ECE 320 SIGNALS & SYSTEMS II

Matlab Project I

Fall 2008

Assigned: Saturday, September 6, 2008 **Report Due:** Tuesday, September 23, 2008

The purpose of this project is to reinforce your your understanding of discrete-time convolution. After familiarizing yourself with the use of Matlab's `conv` command, you will develop a function that implements overlap-add block convolution. Block convolution is used frequently in digital signal processing. You will use the function you develop to filter a noisy audio signal.

Please read the lab report guidelines on the ECE 320 website before beginning the project.

1 Basic Convolution Exercises

In the problems below you will use the `conv` command to find the output of several discrete-time systems given the input $x[n]$ and the impulse response $h[n]$. Before beginning these exercises, read the help for the `conv` function (`help conv`). Note that the `conv` function computes the convolution of two sequences, but does not keep track of the associated time indices. In order to plot the results, you must do some additional calculations to generate a vector of time indices to plot against. The following problems will help you learn to do this.

(a) Consider the impulse response $h[n] = \delta[n+1] + \delta[n-1]$ and the input $x[n] = \delta[n] - 3\delta[n-2]$. In Matlab define the vectors `h` and `x` corresponding to these sequences. Use `conv` to compute the output signal $y[n]$. Determine a vector of time indices corresponding to `y` and store those in the vector `ny`. Plot $y[n]$ as a function of n using the command `stem(ny,y)`.

(b) Consider two finite-length sequences stored in the vectors `h` and `x`. The time indices corresponding to `h` and `x` are stored in the vectors `nh=[a:b]` and `nx=[c:d]`. (a and b represent the starting and ending time samples of the `h` vector. c and d are the starting and ending time samples of the `x` vector.) The command `y=conv(h,x)` computes the convolution of the two sequences. Can you use the information stored in `nh` and `nx` to construct the vector of time indices for the output `y`? To see how to do this, do the simple problems below:

(i) Consider the following simple signals

$$h[n] = \delta[n-a] + \delta[n-b];$$

$$x[n] = \delta[n-c] + \delta[n-d].$$

Analytically compute the convolution $y[n] = h[n] * x[n]$.

(ii) From your answer to (b-i), determine what the vector `ny` should be in terms of a, b, c, and d.

(iii) Check your result by verifying that $y[n]$ is of length $M + N - 1$ when $a = 0$, $b = N - 1$, $c = 0$, and $d = M - 1$.

(c) Consider the impulse response $h[n]$ and input $x[n]$ defined below

$$h[n] = u[n+2];$$

$$x[n] = \left(\frac{3}{5}\right)^{n-2} u[n-2].$$

(i) Analytically compute and sketch the convolution of $h[n]$ and $x[n]$.

(ii) Define a vector `h` in Matlab that contains the values of $h[n]$ $-2 \leq n \leq 14$, and define a vector `x` in Matlab that contains the values of $x[n]$ for $0 \leq n \leq 24$. Define vectors `nh` and `nx` containing the corresponding time indices. Compute the convolution of these two finite-length vectors using `y=conv(x,h)`. Using the results of part (b), compute the vector of time indices for `y` and store that in `ny`. Plot `y` using the stem command. Note that since the `h` and `x` vectors are truncated versions of the true signals, only a portion of the output vector will contain the true values of $y[n]$. Specify which values in the output vector `y` are correct and which are not.

2 Block Convolution

In the problems below you will learn how to implement block convolution. In block convolution the input signal $x[n]$ is split into a number of segments (or "blocks"), and each segment is processed separately. Since the system that does the processing is LTI, the overall output signal $y[n]$ can be computed as a superposition of the outputs for the individual segments. The motivation for using block convolution is that it permits computation of the inital parts of the output even when the entire input signal is not yet available, thus it facilitates the real-time processing needed in many speech and audio processing applications. Block convolution is typically used when the impulse response $h[n]$ of the LTI filter is short compared to the length of the input $x[n]$.

