Lab 3: Inverse Discrete Fourier transform (DFT)

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

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

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

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

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,*

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

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

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

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),

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

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

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

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

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

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.

Brief comments on the implementation of the work

We have successfully implemented DFT transforming signals from time domain to frequency domain. However, can we transform these signals back to time domain without losing any information? Why are DFT important for signal and information processing? In this lab, we will learn Inverse Discrete Fourier Transform that recovers the original signal from its counterpart in the frequency domain. We will first prove a theorem that tells a signal can be recovered from its DFT by taking the Inverse DFT, and then code a Inverse DFT class in Python to implement this process.

We will then introduce an important application of DFT and Inverse DFT that is signal reconstruction and compression. We will use DFT and Inverse DFT Python classes to approximate some signals we have seen in previous labs, such as square pulse and triangular pulse, and study how well these approximations are compared with the original signal. We will further deal with the real-world signal – our voice. We will code a Python class that can record and play our own voice, based on which we will implement DFT and Inverse DFT for voice compression and masking. We will end up with an interesting problem allowing you to uncover secret messages from a signal that you may consider normal.

Computation of iDFT

In this first part of the lab, we will consider the inverse discrete Fourier transform (iDFT) and its practical implementation. As demonstrated in the lab assignment, the iDFT of the DFT of a signal $\mathbf{x}$ recovers the original signal $\mathbf{x}$ without loss of information. We begin by proving Theorem 1 that formally states this fact.

Proof of Theorem 1

Given a discrete signal $x : [0, N-1] \to \mathbb{C}$, let $X = \mathcal{F}(x) : \mathbb{Z} \to \mathbb{C}$ be the DFT of x and $\tilde{x} = \mathcal{F}^{-1}(X) : [0, N-1] \to \mathbb{C}$ be the iDFT of X. From the definition of the iDFT, we have

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

Now substituting the definition of the DFT for $X(k)$ in (1) yields

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

We may exchange the order of the summation, so that we first sum over k, and then pull out $x(n)$ since it is independent of k, i.e.

$$\tilde{x}(\tilde{n}) = \sum_{n=0}^{N-1} x(n) \Big(\sum_{k=0}^{N-1} \frac{1}{\sqrt{N}} e^{-j2\pi kn/N} \frac{1}{\sqrt{N}} e^{j2\pi k\tilde{n}/N} \Big) \tag{3}$$

Due to the orthonormality proved in 2.4 of Lab 1, we obtain

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

Therefore, (3) reduces to

$$\tilde{x}(\tilde{n}) = \sum_{n=0}^{N-1} x(n)\delta(n-\tilde{n}) = x(n) \tag{5}$$

with $\tilde{n} = n$ since the only nonnegative term in the sum is when $\tilde{n} = n$.

In the following, We will implement the iDFT in practice and employ it together with the DFT for signal reconstruction and compression on different signals, such as the square pulse, the triangular pulse, etc.

Signal Reconstruction

The original signal x can be recovered exactly by using N summands in the iDFT expression. However, we can also 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 as

$$\tilde{x}_K(n) := \frac{1}{\sqrt{N}}\left[X(0) + \sum_{k=1}^{K}\left(X(k)e^{j2\pi kn/N} + X(-k)e^{-j2\pi kn/N}\right)\right]. \tag{6}$$

As we increase K, i.e., adding more DFT coefficients in the truncated sum, we make the approximation closer to the actual signal. Based on the relation between the signal and its DFT we have learned from last lab assignment, we conclude that if we have a signal that varies slowly, a representation with just a few coefficients is sufficient, while for signals that vary faster, we need to add more coefficients to obtain a reasonable approximation.

We than first implement the signal reconstruction of a square pulse of duration $T = 32$s sampled at a rate $f_s = 8$Hz and length $T_0 = 4$s. According to the definition, the original signal and its DFT are shown in the following figures.

