A2: Sinusoids and DFT

Audio Signal Processing for Music Applications

Introduction

The second programming assignment is for you to get a better understanding of some basic concepts in audio signal processing related with the Discrete Fourier Transform (DFT). You will write snippets of code to generate sinusoids, to implement the DFT and to implement the inverse DFT. There are five parts in this assignment. 1) Generate a sinusoid, 2) Generate a complex sinusoid, 3) Implement the DFT, 4) Implement the IDFT and 5) Compute the magnitude spectrum of an input sequence. The fifth part of this assignment is optional and will not contribute to the final grade.

Relevant Concepts

Sinusoid: A real sinusoid in discrete time domain can be expressed by:

$$x[n] = A\cos(2\pi fnT + \varphi) \tag{1}$$

where, x is the array of real values of the sinusoid, n is an integer value expressing the time index, A is the amplitude value of the sinusoid, f is the frequency value of the sinusoid in Hz, T is the sampling period equal to $1/fs$, fs is the sampling frequency in Hz, and φ is the initial phase of the sinusoid in radians.

Complex sinusoid: A complex sinusoid in discrete time domain can be expressed by:

$$\bar{x}[n] = Ae^{j(\omega nT+\varphi)} = A\cos(\omega nT + \varphi) + jA\sin(\omega nT + \varphi) \tag{2}$$

where, $\bar{x}$ is the array of complex values of the sinusoid, n is an integer value expressing the time index, A is the amplitude value of the sinusoid, e is the complex exponential number, ω is the frequency of the sinusoid in radians per second (equal to $2\pi f$), T is the sampling period equal $1/fs$, fs is the sampling frequency in Hz and φ is the initial phase of the sinusoid in radians.

Discrete Fourier Transform (DFT): The N point DFT of a sequence of real values x (a sound) can be expressed by:

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N} \qquad k = 0, ..., N-1 \tag{3}$$

where n is an integer value expressing the discrete time index, k is an integer value expressing the discrete frequency index, and N is the length of the DFT.

Inverse Discrete Fourier Transform: The IDFT of a spectrum X of length N can be expressed by:

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{j2\pi kn/N} \qquad n = 0, ..., N-1 \tag{4}$$

where, n is an integer value expressing the discrete time index, k is an integer value expressing the discrete frequency index, and N is the length of the spectrum X.

Magnitude spectrum: The magnitude of a complex spectrum X is obtained by taking its absolute value: $|X[k]|$

Part-1: Generate a sinusoid (*2 points*)

Complete the function `genSine(A, f, phi, fs, t)` in the file `A2Part1.py` to generate a real sinusoid (use np.cos()) given its amplitude A, frequency f (Hz), initial phase phi (radians), sampling rate fs (Hz) and duration t (seconds).

All the input arguments to this function (A, f, phi, fs and t) are real numbers such that A, t and fs are positive, and fs > 2*f to avoid aliasing. The function should return a numpy array x of the sinusoid generated using Equation 1.

if you run your code using `A=1.0`, `f = 10.0`, `phi = 1.0`, `fs = 50.0` and `t = 0.1`, the output `x` should be the following numpy array: `array([ 0.54030231, -0.63332387, -0.93171798, 0.05749049, 0.96724906])`

```
def genSine(A, f, phi, fs, t):
    """
    Inputs:
        A (float) =  amplitude of the sinusoid
        f (float) = frequency of the sinusoid in Hz
        phi (float) = initial phase of the sinusoid in radians
        fs (float) = sampling frequency of the sinusoid in Hz
        t (float) =  duration of the sinusoid (is second)
    Output:
        The function should return a numpy array
        x (numpy array) = The generated sinusoid (use np.cos())
    """
    ## Your code here
```

Part-2: Generate a complex sinusoid (*2 points*)

Complete the function `genComplexSine(k, N)` in the file `A2Part2.py` to generate the complex sinusoid that is used in DFT computation of length N (samples), corresponding to the frequency index k. Note that the complex sinusoid used in DFT computation has a negative sign in the exponential function.

The amplitude of such a complex sinusoid is 1, the length is N, and the frequency in radians is $2\pi k/N$.

