ECEN220: Signals & Systems
Lab 4: The Discrete Time Fourier Transform

Due: 1:10 p.m., Thursday, 26 September 2019

1 Computing the DTFT in MATLAB

Suppose for a finite signal $x[n]$ we wish to plot the discrete-time Fourier transform $X(e^{j\omega})$ over the range $\omega_0 \leq \omega \leq \omega_M$. Now, ω is a continuous variable, but MATLAB can only deal with a finite number of values of ω. We overcome this problem by sampling the frequency axis. Let ω_k be the k-th sample of ω. The sampled frequency axis is then generated from:

```
omega0=-2*pi; %first frequency sample
omegaM=2*pi;  %last frequency sample
k=(0:M)';
omega=omega0+(omegaM-omega0)*k/M;
```

The discrete-time Fourier transform is

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

This can be implemented using vector operations. At the k-th frequency sample, $X(e^{j\omega k})$ is given by the vector inner product

$$X(e^{j\omega_k}) = \begin{bmatrix} e^{-j\omega_k N_1} & \dots & e^{-j\omega_k N_2} \end{bmatrix} \begin{bmatrix} x[N_1] \\ \vdots \\ x[N_2] \end{bmatrix}$$

Here the n-th term of the row vector is multiplied by the n-th term of the column vector and the products are summed. n runs from N_1 to N_2.

To determine the DTFT at all frequencies we can replace the above vector-vector multiplication with a vector-matrix multiplication

$$\begin{bmatrix} X(e^{j\omega_0}) \\ \vdots \\ X(e^{j\omega_M}) \end{bmatrix} = \begin{bmatrix} e^{-j\omega_0 N_1} & \dots & e^{-j\omega_0 N_2} \\ \vdots & & \vdots \\ e^{-j\omega_M N_1} & \dots & e^{-j\omega_M N_2} \end{bmatrix} \begin{bmatrix} x[N_1] \\ \vdots \\ x[N_2] \end{bmatrix}$$

In other words, if we put the signal $x[n]$ into the column vector $\mathbf{x}$ and the frequency-sampled DTFT $X(e^{j\omega k})$ into the column vector $\mathbf{X}$, then the DTFT is given by the matrix-vector multiplication

$$\mathbf{X} = \mathbf{V}\mathbf{x} \tag{1}$$

where the (k, ℓ)-th entry of matrix $\mathbf{V}$ is given by $e^{-j\omega_k n_\ell}$.

$\mathbf{V}$ can be calculated using the vector outer product

$$V = \exp(-j\boldsymbol{\omega}\mathbf{n}^T)$$

where $\mathbf{n}$ is a column vector containing the discrete time points and $\boldsymbol{\omega}$ is a column vector containing the frequency samples. This can be achieved with the following code (assuming x, n and omega are already defined)

```
V=exp(-j*omega*n'); %the V matrix
X=V*v; % the DTFT
```

1a) Package the above code into a function with the calling syntax

```
function [X] = dtft( x, n, omega )
% Computes the DTFT
% X = dtft( x, n, omega )
% where X = DTFT at each value of omega
% x = samples of a discrete-time signal
% n = time indices for the signal
% omega = vector of frequency samples
```

You will need to save this in a separate file called `dtft.m`

1b) Using your function compute and plot the DTFT of the following signal over $-2\pi \leq \omega \leq 2\pi$

$$x[n] = -3, \underset{\uparrow}{2}, 5, 6, 4$$

```
x = [-3,2,6,6,4]';
```

```
n = [0:length(x)-1]'-1;
```

(The arrow denotes the location of $n = 0$.) To do this you will need to set up the `x`, `n` and `omega` *column* vectors in a script file and call the `dtft()` function. What do you think is a reasonable number M of frequency samples? Is $X(e^{j\omega})$ complex? If so, plot the magnitude and phase on separate graphs.

(If you want you can use the subplot command to create two plots in one window.)

1c) Describe what happens to $X(e^{j\omega})$ when the location of $n = 0$ changes.

2 The inverse DTFT in MATLAB

The inverse discrete-time Fourier transform equation is

$$x[n] = \frac{1}{2\pi} \int_{2\pi} X(e^{j\omega}) e^{j\omega n} \, d\omega$$

This is rarely used in practice to find the DTFT; a table of transform pairs is often used. One approach to implement this equation in MATLAB would be to use numerical integration.

An alternative is to solve the matrix equation (1) for x. $\mathbf{V}$ is not necessarily square, so it is not necessarily invertible. However, if we have more frequencies than time points, it can be solved in a least square since by `x = pinv(V)*X` or (better) by `x = V\X`

2a) Define a function with the calling syntax function `x = invdtft(X,n,omega)`

2b) Apply your `invdtft()` function to the Fourier transform $X(e^{j\omega})$ you calculated in 1b).

Use the stem command to plot the signal. Does it match the original sequence Are there any imaginary components? Are they significant? Where do you think they come from?

3 Filtering

Thanks to the convolution property, filtering of a signal becomes a simple multiplication operation in the frequency domain. The Fourier transform of a filtered signal $Y(e^{j\omega})$ is related to the Fourier transform of the signal $X(e^{j\omega})$ and frequency response of the filter $H(e^{j\omega})$ by

$$Y(e^{j\omega}) = H(e^{j\omega})X(e^{j\omega})$$

Here we will apply some filters to some NASA climate data. The file Temperature.txt contains two columns. The first column is the year and the second column is the change in global temperature in °C relative to the 1951–1980 average.

3a) Import the data into column vectors `n` and `x` using the following code

```
data=importdata('Temperature.txt');
n=data(:,1);
x=data(:,2);
```

Then using the same approach used earlier, calculate the DTFT, `X`, of the signal `x`.

Plot the magnitude and phase of X on the frequency range $-\pi \leq \omega \leq \pi$.

3b) The DTFT of a moving-average filter of length m is

$$H(e^{j\omega}) = \begin{cases} 1 & \text{for } \omega = 0 \\ \frac{1}{m} e^{-j\omega(m-1)/2} \left(\frac{\sin(\omega m/2)}{\sin(\omega/2)} \right) & \text{otherwise} \end{cases}$$

Create a column vector `Hav` containing these values using the same frequency scale as in 3a). Plot the magnitude and phase of `Hav` for $m = 5$.

3c) Now perform an element-wise multiplication of the vectors `Hav` and `X` to get the output Fourier transform vector `Y` using `Y=Hav.*X;` Finally perform an inverse DTFT using the function you made in 2a) and plot the output `y` and the input `x` as a function of time on the same graph. Such a plot can be made using `plot(n,x,n,y)`

What does changing m do?

Optional

If time permits, create a high-pass filter that eliminates frequencies below a certain value ω_c . An ideal high-pass filter is given by

$$H(e^{j\omega}) = \begin{cases} 0 & \text{for } |\omega| < \omega_c \\ 1 & \text{for } -\pi \leq \omega \leq -\omega_c \\ 1 & \text{for } \omega_c \leq \omega \leq \pi \end{cases}$$

Apply your filter to the climate data and see what you get. You might like to remove the large low-frequency peaks in X for example. Is there a possibility that low-frequency oscillations account for part of the observed increase in temperature?

Hand in your code, plots and answers to questions