# ECE 3300
# MATLAB Assignment 4

This assignment gives an overview of matrix manipulation techniques in MATLAB. It then explains how to use MATLAB to determine the Fourier transform of continuous-time and discrete-time signals. The approach simply implements sums directly and implements integrals as sums of rectangles, similar to the approach of previous assignments.

It should be noted that MATLAB has an implementation of the Fast Fourier Transform (FFT) that is significantly more computationally efficient. We choose not to use the FFT approach because it requires additional theory normally taught in a Digital Signal Processing (DSP) course and because applying the FFT to the Fourier transform requires additional manipulations of the FFT output that are difficult to explain simply. The approach presented in this assignment, although computationally inefficient, works quite satisfactorily and is relatively simple to understand.

**Matrices in Matlab**

A matrix in Matlab is essentially a two-dimensional list. The command `b=[1 2 3; 4 5 6]` produces the matrix

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

The *transpose* of a Matrix converts the first row into the first column, the second row into the second column, and so on. In mathematics, the transpose of a matrix $A$ is denoted $A^T$. In MATLAB, a transpose is obtained by following the variable with a period followed by an apostrophe, `.'`. For example, with `b` as defined above, the command `b.'` produces the matrix

$$\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

Matrix multiplication produces a matrix in which the element in the $i$th row and $j$th column is equal to the sum of the element-by-element products of the $i$th row of the first matrix and the $j$th column of the second. Two examples are

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} = \begin{bmatrix} (1)(1)+(2)(2)+(3)(3) & (1)(4)+(2)(5)+(3)(6) \\ (4)(1)+(5)(2)+(6)(3) & (4)(4)+(5)(5)+(6)(6) \end{bmatrix} = \begin{bmatrix} 14 & 32 \\ 32 & 77 \end{bmatrix}$$

and

$$\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} (1)(1)+(4)(4) & (1)(2)+(4)(5) & (1)(3)+(4)(6) \\ (2)(1)+(5)(4) & (2)(2)+(5)(5) & (2)(3)+(5)(6) \\ (3)(1)+(6)(4) & (3)(2)+(6)(5) & (3)(3)+(6)(6) \end{bmatrix} = \begin{bmatrix} 17 & 22 & 27 \\ 22 & 29 & 36 \\ 27 & 36 & 45 \end{bmatrix}$$

In MATLAB the asterisk `*` is used for matrix multiplication. Thus the first example is equal to `b*b.'` and the second is equal to `b.'*b`.

**Discrete-Time Fourier Transform**

The discrete-time Fourier transform of $x[n]$ is defined $X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}$. Suppose we represent $x[n]$ in MATLAB via `n` and `x`, and using vector/matrix notation let us write $\mathbf{n} = [n_1\ n_2\ \cdots\ n_L]$ and $\mathbf{x} = [x_1\ x_2\ \cdots\ x_L]$. Similarly, let us use $\mathbf{w} = [\omega_1\ \omega_2 \cdots\ \omega_K]$ to denote the frequencies at which we wish to determine the Fourier transform, and let us use $\mathbf{X} = [X_1\ X_2\ \cdots\ X_K]$ to denote the corresponding Fourier transform values at these frequencies. Using these definitions, the Fourier transform definition can be written as $X_k = \sum_{\ell=1}^{L} x_\ell e^{-jn_\ell\omega_k}$. Our goal is to determine an expression for the *vector* $\mathbf{X}$ in terms of the other vectors. We can begin by writing

$$\mathbf{n}^T\mathbf{w} = \begin{bmatrix} n_1 \\ n_2 \\ \vdots \\ n_L \end{bmatrix} \begin{bmatrix} \omega_1 & \omega_2 & \cdots & \omega_K \end{bmatrix} = \begin{bmatrix} n_1\omega_1 & n_1\omega_2 & \cdots & n_1\omega_K \\ n_2\omega_1 & n_2\omega_2 & \cdots & n_2\omega_K \\ \vdots & \vdots & \ddots & \vdots \\ n_L\omega_1 & n_L\omega_2 & \cdots & n_L\omega_K \end{bmatrix}$$

If $A$ is a matrix, we use the notation $e^A = \exp(A)$ to mean that we take the exponential function on each term in the matrix. (This is exactly what MATLAB does.) Also note that a constant times a matrix means that every element in the matrix is multiplied by that constant. Then we can write

$$\begin{aligned} \mathbf{x}\exp(-j\mathbf{n}^T\mathbf{w}) &= \begin{bmatrix} x_1 & x_2 & \cdots & x_L \end{bmatrix} \begin{bmatrix} e^{-jn_1\omega_1} & e^{-jn_1\omega_2} & \cdots & e^{-jn_1\omega_K} \\ e^{-jn_2\omega_1} & e^{-jn_2\omega_2} & \cdots & e^{-jn_2\omega_K} \\ \vdots & \vdots & \ddots & \vdots \\ e^{-jn_L\omega_1} & e^{-jn_L\omega_2} & \cdots & e^{-jn_L\omega_K} \end{bmatrix} \\ &= \begin{bmatrix} \sum_{\ell=1}^{L} x_\ell e^{-jn_\ell\omega_1} & \sum_{\ell=1}^{L} x_\ell e^{-jn_\ell\omega_2} & \cdots & \sum_{\ell=1}^{L} x_\ell e^{-jn_\ell\omega_K} \end{bmatrix} \end{aligned}$$

This is our desired list of Fourier transform values! Thus $\mathbf{X} = \mathbf{x}\exp(-j\mathbf{n}^T\mathbf{w})$. In MATLAB, this is implemented via `X=x*exp(-i*n.'*w)`.

