4

DTFT Lab

©2021 T. Holton (tholton@sfsu.edu)

4.1 Purpose

The purpose of this laboratory is to learn how to compute and plot the discrete-time Fourier transform (DTFT). The Fourier transform has some interesting properties, particularly the transform of real sequences (which is all we are generally interested in working on anyway). These symmetry properties are of practical as well as theoretical interest.

4.2 Background

To prepare for this exercise, review Chapter 4. Here is a little theoretical work you should do and remember:

1. Given that $x[n]$ is real (that is, $x[n] = x^*[n]$)
   - Show that $X(\omega)$ is conjugate symmetric (that is, $X(\omega) = X^*(-\omega)$).
   - Show that if $X(\omega)$ is conjugate symmetric implies that $|X(\omega)| = |X(-\omega)|$ that is, the magnitude is an even function of ω and $\measuredangle X(\omega) = -\measuredangle X(-\omega)$, that is, the phase is an odd function of ω . This result has a very practical consequence. It says that the only unique part of the DTFT of a real sequence is that part for $\omega \ge 0$. If you know, say, the magnitude of the Fourier transform for all $\omega > 0$, then you automatically know the magnitude for all $\omega < 0$, since the magnitude is an even function of ω , $|X(\omega)| = |X(-\omega)|$. Similarly, if you know the phase of the Fourier transform for all $\omega > 0$, then you automatically know the phase for all $\omega < 0$, since the phase is an odd function of ω , $\measuredangle X(\omega) = -\measuredangle X(-\omega)$.
2. Show that you can write any $x[n]$ as the sum of an even sequence and an odd sequence, $x[n] = x_e[n] + x_o[n]$, where $x_e[n]$ is a sequence such that $x_e[n] = x_e[-n]$, and $x_o[n]$ is a sequence such that $x_o[n] = -x_o[-n]$. Further show that you can compute

$$\begin{aligned} x_e[n] &= \frac{x[n]+x[-n]}{2} \\ x_o[n] &= \frac{x[n]-x[-n]}{2} \end{aligned}.$$

3. Taking the discrete-time Fourier transform, it follows that $X(\omega) = X_e(\omega) + X_o(\omega)$, where $X_e(\omega)$ and $X_o(\omega)$ are respectively the even and odd parts of $X(\omega)$. If $x[n]$ is real, show that $X_e(\omega)$ is a real and even function of ω and $X_o(\omega)$ is a purely imaginary and odd function of ω . Further, show that if $x[n]$ is a real and even sequence, then $X(\omega)$ is real and even, too. Verify this by considering the Fourier transform of some simple sequences which are even and odd. For example, here are some even sequences:
   - $x[n] = \delta[n+1] + \delta[n] + \delta[n-1]$
   - $x[n] = \cos \pi n/4$

   And here are some odd sequences:

- $x[n] = \delta[n+1] - \delta[n-1]$
- $x[n] = \sin \pi n/4$

4.3 Assignment

4.3.1 Even and odd

First, you will develop a few very simple utility functions to derive the even and odd parts from an input sequence. We will assume that the sequences are specified in the manner defined in Laboratory #1, namely as sequence objects with the data array in `x.data`, and an offset integer in `x.offset`. Here are the functions:

```
function y = even(x)
% EVEN   Take a Matlab sequence object, x, which is possibly complex, and create
%        a Matlab sequence structure, y, that corresponds to the even part

function y = odd(x)
% ODD    Take a Matlab sequence object, x, which is possibly complex, and create
%        a Matlab sequence structure, y, that corresponds to the odd part

function y = trim(x)
% TRIM Take a Matlab sequence object and remove leading and trailing zeros,
%      appropriately adjusting offset
```

For example, given x

```
» x = sequence([1 4 3 -2 6], -1);
» xe = even(x)
xe =
     data: [3 -1 2 4 2 -1 3]
   offset: -3

» xo = odd(x)
xo =
     data: [-3 1 -1 0 1 -1 3]
   offset: -3

» xe + xo
ans =
     data: [1 4 3 -2 6]
   offset: -1
```

These functions should be methods inside *sequence.m*.

