ECE 345: Linear Systems and Signals — Fall 2020

Professors Sarwate and El Rouayheb — Due: 5 PM, October 26, 2020

Lab 2

This lab is made up of two mini-labs. In order to do this lab you will need the Systems Toolbox and the Wavelet Toolbox installed in your MATLAB installation. See the Canvas page for the lab for details.

To turn in this lab, you need to submit a `.zip` file containing the following three files:

- a completed lab report **in PDF format**.
- your code for part 1 of the minilab
- your code for part 2 of the minilab (in a separate file/set of files)

Make sure your file names are labeled so that we can easily find your material. For example, `lab2problem1c.m` for part c of the first half of the minilab.

Points will be taken off for non-PDF submissions. Why are we being sticklers about formatting? In the real world, your work will also have to fit formatting guidelines. If you bid on a government contract, for example, and don't follow the guidelines, your bid/proposal could be rejected without review. Get in the habit of following the formatting rules!

Along those lines, do not forget to label your plots – it's part of the rubric!

Introduction: two minilabs

This lab covers the following topics:

- DT convolution
- CT convolution
- Laplace transforms

This lab is made of two "mini labs" focused on DT and CT systems, and in particular on two application areas which are important follow-ons to the material in LSS: image processing, and feedback control.

1 Mini-lab 1: image processing

Since we don't get to be on campus for instruction this fall, perhaps we can do a little imagination using image processing. Download `WeeksHallSmall.jpg` from the assignment website.

This exercise will do some blurring and edge detection on images to demonstrate the visual impact of different filters. If you're interested in fancier edge detection, you might want read about the Canny edge detector [1].

MATLAB has two image display commands: `imshow` and `imagesc`. They behave slightly differently and by default `imagesc` will show a color image. To shift it to grayscale, add the line `colormap gray` after running `imagesc`. To run `imshow` you will need to provide it a range of values, which for our 8-bit grayscale images will be $= [0, 255]$.

(a) Use `imread` to read in the image into MATLAB and `imshow` to display it in a figure window. Use `rgb2gray` to convert it into a grayscale image and verify that it worked using `imshow`. Use `double` to convert the grayscale image from unsigned 8-bit integer (values from 0 to 255) to double-precision floating point so that you can do some filtering.

(b) The filter

$$b[n] = \frac{1}{M} \sum_{k=0}^{M-1} \delta[n-k] \tag{1}$$

is a rectangle of length M. The output of the filter at $n = \ell$ is the average of the signal over a window of length M. Visually, we will see how this corresponds to a *blur*. Use `conv` to convolve the filter $b[n]$ with each row of the image matrix. Put the filtered rows into a matrix and view it with `imagesc` or `imshow`. Now do the same thing for the columns of the image. Try this for different values of $M = 2, 4, 8, 16, 32$. What happens as M increases?

(c) The *first difference* filter is

$$c[n] = \delta[n] - \delta[n-1]. \tag{2}$$

This filter computes the difference of two consecutive pixels. If the difference is large, that indicates an *edge* in the image. Use `conv` to convolve the filter $c[n]$ with each row of the image matrix and view it with `imagesc`. The result might be hard to interpret – try plotting one row with `stem` to see what the output values look like. To see edges more clearly, try thresholding the output image: if the image is `Y`, try looking at `Z = (Y > tau)`. Do the same thing on the columns of the image. Try a couple of different values of τ to see how the threshold affects the edge detection performance.

