terms to model the signal well. This means that discontinuities are associated with high frequencies.

What this means in practice when we used FFTs is that we need to be careful at the edges of the signal that we are windowing. This is because the FFT will cause our signal to repeat outside of our window in time and if there are discontinuities at the windowed edges, the FFT will add high frequencies to model the discontinuities, even if these frequencies are not present in the original signal.

This is illustrated in Fig. 10.16 where we start with a 64 Hz cosine (top left). We pick the frequency of the cosine and FFT length ($N = 1024$) such that when the FFT causes the signal to repeat outside of the $N = 1024$ rectangular window, the period of the 64 Hz cosine aligns well with the effective periodization. This is shown in the top center plot where the left side (in red) is at the end of the signal window and the green segment on the right side is where the signal has been effectively replicated in time. The FFT spectrum in this case is shown (top right). The rectangular window used had $N = 1024$ points, which resulted in a very narrow sinc spectrum in the frequency domain. The noise floor of the FFT spectrum is very low (−314 dB) because of the cosine that aligned perfectly with the window edges as it was replicated in time. This perfect alignment rarely happens in practice except in contrived cases like this one shown.

What typically happens is that the frequency components do not align nicely with the window replication. This is modeled in the example shown in the middle row of Fig. 10.16. Here we illustrate a typical practice of zero padding in order to get a power of two length that we want when applying the FFT. In this example we just set the last three points to zero. This causes a discontinuity to occur when the FFT replicates the signal (middle plot). The left side (in red) is at the end of the signal window and the green segment on the right side is where the signal has been effectively replicated in time. The FFT spectrum is now significantly different where the noise floor changed from −314 dB (shown as the green signal) to −55 dB (in blue). The discontinuity at the window edge has significantly changed the spectrum.

To get rid of discontinuities at the window edges, we can no longer use a rectangular window. We need a window that squashes the ends down to zero so that when the signal is replicated by the FFT, there are no discontinuities. A commonly used window that does this is is the **Hanning** (or Hann) window. A Hanning window ($N = 1024$) applied to the 64 Hz cosine signal with the edge discontinuity is shown in the bottom left plot in Fig. 10.16. We can see that there are no discontinuities at the window edges (middle plot). This improves the spectrum representation where the FFT floor drops from −55 dB (green) to −140 dB (blue) as seen in the bottom right plot. The −314 dB noise floor is also plotted in green. Note that the spectrum of the Hanning window has been convolved with the cosine line spectrum, so the Hanning spectrum can be seen in the blue spectrum curve. The Hanning window is only one of many windows that we can use, which gets us into the next section on window trade-offs.

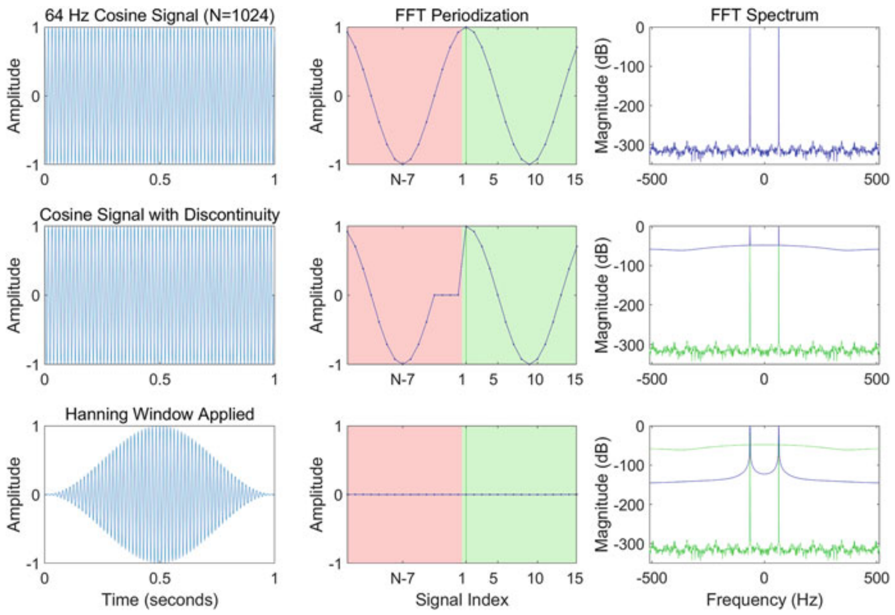

Fig. 10.16: Window edge effects

10.5.4 Window Trade-Offs

When you use the FFT, you are always using a window. If you have not explicitly applied a window, you have used the rectangular window, which means that you have convolved the spectrum of the rectangular window (sinc function) with the signal's spectrum (a blurring operation). In the previous section we discussed why using a Hanning window is better than using a rectangular window to eliminate edge discontinuities. A question you probably have is why a Hanning window? Are there other windows that could be used? The answer is that there are many other windows that can be used and this gets us into window trade-offs.