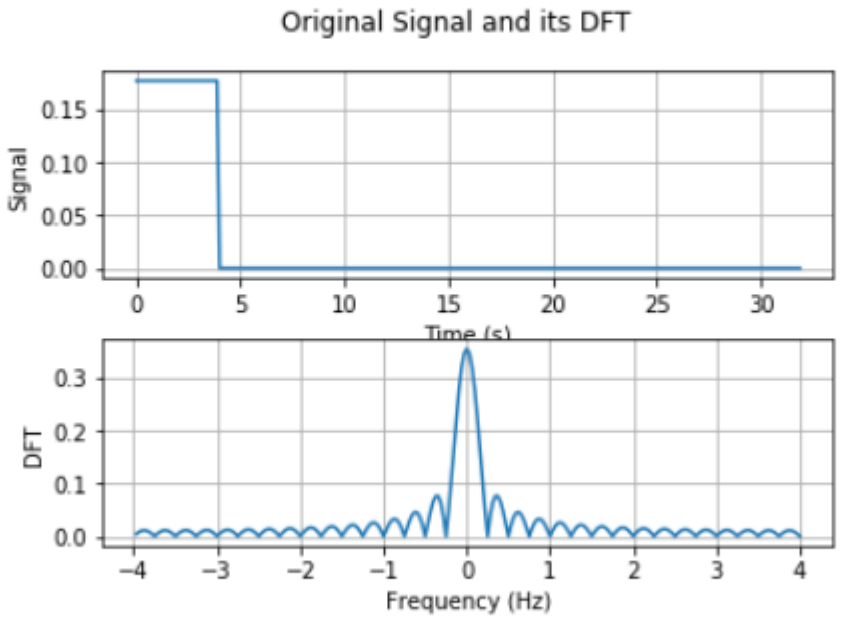

Figure 1: Square Pulse of $T_0 = 4s$

We than reconstruct the signal with the truncated iDFT process. By selecting different truncated parameters K, we can reconstruct different approximate signals as follows.

