Convolution Lab

In this lab, we are going to create a Matlab demo to show off the wonders of convolution. This will be a demonstration geared toward introductory Signals and Systems students in college, and ultimately to be associated with a Matlab lab on convolutions.

Convolutions have many uses in the study of signals and systems, and we cannot hope to provide a proper cross-section of those applications in our demo, for lack of development time. Instead, we will content ourselves with exploring some methods for calculating them, and just put a short note at the beginning of the associated lab that convolutions are a way to calculate the output of a system characterized by a time-domain impulse response function.

0.1 Short Discussion of Calculating Convolutions

0.1.1 Using the Impulse Response

In hopes of not overwhelming the students, you want to sneak up on the calculation of convolutions by first considering "impulse responses". An impulse is a signal

$$\delta[n] = \begin{cases} 1 & \text{if } x = 0; \\ 0 & \text{otherwise.} \end{cases}$$

The impulse response of a system is the system's output when its input is an impulse. That is,

$$x[n] = \delta[n] \longrightarrow \boxed{\text{h[n]}} \longrightarrow y[n] = h[n].$$

Where $h[n]$ is the function that characterizes the system, called the impulse response function.

Then, one simple way to calculate convolutions is to break down $x[n]$ into a series of impulse responses and add together the results. In other words,

$$y[n] \quad = \quad x[n] * h[n]$$

$$
\begin{aligned}
&= x[0]h[n] + x[1]h[n-1] + x[2]h[n-2] + \cdots \\
&= \sum_{l=-\infty}^{\infty} x[l]h[n-l]
\end{aligned}
$$

where the last line is the typical expression for convolution.

0.1.2 Using the "Graphical Method"

The graphical method is a way to calculate the points of the result signal, $y[n]$, one point at a time. One reason for using the graphical method rather than a train of impulse responses is because the same method can be used equally well for continuous-time convolution, although we will not discuss that here. According to the graphical method, the convolution of two signals can be calculated using the following steps:

1. Flip on signal about the y-axis ($h[n] \rightarrow h[-n]$)
2. Line up the signals next to each other (one above and one below), but with one the left of the other (so that no non-zero points overlap). We will call one of the signals "above" the other, as their plots might be laid out on a sheet of paper, with the "lower" one to the left of the upper one.
3. Shift the lower signal to the right one point.
4. Multiply overlapping points together. Sum the result. Assign this value to y[0].
5. Repeat from step 3 for successive points in y[n].

We do not need here a more extensive description since the method was acted out in class; however, one of our goals is to create a demonstration that will show this process with plots, for the users of our demo.

0.2 Block Diagrams

It is possible to show off these different methods by breaking them down into block diagrams. Block diagrams are a way of describing the relationships between "active" processes in a system by tracking the flow of data (signals) between them. We can make quick progress toward making our demonstration by developing an underlying framework of systems.

For warm-up let us make some basic system elements (blocks) for scaling signals, adding signals together, and delaying signals in time. With just these operations, we can make an enormous variety of systems. See section 5.4.1 for graphical representations of these blocks. For our purposes, blocks will be m-files, referenced by a unique id number.

0.2.1 Basic Add and Multiply Blocks

Make an m-file for adding together two signals and for multiplying two signals. They should be of the form

`z = mulblock(id, x, y)` and `z = addblock(id, x, y)`

where `x`, `y`, `z`, and `id` are all single numbers (not vectors). These functions can be applied multiple times to multiple and add full signals together.

0.2.2 The Delay Block

Make a block for a single element delay, `z = dlyblock(id, x)`. Use a "persistent" variable, `dlydata`, a vector for which the *id*th element, `dlydata(id)`, contains information for the delay block with the same id. You will want to use `dlydata(id)` to keep track of "delayed" data. In `dlyblock`, you need to make sure that the `dlydata` vector is large enough to have an element for the `id` you want to use.

For information on persistent variables, use Matlab help (`help persistent`).

0.2.3 Improving the Blocks

One advantage to having functions to denote simple operations like + and × is that we can add additional features to the operations for our demonstration. For now, we will add plotting capabilities to the functions, through a number of "global" variables, with the same form as `dlydata`.

- `addplot`, `mulplot`, and `dlyplot` may contain x-axis indices corresponding to different blocks. Using these variables, we want to be able to plot the output of specific blocks. If `xxxblock(id) == 0`, do not plot the output of that block; if `xxxblock(id) != 0`, plot the output at location `n = xxxblock(id)`.

