CS 370 - A3

Bilal Khan
bilal2vec@gmail.com

November 16, 2023

Contents

1 1

Consider the signal $f(t) = \sin(8\pi t/T)$ with period of T seconds. During one period, N samples $f_n = f(t_n)$ were obtained at $t_n = nT/N$, where N is a given number. Compute by hand the Fourier coefficients $\{F_k\}$ of the discrete signal $\{f_n\}, k = 0, 1, \cdots, N-1$, using the formula

$$F_k = \frac{1}{N}\sum_{n=0}^{N-1} f_n W^{-nk}$$

Where $W = e^{\frac{2\pi i}{N}}$.

$$\begin{aligned}
f(t) &= \sin(8\pi t/T) \\
f_n &= f(t_n) = \sin(8\pi nT/NT) = \sin\left(\frac{8\pi n}{N}\right) \\
F_k &= \frac{1}{N}\sum_{n=0}^{N-1} \sin\left(\frac{8\pi n}{N}\right) e^{\frac{-2\pi ink}{N}} \\
&= \frac{1}{N}\sum_{n=0}^{N-1} \frac{1}{2i}\left(e^{i\frac{8\pi n}{N}} - e^{i\frac{-8\pi n}{N}}\right) e^{i\frac{-2\pi nk}{N}} \\
&= \frac{1}{2iN}\sum_{n=0}^{N-1} e^{i\frac{8\pi n}{N}} e^{i\frac{-2\pi nk}{N}} - \frac{1}{2iN}\sum_{n=0}^{N-1} e^{i\frac{-8\pi n}{N}} e^{i\frac{-2\pi nk}{N}} \\
&= \frac{1}{2iN}\sum_{n=0}^{N-1} e^{i\frac{\pi n(8-2k)}{N}} - \frac{1}{2iN}\sum_{n=0}^{N-1} e^{-i\frac{\pi n(8+2k)}{N}}
\end{aligned}$$

When $k = 4$, the expression reduces to

$$\frac{1}{2iN}\sum_{n=0}^{N-1} 1 - \frac{1}{2iN}\sum_{n=0}^{N-1} e^{-i\frac{16\pi n}{N}}$$

The first summation term amounts to N, and the second term is a geometric series with ratio $e^{-i\frac{16\pi}{N}}$, which disappears because $e^{i2j\pi} = 1$ for any integer j. Thus, the expression reduces to $F_4 = \frac{1}{2i}$.

Similarly, when $k = (N-8)/2$, the first summation term disappears and the second reduces to $F_{(N-8)/2} = \frac{1}{2i}$

For any other value of k in the range provided, both summation terms are geometric series that vanish in this way. Therefore the Fourier coefficients for any k in the range provided that is not 4 or $(N-8)/2$ is 0.

2 2

Let $[F_0, \cdots, F_{N-1}]$ be the DFT of a sequence of data $[f_0, \cdots, f_{N-1}]$ where

$$F_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n W^{-nk}$$

And $W = e^{\frac{2\pi i}{N}}$.

2.1 a

Give a simplified expreession for F_k when $f_n = W^{5n}$. For simplicity, assume $N >> 5$.

$$\begin{aligned} F_k &= \frac{1}{N} \sum_{n=0}^{N-1} W^{5n} W^{-nk} \\ &= \frac{1}{N} \sum_{n=0}^{N-1} W^{n(5-k)} \\ &= \frac{1}{N} \sum_{n=0}^{N-1} e^{\frac{2\pi i n(5-k)}{N}} \end{aligned}$$

When $k = 5$, the expression reduces to $\frac{1}{N} \sum_{n=0}^{N-1} 1 = N/N = 1$. For all other values of k, this reduces to a geometric series with ratio $e^{\frac{2\pi i(5-k)}{N}}$, which converges to zero as $N >> 5$.

2.2 b

Give a simplified expression for F_k when $f_n = (-1)^n$ for odd values of N.

$$\begin{aligned} F_k &= \frac{1}{N} \sum_{n=0}^{N-1} (-1)^n W^{-nk} \\ &= \frac{1}{N} \sum_{n=0}^{N-1} e^{\pi i n} e^{\frac{-2\pi i n k}{N}} \\ &= \frac{1}{N} \sum_{n=0}^{N-1} e^{\pi i n(1-2k/N)} \end{aligned}$$

Since N is odd, the expression $2k/N$ in the exponent is never an integer (and never -1) and so the expression does not reduce to 1 for certain values of k.

