Module 1, B-set 2: More Frequency Domain, The SigSys Abstraction

November 3, 2016

Learning Goals

By the end of this building block, you should be able to...

- Analyze the and manipulate the frequency content of DT and CT signals using the DTFT and CTFT.
- Use the SigSys abstraction to represent systems in block diagrams.

Overview and Orientation

In the last B-set you started to deal with *signals* – both discrete time and continuous time, and both in time and in frequency. We are doing to develop a few more tools to analyze signals in time and frequency, extending frequency domain analysis to continuous time. We are also going to dive into using the SigSys abstraction to represent systems as block diagrams involving inputs and outputs.

WARNING: By the time you're 4 or 5 hours into this, you should feel confident that you can complete the B-set in another 4-5 hours. If not, this is when you should **ask for help.** This means **talk to a colleague**, or **talk to a ninja**, or **track down an instructor**, or **send an email to an instructor**.

Frequency-Domain Analysis

Discrete-time Fourier Transform (DTFT) (appx 3.5 hours including reading and exercises)

In this section, we are going to develop the Discrete-Time Fourier Transform (DTFT), which will provide better frequency resolution than the DFT, and serve as a bridge to the CT Fourier transform (CTFT), enabling us to do frequency analysis on continuous signals.

The Discrete-Time Fourier Transform (DTFT) of a DT signal is defined for any angular frequency Ω radians per sample as follows

$$X(\Omega) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\Omega n} \qquad (1)$$

The DTFT of $x[n]$, denoted by $X(\Omega)$ tells us `how much` in magnitude and phase, of a complex exponential $e^{j\Omega n}$ is present in the signal $x[n]$.

Compared to the DFT, the DTFT evaluates the frequency content in $x[n]$ at a continuum of frequencies Ω. Ω here is in units of radians/sample.

Since Ω can take a continuum of values, the inverse transform cannot take the form of a summation. The Inverse DTFT takes the form of an integral over a 2π interval. The IDTFT is defined as follows

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

$x[n]$ is therefore made up of a dense sum (i.e. an integral) of complex exponentials (frequency components), where the frequency component corresponding to $e^{j\Omega n}$ is weighted by $X(\Omega)$. In general $X(\Omega)$ can be complex.

The DTFT has a number of useful properties which are carried over from the DFT. If $W(\Omega)$ is the DTFT of $w[n]$, $X(\Omega)$ is the DTFT of $x[n]$, $Y(\Omega)$ is the DTFT of $y[n]$, we have the following properties

- If $y[n] = x[n] + w[n]$, then $Y(\Omega) = X(\Omega) + W(\Omega)$. This property enables us to decompose a signal into the sum of simpler signals, perform the DTFT on the simpler signals, and re-assemble them.
- If $y[n] = x[n - n_0]$, then $Y(\Omega) = e^{-j\Omega n_0}X(\Omega)$. This property is useful in analyzing the effects of difference equations in the frequency domain.
- If $y[n] = x[n]e^{j\Omega_0 n}$, then $Y(\Omega) = X(\Omega - \Omega_0)$. This property is useful for moving things around in frequency, like what we did in class on Monday.

The following example of a moving-average filter helps illustrate how the DTFT can be used to do understand the effects of a difference equation in the frequency domain.

1. Suppose that we have a signal $x[n]$. A 3-point moving average of this signal is given by $y[n]$, which is defined as follows

$$y[n] = \frac{1}{3}(x[n+1] + x[n] + x[n-1]). \tag{3}$$

The result of the moving average at time n is the average of the original signal at the current time, one sample ahead, and one sample before. The moving average filter has the effect of smoothing out a signal.

(a) Using the properties of the DTFT in the bullet points above, find the DTFT of $y[n]$, denoted by $Y(\Omega)$, in terms of $X(\Omega)$. Simplify your expression using $\cos\theta = \frac{1}{2}e^{j\theta} + \frac{1}{2}e^{-j\theta}$.

(b) How is the frequency content of $y[n]$ related to the frequency content of $x[n]$? Hint: think about how $Y(\Omega)$ is related to $X(\Omega)$ at $\Omega = 0$ (low frequencies) vs $\Omega = \pi$ (high frequencies).