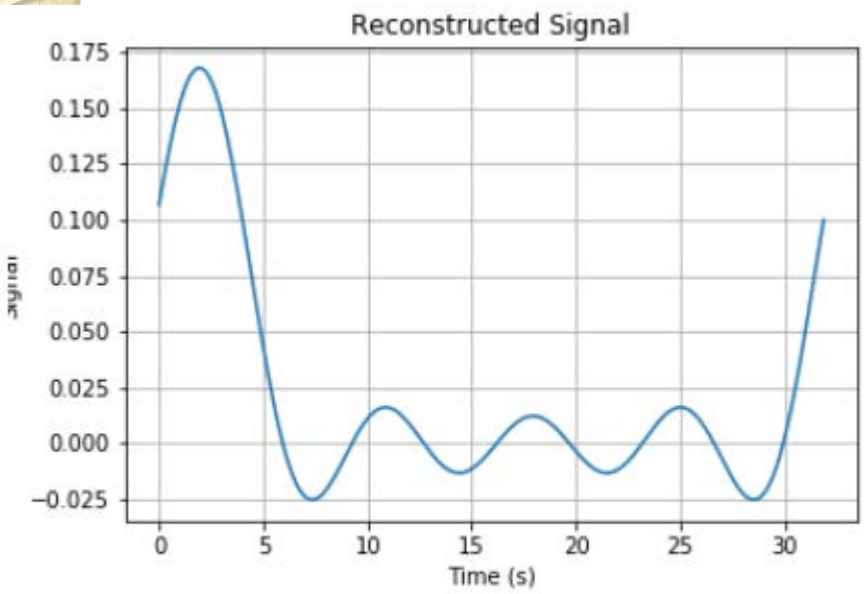

Figure 2: Reconstructed Square Pulse with $K = 4$

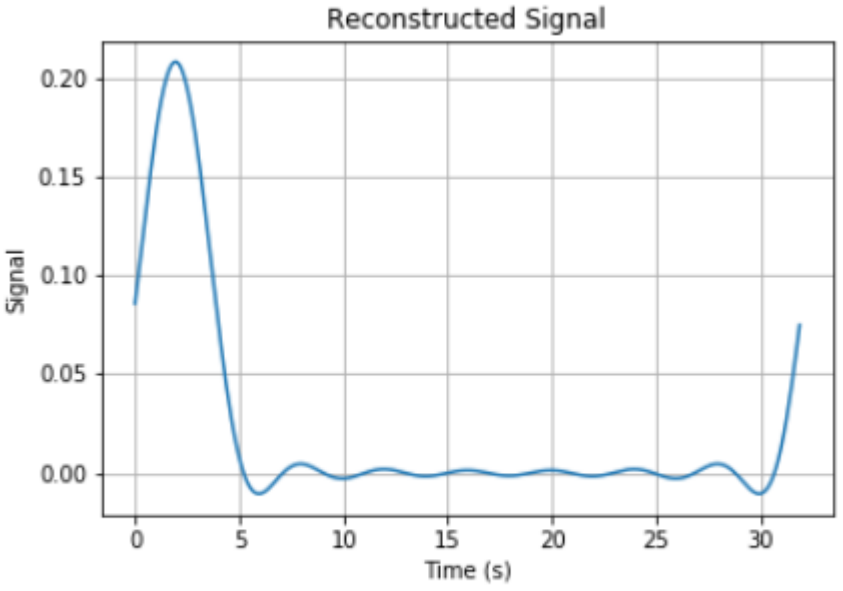

Figure 3: Reconstructed Square Pulse with $K = 8$

Here we select $K = 4$ and 8 as examples. While in the report, you should try more numbers of K to observe difference between these reconstructed signals. We then implement the signal reconstruction on the second example, the triangular pulse. We show the original signal x and its corresponding DFT coefficients in the following figure.

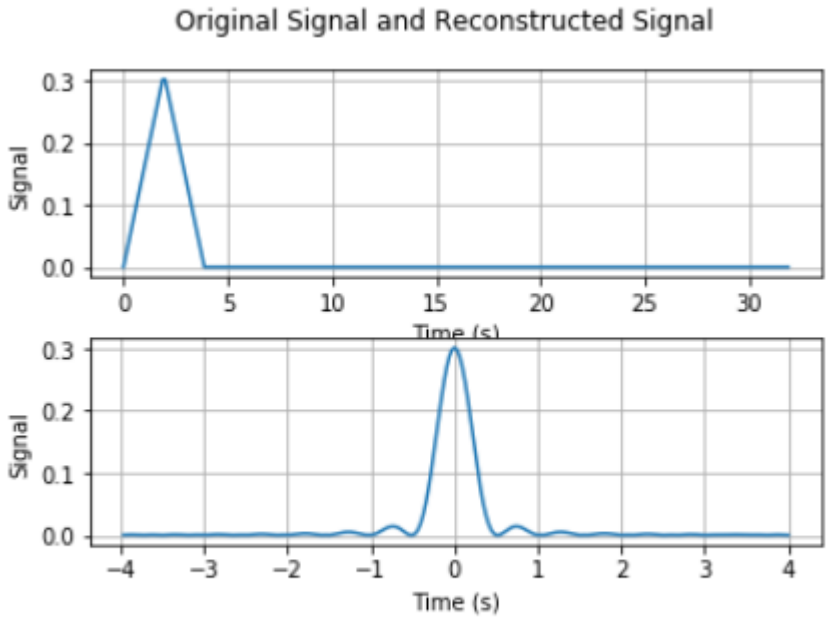

Figure 4: Triangular Pulse of $T_0 = 4s$

We then use the truncated K DFT coefficients to reconstruct the signal as $\tilde{x}_K$.

