Inverse Discrete Fourier transform (DFT)

Alejandro Ribeiro

January 29, 2021

Suppose that we are given the discrete Fourier transform (DFT) $X : \mathbb{Z} \to \mathbb{C}$ of an unknown signal. The inverse (i)DFT of X is defined as the signal $x : [0, N-1] \to \mathbb{C}$ with components $x(n)$ given by the expression

$$x(n) := \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} X(k) e^{j2\pi kn/N} = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} X(k) \exp(j2\pi kn/N) \tag{1}$$

When x is obtained from X through the relationship in (1) we write $x = \mathcal{F}^{-1}(X)$. Recall that if X is the DFT of some signal, it must be periodic with period N. That means that in (1) we can replace the sum over the frequencies $k \in [0, N-1]$ with a sum over any other set of N consecutive frequencies. In particular, the iDFT of X can be alternatively written as

$$x(n) = \frac{1}{\sqrt{N}} \sum_{k=-N/2+1}^{N/2} X(k) e^{j2\pi kn/N} \tag{2}$$

To see that (2) is correct, it suffices to note that $X(k+N) = X(k)$ and that $e^{j2\pi(k+N)n/N} = e^{j2\pi kn/N}$ to conclude that all of the terms that appear in (1) are equivalent to one, and only one, of the terms that appear in (2).

It is not difficult to see that taking the iDFT of the DFT of a signal x recovers the original signal x. This means that the iDFT is, as its names indicates, the inverse operation to the DFT. This result is of sufficient importance to be highlighted in the form of a theorem that we state next.

Theorem 1 *Given a discrete signal $x : [0, N-1] \to \mathbb{C}$, let $X = \mathcal{F}(x) : \mathbb{Z} \to \mathbb{C}$ stand in for the DFT of x and $\tilde{x} = \mathcal{F}^{-1}(X) : [0, N-1] \to \mathbb{C}$ be the iDFT of X. We then have that $x \equiv \tilde{x}$, or, equivalently,*

$$\mathcal{F}^{-1}[\mathcal{F}(x)] = x. \tag{3}$$

Proof: Write down a proof of Theorem 1. ■

The result in Theorem 1 is important because it tells us that a signal x can be recovered from its DFT X by taking the inverse DFT. This implies that x and X are alternative representations of the same information because we can move from one to the other using the DFT and iDFT operations. If we are given x we can compute X through the DFT, and if we are given X we can compute x through the iDFT.

An important practical consequence of this equivalence is that if we are given one of the representations, say the signal x, and the other one is easier to interpret, say the DFT X, we can compute the respective transform and proceed with the analysis. This analysis will neither introduce spurious effect, nor miss important features. Since both representations are equivalent, it is just a matter of which of the representations makes the identification of patterns easier. There is substantial empirical evidence that it is easier to analyze signals in the frequency domain—i.e., the DFT X—than it is to analyze signals in the time domain—the original signal, x.

1 Signal reconstruction and compression

A more mathematical consequence of Theorem 1 is that any signal x can be written as a sum of complex exponentials. To see that this is true, we just need to reinterpret the equations for the DFT and iDFT. In this reinterpretation, the components of the signal x can be written as [cf. (1) and (2)]

$$x(n) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} X(k) e^{j2\pi kn/N} = \frac{1}{\sqrt{N}} \sum_{k=-N/2+1}^{N/2} X(k) e^{j2\pi kn/N} \tag{4}$$

with coefficients $X(k)$ that are given by the formula [cf. equation (1) in lab assignment 2]

$$X(k) := \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N} \tag{5}$$

This is quite a remarkable fact. We may have a signal that doesn't look at all like an oscillation, but it is a consequence of Theorem 1 that such signal can be written as a sum of oscillations.

It is instructive to rewrite (4) in an expanded form that makes the latter observation clearer. To do so, consider the rightmost expression. Write the N summands explicitly and reorder the terms so that the terms corresponding to positive frequency k and its opposite frequency $-k$ appear together. Doing so and noting that frequencies $k = 0$ and $k = N/2$ have no corresponding opposites, it follows that (4) is equivalent to

$$\begin{aligned}\left(\sqrt{N}\right) x(n) = \quad & X(0) && e^{j2\pi 0n/N} \\ & + X(1) && e^{j2\pi 1n/N} && + X(-1) && e^{-j2\pi 1n/N} \\ & + X(2) && e^{j2\pi 2n/N} && + X(-2) && e^{-j2\pi 2n/N} \\ & \vdots && \vdots && \vdots && \vdots \\ & + X\left(\frac{N}{2}-1\right) && e^{j2\pi\left(\frac{N}{2}-1\right)n/N} && + X\left(-\frac{N}{2}+1\right) && e^{-j2\pi\left(\frac{N}{2}-1\right)n/N} \\ & + X\left(\frac{N}{2}\right) && e^{j2\pi\left(\frac{N}{2}\right)n/N} \end{aligned} \tag{6}$$

where we have multiplied both sides of the equality by $\sqrt{N}$ to simplify the expression. Observe that the term that corresponds to frequency $k = 0$ is simply $X(0)e^{j2\pi 0n/N} = X(0)$. We write the exponential part of this factor to avoid breaking the symmetry of the expression.

