ECE 438 - Laboratory 6a
Discrete Fourier Transform and Fast Fourier Transform Algorithms (Week 1)

Last updated on February 22, 2022

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from helper import DTFT, hamming
import time
```

In [2]:
```
# make sure the plot is displayed in this notebook
%matplotlib inline
# specify the size of the plot
plt.rcParams['figure.figsize'] = (16, 6)

# for auto-reloading extenrnal modules
%load_ext autoreload
%autoreload 2
```

1. Introduction

This is the first week of a two week laboratory that covers the Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT) methods. The first week will introduce the DFT and associated sampling and windowing effects, while the second week will continue the discussion of the DFT and introduce the FFT.

In previous laboratories, we have used the Discrete-Time Fourier Transform (DTFT) extensively for analyzing signals and linear time-invariant systems.

$$\text{(DTFT)} \quad X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n} \tag{1}$$

$$\text{(inverse DTFT)} \quad x[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{j\omega})e^{j\omega n}\,d\omega \tag{2}$$

While the DTFT is very useful analytically, it usually cannot be exactly evaluated on a computer because equation (1) requires an infinite sum and equation (2) requires the evaluation of an integral.

The discrete Fourier transform (DFT) is a sampled version of the DTFT, hence it is better suited for numerical evaluation on computers.

$$\text{(DFT)} \quad X_N[k] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N} \tag{3}$$

$$\text{(inverse DFT)} \quad x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X_N[k]e^{j2\pi kn/N} \tag{4}$$

Here $X_N[k]$ is an N point DFT of $x[n]$. Note that $X_N[k]$ is a function of a discrete integer k, where k ranges from 0 to $N-1$.

In the following sections, we will study the derivation of the DFT from the DTFT, and several DFT implementations. The fastest and most important implementation is known as the fast Fourier transform (FFT). The FFT algorithm is one of the cornerstones of signal processing.

2. Deriving the DFT from DTFT

2.1 Truncating the Time-domain Signal

The DTFT usually cannot be computed exactly because the sum in equation (1) is infinite. However, the DTFT may be approximately computed by truncating the sum to a finite window. Let $w[n]$ be a rectangular window of length N:

$$w[n] = \begin{cases} 1 & 0 \le n \le N-1 \\ 0 & \text{else} \end{cases} \tag{5}$$

