University of Massachusetts Dartmouth
Department of Electrical and Computer Engineering

ECE 320 DISCRETE-TIME LINEAR SYSTEMS

Fall 2013

MATLAB **Project 1 Solution**

Project 2.7: Discrete-time Convolution

This project provides experience using the `conv` command and keeping track of the time indices associated with each signal vector.

(a) This part requires us to define a vector `x` to represent $x[n] = \delta[n]+\delta[n-2]$ and `h` to represent the signal $h[n] = 2\delta[n+1] - 2\delta[n-1]$. The output `y` is defined using the `conv` command. The challenge is to find the time indices associated with the vector `y`. Remember from class that the index for the first sample of the output is the sum of the first indices for each of the input signals. Likewise, the index of the last sample of the output is the sum of the last indices of the input signals. The following Matlab code defines the signals and their time indices, and plots the output $y[n]$ against the proper time axis. The plot is shown in Figure 1.

```
% Basic Problems
% Part a

% Define amplitude vectors and associated time index vectors for
% each signal
ha = [2 0 -2];
nha = [-1:1]; % indices for ha
xa = [1 0 1];
nxa = [0:2]; % indices for xa
ya = conv(xa,ha);
nya = [(nha(1)+nxa(1)):(nha(end)+nxa(end))];

% Since h goes from n=-1 to n=1 and x goes from 0 to 2, the result of
% the convolution will be 5 points long and go from n=-1 to 3

% plot signal
figure(1)
clf
stem(nya,ya)
title('y[n] for Project 2.7, Part (a)')
xlabel('n (samples)')
ylabel('y[n]')
% Add vertical axis line at 0 just to be fancy.
hold on
plot([nya(1)-0.5,nya(end)+0.5],[0 0])
set(gca,'xtick',nya)
hold off
```

(b) For this part, we want to convolve $x[n] = \delta[n-a] + \delta[n-b]$ with $h[n] = \delta[n-c] + \delta[n-d]$. We know that convolving with an impulse at time n_0 delays the signal by n_0. We can find $y[n]$ using additivity

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

$$
\begin{aligned}
&= x[n] * (\delta[n-c] + \delta[n-d]) \\
&= x[n-c] + x[n-d] \\
&= \delta[n-c-a] + \delta[n-c-b] + \delta[n-d-a] + \delta[n-d-b]
\end{aligned}
$$

Since we are told that $a < b$ and $c < d$ in the problem, the first of these impulses will be at $n = a + c$ and the last impulse will be at $n = b + d$. This confirms what we said in part (a) about when the output time indices should start and stop. If `x` has `nx=a:b` and `h` has `nh=c:d`, then we want `ny=(a+c):(b+d)`. For instance, if $a = 0, b = N-1, c = 0$ and $d = M-1$, then $x[n]$ is N points long, and $h[n]$ is M points long, and $y[n]$ goes from $n = 0$ to $n = N + M - 2$, which is a total of $N + M - 1$ samples. In general, convolving two finite length signals of length N and M produces a result of length $N + M - 1$.

(c) For this part, we approximate the convolution of two infinitely long signals using finite-length segments of the signals. When we do this, we must think carefully to understand when the result matches the result of convolving the infinite signals, and when it displays artifacts due to the finite-length approximations to infinite-length signals. The commands to define the signals and their time indices for the interval specified are

```
% Part c
% define x[n] for 0 <= n <= 24
xc = [0 0 (1/2).^(0:22)];
nxc = 0:24;
% define h[n] for -2 <= n <= 14
% note mistake in book: u[n+2] begins at n=-2
hc = ones(1,17);
nhc = -2:14;
% convolve x and h to get y
yc = conv(xc,hc);
nyc = -2:38;
figure(2)
stem(nyc,yc)
title('Project 2.7, Part (c)')
ylabel('y[n]')
xlabel('n (samples)')
```