(d) For two-dimensional signals like images, we can generalize our one-dimensional convolutions to two-dimensional convolutions. In this case the filter impulse response looks like a matrix. The row and column first difference filters above are effectively computing the *gradient* of the image along the rows and columns: it's a discrete analogue of taking the first derivative. If we represent these as 2D filters we would have:

$$D_{\text{row}} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{pmatrix} \qquad D_{\text{col}} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \end{pmatrix} \tag{3}$$

With a 2D filter we can take diagonal gradients too. This filter computes the gradient along a diagonal.

$$D_{\text{ur}} = \begin{pmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{4}$$

[1] `https://en.wikipedia.org/wiki/Canny_edge_detector`

Use the function `conv2` to convolve D_{ur} with the image and threshold the output to reveal the diagonal edges. Try it again for the filter

$$D_{\text{ul}} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{5}$$

(e) Using the same ideas in the previous part, try developing a 5×5 matrix D that is tuned to detect the edge formed by the downward sloping diagonal angle of the roof of Weeks Hall. Specify the matrix and threshold that you used.

(f) One way to think about edges is that that they are "details" in an image: this is the perspective we take when using the *wavelet* transform. Wavelets are used in what is called multirate signal proccessing, in which we try to analyze signals based on patterns in the signals that appear at different scales. This is especially effective image compression: the JPEG 2000 standard (the latest version of JPEG) uses the 2D discrete wavelet transform (DWT) transform for image compression.

Look up the documentation for the function `dwt2`, which implements a 2D DWT. In the examples given in the documentation, they process an image called `sculpture` using `haar` wavelets. Use the same steps to process the image of Weeks Hall and give a 2×2 plot of the approximation, horizontal, vertical, and diagonal coefficients. The approximation is a downscaled version of the original image, whereas the others contain edge information. To see this better, modify the code to use `imagesc` for the horizontal, vertical, and diagonal coefficients. What are the similarities and differences from the edges you found in the earlier parts?

(g) To reconstruct an image of the appropriate size, use the inverse DWT in the function `idwt2` and display it. Calculate the difference between the reconstructed image and the original and display it/ Calculate the total squared error per pixel of this difference.. Calculate the maximum absolute difference between the reconstructed image and the original.

(h) Now let's try some crude image compression. Look at the sizes of of the matrices produced in the previous part as well as the size of the original matrix. How many numbers do you need to store the DWT versus the image?

One thing we could do to compress is to just zero-out the horizontal, vertical, and diagonal coefficients and just keep the approximation. How much would that compress the image in terms of number of values kept over number of values in the original image?

Run `idwt2` on the approximation matrix and three all-zero matrices for the detail matrices. Look at the reconstructed image – can you see what changed between the original and the reconstruction? How would you describe that change?

Calculate the difference between the reconstructed image and the original and display it. Use a threshold of $\tau = 10$ to view where the difference is large. What do you see? Calculate the total squared error per pixel of this difference. Calculate the maximum absolute difference between the reconstructed image and the original. Do these measures of performance seem like the right ones to use for this problem? Why or why not?

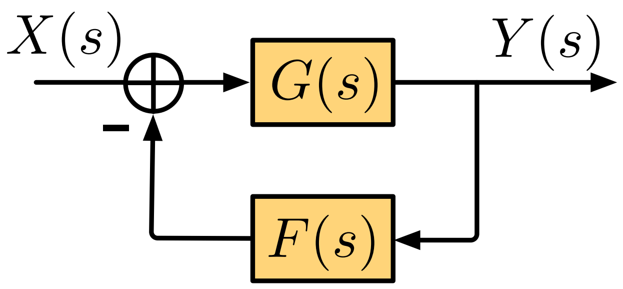

Figure 1: A generic feedback control system. The output of the plant $G(s)$ is used as the input to the feedback system $F(s)$, creating a new input-output transfer function.

2 Mini-lab 2: feedback control

In this mini-lab we're going to try to stabilize an unstable system use *feedback control.* Feedback is one of those important concepts that every engineer should learn about. Your own body uses feedback all the time (think about the brain). Feedback is also what makes the internet work (it controls congestion), vehicles like cars and planes (steering), and robots. The central idea in feedback control systems is to "close the loop" by using the output of an unstable system to modify the input so that the overall system is stable. The basic picture is in Figure 1. The feedforward system $G(s)$ is called the *plant* (think of a power plant) and we generally consider unstable plants. Recall this means that the ROC of $G(s)$ does not contain the imaginary axis, so there is a pole in the right-half plane.