Let us look at the spectrum of the rectangular window (sinc function) shown in Fig. 10.17. Note that there are two parameters associated with this window spectrum (and all window spectrums). The width of the main-lobe (width at half-height or width at the -3 dB point) and how high the side-lobes are relative the height of the main-lobe. If we want good frequency resolution, we want the main-lobe to be as narrow as possible. An easy way to control the width of the main-lobe is to use longer windows in time. However, we can also affect the main-lobe width just by the choice of window where the windows have the same length in time. This is where the window trade-offs come into play. Choosing a window that has a narrower main-lobe width typically causes the side-lobe attenuation to be reduced (side-lobe peaks become more pronounced). Why is this an issue? Remember that this window spectrum gets convolved with the signal's spectrum. This means that energies associated with other frequencies will "leak" into other frequency locations, affecting the fidelity of energy measurement for frequencies of interest (this is known as "leakage"). Thus the window used is a design trade-off that depends on the application. If you are more interested in what frequencies are present and being able to resolve frequencies, choose a window with a narrower main-lobe. If you are wanting to measure the power that exists at a particular frequency, use a window that has good side-lobe attenuation, so energy from other frequencies does not leak into your measurement.

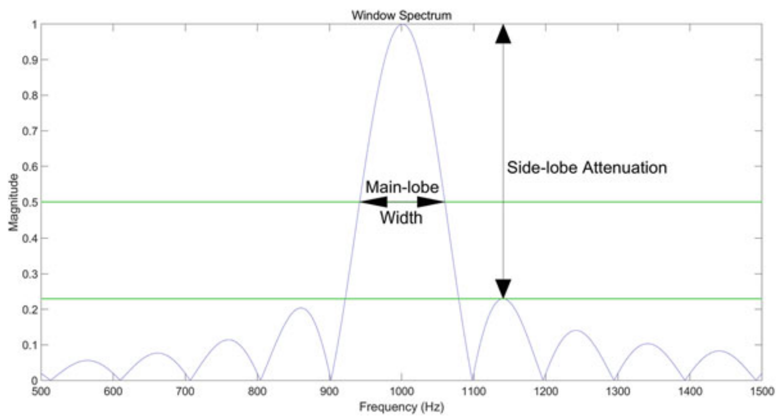

Fig. 10.17: Window trade-offs

The trade-off between main-lobe width and side-lobe attenuation can be seen in Fig. 10.18 for all the windows listed in Tables 10.4 and 10.5 . All the windows except for the rectangular window squash the window edges to zero in time to eliminate discontinuities. This is why the rectangular window is rarely used (except if you forget to apply a window). The Hanning (or Hann) window (blue circle #18) is typically chosen if you do not know what window to apply since it is a good balance between main-lobe width and side-lobe attenuation.

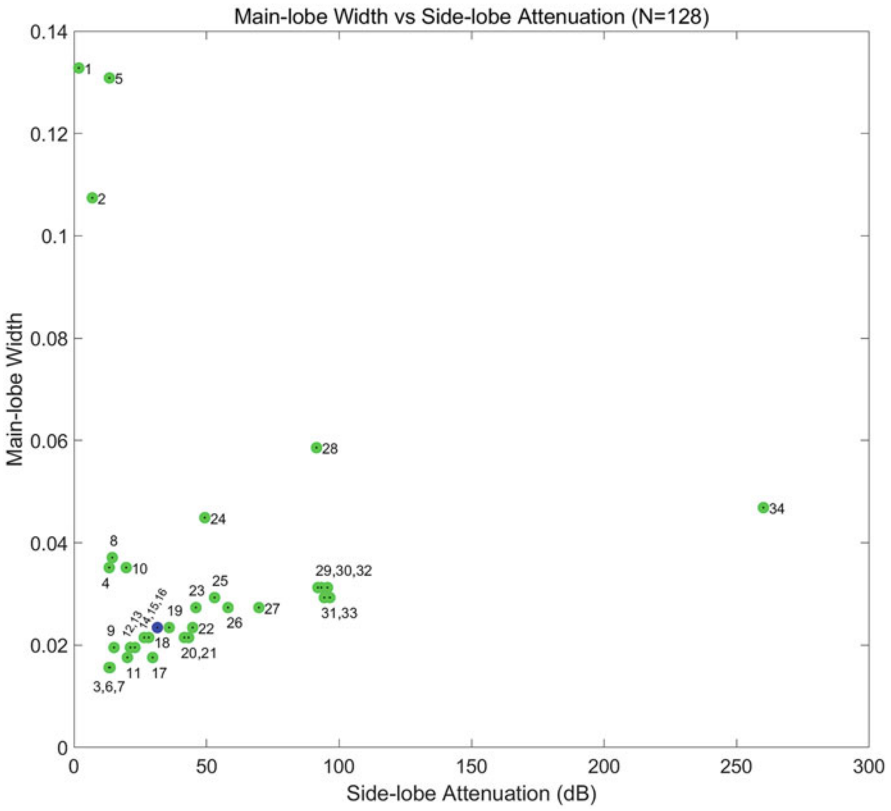

Fig. 10.18: Main-lobe width vs side-lobe attenuation for the windows listed in Tables 10.4 and 10.5. The commonly used Hann (or Hanning) window (number 18 in figure and colored blue) is a good compromise between main-lobe width and side-lobe attenuation. Windows were computed from Joe Henning's window utilities [6]