(c) How might you actually apply the moving average filter to a given vector? In other words, in MATLAB, how would you operate on a vector $x[n]$ to produce $y[n]$? Think about three different approaches:

i. Use a for loop.
ii. Use vector indices and vector addition.
iii. Use a matrix multiply.

Choose one that makes sense to you, and implement it. Note that you'll have to think about how to deal with the terms at the edges: there is not a $n-1$ term if $n = 0$!

(d) At the MATLAB prompt, type `load handel` to load an audio segment provided with MATLAB, which you used in the last BSet. Write code to implement the moving average filter, and apply it to the audio sample in `y`. Using a for loop may be the simplest approach here. (Unfortunately, MATLAB calls this signal `y`, but this will be the input to our moving average filter). Plot the original signal `y`, and the result of applying the moving average to it on the same axes. Zoom in on a segment of the audio signal to see the effect of the moving average in the time domain. Do you see what you expected to see? Play the original signal and the result of the moving average. Does the filtered signal sound as expected? Please turn in your plots with your answers to the questions.

(e) On the same axes, plot the magnitude of the DTFT of $x[n]$ and $y[n]$. You can use the following code segment to do this.

```
X = fft(x); % Take the FFT of x
Y = fft(y); % Take the FFT of y
Nx = length(x); % Find the length of x
Ny = length(y); % Find the length of y
% make vectors for the frequency axis points in radians
% per sample
wx = linspace(-pi, pi*(Nx-1)/Nx, Nx);
wy = linspace(-pi, pi*(Ny-1)/Ny, Ny);
% plot the magnitudes on the same axes
% use fftshift to reorder the FFT
plot(wx, fftshift(abs(X)));
hold on;
plot(wy, fftshift(abs(Y)), 'm');
```

Please submit all plots you made, and write a sentence or two explaining qualitatively how the frequency content of the signal before and after the moving average filter is applied.

In the following example, we shall re-examine a problem from BSet1, where you use the DFT and IDFT to remove a tone in an audio signal (we do have a better audio clip to use in this BSet). This time, you will use the DTFT to help you design a notch filter, which is a filter that attenuates a very specific frequency by a large amount, leaving other frequencies intact. In particular, by using the DTFT to design the notch filter, we are able to place a notch at an arbitrary frequency, one which is not computed by the DFT. Besides being a useful method to remove things like 60Hz noise for instance, this example also helps improve your understanding of the DTFT in general.

Consider a system whose input $x[n]$ and output $y[n]$ are related by the following difference equation. q and Ω_0 are parameters of the system. q which is in the range $0 < q < 1$ controls the width of the notch, and Ω_0 is a constant parameter that controls which frequencies the notch is at. The difference equation for the notch filter is

$$y[n] = Ay[n-1] + By[n-2] + Px[n] + Qx[n-1] + Rx[n-2]$$

where

$$\begin{aligned}A &= 2q\cos(\Omega_0)\\ B &= -q^2\\ P &= 1\\ Q &= -2\cos(\Omega_0)\\ R &= 1\end{aligned}$$

The frequency response of this filter $H(\Omega)$ is

$$H(\Omega) = \frac{\left(1 - e^{-j(\Omega_0+\Omega)}\right)\left(1 - e^{-j(\Omega-\Omega_0)}\right)}{\left(1 - qe^{-j(\Omega_0+\Omega)}\right)\left(1 - qe^{-j(\Omega-\Omega_0)}\right)} \tag{4}$$

so that the value of the output at a given frequency Ω is given by $Y(\Omega) = H(\Omega)X(\Omega)$. This magnitude of this response is shown in Figure 1. The closer q is to 1, the narrower the notch. Can you see why?is While we don't expect you to `derive` this picture, there's a video link on the BSet 2 page that shows you how we get this picture from the difference equation.

2. Load the wav files Disturbance275.wav, Disturbance261.wav, and Disturbance.wav in MATLAB using

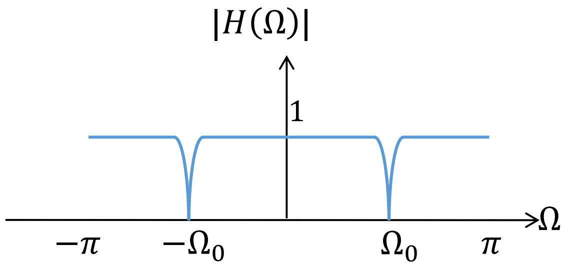

Figure 1: Frequency Response of Notch Filter.

```
[x275, Fs] = audioread('Disturbance275.wav');
[x261, Fs] = audioread('Disturbance261.wav');
[x_orig, Fs] = audioread('DisturbanceOrig.wav');
```