We can interpret (6) as a set of successive approximations of $x(n)$ that introduce ever finer details in the form of faster signal variations. I.e., we can choose to approximate the signal x by the signal $\tilde{x}_K$ which we define by truncating the DFT sum to the first K terms in (6),

$$\tilde{x}_K(n) := \frac{1}{\sqrt{N}}\left[X(0) + \sum_{k=1}^{K}\left(X(k)e^{j2\pi kn/N} + X(-k)e^{-j2\pi kn/N}\right)\right]. \tag{7}$$

The approximation that uses $k = 0$ only, is a constant approximation of the signal x. The approximation that uses $k = 0$ and $k = \pm 1$ approximates x with a constant and a single oscillation, the approximation that adds $k = \pm 2$, refines the signal by adding finer details in the form of a (more rapid) double oscillation. In general, when adding the kth frequency and its opposite $-k$, we add an oscillation of frequency k that makes the approximation closer to the actual signal. If we have a signal that varies slowly, a representation with just a few coefficients is sufficient. For signals that vary faster, we need to add more coefficients to obtain a reasonable approximation.

Alternatively, if only gross details are important, we can eliminate the finer irrelevant features by studying the approximated signal instead of the original signal. This observation is related to our digression on the empirical value of the DFT as a tool for pattern identification. The representation of x as a sum of complex exponentials facilitates identification of relevant time features that tend to correspond to variations that are slower than patterns. E.g., weather varies from day to day, but there is an underlying slower pattern that we call climate. Weather will manifest in the DFT coefficients for large frequencies and climate in the DFT coefficients associated with slower frequencies. We can study climate by reconstructing a weather signal x with a small number of DFT coefficients.

In this part of the lab we will study the quality of the reconstruction of x with approximating signals $\tilde{x}_K$ as we increase K.

1.1 Computation of the iDFT. Consider a DFT X corresponding to a real signal of even duration N and assume that we are given the $N/2+1$ coefficients corresponding to frequencies $k = 0, 1, \ldots, N/2$. Create a Python class that takes these $N/2$ coefficients as input, as well as the associated sampling frequency f_s, and returns the iDFT $x = \mathcal{F}^{-1}(X)$ of the given X. Return also a vector of real times associated with the signal samples.

1.2 Signal reconstruction. Suppose now that we are given the first $K+1$ coefficients of the DFT of a signal of duration N. Create a Python class that returns the approximated signal $\tilde{x}_K$ with elements $\tilde{x}_K(n)$ as given in (7). The inputs to this class include the $K+1$ coefficients, the signal duration N, and the sampling frequency f_s. Return also a vector of real times associated with the signal samples. *Hint: You can use what you solved in Part 1.1 to help solve this part.*

1.3 Reconstruction of a square pulse. Generate a pulse[1] of duration $T = 32$s sampled at a rate $f_s = 8$Hz and length $T_0 = 4$s and compute its DFT[2]. Use the class in Part 1.2 to create successive reconstructions of the pulse. Compute the energy of the difference between the signals x and $\tilde{x}_K$. This energy should decrease for increasing K. Report your results for $K = 2$, $K = 4$, $K = 8$, and $K = 16$ $K = 32$. Repeat for a pulse of length $T_0 = 2$s. Since this pulse varies faster, the reconstruction should be worse. Is that the case?

[1]It is recommended that you use the class `sqpulse()` provided on the website. Remember to type `help(sqpulse)` in the console to learn how to use the class.

[2]Use of the class `dft()` provided on the course website (Lab2) is recommended. Please, remember to type `help(dft)` in the console to learn how to use the class

1.4 Reconstruction of a triangular pulse. Generate a triangular pulse[3] of duration $T = 32$s sampled at a rate $f_s = 8$Hz and length $T_0 = 4$s and compute its DFT. Use the class in Part 1.2 to create successive reconstructions of the pulse. Compute the energy of the difference between the signals x and $\tilde{x}_K$. Report your results for $K = 2$, $K = 4$, $K = 8$, and $K = 16$ $K = 32$. This pulse should be easier to reconstruct than the square pulse. Is that true?

1.5 The energy of the difference signal. In parts 1.3 and 1.4 you have computed the energy of the difference between the signals x and $\tilde{x}_K$. Just to be formal, define the error signal ρ_K as the one with components $\rho_K(n) = x(n) - \tilde{x}_K(n)$. The energy you have computing is therefore given by

$$\|\rho_K\|^2 = \sum_{n=0}^{N-1} \left|\rho_K(n)\right|^2 = \sum_{n=0}^{N-1} \left|x(n) - \tilde{x}_K(n)\right|^2. \tag{8}$$

Using Parseval's theorem, this energy can be computed from the values of the DFT coefficients that you are neglecting to include in the signal approximation. Explain how this can be done, and verify that your numerical results coincide.