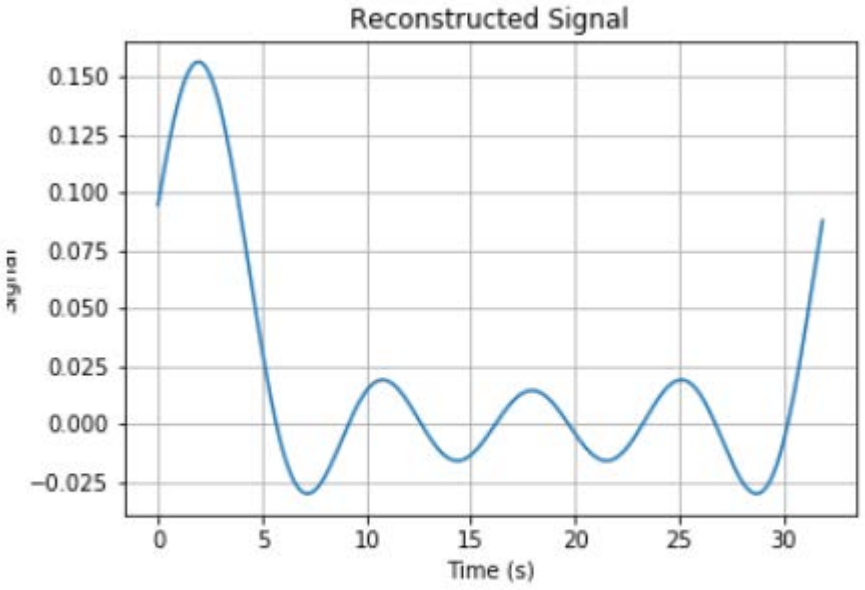

Figure 5: Reconstructed Triangular Pulse with $K = 4$

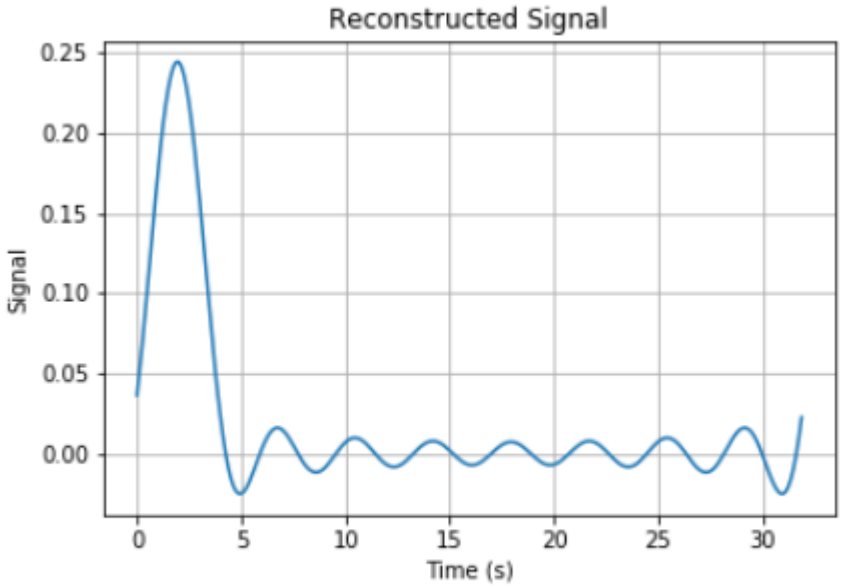

Figure 6: Reconstructed Triangular Pulse with $K = 8$

Since the triangular pulse varies more slowly, it should be easier to reconstruct with truncated DFT coefficients. You will find this result more precisely when testing more truncated numbers K.

Energy Difference

We study the energy of the difference signal. Denote $\tilde{X}_K$ by the DFT of the reconstructed signal $\tilde{x}_K$, and apply Parseval's Theorem to have

$$\|\rho_K\|^2 = \sum_{n=0}^{N-1} |x(n) - \tilde{x}(n)|^2 = \sum_{k=0}^{N-1} |X(k) - \tilde{X}(k)|^2 = \sum_{k=-N/2+1}^{N/2} |X(k) - \tilde{X}(k)|^2. \tag{7}$$

Using the fact that $X(k) - \tilde{X}(k) = 0$ for all $|k| \leq |K|$ and $X(k) - \tilde{X}(k) = X(k)$ for all $|k| > |K|$, the energy of the difference signal becomes

$$\|\rho_K\|^2 = \sum_{|k|>|K|} |X(k)|^2. \tag{8}$$

From above results, the larger K is, the smaller the energy difference is. Therefore, the constructed signal $\tilde{x}_K$ becomes closer to the original signal x if we increase K.

