

## EE432: Digital Signal Processing Fall 2011

### **Project 05: Difference Equations, Impulse Responses and Filtering**

**Assigned: Tues 10/4/11**

**Due: Tues 10/11/11**

This project is intended to give you some practice working with difference equations and impulse responses, and an introduction to how they are handled in MATLAB.

#### **I. Preliminaries**

1. Sketches of some of the basic discrete-time functions are attached to this document, along with MATLAB code that implements four of them (MATLAB functions called *impD*, *rampD*, *uD*, *impND*). Our book does not use a ramp function (*rampD*), or the periodic impulse function (called *impND*), but they are provided as well and we will use them. Create MATLAB function files that implement these discrete-time functions. Just copy the code, all the comments are not necessary unless you want them...I will not collect them.
2. **Test your new functions by creating stem plots for the following signals. Put them on a 4 x 1 subplot, with proper labels, and turn on the grid. When using the *stem* function, be sure to use the “filled” option so that the lollipops contain filled circles. Also, use the *axis* function to set the axis limits so that the lollipops all fall within the display. You pass the function the values you want the plot axes limits to be. You would use it like:**

```
>> stem(n,x,'filled'), axis([xmin xmax ymin ymax]),grid on
% xmin, xmax are the min/max x-values to display
% ymin, ymax are the min/max y-values to display
```

For example, for the first plot below, if you don't use the axis command, the left end of the plot will be at  $n=0$  and the right end will be at  $n=10$ . The lollipops that occur there will be hard to see. Also, the plot will limit the vertical axis to fall from 0 to 1.0, so the lollipops with height 1 will be hard to see. Instead, you might use:

```
>> axis([-5.5 10.5 -0.5 1.5])
```

Now create the plots for these signals and make sure they are correctly plotted:

- a)  $x[n] = u[n] - u[n-3], \quad -5 \leq n \leq 10.$
- b)  $x[n] = \delta[n] + 0.5 \delta[n-2] - 0.5 \delta[n+2], \quad -5 \leq n \leq 5.$
- c)  $x[n] = \text{ramp}[-n] \cdot u[n+5], \quad -10 \leq n \leq 5.$
- d)  $x[n] = \delta_2[n] \text{ramp}[2n], \quad 0 \leq n \leq 15.$

#### **II. Impulse Response, Difference Equations and Convolution**

1. **By hand (turn in a table), find the impulse response of the system defined by  $y[n] = x[n] - 0.8 y[n-1]$ . By hand, plot this impulse response from  $n = -1$  up to  $n=8$ .**
2. The general form of the difference equation is given by:

$$\begin{aligned} a_0 y[n] + a_1 y[n-1] + a_2 y[n-2] + \cdots + a_N y[n-N] \\ = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] + \cdots + b_M x[n-M] \end{aligned}$$

We use the coefficients as vector variables in many of the MATLAB DSP functions. For example, the output  $y$ -coefficients are the  $a_k$ 's, which we put in vector form as  $a = [a_0 \ a_1 \ a_2 \ \dots \ a_N]$ , and the input  $x$ -coefficients are the  $b_k$ 's, which we put in vector form as  $b = [b_0 \ b_1 \ b_2 \ \dots \ b_M]$ . That is how the coefficients are passed into MATLAB.

It is important that each delay of an input or output is accounted for in the vector. For example, if any of the  $b_k$  values are zero, the zeros should appear between the 1<sup>st</sup> delay and the last delay. For example, if the right side of the general difference equation was: “ $= x[n] - x[n-3]$ ”, the delay of 1 and the delay of 2 coefficients must appear, so that  $b = [1 \ 0 \ 0 \ -1]$ ;

MATLAB has a function called *impz* which will calculate a system's impulse response, given the A and B coefficient vectors. To find the impulse response of a system, use the function as in the following:

```
>> [h, n]=impz (b, a) ;
```

Here,  $b$  is the vector of input coefficients,  $a$  is the vector of output coefficients,  $h$  is the resulting impulse response, and  $n$  is a vector of the sample numbers corresponding to the impulse response values.

- a) **Create a MATLAB stem plot of the impulse response of the difference equation in part II, step 1 using the *impz* function. Do the MATLAB values of  $h[n]$  match your values?**
- b) **Determine by hand (turn in a table) the impulse response for the system with difference equation given by impulse**

$$y[n] = 0.2(x[n] + x[n-1] + x[n-2] + x[n-3] + x[n-4])$$

**What kind of filter is this? Create a MATLAB plot of the of this impulse response using the coefficients of the difference equation as inputs to *impz*. Do the MATLAB values of  $h[n]$  match your values?**

- c) **Determine by hand (turn in a table) the impulse response for the system with difference equation given by impulse**

$$y[n] = y[n-1] + 0.2 x[n] - 0.2 x[n-5]$$

**What kind of filter is this? Create a MATLAB plot of the of this impulse response using the coefficients of the difference equation as inputs to *impz*. What kind of filter is this? Do the MATLAB values of  $h[n]$  match your values?**

3. If the system impulse response is known, then convolution can be used to compute the output for any input.

- a) **By hand (turn in your convolution table), determine the output of the system with impulse response:**

$$h[n] = \delta[n] + 0.5 \delta[n-1] + 0.25 \delta[n-2] + 0.125 \delta[n-3] - 1.875 \delta[n-4]$$

**if the input to the system is  $x[n] = u[n]$ .**

MATLAB does have a convolution function called *conv*. If you input two vectors, it will convolve the result. You use it as in:

```
>> y = conv(x, h) ;
```

Unlike the *impz* function, it does not return the  $n$ -vector for the  $x$ -axis, so it is a little tricky to determine how to plot its result.