2.3 c

Give a simplified expression for F_k when $f_n = (-1)^n$ for even values of N (it may be useful to recall that $e^{\pi i} = -1$)

$$\begin{aligned} F_k &= \frac{1}{N}\sum_{n=0}^{N-1}(-1)^n W^{-nk} \\ &= \frac{1}{N}\sum_{n=0}^{N-1} e^{\pi i n} e^{\frac{-2\pi i n k}{N}} \\ &= \frac{1}{N}\sum_{n=0}^{N-1} e^{\pi i n(1-2k/N)} \end{aligned}$$

When $k = N/2$, the expression reduces to $\frac{1}{N}\sum_{n=0}^{N-1} e^0 = 1$. For all other values of k, this reduces to $e^{\pi i(1-2k/N)}$, the exponent of which results in values < 0 for $k < N/2$ and values > 0 for $k > N/2$. Note that values of k that are equally far away from $N/2$ result in values with the same magnitude but opposite sign, and so they cancel each other out in the summation. Thus, the expression reduces to 0 for all values of k except $k = N/2$.

3 3

Consider the real, even vector $f = [f_0, \cdots, f_{N-1}]$ in which $f_n \in \mathbb{R}$ and $f_n = f_{N-n}$ (Analogous to an even function, where $f(x) = f(-x)$). Given the DFT,

$$F_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n W^{-nk}$$

and $W = e^{\frac{2\pi i}{N}}$, prove that the vector of Fourier coefficients $F = [F_0, \cdots, F_{N-1}]$ is also a real and even vector. Justify your steps.

Note that $e^{\pi i n} = 1$ for all integers n. Computing the value of F_{N-k}:

$$\begin{aligned}
F_{N-k} &= \frac{1}{N} \sum_{n=0}^{N-1} f_n W^{-n(N-k)} \\
&= \frac{1}{N} \sum_{n=0}^{N-1} f_n e^{\frac{-2\pi i n(N-k)}{N}} \\
&= \frac{1}{N} \sum_{n=0}^{N-1} f_n e^{-2\pi i n + \frac{2\pi i n k}{N}} \\
&= \frac{1}{N} \sum_{n=0}^{N-1} f_n e^{\frac{2\pi i n k}{N}} \\
&= \frac{1}{N} \sum_{n=0}^{N-1} f_n e^{\frac{2\pi i n k}{N}}
\end{aligned}$$

Now lets compute the value of F_k using the fact that $f_n = f_{N-n}$ (equivalently: $n = N - n$)

$$\begin{aligned}
F_k &= \frac{1}{N} \sum_{n=0}^{N-1} f_n W^{-nk} \\
&= \frac{1}{N} \sum_{n=0}^{N-1} f_n e^{\frac{-2\pi i n k}{N}} \\
&= \frac{1}{N} \sum_{n=0}^{N-1} f_{N-n} e^{\frac{-2\pi i (N-n) k}{N}} \\
&= \frac{1}{N} \sum_{n=0}^{N-1} f_{N-n} e^{-2\pi i k + \frac{2\pi i n k}{N}} \\
&= \frac{1}{N} \sum_{n=0}^{N-1} f_{N-n} e^{\frac{2\pi i n k}{N}} \\
&= \frac{1}{N} \sum_{n=0}^{N-1} f_n e^{\frac{2\pi i n k}{N}}
\end{aligned}$$

The two expressions are equivalent for all k from 0 to $N-1$ and so the Fourier coefficients F_k are real and even.

4 4

Consider the sequence of 8 numbers: $f = [-1, 2, 3, 4, 5, 2, 3, 4]$. Perform a complete butterfly Fast Fourier Transform process in order to compute the DFT of f. Show your work in a butterfly diagram.

$$\begin{bmatrix} -1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 2 \\ 3 \\ 4 \end{bmatrix} \rightarrow \begin{bmatrix} (-1+5)/2 = 2 \\ (2+2)/2 = 2 \\ (3+3)/2 = 3 \\ (4+4)/2 = 4 \\ (-1-5)/2 * W^{-0} = -3 \\ (2-2)/2 * W^{-1} = 0 \\ (3-3)/2 * W^{-2} = 0 \\ (4-4)/2 * W^{-3} = 0 \end{bmatrix} \rightarrow \begin{bmatrix} (2+3)/2 = 2.5 \\ (2+4)/2 = 3 \\ (2-3)/2 * W^{-0} = -0.5 \\ (2-4)/2 * W^{-1} = -1 * W^{-1} \\ (-3+0)/2 = -1.5 \\ (0+0)/2 = 0 \\ (-3-0)/2 * W^{-0} = -1.5 \\ (0-0)/2 * W^{-1} = 0 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} (2.5+3)/2 = 2.75 \\ (2.5-3)/2 * W^{-0} = -0.25 \\ (0.5 + -1 * W^{-1}) * W^{-0} \\ (0.5 - -1 * W^{-1}) * W^{-1} \\ (-1.5+0)/2 = -0.75 \\ (-1.5-0)/2 * W^{-0} = -0.75 \\ (-1.5+0)/2 = -0.75 \\ (-1.5-0)/2 * W^{-1} = -0.75 * W^{-1} \end{bmatrix} \rightarrow \begin{bmatrix} 2.75 \\ -0.75 \\ (0.5 + -1 * W^{-1}) * W^{-0} \\ -0.75 \\ -0.25 \\ -0.75 \\ (0.5 - -1 * W^{-1}) * W^{-1} \\ -0.75 * W^{-1} \end{bmatrix}$$