4.3.2 Conjugate

Create a function to take the conjugate of a sequence structure. Since this function overloads the Matlab `conj` function for doubles, it should also be inside the methods block of *sequence.m*.

```
function y = conj(x)
% CONJ Take a Matlab sequence object and return the complex conjugate
```

For example, given x

```
» x = sequence([1+1j 0 1-1j], -1);
» xconj = conj(x)
xconj =
     data: [1-1i  0+0i  1+1i]
   offset: -1
```

4.3.3 DTFT

Next, you will write a function to perform the DTFT:

```
function y = dtft(x, w)
% DTFT   Evaluate the DTFT of Matlab sequence object, x, at radial frequencies
%        given by double array w. Return a double array, y.
```

For example, start with our old friend, sequence $x[n] = \delta[n+1] + \delta[n] + \delta[n-1]$:

```
» x = sequence([1 1 1], -1);
```

Make an array of frequency from $-\pi$ to π. This is most easily done using Matlab's `linspace` command. Choose an odd number of points so the center point will correspond to a frequency of $\omega = 0$. For example, the following command produces an array of 11 points spanning the range from $-\pi$ to π:

```
» w = linspace(-pi, pi, 11)
w =
  Columns 1 through 7
  -3.1416 -2.5133 -1.8850 -1.2566 -0.6283 0 0.6283
  Columns 8 through 11
  1.2566 1.8850 2.5133 3.1416
```

and then, call your `dtft` function with the sequence object, `x`, and frequency array, `w`:

```
» dtft(x, w)
ans =
   Columns 1 through 7
   -1.0000 -0.6180 0.3820 1.6180 2.6180 3.0000 2.6180
   Columns 8 through 11
   1.6180 0.3820 -0.6180 -1.0000
```

In this case, the result is purely real, but in general, it will be complex.

Since Matlab supports full complex arithmetic, it is extremely easy to compute the DTFT from the definition of the DTFT,

$$X(\omega) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}$$

While you can implement this equation any way you like, if you are clever, you can do it using only a single, well-written line of Matlab code. No for loop is required, only a matrix multiplication. Here's one way to think about the problem. You are trying to form Express $X(\omega)$ as $\mathbf{X}$, a row vector of length M (in the example above, $M = 11$). Express $x[n]$ as $\mathbf{x}$, is a row vector of length N (in the example above, $N = 3$). Now make an $N \times M$ matrix, $\mathbf{Q}$, which implements ωn. No for loops necessary! Then $\mathbf{X} = \mathbf{x}\exp(-j\mathbf{Q})$.

4.3.4 Real and imaginary parts

Most computer languages do not natively support complex arithmetic, nor do they support vectorized operations as Matlab does. Hence let's write another, more primitive, short function that will return the real and imaginary part the DTFT in a structure comprising two fields, `y.real` and `y.imag`:

```
function y = dtft2(x, w)
% DTFT2   Evaluate the DTFT of Matlab sequence object, x, at frequencies given
%         by array w. Return values are a structure with
%         y.real (real part) and y.imag (imaginary part)
```

In this function, you cannot use the `j` operator, or any explicitly complex numbers or notations (e.g. `1j`, `sqrt(-1)`, ...). Nor can you use vectorized operations. That means you'll need to implement this with nested `for` loops, `sine` and `cosine`. The point of this exercise is to demonstrate that you understand how to decompose the DTFT into its real and imaginary parts.

```
» x
x =
     data: [1 4 3 -2 6]
   offset: -1

» y = dtft2(x, w)
y =
    real: [-8 2.0000 9.2361 2 4.7639 12 4.7639 2 9.2361 2.0000 -8]
    imag: [0 8.7840 -0.4490 -2.8002 4.9798 0 -4.9798 2.8002 0.4490 -8.7840 0]
```