- `x_orig` contains an audio sample from an obscure movie from the late 1970s. This is the original, **uncorrupted** signal.
- `x275` contains the same sample corrupted by a noise signal in the form of a cosine at 275.6181Hz (this value is carefully selected so that the two frequency components in the cosine are frequencies that the DFT computes in this case).
- `x261` contains the same sample corrupted by a noise signal in the form of a cosine at 261.626Hz (this is the frequency of the middle 'C' on a piano, no other significance here).

Play all three sound files using the following commands in MATLAB, making sure that you wait until the first is done playing before running the second command.

```
sound(x_orig, Fs);
sound(x275, Fs);
sound(x261, Fs);
```

On the same axes, plot the magnitude of the FFT of the three signals using the following code segment. The frequencies are given in Hz here.

```
f = linspace(-Fs/2, Fs/2*(length(x261)-1)/length(x261),
length(x261));
X275 = fft(x275);
plot(f, fftshift(abs(X275)), 'b');
hold on;
X261 = fft(x261);
plot(f, fftshift(abs(X261)), 'm');
X_orig = fft(x_orig);
plot(f, fftshift(abs(X_orig)), 'r');
```

(a) First, we will try to remove the noise in `x275`, using two different approaches. Recall that the two frequencies present in the cosine noise added to the `x275` was picked to be frequencies that the DFT computes.

i. Take the FFT of `x275`. Zero out the frequency components with the two highest magnitudes in the result of the FFT. MATLAB's `sort` command could help here. Take the IFFT of the result to go back to the time domain. Plot the magnitude of the fft of the resulting signal. Play the resulting signal to hear if you have successfully removed the noise.

ii. Write MATLAB code to filter out this signal using a notch filter. Note that this is in many ways equivalent to the way you implemented the moving average: you are implementing a set of difference equations in MATLAB.

   A. First, you will need to determine the angular frequency in radians per sample, corresponding to a 275.6181 Hz CT signal. You can do this by noting that the CT signal has a frequency of $2 \times 275.6181\pi$ radians per second, and the sampling rate used to convert the CT signal to DT is `Fs` samples per second. Hence the frequency in radians per sample of this is $2 \times 275.6181\pi/Fs$.

   B. Next you'll want to implement the notch filter difference equation to produce an output vector `y` for an input (signal) vector `x`. Please assume any signal data that you do not have is zero.

iii. After running `x275` through the notch filter, play the audio segment to verify that you have successfully removed the noise. Plot the magnitude of the fft of the resulting signal on the same axes as the last part. Play the resulting signal to hear if you have successfully removed the noise.

(b) Repeat two filtering approaches with `x261`. The frequency of the noise here is not one of those computed by the DFT.

(c) Please explain the differences in the ability of the FFT-IFFT approach vs the notch filter approach in removing the noise in the two cases.

Please turn in all code and plots you generated in this problem.

Continuous-time Fourier Transform (2 hours)

If we interpolate a DT signal using a particular sampling frequency, we get a CT signal. Like DT signals, we can analyze and manipulate the frequency content of CT signals. For instance, in ISIM, you have already used CT filters of various kinds, which change the frequency content of signals. Examples of these include RC, and RLC circuits.

Here, we shall dig deeper into what is going on mathematically when you do operations such as low-pass filtering.

The frequency content of the CT signal can be analyzed using the Continuous-Time Fourier Transform (CTFT). Since we like doing things backwards here, we start with the inverse CTFT, which is defined as follows.

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)e^{j\omega t}d\omega \tag{5}$$

Compare this to the expression for the IDTFT. By convention, we shall use a lower-case ω to describe the angular frequency of the CT signal. We can think of an integral as a dense sum. Hence, the expression above tells us that $x(t)$ is made up of a sum of complex exponentials at all possible frequencies from $\omega = -\infty$ to $\omega = \infty$, with $X(\omega)$ being the weight applied to the complex exponential with frequency ω.

The CTFT is defined as follows

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt\,. \tag{6}$$

Compared to the DTFT, we see that, the summation becomes an integration, and the rest of the expression remains pretty much the same.