Signal Reconstruction with K Largest DFT Coefficients

In this subsection, we consider the signal reconstruction with K largest DFT coefficients, which is a different way for signal compression compared with (6). In this case, we consider the square wave signal of duration $T = 32$s sampled at a rate $f_s = 8$Hz and frequency 0.25Hz. We first compute its DFT, find out its largest K DFT coefficients, and reconstruct its approximate signal with iDFT. According to its definition, the original signal and its DFT coefficients are shown in the following figure.

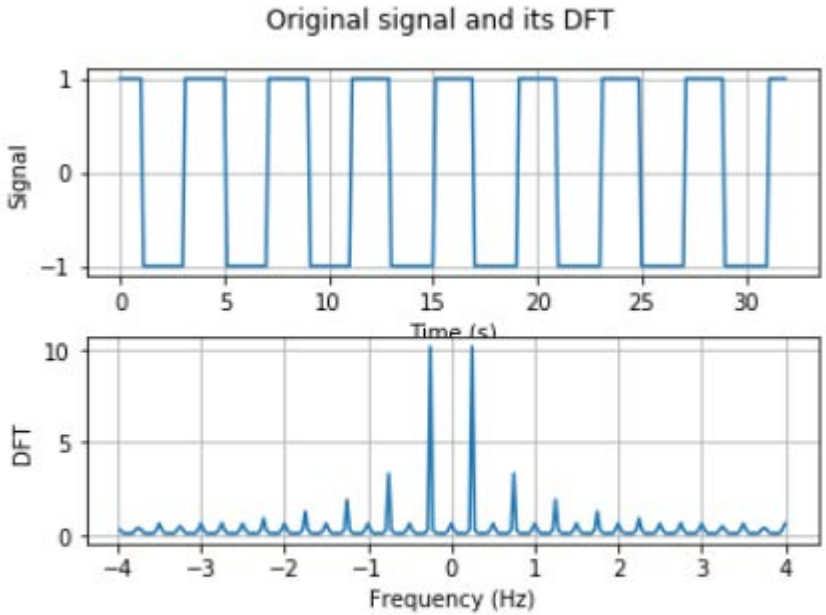

Figure 7: Square Wave Signal

We then reconstruct the signal with K largest DFT coefficients shown as follows.

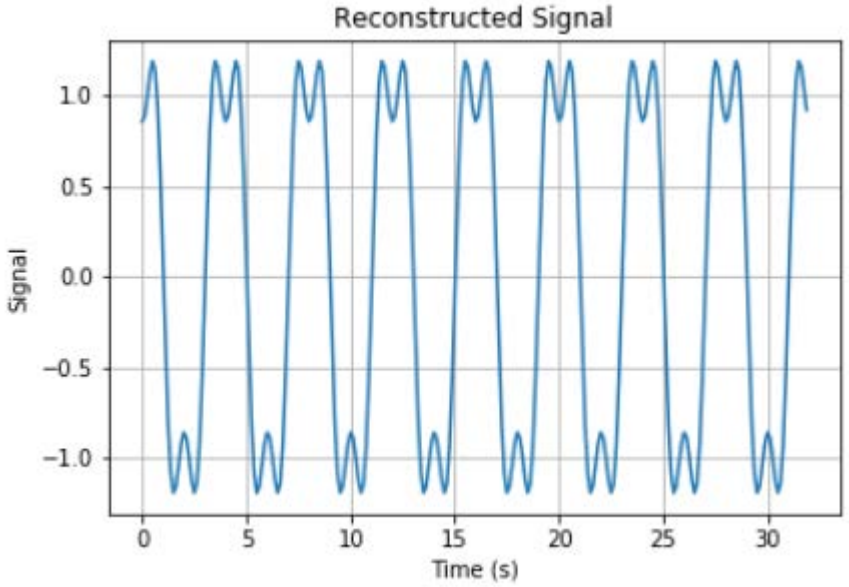

Figure 8: Reconstructed Square Wave with $K = 4$

Here we also consider $K = 4$ as an example, and you should try different numbers of K to see the difference. We observe that a square wave can be approximated better than a square pulse if you keep the same number of coefficients. This is because the square wave has periodic structure throughout its entire domain, so that we can easily approximate it with a few dominant DFT coefficients. However, the square pulse has a particular structure for the values $0 \leq n \leq M$ for fixed M. Lacking periodic structure, we need more DFT coefficients to effectively reconstruct the signal.