4.3.5 Magnitude and phase

Magnitude is simply computed by the Matlab `abs` function. The phase is computed with the `angle` function. Please write your own little function which computes magnitude and phase without using `abs` or `angle`. The input will be the structure comprising `y.real` and `y.imag` that we computed in the previous part and we will return the real and imaginary part the DTFT in a structure comprising two fields, `y.mag` and `y.phase`.

```
function y = mag_phase(x)
% MAG_PHASE Input argument is a structure with x.real and x.imag
%         Return values are y.mag (magnitude) and y.phase (phase in radians)
```

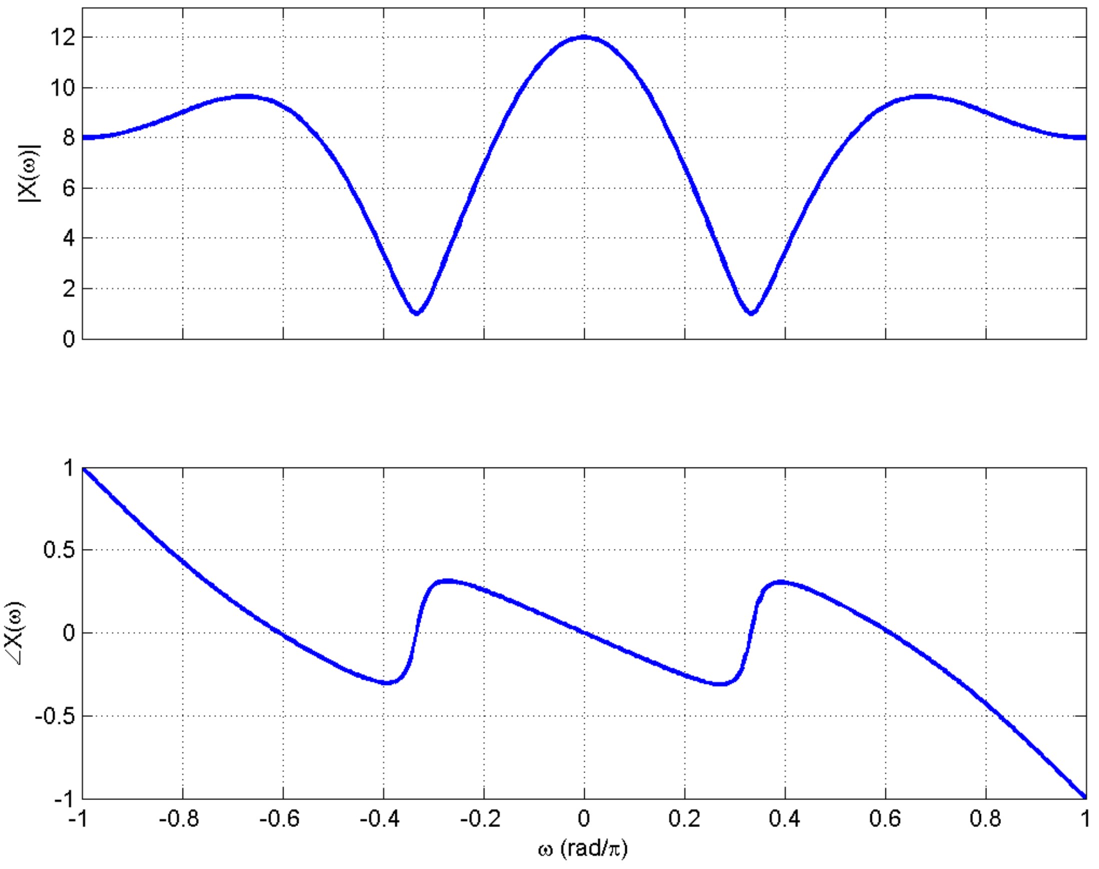
For example, given the data in the previous example,

```
» z = mag_phase(dtft2(x, w));
z =
      mag: [8 9.0088 9.2470 3.4411 6.8915 12 6.8915 3.4411 9.2470 9.0088 8]
    phase: [3.1416 1.3469 -0.0486 -0.9506 0.8075 0 -0.8075 0.9506 0.0486 -1.3469
3.1416]
```