The feedback system $F(s)$ is the thing we get to design. In order to build such a system we need some sort of sensor that measures the output $y(t)$ (corresponding to $Y(s)$). We can then design a *control signal* $u(t)$ as the output of $F(s)$. We're assuming negative feedback, so we subtract the control from the input $x(t)$ so that $x(t) - u(t)$ is now the input to the feedforward system. This reverse connection creates a new transfer function $H(s)$ between $X(s)$ and $Y(s)$. We're going to explore what happens to the pole-zero diagram as we change $F(s)$: hopefully we can make it so that the poles of $H(s)$ are all in the left-half plane so the system is stable. We can think of this as "moving" the poles of $G(s)$ around. This is the idea behind the *root locus* method used in control system design.

(a) Suppose that our system $G(s)$ is with Laplace transform given by

$$G(s) = \frac{s-1}{s^2 - 2s + 50}. \tag{6}$$

Find the impulse response of this system.

We can specify a rational Laplace transform using the System Toolbox in MATLAB using the function `tf`. Create a system object `plant` for $G(s)$ using the function `tf` and verify that it is correct (you can just type `plant` (no semicolon) and it should give the rational transform formula). Use `impulse` to plot the system output for 5 seconds and verify that the system is indeed unstable. When using `impulse` get the output signals using `[Y,T] = impulse()` so that you can plot the impulse response with a wider `LineWidth` for visibility.

(b) Use the `pzplot` function to plot the pole-zero diagram. You may notice that it is a bit hard to read. Try running

```
h = findobj(gca, 'type', 'line');
set(h, 'markersize', 9);
set(h, 'linewidth', 3);
axis equal;
```

to get a more nice-looking plot. Verify that these are the poles and zeros you saw in the previous part.

(c) We can see that the poles of $G(s)$ are in the right half-plane, which means the system is unstable. To try and stabilize it, let's use the feedback setup given in Figure 1. Write down the transfer function $H(s)$ between $X(s)$ and $Y(s)$ in the feedback control system for general $F(s)$ and $G(s)$.

Suppose we set $F(s) = K_p$ for some constant K_p. This is what is called *proportional control* since we are feeding back a proportion of the input to the output. Write down the closed-loop transfer function $H(s)$ with this choice of $F(s)$ and calculate the locations of the poles and zeros as a function of K_p (this will be some messy formula, so don't worry). Find a value for K_p such that the poles are on the imaginary axis.

(d) Now, choose $K_p = 0.5$, generate a new transfer function in MATLAB for $H(s)$ and plot its the pole-zero diagram. Where are the poles of the closed-loop system? Have you stabilized the system? Do the same for $K_p = 1$, K_p equal to the value you found to make the poles on the imaginary axis, and twice that value.

From your analysis, describe the benefits and costs of using proportional control in terms of implementing the feedback system. For context, imagine the output of the plant is a very high-voltage signal. What does the feedback signal look like?

(e) Proportional control is a simple choice, but may have some drawbacks. We could instead try *proportional-integral (PI)*, *proportional-derivative (PD)*, or *proporational-integral-derivative (PID)* control. Let's focus on PD control for this lab. Under PD control the feedback system is $F(s) = K_d s + K_p$. Since multiplication by s means taking the derivative in time, this takes the output signal and and its derivative to try and use as feedback. Since there are now two parameters (K_p and K_d), we have two degrees of freedom for our control.

Write down the closed-loop transfer function $H(s)$ with this choice of $F(s)$. How does the presence of K_d help in lowering the value of K_p needed to stabilize the system? Find a pair of values K_p and K_d that should lead to a stable system (you do not need to compute the poles explicitly).

Simulate the closed-loop system with PD control and try your values for K_p and K_d. Plot the pole-zero diagram and show why the system is stable. Is PD control better than P (proportional) control in this setting based on your cost/benefit analysis?