5 5

The sound file train bird.wav can be downloaded from the Crowdmark assignment page. This sound file can be loaded into your Jupyter notebook and played using:

```
from IPython.display import Audio
import scipy.io.wavfile
Fs, y = scipy.io.wavfile.read('train_bird.wav')
Audio(y, rate=Fs)
```

The sound data consists of a signal array y and a sampling rate Fs. You should hear a bird chirping and a train whistle. The combined recording has been generated by superimposing the two signals. As seen in the time-domain plot below, it would be rather difficult to separate the bird chirps from the train whistle directly from this mixed signal, since they heavily overlap in time. Instead, you will attempt to separate them in the frequency-domain using the DFT, exploiting the fact that the sounds have different frequency ranges.

1. Plot the original signal, and plot the magnitudes (abs) of the DFT of this signal, with appropriate titles.
2. Read the Appendix (page 5) to this assignment on Filtering. Then design a low pass filter which isolates the train sound and a corresponding high pass filter to instead isolate the bird chirps. You should be able to do this by inspecting the frequency plot of the input, and using a little bit of trial and error. (Do not worry if you cannot absolutely perfectly separate them — separate them as best you can with a reasonable effort.) Playing back the resulting filtered sounds should allow you to hear the separated train and bird sounds, respectively.
3. Produce plots of each of the final filtered signals, for both the magnitudes of the filtered frequency-domain data and the new time-domain data.

Notes: 1) Theoretically, the inverse transform of the filtered signal should be real. However, due to roundoff errors, the imaginary part may not be identically zero (but should be very small, close to machine epsilon). You may just take the real part and ignore the imaginary part of the inverse transform. 2) When applying the low/high pass filter, make sure you set to zero for both of the complex conjugate pairs. Otherwise, the inverse transform will lead to a complex signal where the imaginary part is not the size of the machine epsilon (e.g. $O(10^{-4})$).

6 6

In Jupyter, a grayscale image can be loaded, converted into a 2D array of floating point values, and viewed with the following commands

```
f = np.array(plt.imread('operahall.png'), dtype=float)
plt.imshow(f,cmap='gray');
```

(provided that numpy and matplotlib.pyplot are loaded). In this problem, we study a simple form of compression on grayscale images. We will obtain compression by dropping (relatively) small Fourier coefficients on 32×32 pixel subblocks of an input image. By this we mean that if $\{f_i, j\}$ are the original pixel values in a given subblock and $\{F_k, l\}$ are the corresponding DFT values, then we drop (i.e., set to zero) any F_k, l such that

$$|F_k, l| \leq F_{max} \cdot tol$$

Here F_{max} is the maximum of $\{|F_k, l|\}$ within a given block and tol is a specified (global) drop tolerance. The file operahall.jpg included with the assignment is a grayscale image which we will use for all parts of this compression question. Prepare a Jupyter notebook to carry out the operations described below.

1. Perform the following:
   (a) Load the image, extract the top-left 32×32 sub-block of pixels, and plot it.
   (b) Perform a 2D FFT on this block using fft2 to get a new 2D array of complex numbers, which we'll denote as F.
   (c) Use imshow, again with the gray scale colormap, to visualize the magnitudes (i.e., modulus or absolute value) of the 2D array of DFT data. State which pixel is the brightest (whitest) and explain what significance this value has.
   (d) Zero out the DC coefficient of F, and use imshow once more to visualize the data.

