ECE-205 Lab 3: A tale of two disjoin sub- labs

Damping Ratio, Natural Frequency, Static Gain, and Fourier Series Too!

Overview

This lab has two disjoint parts.

In the first part of this lab you will be given the measured step response of four "unknown" ideal second order systems. You need to determine the correct values for the damping ratio, natural frequency, and static gain using a guess and check method. The goal of this part of the lab is for you to become more familiar with how these parameters affect the step response of a second order system.

In the second part of this lab you will learn how to use Matlab to perform numerical integration of two functions. As an example of this, you will determine a Fourier series representation of periodic functions. While we will not be using Fourier series in this class, they will be used in ECE300 (Continuous-Time Signals and Systems) and doing this part of the lab will hopefully help you get more comfortable with Matlab.

You will need to download and unzip the file **Lab3 Files.rar** for this lab.

PART A: Second Order System Response

The differential equation model of an ideal second order system is

$$\ddot{y}(t) + 2\zeta\omega_n\dot{y}(t) + \omega_n^2 y(t) = \mathrm{K}\omega_n^2 x(t)$$

Here K is the static gain, ω_n is the natural frequency, and ζ is the damping ratio. These are the parameters we need to determine for these models.

For this part of the lab you will primarily using the Matlab file **second_order_driver.m**. This file reads in data from a file (the measured data files for an unknown system) and extracts the time, input, and output values for the system. After this, you are to modify your guesses for the damping ratio, the natural frequency, and the static gain. The Matlab script **second_order_driver.m** then runs the Simulink model file **Second_Order_System.slx** and plots the results of your model with the measured data.

You are to change the parameters in the Matlab driver file to try and get your model to match the step response of the unknown system as well as possible. Once you are done, save the graph with the step responses of the two systems and put it in your memo.

You will need to do this for the data files **measured_1**, **measured_2**, **measured_3**, and **measured_4**.

Your memo should contain four figures for this part of the lab.

Again, the goal is for you to become familiar with how the different parameters affect the response of a second order system. This part of the lab should not take very long.

PART B: Numerical Integration and Fourier Series

Your memo for this part of the lab will need to include code and results from part I, three plots from part III, one plot from part IV, and your final code. Your code should be neat and organized (you can comment out the functions of x(t) you are not using)

PART I : Simple integration using Matlab

Maple is often used for symbolically integrating a function. Sometimes, though, what we really care about is the numerical value of the integral. Rather than integrating symbolically, we might want to just use numerical integration to evaluate the integral. In order to do this we will learn to use one of Matlab's built-in functions for numerical integration. In order to efficiently use this function, we need to use *anonymous* functions, which you have used already. We will then use this information to determine the *average* and *rms* value of a function. Some of this is going to seem a bit strange at first, so just try and learn from the examples.

Let's assume we want to numerically integrate the following:

$$I = \int_0^{2\pi} (t^2 + 2)dt$$

In order to do numerical integration in Matlab, we will use the built-in command **quadl.** The ***arguments*** to **quadl,** e.g., the information passed to **quadl**, are

- A function which represents the integrand (the function which is being integrated). Let's call the integrand $x(t)$. This function must be written in such a way that it returns the value of $x(t)$ at each time t. Clearly here $x(t) = t^2 + 2$
- The lower limit of integration, here that would be 0
- The upper limit of integration, here that would be 2π

Note that an optional fourth argument is the tolerance, which defaults to 10^{-6}. When the function value is very small, or the integration time is very small, you will have to change this, which we will do at the end of this lab so you will see how it works.