The input arguments to the function are two positive integers, `k` and `N`, such that $k < N - 1$. The function should return `cSine`, a numpy array of the complex sinusoid.

If you run your function using `N=5` and `k=1`, the function should return the following numpy array `cSine`: `array([1.0 + 0.j, 0.30901699 - 0.95105652j, -0.80901699 - 0.58778525j, -0.80901699 + 0.58778525j, 0.30901699 + 0.95105652j])`

```
def genComplexSine(k, N):
    """
    Inputs:
        k (integer) = frequency index of the complex sinusoid
                      of the DFT
        N (integer) = length of complex sinusoid in samples
    Output:
        The function should return a numpy array
        cSine (numpy array) = The generated complex sinusoid
                              (length N)
    """
    ## Your code here
```

Part-3: Implement the discrete Fourier transform (DFT) (*3 points*)

Complete the function `DFT(x)` in the file `A2Part3.py` so that it implements the discrete Fourier transform (DFT). Given a sequence x of length N, the function should return its DFT of length N, its spectrum, with the frequency indexes ranging from 0 to $N-1$.

The input argument to the function is a numpy array x and the function should return a numpy array `X`, which is the DFT of `x`.

If you run your function using `x = np.array([1, 2, 3, 4])`, the function shoulds return the following numpy array: `array([ 10.0 + 0.0j, -2. +2.0j, -2.0 - 9.79717439e-16j, -2.0 - 2.0j])`

Note that you might not get an exact 0 in the output because of the small numerical errors due to the limited precision of the data in your computer. Usually these errors are of the order 10^{-15} depending on your machine.

```
def DFT(x):
    """
    Input:
        x (numpy array) = input sequence of length N
    Output:
        The function should return a numpy array of length N
        X (numpy array) = The N point DFT of the input sequence x
    """
    ## Your code here
```

Part-4: Implement the inverse discrete Fourier transform (IDFT) (*3 points*)

Complete the function `IDFT(X)` in the file `A2Part4.py` so that it implements the inverse discrete Fourier transform (IDFT). Given a frequency spectrum X

of length N, the function should return its IDFT x, also of length N. Assume that the frequency index of the input spectrum ranges from 0 to $N-1$.

The input argument to the function is a numpy array `X` of the frequency spectrum and the function should return a numpy array `x`, the IDFT of `X`. Remember to scale the output appropriately.

If you run your function using `X = np.array([1 ,1 ,1 ,1])`, the function should return the following numpy array `x`:

`array([ 1.0 +0.0 j, -4.5924255e-17 +5.5511151e-17j, 0.000000e+00 +6.12323400e-17j, 8.22616137e-17 +8.32667268e-17j])`

Notice that the output numpy array is essentially [1, 0, 0, 0]. Instead of exact 0 we get very small numerical values of the order of 10^{-15}, which can be ignored. Also, these small numerical errors are machine dependent and might be different in your case.

In addition, an interesting test of the IDFT function can be done by providing the output of the DFT of a sequence as the input to the IDFT. See if you get back the original time domain sequence.

```
def IDFT(X):
    """
    Input:
        X (numpy array) = frequency spectrum (length N)
    Output:
        The function should return a numpy array of length N
        x (numpy array) = The IDFT of the frequency spectrum X
                          (length N)
    """
    ## Your code here
```

Part-5: Compute the magnitude spectrum (*optional*)

Complete the function `genMagSpec(x)` in the file `A2Part5.py` so that it that computes the magnitude spectrum of an input sequence x of length N. The function should return an N point magnitude spectrum with frequency index ranging from 0 to $N-1$.

The input argument to the function is a numpy array `x` and the function should return a numpy array `magX`, of the magnitude spectrum of `x`.

If you run your function using `x = np.array([1, 2, 3, 4])`, the function should return the following numpy array `magX`: `array([10.0, 2.82842712, 2.0, 2.82842712])`

```
def genMagSpec(x):
    """
    Input:
        x (numpy array) = input sequence of length N
    Output:
        The function should return a numpy array
        magX (numpy array) = The length N magnitude spectrum of
                             the input sequence x
```

```
    """
    ## Your code here
```

Grading

Only the first four parts of this assignment are graded and the fifth part is optional. The maximum number of points obtained in this assignment is 10.