Figure 2 shows the resulting plot. The values shown in the plot are correct for the range $-2 \le n \le 16$. To find this, we make sketches of $x[n]$ with flipped and shifted versions of the truncated $h[n] = u[n+2]$ underneath. The plots in Figure 3 were not required for part (c), but help us to see when the overlap between these finite segments of $x[n]$ and $h[n]$ is the same as if they were infinitely long. Computing the convolution at $n = 0$ would be the result of multiplying $x[k]$ in the top panel with $h[-k]$ in the second panel, then adding across k. We can see that this pair for $n = 0$ is the first value when the signals overlap, and that the result wouldn't change if $h[-k]$ extended infinitely far to the left, since those values are all multiplied by zeros. In fact, we can see from this figure that the output would be zero for $n = -2$ or -1 as well, so those values in Figure 2 are also correct. Next, we want to find the last value when the overlap between the signals is the same as if they $h[-k]$ extended infinitely far to the left. Comparing the top panel with the bottom panel, we see that when $n = 16$ the finite length segment $h[-k]$ still has the same overlap it would if $h[-k]$ went infinitely far to the left. However, if we shift even one sample more to the right with $h[17-k]$, the left edge of the finite segment falls inside $x[k]$ with a zero where the infinite signal would still be one.

(d) For this part, we convolve the complicated signal

$$x[n] = \cos(n^2)\sin(2\pi n/5)u[n]$$

with the impulse response

$$h[n] = (0.9)^n\left(u[n] - u[n-10]\right).$$

The following commands define the signal and compute the convolution:

```
% Part d
% define h[n]
nhd = 0:9;
hd = (0.9).^nhd;
% define x[n] to get region needed to find y[n] on 0 <= n <= 99
nxd = 0:99;
xd = cos(nxd.^2).*sin((2*pi/5)*nxd);
yd = conv(xd,hd);
nyd = 0:108;
% remove the last nine points of yd because these are transient artifacts
% due to the fact we are working with a finite segment of x
yd = yd(1:100);
nyd = nyd(1:100);

figure(3)
stem(nyd,yd)
title('Matlab I, Part d')
ylabel('y[n]');
xlabel('n (samples)');
```

(e) This part does a simple example of block convolution where we convolve each block with $h[n]$ separately, then add the results back together. The idea here is that we can break the input signal into blocks of length L, do the convolution separately on each block, then put them back together again using LTI properties to find the overall output. This technique relies on the distributive property

$$\begin{aligned} y[n] &= h[n] * x[n] \\ &= h[n] * \left(\sum_{r=0}^{\infty} x_r[n - rL] \right) \\ &= \sum_{r=0}^{\infty} (h[n] * x_r[n - rL]) \\ &= \sum_{r=0}^{\infty} y_r[n - rL] \end{aligned}$$

where $y_r[n] = x_r[n] * h[n]$, i.e., the convolution for an individual block.

For this problem, $k = 50$, since the second block is shifted 50 samples from the first block.

```
% Part (e)
% Break input signal into segments of length 50.
x0e = xd(1:50);
x1e = xd(51:100);

% Convolve each segment with h[n] to get y0 and y1
y0e = conv(x0e,hd);
y1e = conv(x1e,hd);

% add segments together with y1e delayed by 50;
ye = [y0e zeros(1,50)]+[zeros(1,50) y1e];
% discard last nine samples as transient
nye = 0:99;
ye = ye(1:100);

figure(4)
```

```
stem(nye,ye)
title('Matlab I, Part e')
xlabel('n (samples)')
ylabel('y[n]')

% verify no difference between y computed in one convolution and
% y computed with block convolution

max(abs(yd-ye))

% This command gives
%ans =
%  2.2204e-16
% which is the value of eps, Matlab's numerical threshhold.
```