- b) **Create a unit step function for  $-10 < n < 5$ , then convolve it with the impulse response above. Use a stem plot to plot the result, but do not supply the vector for  $x$ -axis values (which would be  $n$ ).**

Compare the values of this stem plot with your result obtained by hand. Do they match? What is the reason why the values go haywire on the right edge?

### III. Difference Equations and Filtering

1. System outputs can be found by convolution, or via the difference equation, as seen in previous steps. In MATLAB, the easier way to calculate the output is using the difference equation coefficients and the *filter* command. The filter command is used as in

```
>> y = filter(b, a, x)
```

Where *a* and *b* are the difference equation coefficient vectors as before, and *x* is the input that you wish to filter.

2. Create a 25 point moving average filter in MATLAB. Moving average filters tend to smooth out data, so it removes high frequency information (that is, it is a low pass filter).

- a) Use this filter on one of your voice files (you'll have to read in the file using *wavread*, then use the filter function). Note the size of the original signal and of the filtered signal. Why are they different?

Size of original signal: \_\_\_\_\_ Size of filtered signal: \_\_\_\_\_

- b) Plot (use the *plot* command) the 1<sup>st</sup> 1000 samples of the original voice signal on the same plot as the 1<sup>st</sup> 1000 samples of the filtered voice signal. Why are the two signals so different in the beginning?

- c) Try plotting (use the *plot* command) the 1<sup>st</sup> 1000 samples of the original signal and then samples 25:1024 of the filtered signal. Is the effect of smoothing apparent in the plot? Do the plots look more comparable? Why would you not be so interested in the 1<sup>st</sup> 24 samples of the filtered signal?

- d) Play the original, and then the filtered signal. Can you hear the smoothing effect? If so, what does it sound like?

- e) Now try filtering one of your music clips with a 100 point moving average filter. Can you hear the difference in the outputs? Plot (use the *plot* command) the 1<sup>st</sup> 1000 samples of the original signal along with the 1<sup>st</sup> 1000 "good" samples of the filtered signal on an axis with accurate time values. Label the plot and turn it in.

**For this lab, answer the questions that are in bold-face font. Turn in all plots called for along with your code to create the plots. I do not need the functions you copied in the 1<sup>st</sup> step.**

## Some Basic Discrete-time (DT) Functions

Function	Mathematical Definition	Sketch
<b>Unit Sequence</b>	$u[n] = \begin{cases} 1, & n \geq 0 \\ 0, & n < 0 \end{cases}$	
<b>Unit Ramp</b>	$\text{ramp}[n] = \begin{cases} n, & n \geq 0 \\ 0, & n < 0 \end{cases}$	
<b>Unit Impulse</b>	$\delta[n] = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases}$	
<b>Periodic Impulse, or Impulse Train</b>	$\delta_N[n] = \sum_{m=-\infty}^{\infty} \delta[n - mN]$	

```

%      Function to generate the discrete-time impulse
%      function defined as one for input integer arguments
%      equal to zero and zero otherwise. Returns "NaN" for
%      non-integer arguments. Works for vectors and scalars
%      equally well.
%
%      function y = impD(n)
function y = impD(n)
    y = double(n == 0) ;      % Impulse is one where
                             % argument is zero and zero
                             % otherwise
    ss = find(round(n) ~= n); % Find non-integer values
                             % of "n"
    y(ss) = NaN ;           % Set corresponding outputs
                             % to "NaN"

```

+++++

```

%      Unit sequence function defined as 0 for input integer
%      argument values less than zero, and 1 for input
%      integer argument values equal to or greater than
%      zero. Returns "NaN" for non-integer arguments. Works
%      for vectors and scalars equally well.
%
%      function y = uD(n)
function y = uD(n)
    y = double(n >= 0) ;      % Set output to one for
                             % non-negative arguments
    ss = find(round(n) ~= n) ; % Find all non-integer
                             % "n's"
    y(ss) = NaN ;           % Set the corresponding
                             % outputs all to "NaN"

```

+++++

```

%      Unit discrete-time ramp function defined as 0 for
%      input integer argument values equal to or less than
%      zero, and "n" for input integer argument values
%      greater than zero. Returns "NaN" for non-integer
%      arguments. Works for vectors and scalars equally
%      well.
%
%      function y = rampD(n)

```

```

function y = rampD(n)

    pos = double(n>0) ;      % Set output to "n" for
                            % positive
    y = n.*pos ;           % "n"
    ss = find(round(n)~=n) ; % Find all non-integer
                            % "n's"
    y(ss) = NaN ;         % Set the corresponding
                            % outputs all to "NaN"

```

+++++

```

%      Discrete-time periodic impulse function defined as 1
%      for input integer argument values equal to integer
%      multiples of "N" and 0 otherwise. "N" must be an
%      integer. Returns "NaN" for non-integer input values.
%      Works for vectors and scalars equally well.
%
%      function y = impND(N,n)

```

```

function y = impND(N,n)
    if N == round(N),
        y = double(n/N == round(n/N)) ; % Set output
                                        % to one
                                        % for all n's
                                        % which are
                                        % integer
                                        % multiples
                                        % of N and
                                        % zero
                                        % otherwise
        ss = find(round (n) ~=n) ;      % Find all
                                        % non-
                                        % integer
                                        % n's"
        y(ss) = NaN ;                 % Set the
                                        % corresponding
                                        % outputs all
                                        % to "NaN"
    else
        disp('In impND, period parameter, N, is not an integer');
    end

```