A few properties can be derived using some standard calculus/algebra steps and are summarized in Table 1.

signal	CTFT
$\frac{d}{dt}x(t)$	$j\omega X(\omega)$
$\int_{-\infty}^{t} x(t)dt$	$\frac{1}{j\omega}X(\omega)$
$e^{j\omega_0 t}x(t)$	$X(\omega-\omega_0)$
$x(t-t_0)$	$e^{-j\,\omega\,t_0}\,X(\omega)$

Table 1: Table of Fourier transform properties

3. To get ourselves comfortable with the CTFT, let's derive the CTFT of a few different signals. At this point, we are just trying to get comfortable with the CTFT, so some of these examples may not correspond to physical systems.

   (a) In ISIM, and in the review Bset, you encountered signals of the form $x(t) = e^{-\frac{t}{\tau}}u(t)$ where $\tau > 0$ and $u(t)$ is the unit step function defined as follows

   $$u(t) = \begin{cases} 1 & \text{if } t \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

   Derive the CTFT of $x(t)$ as a function of ω.

(b) Suppose that a signal $y(t) = \cos(\omega_0 t)x(t)$. Using the fact that $\cos\theta = \frac{1}{2}e^{j\theta} + \frac{1}{2}e^{-j\theta}$, and properties in Table 1, find an expression for $Y(\omega)$. This problem captures the idea of modulation in CT, and is what is done in AM radio transmissions to move a signal from low frequencies to high frequencies to be transmitted wirelessly.

(c) Find the relationship between $Y(\omega)$ and $X(\omega)$ if

$$y(t) = \frac{d}{dt}x(t) + 3x(t) \tag{8}$$

This example is meant to illustrate how the frequency content of one signal is related to another, when they are related by a differential equation in the time domain.

We have now introduced the main tools that we will use in this course to understand signals both as functions of time and frequency. Fundamentally, when we go from time to frequency, we are expressing signals in terms of a different basis. In the case of the DFT, the basis is complex exponential functions of a certain set of frequencies. In the case of the DTFT and CTFT, it is complex exponentials of all frequencies. We shall revisit this idea in class again on Thursday as this is one of the big ideas in this course.

As we move forward in this module, we will go back and forth between analyzing signals, and systems in both time and frequency and keep using these tools, as well as deriving new properties where needed. We shall use the CTFT in a later problem in this BSet.

The Sig-Sys Abstraction and Block Diagrams (3.5 hours)

You've dealt with a variety of different types of abstraction in the past – for example, you've represented physical situations using stock and flow diagrams, using free body diagrams, using differential equations, etc.

A key type of abstraction that gets widely used in engineering is the so-called "sig-sys" abstraction. This abstraction says that you can think of things as being broken down into *signals* (which we discussed a bit last time), and *systems* which operate on signals (ie., they take signals as input, and produce signals as output).

We tend to use "block diagrams" to represent the sig-sys abstraction, as we discussed in class. You're actually quite used to doing this already: you did so for the bad modem. Block diagrams are frequently used in designing and representing systems, from communication systems to control systems.

One of the very handy aspects of block diagrams is the extent to which they lend themselves to different *levels of abstraction* - i.e., you can always "drill down" to ask "what is the block diagram of that block?", and you can also always draw a dotted line around a set of blocks and call it a new single block. The ability to express different levels of abstraction conveniently is powerful since different situations in a design process require different levels of granularity.

Often, block diagrams can also be translated to hardware and software more directly than mathematical expressions. As such, block diagrams using the SigSys abstraction are powerful tools for the design of real-world systems, and can offer insight that may not be as apparent by examining mathematical relationships alone. Those of you in PoE will learn to appreciate this fact in a month or so.

Lets think first about block diagrams for discrete signals. In the following discussions $x[n]$ represents the input signal, $y[n]$ represents the output signal, n is an integer and represents the time index of the signal. Consider the following relationship between the input and output.

$$y[n] + ay[n-1] = bx[n]. \tag{9}$$

a and b are constant coefficients here. With appropriate choices for a and b, this expression can implement a digital filter (you've seen this kind of idea above). The above equation involves three basic operations.

- Addition
- Multiplication by a coefficient
- A delay which takes $y[n]$ as its input and outputs the previous sample $y[n-1]$.

The operations described above can be represented using elementary blocks shown in Figure 2.