Now, we need a routine to plot the magnitude and phase of the DTFT:

```
function plot_magph(x, w)
% PLOT_MAGPH Plot the magnitude and phase of Matlab sequence object, x
%            at frequencies given by array w
» w = linspace(-pi, pi, 1001);
» plot_magph(x, w)
```

Here's the resulting plot:



Note a few things:

- Please put your `dtft`, `dtft2`, `mag_phase` and `plot_magph` functions in your working directory, i.e., NOT in *sequence.m*.
- Choose a nice big number of points (1001) so the plots look continuous.
- The abscissa limits of both the magnitude and phase plots are from -1 to 1, corresponding to $-\pi$ to π.
- The ordinate limits of the magnitude plot are from 0 to the maximum of the magnitude.
- The ordinate limits of the phase plot are from -1 to 1, corresponding to $-\pi$ to π.
- The plots are labelled. To implement this plot, you will have to learn about a few of the Matlab plotting functions.

A few modifications of the plot are easy, and can be learned by looking at Matlab's help files:
`plot`
`grid`
`xlabel`
`ylabel`
`axis` (specifically `axis tight`)

More subtle control of the plot is possible with the Matlab `set` and `get` functions. Some of the characteristics of the plot are associated with the axis. For example, to turn the abscissa labels off in the magnitude plot, you can do:

```
    set(gca, 'XtickLabel', '');
```

`gca` is 'get current axis', the internal Matlab variable that is shorthand for the 'handle' of the current axis, namely, the one that you are currently plotting into using the subplot command. Obviously, you have to be working on the magnitude axis before you issue this command, or you have to have access to the handle to the axis. To see a full range of the parameters that you can adjust with the `set` and `get` functions, type `get(gca)`.

Some features of the plot, such as the thickness and color of the lines of the plot are governed by the plotting command
itself. For example, to get a thicker line in the plot, you can do:

```
    plot(x, y, 'LineWidth', 2);
```

You can also use the `set` and `get` functions to change the parameters of the plot, such as the line width and color, even after it has been drawn. In this case, one would do the following:

```
    h = plot(x, y)
    set(h, 'LineWidth', 2, 'Color', 'r');
```

Here, the variable, `h`, is the 'handle' to the plot. If you type `get(h)`, you will see the full range of parameters in the plot that you can adjust using the `set` and `get` functions. These include the data in

©2021 T. Holton (tholton@sfsu.edu)

the plot, as well as the line width and color. Note also that the label on the frequency axis has nice Greek characters for π and ω. You can dig into the documentation and figure out how to do this, if you wish.

4.3.6 Putting it together

You should now be able to use all the functions you have written for this laboratory and put them together to investigate the claims made at the beginning of this write-up. For example, let's create a sequence as before:

```
» x =
     data: [1 4 3 -2 6]
   offset: -1
» w = linspace(-pi, pi, 1001);
» plot_magph(x, w);
» plot_magph(even(x), w);
» plot_magph(odd(x), w);
```

- Verify that the magnitude of the all plots is even and the phase is odd.
- Verify that the DTFT of the even function is purely real and even.
- Verify that the DTFT of the odd function is purely imaginary and odd. How can you tell by looking at the magnitude and phase that this is a purely imaginary and odd function?

Download the lab4.zip file and put them in the same directory as your `dtft`, `dtft2`, `mag_phase` and `plot_magph` functions. Then `publish lab4` to a pdf file