- `addsubplot`, `mulsubplot`, and `dlysubplot`, each of which contain a "handle" to a subplot in which the data is to be graphed. This handle is the result of a `subplot(...)` function call and may be used by another subplot function to recall the same plotting area.

In plotting your data, make sure you `hold` the contributions of previous blocks.

0.2.4 Creating Systems

A system will combine a number of blocks to process a single input to a single output. We will then call the system for successive inputs to calculate the impulse response of full signals. For example, a system to double its input would just calculate

`output = mulblock(1, input, 2)`.

(Where the multiplication block here is assigned ID = 1).

Create two systems, `output = system1(impresp, input)` and `output = system2(impresp, input)`, to implement figures 5.13 and 5.14 to calculate convolution (`impresp` is the impulse response vector). Again, `output` and `input` should be single numbers (not vectors). You may want to use a `for` loop to calculate the segment of the system associated with each element of the impulse response.

Test your functions with an impulse response `[.1 .15 .25 0 .25 .15 .1]` by inputting 1, followed by several 0's to collect the impulse response.

0.2.5 Making the Demos

Create a new function, `outv = convolve1(inv)` to automate the process of feeding numbers into `system1`, while graphing the process that form the convolution. Specifically, make a four row plot containing (1) the impulse response, (2) the elements from `inv` that have been fed into the system, in the order they were entered, (3) the outputs of the multiplication blocks, from left to right, and (4) the output. Use a `for` loop to input successive values from `inv` and plot the updates; at the end of every loop, execute a `pause(.5)` command, to give the viewers of our demonstration the illusion of movement ("signals marching left to right"). Also, use the `axis(XMIN XMAX YMIN YMAX)` command to make all of the plots line up (so that `XMIN` is `1` and `XMAX` is `length(impresp)` (choose appropriate values for `YMIN` and `YMAX`, which need not be consistent between graphs).

Make a second function `outv = convolve2(inv)`, almost identical to this one but using `system2`.

Run your programs and interpret the results. How is convolution being calculated? What other data could you graph from the systems to make more clear the processes underlying convolution? Write descriptions to go with the demos to explain to the students what they are seeing.

0.3 Demo Sound-track

Any good demo should combine auditory and visual aspects. It is time to give our demo a sound track.

0.3.1 Loading Music

Select a `.wav` file of some music that you like (you can also find `.wav` files for many songs at `http://www.chivalry.com/cantaria/`).

Read it into a Matlab variable using `wavread(FILE)`. If the `.wav` file is large, you may want to limit the number of samples with `wavread(FILE, N)`.

Use `sound()` to play the song; make sure that the sampling frequency that you use in Matlab is the same on used for the `.wav` file samples.

Generate a spectrogram for a small piece of your song (about 1 second). Include your `.wav` file and your spectrogram graph in your lab report.