We can represent (9) in a block diagram as shown in Figure 3. Note the output is fed back through a delay element, which is typically implemented as a memory element, multiplied by a constant and added to $bx[n]$. Such a representation can be readily translated to hardware, as one can obtain hardware components that act as memory elements (called registers), adders, and multipliers.

Now let's think about continuous time. Consider the differential equation

$$\frac{dy(t)}{dt} + ay(t) = bx(t) \tag{10}$$

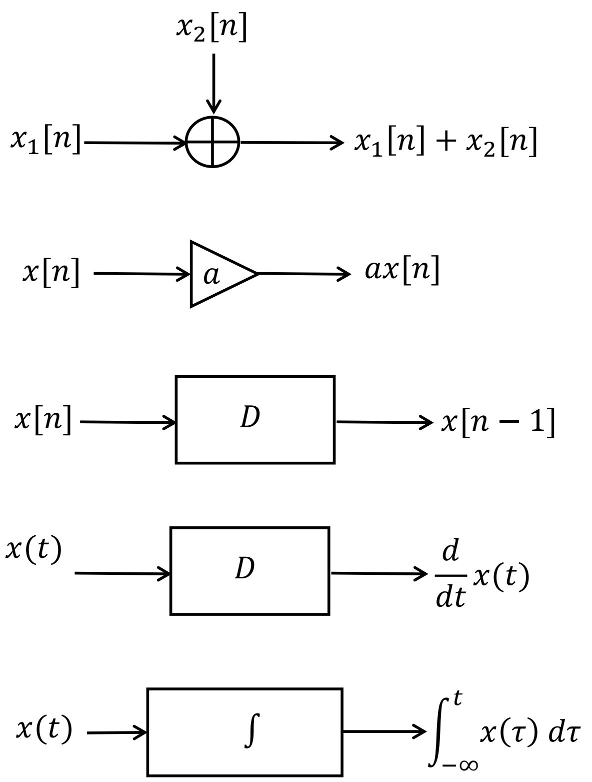

Figure 2: Basic blocks for DT and CT signals.

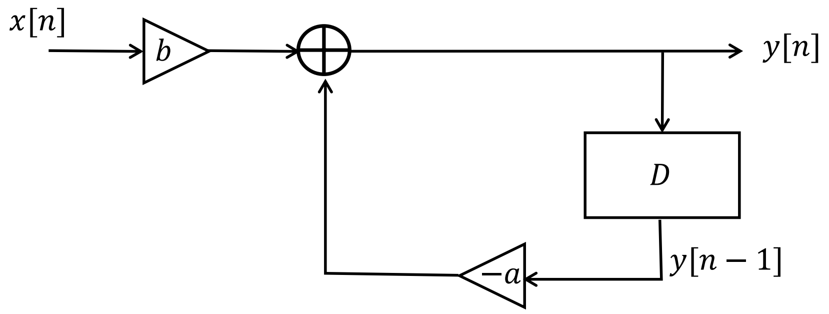

Figure 3: Block diagram representation of (9).

We can re-write this as

$$y(t) = -\frac{1}{a}\frac{dy(t)}{dt} + \frac{b}{a}x(t) \tag{11}$$

Equation (11) can be realized in the block diagram given in Figure 4, where the block labeled D is a differentiator. NOTE THAT THIS IS UNFORTUNATE NOTATION! A DIFFERENTIATOR IS NOT THE SAME AS A DELAY!!

An alternate way of writing (10) is

$$\frac{dy(t)}{dt} = bx(t) - ay(t) \tag{12}$$

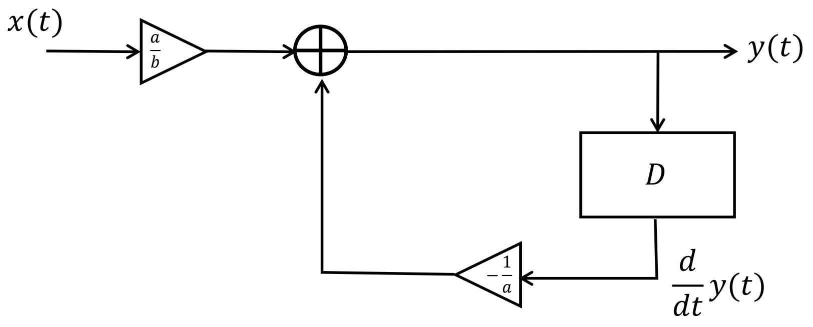

Figure 4: Block diagram representation of (11).

and taking the integral from $\tau = -\infty$ to t yields

$$y(t) = \int_{-\infty}^{t} [bx(\tau) - ay(\tau)]\, d\tau \tag{13}$$

Note that τ here is a dummy integration variable. Using a new block that performs integration, we can draw a block diagram representation of this system as in Figure 5.