Table 10.4: Windows associated with Fig. 10.18. Windows sorted according to side-lobe attenuation. Windows computed from [6]

Window number	Window name	Side-lobe attenuation	Main-lobe width
1	Ultraspherical (0,0.5)	1.8255	0.13281
2	Ultraspherical (0.5,0.5)	6.9263	0.10742
3	Rectangular	13.2619	0.015625
4	Generalized Normal (P = 50)	13.3164	0.035156
5	Ultraspherical (−0.5,0.5)	13.3978	0.13086
6	Planck-taper (0.1)	13.3994	0.015625
7	Kaiser	13.6187	0.015625
8	Generalized Normal (P = 10)	14.4567	0.037109
9	Tukey	15.1229	0.019531
10	Generalized Normal (P = 4)	19.724	0.035156
11	Exponential (tau = 64)	20.158	0.017578
12	Welch	21.3084	0.019531
13	Sine	23.0021	0.019531
14	Triangular	26.5129	0.021484
15	Bartlett	26.5129	0.021484
16	Planck-Bessel (0.1, 4.45)	28.2264	0.021484
17	Taylor	29.6914	0.017578
18	Hann	31.4755	0.023438
19	Bartlett–Hann	35.9391	0.023438
20	Hamming	41.6395	0.021484
21	Gaussian (sigma = 0.4)	43.2658	0.021484
22	Slepian (alpha = 2)	44.7997	0.023438
23	Bohman	46.0164	0.027344
24	Exponential (tau = 9.2693)	49.3683	0.044922
25	Parzen	53.0471	0.029297
26	Blackman	58.1236	0.027344
27	Slepian (alpha = 3)	69.7603	0.027344
28	Flat Top	91.5097	0.058594
29	Blackman–Harris	92.0377	0.03125
30	Nuttall	93.3288	0.03125
31	Dolph–Chebyshev	94.4534	0.029297
32	Slepian (alpha = 4)	95.6357	0.03125
33	Blackman–Nuttall	96.5229	0.029297
34	Slepian (alpha = 10)	260.1093	0.046875

Table 10.5: Windows associated with Fig. 10.18. Windows sorted according to main-lobe width. Windows computed from [6]

Window number	Window name	Side-lobe attenuation	Main-lobe width
3	Rectangular	13.2619	0.015625
6	Planck-taper (0.1)	13.3994	0.015625
7	Kaiser	13.6187	0.015625
11	Exponential (tau = 64)	20.158	0.017578
17	Taylor	29.6914	0.017578
9	Tukey	15.1229	0.019531
12	Welch	21.3084	0.019531
13	Sine	23.0021	0.019531
14	Triangular	26.5129	0.021484
15	Bartlett	26.5129	0.021484
16	Planck-Bessel (0.1, 4.45)	28.2264	0.021484
20	Hamming	41.6395	0.021484
21	Gaussian (sigma = 0.4)	43.2658	0.021484
18	Hann	31.4755	0.023438
19	Bartlett–Hann	35.9391	0.023438
22	Slepian (alpha = 2)	44.7997	0.023438
23	Bohman	46.0164	0.027344
26	Blackman	58.1236	0.027344
27	Slepian (alpha = 3)	69.7603	0.027344
25	Parzen	53.0471	0.029297
31	Dolph–Chebyshev	94.4534	0.029297
33	Blackman–Nuttall	96.5229	0.029297
29	Blackman–Harris	92.0377	0.03125
30	Nuttall	93.3288	0.03125
32	Slepian (alpha = 4)	95.6357	0.03125
4	Generalized Normal (P = 50)	13.3164	0.035156
10	Generalized Normal (P = 4)	19.724	0.035156
8	Generalized Normal (P = 10)	14.4567	0.037109
24	Exponential (tau = 9.2693)	49.3683	0.044922
34	Slepian (alpha = 10)	260.1093	0.046875
28	Flat Top	91.5097	0.058594
2	Ultraspherical (0.5,0.5)	6.9263	0.10742
5	Ultraspherical (−0.5,0.5)	13.3978	0.13086
1	Ultraspherical (0,0.5)	1.8255	0.13281

References

1. D. Gottlieb, C.-W. Shu, On the Gibbs phenomenon and its resolution. SIAM Rev. **39**(4), 644–668 (1997)
2. A.V. Oppenheim, R.W. Schafer, *Digital Signal Processing* (Prentice-Hall, Hoboken, 1975)

3. J.G. Proakis, *Digital Signal Processing: Principles Algorithms and Applications* (Pearson, London, 2001)
4. R.N. Bracewell, *The Fourier Transform and Its Applications* (McGraw-Hill, New York, 1986)
5. E.O. Brigham. *The Fast Fourier Transform and Its Applications* (Prentice-Hall, Hoboken, 1988)
6. J. Henning, Window Utilities. MATLAB Central File Exchange https://www.mathworks.com/matlabcentral/fileexchange/46092-windowutilities. Accessed 18 Jun 2021