(a) Consider the problem of computing the output $y[n]$ of an LTI system for $0 \leq n \leq 99$ when the impulse response and input signal are defined below:

$$h[n] = (0.9)^n(u[n] - u[n-10]),$$

$$x[n] = u[n].$$

(i) First compute the output using the `conv` command on the whole input sequence. Plot $y[n]$ using the `stem` command.

(ii) Break the input into two segments of length $L = 50$. Using the `conv` command, compute the convolution of $h[n]$ with each of the segments, *i.e.*, compute

$$y_0[n] = h[n] * x_0[n] \qquad y_1[n] = h[n] * x_1[n],$$

where $x_0[n]$ contains the first 50 samples of $x[n]$ and $x_1[n]$ contains the last 50 samples of $x[n]$. The output $y[n]$ may be computed as follows

$$y[n] = x[n] * h[n] = y_0[n] + y_1[n - M].$$

Determine the correct value of M to use. Compute $y[n]$ in this manner and plot it using `stem`. Make sure you obtain the same result as you did in part (a-i).

The type of block convolution you implemented in part (a) is known as *overlap-add* because the sequences $y_0[n]$ and $y_1[n]$ overlapped in time and were added together to form the output. Consider the general problem of implementing overlap-add convolution for an impulse response $h[n]$ that is non-zero on the interval $0 \leq n \leq P - 1$. The input sequence $x[n]$ is known to be zero before $n = 0$, and the length of

$x[n]$ is much greater than P (the length of $h[n]$). In this case the input can be written as a sum of a series of L-point segments:

$$x[n] = \sum_{r=0}^{\infty} x_r[n - rL],$$

where

$$x_r[n] = \begin{cases} x[n + rL] & 0 \le n \le L - 1 \\ 0 & \text{otherwise} \end{cases}$$

Given this, the output $y[n]$ can be written as the superposition of the outputs computed for the individual segments

$$y[n] = \sum_{r=0}^{\infty} y_r[n - M_r],$$

where $y_r[n] = x_r[n] * h[n]$ and M_r is the offset for each segment. Based on the results of (a), you should be able to determine how M_r must be defined.

(b) Write a Matlab function called `oafilt` to implement overlap-add block convolution. This function should take as input the impulse response vector `h`, the input vector `x`, and the segment length `L`, and produce the output vector `y`. The first line of your function should be

```
function y=oafilt(h,x,L)
```

Note that your function should allow the vector `x` to have arbitrary length. The function should check that the input segment length `L` is longer than both `h` and `x`. If it is not, the function should return an error. For information on how to do this, type `help error` at the Matlab command line.

(c) Test your `oafilt` function by redoing the convolution for part (a) using several different values of the segment length L. You are also encouraged to try your filter on other test sequences. Your writeup for this part should describe your test procedure. You should include a few plots that demonstrate that your function is working properly.

Important note:

- While many overlap-add implementations use the `fft` command to implement the convolution for each block, I *am not* asking you to do this. You should use the `conv` command instead. A discussion of the efficient `fft`-based implementation is beyond the scope of this class. You will learn more about it if you take a senior- or graduate-level class in signal processing, *e.g.*, ECE 410, which is being offered in spring 2009.

3 Filtering of a Noisy Audio Signal

As the final part of this project, use your function `oafilt` to filter the noisy audio signal contained in the file `proj1_data.mat` (which may be downloaded from the course website). You may load the data file by typing `load proj1_data.mat`. The file contains 3 variables:

- `fs`: sampling frequency in Hz
- `h`: vector containing the impulse response of a 61-point FIR filter

- `sig`: vector containing the noisy audio signal

Once you have loaded the data, do the following.

(a) Play the signal using Matlab's `soundsc` command. What do you hear?

(b) Filter the signal using your `oafilt` function. You should experiment with different block lengths. Play the resulting signal. Is there a difference? Can you identify the words?

Your writeup for this part should include answers to the above questions along with plots of the noisy signal and the filtered signal.

Acknowledgment: The first two parts of this project are based on a project in the book *Computer Explorations in Signals and Systems Using Matlab* by Buck, Daniel, and Singer.