Speech Processing

In this subsection, we consider the signal reconstruction for speech signals in practice. We will record our voice, store it as a signal, and employ the DFT combined with the iDFT to perform the signal reconstruction. To begin with, we need to use the toolbox "sounddevice" to record our voice in Python and you should type "pip install sounddevice" in the console for installation. The recorded voice and its DFT is given by

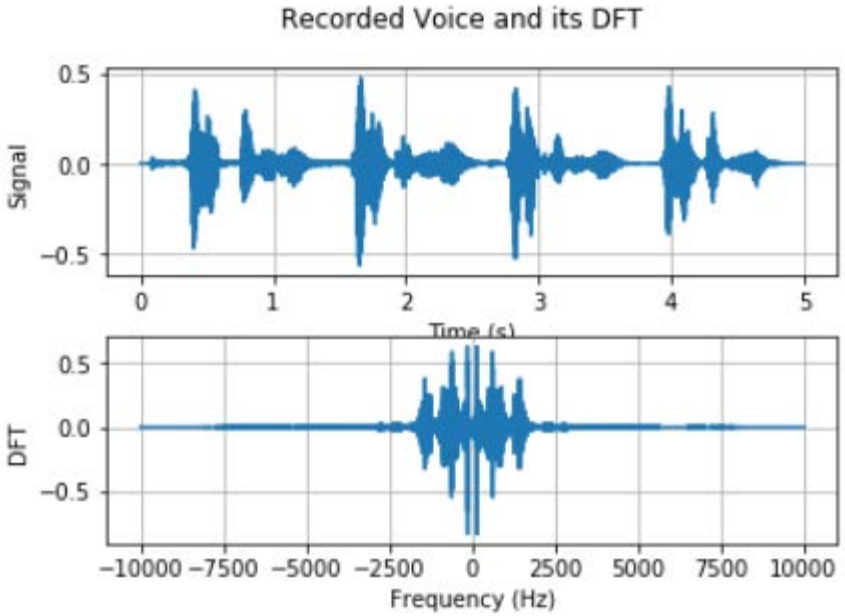

Figure 9: Recorded Voice and its DFT Coefficients

We then use the truncated DFT coefficients for approximate signal reconstruction. We first use the first K DFT coefficients to reconstruct the signal as follows.