Then we may define a truncated signal to be

$$x_{\text{tr}}[n] = w[n]x[n] \tag{6}$$

The DTFT of $x_{\text{tr}}[n]$ is given by:

$$X_{\text{tr}}(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x_{\text{tr}}[n]e^{-j\omega n} \tag{7}$$

$$= \sum_{n=0}^{N-1} x[n]e^{-j\omega n} \tag{8}$$

We would like to compute $X(e^{j\omega})$, but as with filter design, the truncation window distorts the desired frequency characteristics; $X(e^{j\omega})$ and $X_{\text{tr}}(e^{j\omega})$ are generally not equal. To understand the relation between these two DTFT's, we need to convolve in the frequency domain (as we did in designing filters with the truncation technique):

$$X_{\text{tr}}(e^{j\omega}) = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{j\sigma})W(e^{j(\omega-\sigma)})d\sigma \tag{9}$$

where $W(e^{j\omega})$ is the DTFT of $w[n]$. Equation (9) is the periodic convolution of $X(e^{j\omega})$ and $W(e^{j\omega})$. Hence the true DTFT, $X(e^{j\omega})$, is smoothed via convolution with $W(e^{j\omega})$ to produce the truncated DTFT, $X_{\text{tr}}(e^{j\omega})$.

We can calculate $W(e^{j\omega})$:

$$\begin{aligned} W(e^{j\omega}) &= \sum_{n=-\infty}^{\infty} w[n]e^{-j\omega n} \\ &= \sum_{n=0}^{N-1} e^{-j\omega n} \\ &= \begin{cases} \frac{1-e^{-j\omega N}}{1-e^{-j\omega}} & \text{for } \omega \neq 0, \pm 2\pi \\ N & \text{for } \omega = 0, \pm 2\pi \end{cases} \end{aligned} \tag{10}$$

For $\omega \neq 0, \pm 2\pi, \ldots$, we have:

$$\begin{aligned} W(e^{j\omega}) &= \frac{e^{-j\omega N/2}}{e^{-j\omega/2}} \frac{e^{j\omega N/2} - e^{-j\omega N/2}}{e^{j\omega/2} - e^{-j\omega/2}} \\ &= e^{-j\omega(N-1)/2} \frac{\sin(\omega N/2)}{\sin(\omega/2)} \end{aligned} \tag{11}$$

Notice that the magnitude of this function is similar to $\text{sinc}(\omega N/2)$ except that it is periodic in ω with period 2π.

2.2 Frequency Sampling

Equation (8) contains a summation over a finite number of terms. However, we can only evaluate (8) for a finite set of frequencies, ω. We must sample in the frequency domain to compute the DTFT on a computer. We can pick any set of frequency points at which to evaluate (8), but it is particularly useful to uniformly sample ω at N points, in the range $[0, 2\pi)$. If we substitute

$$\omega = 2\pi k/N \tag{12}$$

for $k = 0, 1, \ldots, (N-1)$ in (8), we find that

$$\begin{aligned} X_{\text{tr}}(e^{j\omega})\Big|_{\omega=\frac{2\pi k}{N}} &= \sum_{n=0}^{N-1} x[n]e^{-j\omega n}\Big|_{\omega=\frac{2\pi k}{N}} \\ &= \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N} \\ &= X_N[k]. \end{aligned}$$

