University of Washington
Department of Electrical Engineering
EE 235 Lab 6 Background:
Applications of Fourier Transforms

Matlab Concepts/Functions to Review	New Matlab Concepts/Functions
• Generating a time signal & time sample vector • Computing magnitude of FFT • Plotting basics • Concatenating vectors • Creating a string variable • Using for loops • Using decision statements	• Using inverse transform **ifft** function • Compute output of an LTI system with **lsim** function • Rules for multiplication and division • Labeling a figure window

BACKGROUND SECTION

1) **Matlab Review**

Concepts/Functions	Sample Code
Generating a signal in the time domain	% x(t) = cos(πt) where 0 ≤ t ≤ 5 with Fs = 2 t = 0:(1/Fs):5; % Time samples vector x = cos(pi*t); % Actual signal vector % x(t) = cos(πt) where 0 ≤ t < 5 with Fs = 2 t = 0:(1/Fs):5 – (1/Fs); % Time samples vector x = cos(pi*t); % Actual signal vector

Computing magnitude of FFT	% Convert signal x and use N = 1024 freq samples
	N = 1024;
	x_fft = fftshift(fft(x, N));
	x_abs = abs(x_fft);
	
	% Compute frequency samples vector
	w_period = 2*pi*Fs/N;
	w = (-N/2:(N/2)-1)*w_period;

Using a 2 x 1 subplot and plotting on 1st figure	subplot(2, 1, 1); plot(……..);
Plotting a signal x vs. t	plot(t, x);
Changing axes limits	xlim([0 10]); ylim([-5 5]);

Labeling axis and adding plot title	xlabel('Time');
	ylabel('x(t)');
	title('Signal x(t)');

Vector concatenation	z = [x, y]; % horizontal: x & y must have the same # rows
	z = [x; y]; % vertical: x & y must have the same # columns

if-else Decision Statements	if x == -2
	  % Code
	elseif x > 2 && x < 5
	  % Code
	else
	  % Code
	end

for loops	for i = 1:5 % Code end
Creating string variables	str = 'Here is a string'; % Initialize string str = ['The value is', num2str(x)]; % Concatenation

2) **Function ifft: Computing Inverse Fourier Transform**

- The purpose of the Inverse Fourier Transform is to convert a signal X(jw) in the frequency domain back to signal x(t) in the time domain. We will accomplish this in Matlab by using the function **ifft**.

- Usage of **ifft**: In general, the **ifft** function simply undoes the fft function, so **x=ifft(fft(x))**. However, since we will be doing operations in the frequency domain and using functions that we want to plot, we need to use **fftshift** and account for numerical issues. Suppose we have a Fourier Transform **X_fft** that has **N** = 1024 samples in the frequency domain, that we want to filter using multiplication in the frequency domain:

```
N = 1024;
X_fft = fftshift(fft(x, N));
Y_fft = X_fft.*H_fft;
```

To convert **Y_fft** to the time domain, we would call **ifft** as such:

```
y_ifft = ifft(fftshift(Y_fft), N)
y = real(y_ifft);
```

- A call to function **fftshift** is needed to undo the shift that we performed to center the Fourier Transform **X_fft** around **w = 0**

- A call to function **real** is needed because numerical round-off errors in **fft** and **ifft** can introduce a very small nonzero imaginary component to the output **y_ifft**. In general, the time domain signals we analyze will be real-valued, so we need to remove the insignificant imaginary part. (If the imaginary part is not small, then you probably have a error somewhere.)

3) **Matlab and Filters**
   - LTI filters whose input and output satisfy an LCCDE will have the general frequency response

$$H(jw) = \frac{b_M(jw)^M + b_{M-1}(jw)^{M-1} + \cdots + b_1(jw) + b_0}{a_N(jw)^N + a_{N-1}(jw)^{N-1} + \cdots + a_1(jw) + a_0}$$

   - A filter is defined in Matlab by the numerator and denominator coefficients in H(jw).