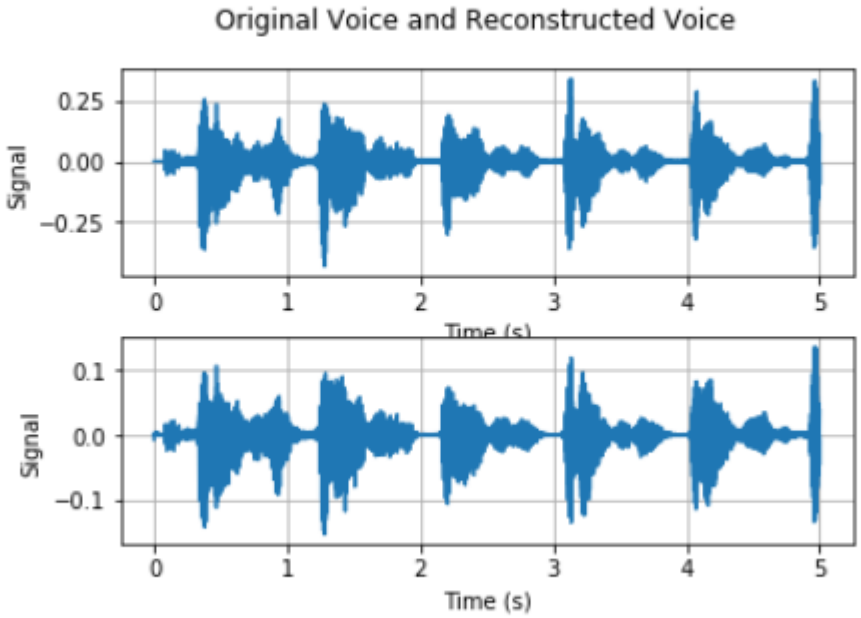

Figure 10: Original Signal and Reconstructed Signal with $\gamma = 16$

Here, we consider the factor $\gamma = 16$, i.e., first $K = 6250$ DFT coefficients. You should try different numbers of K in your report to observe the difference. We then consider another strategy for signal reconstruction. That is, we use the largest $K/2$ DFT coefficients as shown below.

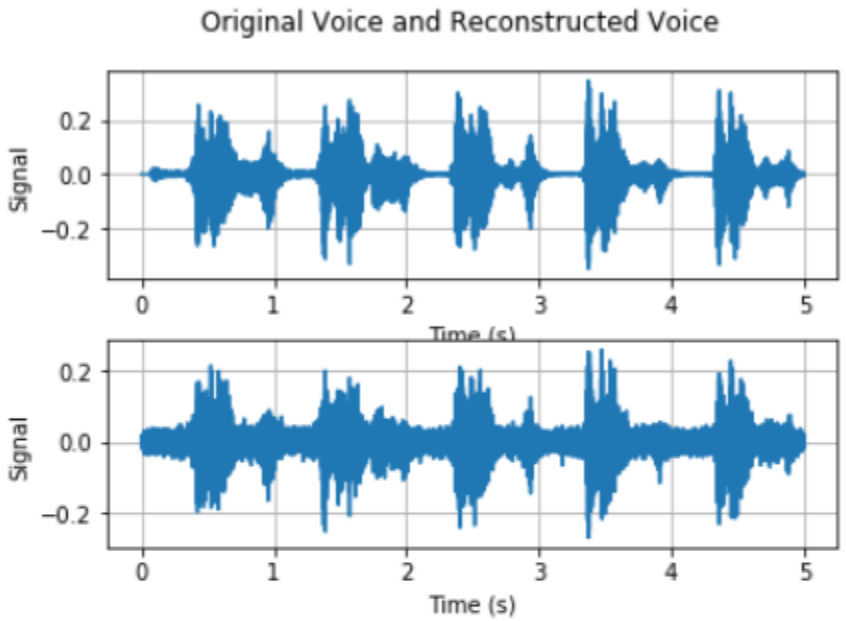

Figure 11: Original Signal and Reconstructed Signal with Largest DFT Coefficients

You should do both signal reconstruction strategies with different numbers K. By comparing these results, we observe that the signal reconstruction with largest $K/2$ DFT coefficients typically works better than the signal reconstruction with first K coefficients, while we note that this result also depends on the number K.

Voice Masking

You can reconstruct your voice signal by different truncation strategies. In particular, you can manipulate the spectrum as you prefer to reconstruct different approximate signals. One possible strategy is that we only store the DFT coefficient whose magnitude is smaller than a preset threshold α, which is shown in the following figure.