In short, the DFT values result from sampling the DTFT of the truncated signal.

$$X_N[k] = X_{\text{tr}}(e^{j2\pi k/N}) \tag{13}$$

Exercise 2: Windowing Effects

We will next investigate the effect of windowing when computing the DFT of the signal $x(n) = \cos\left(\frac{\pi}{4}n\right)$ truncated with a window of size $N = 20$.

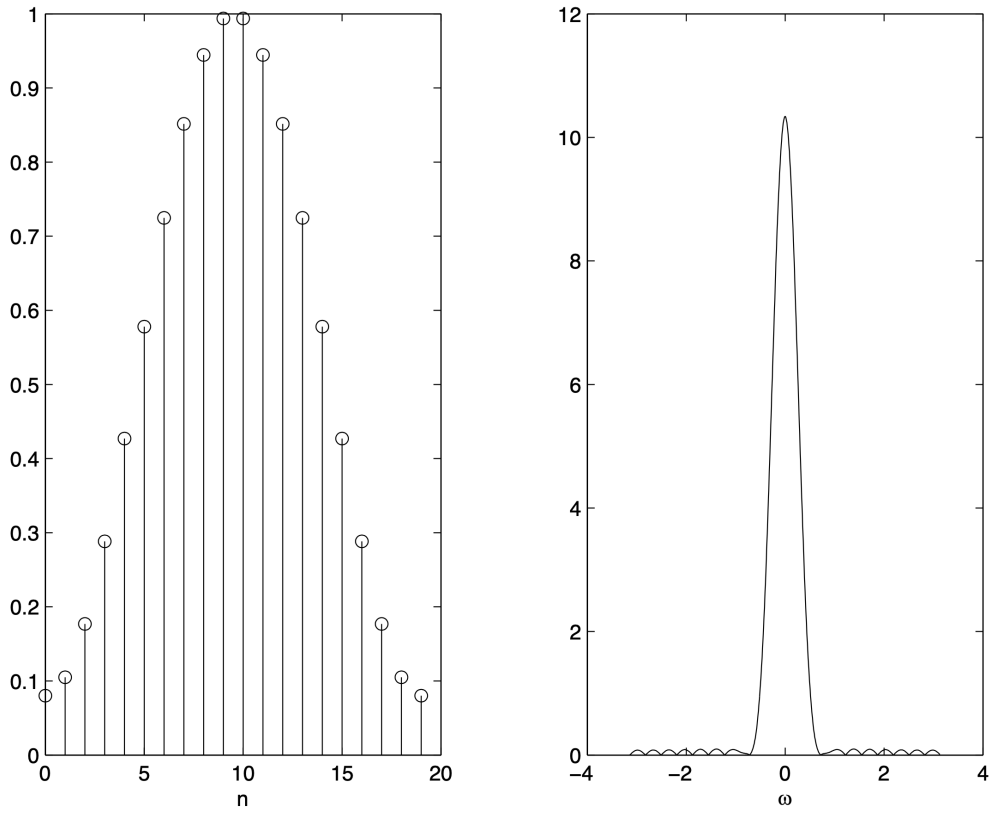

Figure 1: The plot of a Hamming window (left) and its DTFT (right).

1. Plot the magnitude of $W(e^{j\omega})$, using equations (10) and (11).

In [3]:
```
# insert your code here
```

2. Plot the phase of $W(e^{j\omega})$, using equations (10) and (11).

In [4]:
```
# insert your code here
```

3. Determine an analytical expression for $X(e^{j\omega})$ (the DTFT of the non-truncated signal).

insert your answer here

4. Truncate the signal $x[n]$ using a window of size $N = 20$ and then use `DTFT` to compute $X_{\text{tr}}(e^{j\omega})$. Then plot the magnitude of $X_{\text{tr}}(e^{j\omega})$. Make sure that the plot contains a least 512 points.

Hint: Use the command `X, w = DTFT(x, 512)` .

In [5]:
```
# insert you code here
```

5. Describe the difference between $|X_{\text{tr}}(e^{j\omega})$ and $|X(e^{j\omega})|$. What is the reason for this difference?

insert your answer here

6. What would you expect your plots to look like if you had used a Hamming window in place of the truncation (rectangular) window? (See Fig. 1 for a plot of a Hamming window of length 20 and its DTFT.) Submit the plot of the magnitude of the DTFT of the signal $x[n]$ windowed using a Hamming window. (Hint: The Python command for a Hamming window is `hamming(N)` .)

insert your answer here

7. Comment on the effects of using a different window for $w[n]$.

insert your answer here

3. The Discrete Fourier Transform

Exercise 3.1: Computing the DFT

We will now develop our own DFT functions to help our understanding of how the DFT comes from the DTFT.

1. Write your own Python function to implement the DFT of equation (3). Your routine should implement the DFT exactly as specified by (3) using *for-loops* for n and k, and computing the exponentials as they appear.

Hint: initialize `X` as a vector of complex values by using `.astype(complex)` .

In [6]:
```
def DFTsum(x):
    """
    Parameters:
    ---
    x: the input signal, an N point vector contining the values x[0], ..., x[N - 1]