   - The coefficients are defined as a row vectors and are listed out in the same order as the frequency response above, starting with the coefficients from the highest order of jw. Therefore, the filter coefficients for the general frequency above are defined as such:

     b = [b_M b_{M-1} … b_1 b_0]
     a = [a_N a_{N-1} … a_1 a_0]

   - As an example, suppose $H(jw) = \frac{1+2jw}{1-2jw}$. Then, the filter coefficients are given by

```
b = [2  1];
a = [-2  1];
```

   - To simulate filtering and obtain the output response in time of an LTI system to an arbitrary input, we can use the function **lsim** with one return value. The first two arguments are the filter coefficients of the numerator and the denominator, followed by the input and time samples:

```
y = lsim(b, a, x, t);
```

     Note: the time samples for output **y** will be the same as the input **x**, so we can use the vector t for the input and output signals

4) **Rules for Multiplication and Division**
   - So far, we have learned about the symbol * to perform a scalar multiplication, a type of multiplication in which a number is multiplied to every element in a matrix:

     3 * [1 2] = [3 * 1 3 * 2] = [3 6]

   - In Matlab, there are two other types of multiplication operations:
     - Element-by-element array multiplication with operator .*
     - matrix multiplication with operator *

- In elementwise array multiplication, matrices of equal dimensions are multiplied together, where each element in the first matrix is multiplied with a similarly located element in the second matrix. Element-by-element multiplication is sometimes called the Hadamard product or the Schur product.
  - Valid Example with Matrices: $\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} .* \begin{bmatrix}2 & 3\\4 & 5\end{bmatrix} = \begin{bmatrix}1*2 & 2*3\\3*4 & 4*5\end{bmatrix} = \begin{bmatrix}2 & 6\\12 & 20\end{bmatrix}$
  - Valid Example with Vectors: $\begin{bmatrix}1\\3\end{bmatrix} .* \begin{bmatrix}2\\4\end{bmatrix} = \begin{bmatrix}1*2\\3*4\end{bmatrix} = \begin{bmatrix}2\\12\end{bmatrix}$
  - Valid Example with Scalars: 1 .* 2 = 2
- If you have taken linear or matrix algebra before, then you are already familiar with the notion of matrix multiplication. In matrix multiplication, the two matrices must have a common inner dimension. This means the number of columns in the first matrix must equal the number of rows in the second matrix:
  - Valid Example : $\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} * \begin{bmatrix}2 & 3\\4 & 5\end{bmatrix} = \begin{bmatrix}1*2+2*4 & 1*3+2*5\\3*2+4*4 & 3*3+4*5\end{bmatrix} = \begin{bmatrix}10 & 13\\22 & 29\end{bmatrix}$
  - Valid Example: $\begin{bmatrix}1 & 2 & 3\end{bmatrix} * \begin{bmatrix}1\\2\\1\end{bmatrix} = [1*1+2*2+3*1] = [8]$
  - Invalid Example: $\begin{bmatrix}1\\3\end{bmatrix} * \begin{bmatrix}2 & 3\\4 & 5\end{bmatrix}$
  - Valid Example: $[1\ 3] * \begin{bmatrix}2 & 3\\4 & 5\end{bmatrix} = [1*2+3*4 \quad 1*3+3*5] = [14 \quad 18]$
- Note: scalar multiplication can be performed using the operator .*, in addition to the operator * since scalar multiplication is essentially an element-by-element multiplication with the same number. To be consistent, we will use the operator * for scalar multiplication as we have been doing, but be aware of this special exception to the rule.
- Similarly, with division, there are the following division operators:
  - scalar division with operator ./
  - element-by-element array division with operator ./
  - matrix division with operator /
- Note: unlike scalar multiplication, scalar division with / only works when both operands are scalar values. It will not work when either one is a matrix.
- Array division is an element-by-element division operator, and hence can only be performed with matrices of equal dimensions
- Matrix division is a more complex operator, one which we will not use. You just need to be aware that matrix division only works if both matrices have the same number of columns

5) **Labeling Figure Windows**

- So far, we know that the command **figure** will create a new figure window

- In some cases, we will want to go back to a previous figure window after we have already created several other figure windows. We can accomplish this by using the function **figure** with 1 argument. The argument allows you to label each figure window with a number and access a figure window later on.

- Example Code:

```
figure(1);                      % Create figure #1
subplot(2,1,1); plot(t, x);
…
figure(2);                      % Create figure #2
plot(t, y);
…
figure(1);                      % Go back and access figure #1
subplot(2,1,2); plot(t, z);
```