0.3.2 Note-Pass Filter

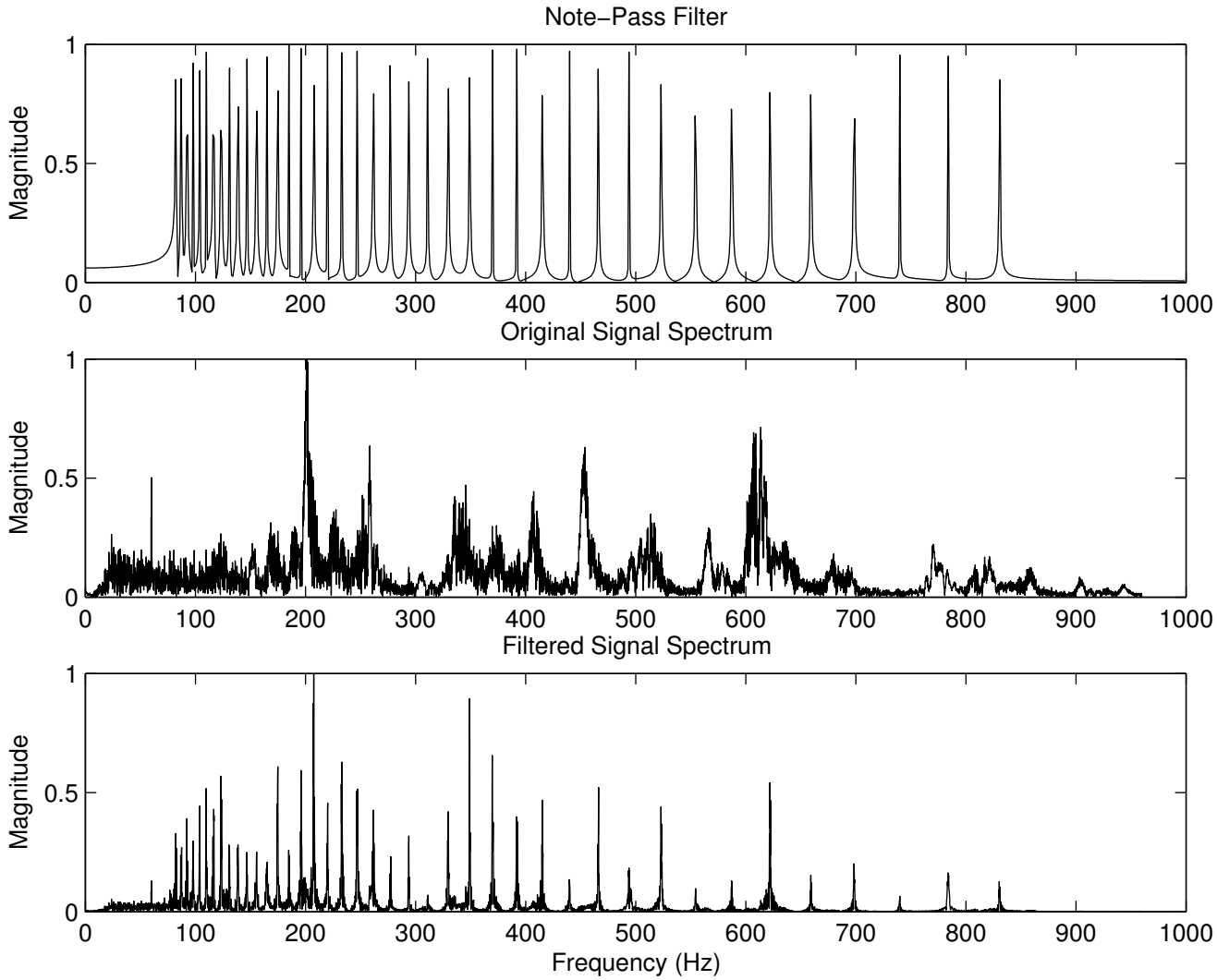
Often real songs, particularly those with words, can be very complex and contain off-key elements. As a result, our students may be distracted from the demonstration of convolution. We will use convolution to simplify the song into a more appropriate background.

A useful property of convolution is that multiplying signals in the frequency domain is equivalent to convolving their corresponding signals in the time domain, and visa-versa. One piece of evidence for this is the result that

$cos(2\pi f_0 t)cos(2\pi f_c t) = \frac{1}{2}cos(2\pi(f_c + f_0)t) + \frac{1}{2}cos(2\pi(f_c - f_0)$.

You can derive this result by convolving the spectra of $cos(2\pi f_0 t)$ and $cos(2\pi f_c t)$.

We will do the opposite, convolving two signals in the time-domain to multiply the spectra in the frequency domain and thereby accent some frequency components and diminish others.



The middle plot in the figure shows what the spectrum of a typical segment of a song might look like. We can generate a "filter" with a spectrum like the top graph, where each peak is at a different musical note. The resulting spectrum, made by multiplying these two spectra, will only contain those sounds that can be played on a keyboard.

Use `sumcos` to generate between 1000 and 10000 points of a signal containing only the

frequencies on a keyboard within some key range (say keys numbered 20 to 60). Use Matlab's built-in convolution function, `conv`, to convolve this signal with a piece of your song (10000 to 100000 points). Beware that convolution can be very computationally intensive and grows with the square of the number of points to be convolved, so you may want to start with fewer points and increase gradually. More samples in the filter will result in a more precise effect.

Play the result (use `soundsc!`). How does it sound? Include in your lab report a `wavwrite` of the result (make sure you scale the vector to between -1 and 1. Make a spectrogram of the same segment as before and include this in your report.

0.4 Final Notes

In your lab report, please remember to include how long this lab took you and with whom you collaborated. Also, please comment on the lab in general and any particular pieces. Did you enjoy it? Do you have any suggestions for improving it for the future?