Anonymous Functions Let's assume we wanted to use Matlab to construct the function $x(t) = t^2 + 2$. We can do this by creating what Matlab calls an **anonymous function**. To do this, we type into Matlab

```
x = @(t) t.*t+2;
```

If we want the value of $x(t)$ at $t = 2$, we just type x(2)

Hence, to evaluate the integral $I = \int_0^{2\pi} (t^2 + 2)dt$ in Matlab we would type

```
x = @(t) t.*t+2;
I = quadl(x,0,2*pi)
```

Note that it is important to define x ***before*** it is used by (passed to) **quadl**

Example 1 To numerically evaluate $I = \int_{-1}^{1} e^{-t}\cos(2t)dt$ we could type

```
x = @(t) exp(-t).*cos(2*t);
I = quadl(x,-1,1);
```

Example 2 To numerically evaluate $I = \int_{-2}^{1} |t| e^{-|t|}dt$ we could type

```
y = @(t) abs(t).*exp(-abs(t));
I = quadl(y,-2,1);
```

Integrating Products of Functions Sometimes we are going to want to integrate the product of functions. While we could just multiply the functions together, it is usually easier to let Matlab do it for us.

Let's assume we want to evaluate the integral $I = \int_0^1 x(t)y(t)dt$, and let's assume that we already have anonymous functions x and y. The function **quadl** needs to be passed a function which is the product of x and y. To do this, we make a new anonymous function z, using the following:

```
z = @(t) x(t).*y(t);
```

and then perform the integration

```
I = quadl(z,0,1)
```

An alternative is to write

```
I = quadl(@(t) x(t).*y(t),0,1);
```

Example 3 To numerically evaluate $I = \int_{-1}^{1} e^{-t}\cos(2t)dt$ we could type

```
x = @(t) exp(-t)
y = @(t) cos(2*t);
z = @(t) x(t).*y(t);
I = quadl(z,-1,1);
```

or

```
I = quadl(@(t) x(t).*y(t),-1,1);
```

Example 4 To numerically evaluate $I = \int_{-2}^{1} |t| e^{-|t|} dt$ we could type

```
x = @(t) abs(t);
y = @(t) exp(-abs(t));
z = @(t) x(t).*y(t);
I = quadl(z,-2,1);
```

or

```
I = quadl(@(t) x(t).*y(t),-2,1);
```

The ***average value*** of a function x(t) is defined as

$$\overline{x} = \frac{1}{b-a}\int_a^b x(t)dt$$

and the ***root-mean-square (rms)*** value of a function is defined as

$$x_{rms} = \sqrt{\frac{1}{b-a}\int_a^b x^2(t)dt}$$

For you to do: Write an m-file to use Matlab to find the *average* and *rms* values of the functions

$$\begin{aligned} x(t) &= \cos(t) \quad 0 < t < \pi \\ x(t) &= \cos(t) \quad 0 < t < 2\pi \\ x(t) &= |t| \quad -1 < t < 1 \\ x(t) &= t\cos(t) \quad -2 < t < 4 \end{aligned}$$

Hint: You will probably find the **sqrt** function useful. *Attach your m-file to your e-mail, and put the answers in your memo.*

PART II : Trigonometric Fourier Series in Matlab

It is often convenient to represent a periodic function in terms of a sum of cosines. If we can do that, then we can use phasor analysis to determine how a system responds to a periodic input. We will begin with a simple Fourier sine series, then convert it to a sine and cosine series. The only new thing we will need is the idea of a loop.

Trigonometric Fourier Series If $x(t)$ is a periodic function with fundamental period T, then we can represent $x(t)$ as a Fourier series

$$x(t) = a_0 + \sum_{k=1}^{\infty} a_k \cos(k\omega_o t) + \sum_{k=1}^{\infty} b_k \sin(k\omega_o t)$$

where $\omega_o = \frac{2\pi}{T}$ is the fundamental period, a_o is the average (or DC, i.e. zero frequency) value, and

$$a_o = \frac{1}{T}\int_T x(t)dt$$

$$a_k = \frac{2}{T}\int_T x(t)\cos(k\omega_o t)dt$$

$$b_k = \frac{2}{T}\int_T x(t)\sin(k\omega_o t)dt$$

Even and Odd Functions Recall that a function $x(t)$ is **even** if $x(t) = x(-t)$ (it is symmetric about the y-axis) and is **odd** if $x(-t) = -x(t)$ (it is antisymmetric about the y-axis). If we know in advance that functions are even or odd, we can determine that some of the Fourier series coefficients are zero. Specifically,

If $x(t)$ is even all of the b_k are zero.