A square wave can be visualized as a train of square pulses pasted next to each other. Mathematically, it is easier to generate a square wave by simply taking the sign of a discrete cosine. Consider then a given frequency f_0 and a given sampling frequency f_s and define the square wave of frequency f_0 as the signal

$$x(n) \;=\; \text{sign}\left[\cos\Big(2\pi(f_0/f_s)n\Big)\right]. \tag{9}$$

This signal can be reconstructed with a few DFT coefficients, but not with the first K. To compress this signal well, we pick the K largest DFT coefficients, which are not necessarily the first K. When reconstructing the signal, we use a modified version of (7) in which we sum over the coefficients that were picked during the compression stage.

[3]It is recommended that you use the class `tripulse()` provided on the website. Remember to type `help(tripulse)` in the console to learn how to use the class.

1.6 Signal compression. Create a Python class that takes as input a signal x of length N, the sampling frequency f_s, and a compression target K. The function outputs a vector with the K largest DFT coefficients and the corresponding set of frequencies at which these coefficients are observed. Notice that each of the coefficients that is kept requires storage of two numbers, the coefficient and the frequency. This is disadvantageous with respect to keeping just the first K coefficients. This more sophisticated compression is justified only if keeping these coefficients reduces the total number of DFT coefficients by a factor larger than 2.

1.7 The why of signal compression. Why do we keep the largest DFT coefficients? This question has a very precise mathematical answer that follows from Parseval's Theorem. Provide that very precise answer. You may want to look at Part 1.5.

1.8 Signal reconstruction. Create a Python class that takes as input the output of the class in Part 1.6 and reconstructs the original signal x. *Hint: You can use what you solved in Part 1.1, and Part 1.2 to help solve this part.*

1.9 Compression and reconstruction of a square wave. Generate a square wave of duration $T = 32$s sampled at a rate $f_s = 8$Hz and frequency 0.25Hz. Compress and reconstruct this wave using the functions in parts 1.6 and 1.8. Try different compression targets and report the energy of the error signal for $K = 2$, $K = 4$, $K = 8$ and $K = 16$. This problem should teach you that a square wave can be approximated better than a square pulse if you keep the same number of coefficients. This should be the case because the square wave looks the same at all points, but the square pulse doesn't. Explain this statement.

2 Speech processing

The DFT, in conjunction with the iDFT can be used to perform some basic speech analysis. In this part of the lab you will record your voice and perform a few interesting spectral transformations. For this part of the Lab, it is recommended that you use the classes `dft()` (Lab2) and `idft()` provided on the website. Please, remember to type `help(dft)` and `help(idft)` to learn how to use these classes.

2.1 Record, graph, and play your voice. Record 5 seconds of your voice[4] sampled at a frequency $f_s = 20KHz$. Plot your voice. Compute the DFT of your voice and plot its magnitude. Play it back on the speakers.

2.2 Voice compression. The 5 second recording of your voice at sampling frequency $f_s = 20KHz$ is composed of 100,000 samples. Use the DFT and iDFT to compress your voice by a factor of 2, i.e., store $K = 50,000$ numbers instead of 100,000, a factor of 4, (store $K = 25,000$ numbers), a factor of 8 (store $K = 12,500$ numbers), and so on. Keep compressing until the sentence you spoke becomes unrecognizable. You can perform this compression by keeping the first K DFT coefficients or the largest $K/2$ DFT coefficients. Which one works better?

2.3 Voice masking. Say that you and your partner speak the same sentence. The DFTs of the respective recording will be similar because it's the same sentence, but also different, because your voices are different. You can use this fact to mask your voice by modifying its spectrum, i.e., by increasing the contribution of some frequencies and decreasing the contributions of others. Design a project to record your voice, make it unrecognizable but intelligible, and play it in the speakers.

As we saw in Part 1.9, it is easier to reconstruct a square wave than it is to reconstruct a square pulse. This happens because the wave looks the same at all points, while the pulse looks different at different points. This suggests a problem with approximating the 5 second recording of your voice, namely, that you are trying to use the same complex exponentials to approximate different parts of your speech. You can overcome this limitation by dividing your signal in pieces and compressing each piece independently.

2.4 Better voice compression. Design a project that divides your speech in chunks of 100ms, and compresses each of the chunks by a given factor γ. Design the inverse system that takes the compressed chunks, reconstructs the individual speech pieces, stitches them together and plays them back in the speakers. You have just designed a rudimentary MP3 compressor and player. Try this out for different values of γ. Push γ to the largest possible compression factor.

[4]The class `recordsound()` provided on the website can be used. Please, remember to type `help(recordsound)` in the console to learn how to use the class.

3 Time management

The effort for this particular lab is evenly divided between parts 1 and 2. There is some overlap between the questions. If you do Part 1 properly, then Part 2 will be easier. Thus, the time split can be 4 and 6 hours or 6 and 4 hours, depending on the sort of person you are.

Do notice that some of the parts are conceptually simple but have finer points that may make them difficult to implement. The teaching assistants will provide substantial help with these fine points.