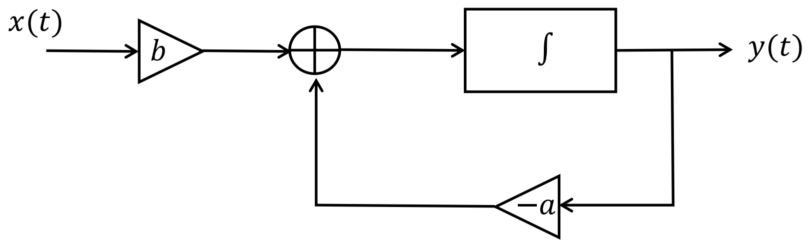

Figure 5: Block diagram representation of (13).

4. Create a block diagram that takes an input signal in time (e.g., a sound wave - a continuous signal), and produces an output that is the original signal with an echo. Create your block diagram at an abstraction level that assumes common operations are individual blocks (e.g., use blocks that represent operations like “add”, “multiply”, “delay”, “differentiate”, etc.)

5. Create a block diagram for either the notch filter or the moving average filter you played with above.

6. Let’s think about modifying block diagrams to represent additional effects.

   (a) The implementations in Figures 4 and 5 offer different benefits and drawbacks in terms of sensitivity to DC offset, and noise. A DC offset is a constant offset applied to a signal, and noise can be modeled as a random signal that is added to your desired signal.

Modify the block diagrams in Figures 4 and 5 to include either a DC offset or noise in the input signal. You can model noise simply as a function $n(t)$.

(b) Which implementation is more sensitive to DC offset errors?

(c) Which implementation is more sensitive to noise?

7. Consider a simple model for the balance in the bank account. A certain amount is deposited every month into the account and assume that there are no withdrawals from the account. The balance amount in the account earns interest at a rate of 1% each month, which is calculated every month. Write the input - output relationship for such a bank account and implement it ung a block diagram. Please indicate clearly which signals are the inputs and outputs in the diagram.

8. Consider the circuit described in Figure 6. Apply Kirchoff's voltage law (KVL) to note that

$$v_{in}(t) - v_R(t) - v_{out}(t) = 0 \tag{14}$$

$$v_{out}(t) = \frac{1}{C}\int_{-\infty}^{t} i(\tau)d\tau\,. \tag{15}$$

Note that $v_R(t)$ is the voltage across the resistor, and $i(\tau)$ is the current flowing clockwise in the circuit.

(a) Find an expression relating v_{out} to v_{in}, and draw a block diagram representation of this system.

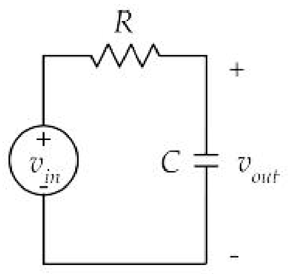

Figure 6: RC Circuit.

(b) Depending on how you approached the previous part, you may have come up with the following differential equation relating $v_{in}(t)$ and $v_{out}(t)$.

$$v_{in}(t) = v_{out}(t) + RC\frac{d}{dt}v_{out}(t) \tag{16}$$

By taking the CTFT of each term in the expression above, find the equation relating $V_{in}(\omega)$ and $V_{out}(\omega)$. In other words, find the relationship between the input voltage and the output voltage in frequency space. You will need to use the CTFT properties derived in the previous section for this.

(c) The ratio of $V_{out}(\omega)$ to $V_{in}(\omega)$ is the frequency response of this system (again, you will see more of this later). Plot the magnitude of the frequency response in the range $\omega = 1$ to $\omega = 10^4$ radians/second. You should use logarithmic axes here. You can use $C = 1\mu F$ and $R = 1k\Omega$, as nominal values here. Does this picture look familiar? You will find MATLAB's `loglog` function useful to make a log-log scale plot.