If $x(t)$ is odd all of the a_k (including a_o) are zero.

For Loops Let's assume we want to generate the coefficients $b_k = \frac{k}{1+k^2}$ for $k = 1$ to $k = 10$. One way of doing this in Matlab is by using the following **for** loop

```
for k=1:10
  b(k) = k/(1+k^2);
end;
```

In this loop, the variable k first takes the value of 1 until the end is reached, then the value 2, all the way up to k = 10. In this loop we are assigning the coefficients to the array b, hence b(1) = b_1, b(2) = b_2, etc.

Similarly, if we wanted to generate the coefficients $a_k = \frac{1}{k+1}$ and $b_k = \frac{2}{k^2+1}$ for $k=1$ to $k=5$, we could do this using for loops as follows:

```
for k=1:5
  a(k) = 1/(k+1);
  b(k) = 2/(k^2+1);
end;
```

Note that Matlab requires the indices in an array to start at 1, and that for loops should usually be avoided in Matlab if possible since they are usually less efficient (Matlab is designed for vectorized operations).

Fourier Sine Series Now we want to generate the Fourier series for the periodic function

$$x(t) = \begin{cases} 0 & -2 \le t < -1 \\ t & -1 \le t < 1 \\ 0 & 1 \le t < 2 \end{cases}$$

We will only need to generate a Fourier series with sine terms for this series (don't worry about why, but if you want to know it is because *x(t)* is an odd function). Download the program **Fourier_Sine_Series.m** from the class website. This program determines the Fourier series for this *x(t).*

If you type (in Matlab's command line) **Fourier_Sine_Series**(5) you should get a plot like that shown in Figure 1. As you increase the number of terms in the Fourier series, you should get a better match to the function. *Run the code for N=100 and include your plot in your memo.*

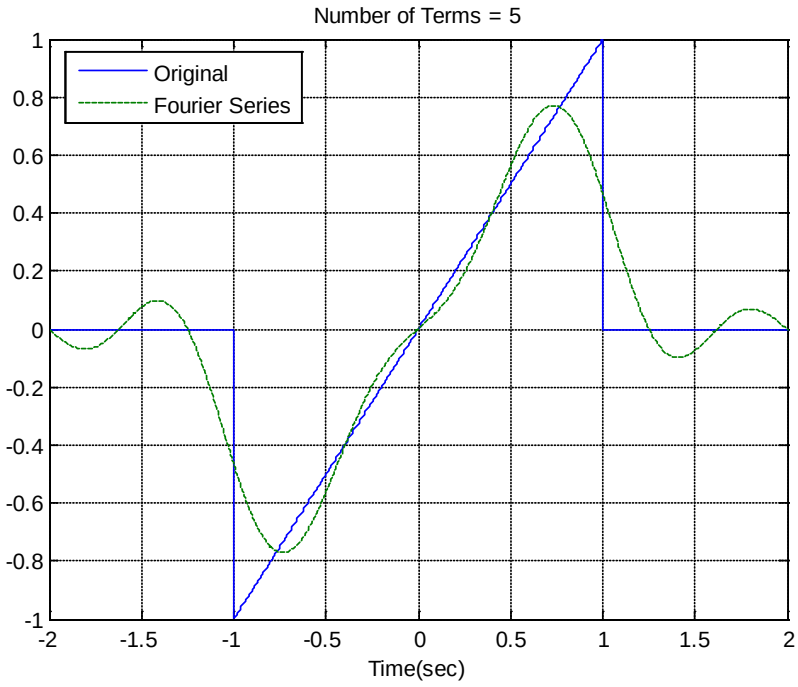

Figure 1. Trigonometric Fourier series example.