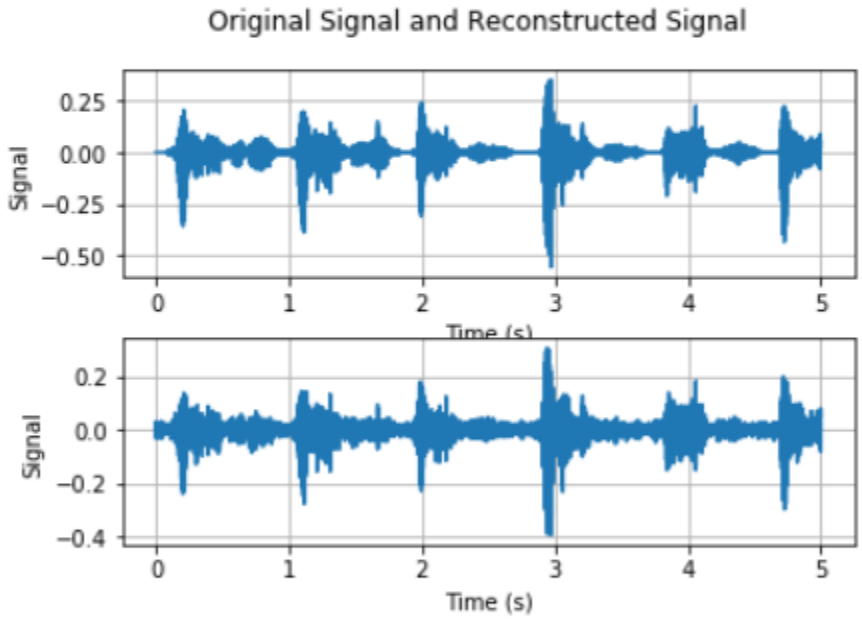

Figure 12: Original Signal and Reconstructed Signal with $\alpha = 0.25$

We consider the threshold $\alpha = 0.25$. In your report, you can try different strategies for signal reconstruction with your creativity and observe your results.

MP3 Compressor

Lastly, we consider a better voice compression strategy that divides your speech in chunks of 100ms, and compresses each of the chunks by a given factor γ. This is also a rudimentary MP3 compressor, and we show the original signal and the reconstructed signal as follows.

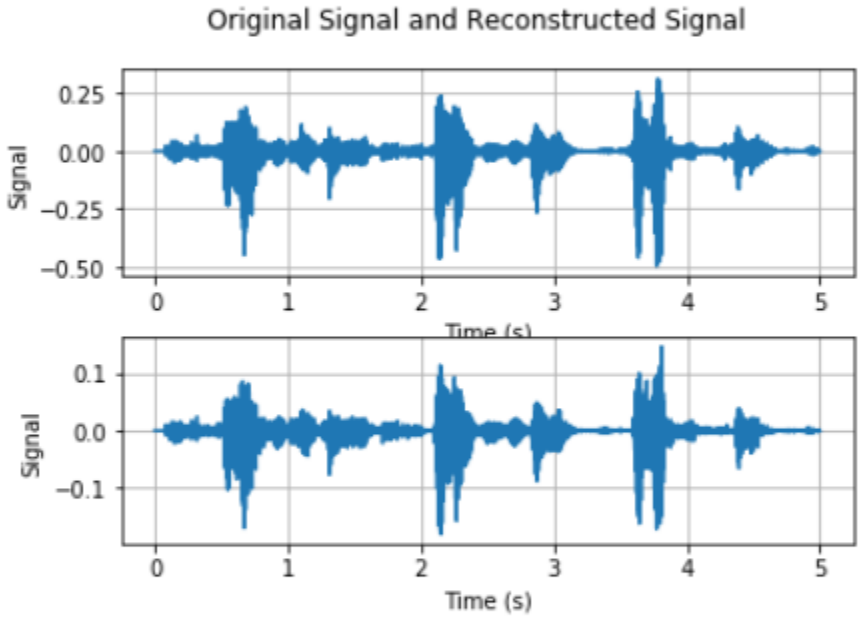

Figure 13: Original Signal and Reconstructed Signal of MP3 Compressor with $\gamma = 16$

We consider the factor $\gamma = 16$ as an example. We may observe that the MP3 compressor recovers the original signal better. In your report, you should try different factors γ and try to push γ the largest possible compression factor.

Code links

The code described here can be downloaded from this site:

Provided_Code_Lab3.py :

The class *recordsoun()* is defined in this file to record voice signals.

The class *idft()* implements the inverse discrete Fourier transform in 2 different ways..

The class *tripulse()* generates the triangular pulse signal.
The class *sqpulse()* generates the square pulse signal.

discrete_signal.py: This file defines the functions that generates different types discrete signals.

Lab3_Main.py: This file defines the functions that we used to solve the problems in the lab assignment, instantiating objects when necessary. This is also where the plots and the audio files are created.

Report presentation

Please remember to label both the x-axis and y-axis of all your figures Remember also to add a legend and/or a title to the plots you're including in your figures (check commands A graph without units and labeled axes makes no sense. The titles help us with grading.

Please include your code along with the lab report.