    Returns:
    ---
    X: the DFT of x
    """
    X = None
    return X
```

2. Test your routine `DFTsum` by computing $X_N(k)$ for each of the following cases:

- $x(n) = \delta(n) \quad \text{for } N = 10$
- $x(n) = 1 \quad \text{for } N = 10$
- $x(n) = e^{j2\pi n/10} \quad \text{for } N = 10$
- $x(n) = \cos(2\pi n/10) \quad \text{for } N = 10$

and plot the magnitude of each of the DFT's.

In [7]:
```
# first case
```

In [8]:
```
# second case
```

In [9]:
```
# third case
```

In [10]:
```
# fourth case
```

3. Derive simple closed-form analytical expressions for the DFT (not the DTFT!) of each signal.

insert your answer here

Exercise 3.2: Computing the Inverse DFT

1. Write a Python function for computing the inverse DFT of (4).

In [11]:
```
def IDFTsum(X):
    '''
    Parameters:
    ---
    X: the N point vector containing the DFT

    Returns:
    ---
    x: the corresponding time-domain signal
    '''
    x = None
    return x
```

2. Use `IDFTsum` to invert each of the DFT's computed in the previous problem. Plot the magnitudes of the inverted DFT's, and verify that those time-domain signals match the original ones. Use `np.real()` to eliminate any imaginary parts which roundoff error may produce.

In [12]:
```
# insert your code here
```

Exercise 3.3: Matrix Representation of the DFT

The DFT of (3) can be implemented as a matrix-vector product. To see this, consider the equation

$$X = Ax \tag{14}$$

where A is an $N \times N$ matrix, and both X and x are $N \times 1$ column vectors. This matrix product is equivalent to the summation

$$X_k = \sum_{n=0}^{N-1} A_{kn} x_n \tag{15}$$

where A_{kn} is the matrix element in the kth row and nth column of A. By comparing equations (3) and (15) we see that for the DFT,

$$A_{kn} = e^{-j2\pi kn/N} \tag{16}$$

1. Write a Python function for computing the $N \times N$ DFT matrix A in equation (16).

In [13]:
```
def DFTmatrix(N):
    """
    Parameters:
    ---
    N: N point DFT

    Returns:
    ---
    A: an N x N DFT matrix
    """
    A = None
    return A
```

2. Print out the matrix A for $N = 5$.

In [14]:
```
# insert your code here
```

3. Use the matrix A to compute the DFT of the following signals.

- $x(n) = \delta(n) \quad$ for $N = 10$
- $x(n) = 1 \quad$ for $N = 10$
- $x(n) = e^{j2\pi n/N} \quad$ for $N = 10$

In [15]:
```
# insert your code here
```

4. Plot the magnitude plots of these 3 DFTs.

In [16]:
```
# the DFT of the first signal
```

In [17]:
```
# the DFT of the second signal
```

In [18]:
```
# the DFT of the third signal
```

5. How many multiplies are required to compute an N point DFT using the matrix method (Consider a multiply as the multiplication of either complex or real numbers.)

insert your answer here

Exercise 3.4: Matrix Representation of the Inverse DFT

As with the DFT, the inverse DFT may also be represented as a matrix-vector product

$$x = BX$$

1. Write an analytical expression for the elements of the inverse DFT matrix `B` , using the form of equation (16).

insert your answer here

2. Write a Python function for computing the $N \times N$ inverse DFT matrix B.

In [19]:
```
def IDFTmatrix(N):
    """
    Parameters:
    ---
    N: N-point IDFT

    Returns:
    ---
    B: the N x N inverse DFT matrix
    """
    B = None
    return B
```

3. Print out the matrix B for $N = 5$.

In [20]:
```
# insert your code here
```

4. Compute the matrices A for $N = 5$. Then compute and print out the elements of $C = BA$.

In [21]:
```
# insert your code here
```

5. What form does C have? Why does it have this form?

insert your answer here

Exercise 3.5: Computation Time Comparison

Although the operations performed by `DFTsum(x)` are mathematically identical to a matrix product, the computation times for these two DFT's in Python are quite different. (This is despite the fact that the computational complexity of two procedures is of the same order!) This exercise will underscore why you should try to avoid using *for* loops in Python, and wherever possible, try formulate your computations using matrix/vector products.

To see this, do the following:

1. Compute the signal $x(n) = \cos(2\pi n/10)$ for $N = 512$.

In [22]:
```
# insert your code here
```

2. Compute the matrix A for $N = 512$.

In [23]:
```
# insert your code here
```

3. Compare the computation time of `DFTsum(x)` with a matrix implementation `X = A.dot(x)` by using the *time* function from *time* library before and after the program execution (See the example below). Do not include the computation of A in your timing calculations.

Report the time required for each of the two implementations.

```
t1 = time.time()
# program execution
t2 = time.time()
print(f"time taken: {t2 - t1:.4f}")
```

In [24]:
```
# insert your code here
```

4. Which method is faster? Which method requires less storage?

insert your answer here