For you do to: Copy **Fourier_Sine_Series.m** to a file named **Trig_Fourier_Series.m** and modify the code to produce a full trigonometric Fourier series representation. This means you will have to compute the average value a_o and the a_k, and then use these values in the final estimate. Using this code use Matlab to find the trigonometric Fourier series representation for the following three functions (defined over a single period)

$$x_1(t) = e^{-t}u(t) \quad 0 \le t < 3$$

$$x_2(t) = \begin{cases} t & 0 \le t < 2 \\ 3 & 2 \le t < 3 \\ 0 & 3 \le t < 4 \end{cases}$$

$$x_3(t) = \begin{cases} 0 & -2 \le t < -1 \\ 1 & -1 \le t < 2 \\ 3 & 2 \le t < 3 \\ 0 & 3 \le t < 4 \end{cases}$$

Use N = 10 *for each function You do not need to include these graphs in your memo, but you need to go through this step to be sure your anonymous functions are working at this point. Note that the values of **low** and **high** will be different for each of these functions!*

PART III : Compact Fourier Series in Matlab

We can now write period functions as a sum of sines and cosines, but we really want to be able to write them as a sum of cosines in the form

$$x(t) \approx d_0 + \sum_{k=1}^{N} d_k \cos(k\omega_o t + \theta_k)$$

and we need to be able to compute the d_k and θ_k from the a_k and b_k. To do this we use the trigonometric identity

$$\cos(\alpha + \beta) = \cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta)$$

to equate the two forms of the Fourier series

$$d_k \cos(k\omega_0 t + \theta_k) = d_k \cos(k\omega_0 t)\cos(\theta_k) - d_k \sin(k\omega_0 t)\sin(\theta_k) = a_k \cos(k\omega_0 t) + b_k \sin(k\omega_0 t)$$

so we have

$$d_k \cos(\theta_k) = a_k$$
$$-d_k \sin(\theta_k) = b_k$$

We can then determine

$$d_k = \sqrt{a_k^2 + b_k^2},\ \theta_k = \tan^{-1}\left(\frac{-b_k}{a_k}\right)$$

In Matlab, we can compute these as

```
d0=a0;
d = sqrt(a.*a + b.*b);
theta = atan2(-b,a);    % atan2 includes the correct angle quadrant
```

For you to do: Determine the compact Fourier series representation for the following functions (defined over a single period), which are the same as those you used previously

$$x_1(t) = e^{-t}u(t) \quad 0 \le t < 3$$

$$x_2(t) = \begin{cases} t & 0 \le t < 2 \\ 3 & 2 \le t < 3 \\ 0 & 3 \le t < 4 \end{cases}$$

$$x_3(t) = \begin{cases} 0 & -2 \le t < -1 \\ 1 & -1 \le t < 2 \\ 3 & 2 \le t < 3 \\ 0 & 3 \le t < 4 \end{cases}$$

Use N = 10 *and include each of your plots in your memo*. *Note that the values of **low** and **high** will be different for each of these functions! (do not include your code)*

PART IV : Compact Fourier Series in Matlab using tolerances

For many engineering systems we are concerned with time functions that occur on small time scale. We can still use a Fourier series to approximate these functions, but we have to be more careful (and look at our results to see if they make any sense).

As an example, use your compact Fourier series code to try and determine a Fourier series for the following function:

$$x(t) = \begin{cases} -1 & 0 \le t < 1\times10^{-6} \\ 1 & 1\times10^{-6} \le t < 2\times10^{-6} \end{cases}$$

(In Matlab you represent 2×10^{-6} as 2e-6). If your look at your approximation, it is not very good. The problem is that Matlab does the numerical integration until it thinks the answer is "good enough", or within a certain tolerance. However, the default tolerance it not good enough for this signal and you will need to change it.

For you to do:

Use the **help** or **doc** command to look up **quadl** and figure out how to set the tolerance for each of your integrals (every time you use **quadl**). Set a variable called **tol** near the beginning of your code and assign this variable to be the tolerance for all of your integrals. Vary the value of **tol** until you think the integral has converged reasonably (note that a tolerance too small is as bad as one too large, so don't just make this ridiculously small). *Once you have a good value for your tolerance, include a plot of your Fourier series approximation (use N = 10) and the original function in your memo, and attach your final code to your e-mail.*