**Continuous-Time Fourier Transform**

The continuous-time Fourier transform $X(j\omega)$ can be approximated by replacing the integral in its definition with a sum of areas of rectangles, much akin to the method used with integrals in previous assignments. Denote the list of time values and signal values as $\mathbf{t} = [t_1\ t_2\ \cdots\ t_L]$ and $\mathbf{x} = [x_1\ x_2\ \cdots\ x_L]$, respectively, and define the list of frequency values and transform values as $\mathbf{w} = [\omega_1\ \omega_2\ \cdots\ \omega_K]$ and $\mathbf{X} = [X_1\ X_2\ \cdots\ X_K]$. If the time between time samples is $b$, we can write

$$X(j\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt \approx \sum_{\ell=1}^{L} x_\ell e^{-j\omega t_\ell} b$$

and at frequency $\omega_k$ this expression becomes $X_k = \sum_{\ell=1}^{L} x_\ell e^{-j\omega_k t_\ell} b$. Except for replacing $\mathbf{n}$ with $\mathbf{t}$ and multiplying by $b$, this is identical to the discrete-time result. Thus $\mathbf{X} = \mathbf{x}\exp(-j\mathbf{t}^T\mathbf{w})b$, and in MATLAB, this is implemented via `X=x*exp(-i*t.'*w).*b`.

**Inverse Transforms**

The inverse continuous-time and discrete-time Fourier transforms are defined via

$$x(t) = \tfrac{1}{2\pi}\int_{-\infty}^{\infty} X(j\omega)e^{j\omega t}d\omega \quad \text{and} \quad x[n] = \tfrac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{j\omega})e^{j\omega t}d\omega$$

If the frequency spacing is $d$, a similar analysis shows that $x(t)$ can be obtained from $X(j\omega)$ via $\mathbf{x} = \mathbf{X}\exp(j\mathbf{w}^T\mathbf{t})\frac{d}{2\pi}$ and $x[n]$ can be obtained from $X(e^{j\omega})$ via $\mathbf{x} = \mathbf{X}\exp(j\mathbf{w}^T\mathbf{n})\frac{d}{2\pi}$. These expressions can be implemented in MATLAB with the code `x=X*exp(i*w.'*t).*d./(2.*pi)` and `x=X*exp(i*w.'*n).*d./(2.*pi)`, respectively.

However, note that whereas many signals are limited to cover a finite amount of time, fewer are limited in frequency. If a signal has infinite frequency duration, the frequency values will need to be truncated beyond a certain point, and this may introduce additional error beyond that caused by sampling. One effect of this error is that the inverse transforms may produce signals that are not purely real even if the transforms were in fact transforms of real signals. In practice, one may wish to take the real part of an inverse transform obtained in MATLAB just in case.

**Magnitude and Phase of Fourier Transforms**

Recall that a Fourier transform is complex-valued in general. We can use `abs(X)` to obtain the magnitude of the Fourier transform and `angle(X)` to obtain the phase. If phase unwrapping is desired, one should use `unwrap(angle(X))`.

**PROBLEM STATEMENT**

Suppose $x(t) = \begin{cases} 2 & \text{if } -0.5 \leq t \leq 0.5 \\ 1 & \text{if } -1 \leq t < -0.5,\ 0.5 < t \leq 1 \\ 0 & \text{otherwise} \end{cases}$.

Further suppose $y(t) = x(t+4) + x(t+2) + x(t) + x(t-2) + x(t-4)$ and
$z(t) = x(t+8) + x(t+6) + x(t+4) + x(t+2) + x(t) + x(t-2) + x(t-4) + x(t-6) + x(t-8)$.

Note that $y(t)$ is the sum of five time-delayed copies of $x(t)$ and $z(t)$ is the sum of nine time-delayed copies. Such signals are sometimes called quasi-periodic because they look like a periodic signal over a finite time duration. Recall that the Fourier transform of periodic signals consists of a series of impulses. It follows that quasi-periodic signals should have Fourier transforms that look more and more impulse-like as more and more delayed copies are added.

Throughout this assignment use a sampling time of 0.01 and a frequency spacing of 0.01. Take $t$ to go from $-10$ to 10, and take $\omega$ to go from $-10$ to 10. In each part with multiple plots on a single graph, make sure to label each transform. When plotting the phase of a Fourier transform, it is your choice whether to use phase unwrapping.

1. Plot $x(t)$. Also plot the magnitude $|X(j\omega)|$ and the phase $\angle X(j\omega)$.

2. Plot $y(t)$ and $z(t)$ on separate graphs. Also plot the magnitudes $|X(j\omega)|$, $|Y(j\omega)|$, and $|Z(j\omega)|$ on a single (third) graph. *Hint:* Can you see the beginnings of the emergence of impulse-like behavior? Note that the time spacing for the copies of $x(t)$ is $T_q = 2$. This is sometimes called the quasi-period. Likewise, the quasi-fundamental frequency is given by $\omega_q = 2\pi/T_q$. See if you can identify $\omega_q$ on your graphs. (You do not need to write anything about this; only the plots are required.)

3. In this part, we explore the effects of time delay on the phase of the Fourier transform of quasi-periodic signals. On a single graph plot the phases of the Fourier transforms of $x(t)$, $x(t-0.5)$, $x(t-1)$, and $x(t-2)$. Repeat (on a second graph) with $y(t)$, $y(t-0.5)$, $y(t-1)$, and $y(t-2)$, and repeat again (on a third graph) with $z(t)$, $z(t-0.5)$, $z(t-1)$, and $z(t-2)$. *Hint:* You should see behavior similar to that discussed about time delay and the phase of the Fourier transform in Section 4.5, although $y(t-2)$ and $z(t-2)$ may behave a little differently. If so, why is this? (You do not need to write anything about this; only the plots are required.)