2. Create a function, `[Y, drop] = Compress(X,tol)`, which takes as inputs an original image as an array of floats, X, and the drop tolerance parameter, tol. It should return a tuple [Y, drop] which contains the compressed image Y, as another array of floats, and the computed drop ratio, drop, which is defined as:

$$\text{drop ratio} = \frac{\text{Total number of nonzero Fourier coefficients dropped}}{\text{Total number of originally nonzero Fourier coefficients}}$$

   If drop ratio = 0, then no nonzero Fourier coefficients have been dropped; if the drop ratio = 1, then all nonzero Fourier coefficients have been dropped. In general, the drop ratio will be between 0 and 1.

   Specifically, your function Compress should:
   (a) compute the 2D Fourier coefficients (fft2) for every 32×32 subblock.
   (b) for each subblock, determine its F_{max} and set each of its Fourier coefficients having magnitude/modulus less than or equal to $F_{max} \cdot tol$ to 0.
   (c) find the total (global) number of nonzero coefficients and number of dropped nonzero coefficients, and use them to compute drop.

(d) reconstruct the new/compressed 32×32 image array by using the inverse 2D Fourier transform (ifft2), and then discarding any imaginary parts of the resulting data.

(e) after all the 32×32 subblocks for all the components have been processed, return the complete reconstructed (”compressed”) image as Y and the computed drop ratio as drop.

3. Determine by trial and error on different tol values (i.e., not by writing any code) four values of tol resulting in drop ratios of approximately 0.5, 0.8, 0.92, and 0.97 (to 2 significant digits on the tolerance). Then in the Jupyter notebook, you should:

   (a) Call your Compress function with the set of different tol values you determined.

   (b) Plot the four compressed images, using imshow for each compressed image Y. Each plot should have a title indicating the tol value used and the resulting drop ratio.

   (c) Of the four images reconstructed above, visualize the error in the result image with the least compression (highest quality), using imshow (again with gray scale colormap) on the 2D array of absolute values of the difference between the original image and the compressed image.

7 Appendix

Recorded sounds are often processed by carrying out a filtering operation in the frequency domain. suppose we are given an input signal $x_i = 0, i = 0, \cdots, N-1$. Let $X = FFT(x)$. The Fourier representation has frequencies in the range $\{0, \cdots, N/2\}$, However, note that $X_{N-k} = X_{-k}, k < N/2$ really represents frequencies of size k, not $N-k$, since we have used a complex representation of the Fourier series, and we have defined $X_{k \pm N} = X_{-k}$. If we use the conventional range of X_k, i.e., $k \in [0, N-1]$, this means that we have to do the following to construct a lowpass filter. Let $p < N/2$ be the maximum value of the frequency which will be allowed to pass our lowpass filter (i.e., retained). In other words, we will eliminate (zero out) any frequencies in the signal with index $>$ p. This is easily accomplished using the lowpass filter

$$\begin{aligned} Q_k &= 0, k = p+1, \cdots, N-p-1 \\ &= 1, \text{otherwise} \end{aligned}$$

Note that due to conjugate symmetry of the DFT of a real signal, the filter should be symmetric about $N/2$.

Another way to think about equation (1) is to imagine plotting X in the range $[-N/2 + 1, \cdots, +N/2]$. Then, we want to zero all the X_k such that $k > p$ or $k < -p$. This defines a filter $Q_k, k \in [-N/2 + 1, \cdots, N/2]$. Now, define the filter in the range $[0, \cdots, N-1]$ by a periodic extension $Q_{N-k} = Q_{-k}, k = 1, \cdots, N/2 - 1$.

The filtered signal in the frequency domain $\hat{X}$ is then

$$\hat{X} = X_k Q_k; k = 0, \cdots, N-1$$

and the filtered signal in the time domain is $\hat{X} = Real(IFFT(\hat{X}))$. A high pass filter is constructed in a similar way, except we want to remove frequencies $< p$.

The convention for the FFT in SciPy (and elsewhere) is slightly different from that used in our class and the course notes (see section 5.5.1). In class, the DFT/IDFT pair was defined as

$$\begin{aligned} F_k &= \frac{1}{N} \sum_{n=0}^{N-1} f_n W^{-nk} \\ f_n &= \sum_{k=0}^{N-1} F_k W^{nk} \end{aligned}$$

where $W = e^{\frac{2\pi i}{N}}$ whereas SciPy defines the above pair as

$$\begin{aligned} F_k &= \sum_{n=0}^{N-1} f_n W^{-nk} \\ f_n &= \frac{1}{N} \sum_{k=0}^{N-1} F_k W^{nk} \end{aligned}$$

The two definitions differ in the place where the normalization by $1/N$ occurs, in the forward or inverse transform; since this is simply multiplication by a constant, it is straightforward to convert between the two. In all analytical work, we will use the definition as given in class unless otherwise specified. For coding questions, using SciPy's convention is fine.