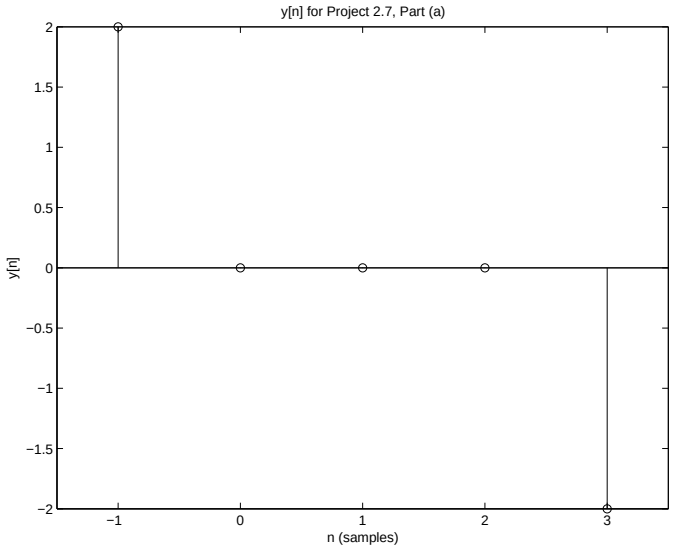

Figure 1: Output signal $y[n]$ for Project 2.7, Part (a)

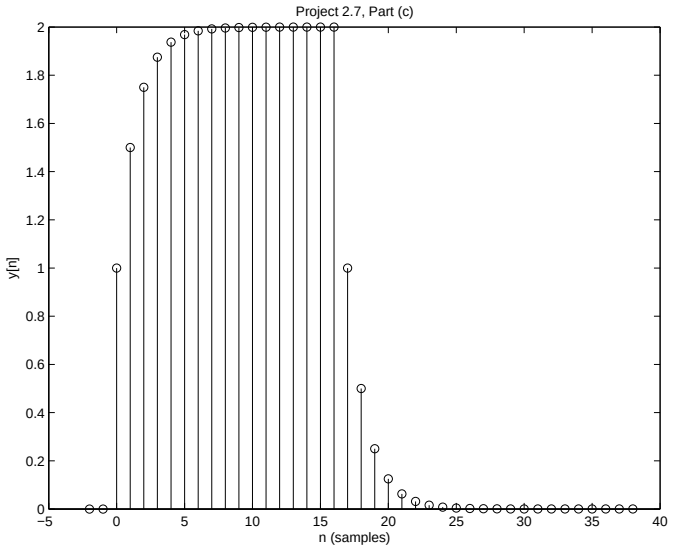

Figure 2: Ouptut signal $y[n]$ for Project 2.7, Part (c)

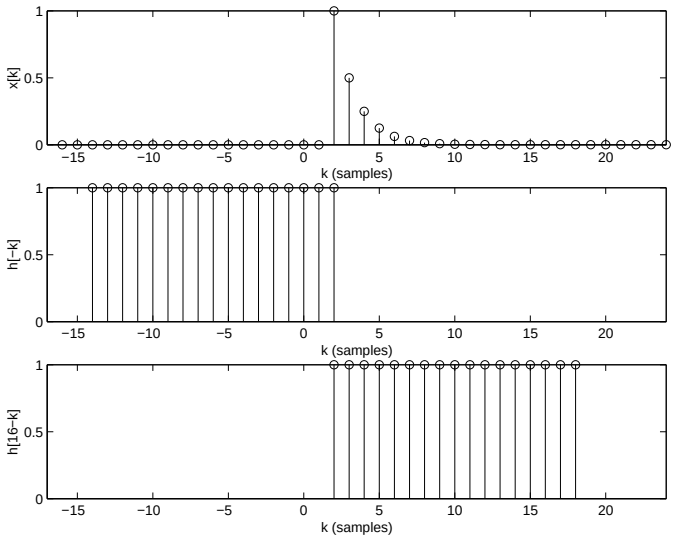

Figure 3: Sketch to shown when y has the correct values for Project 2.7, Part (c)

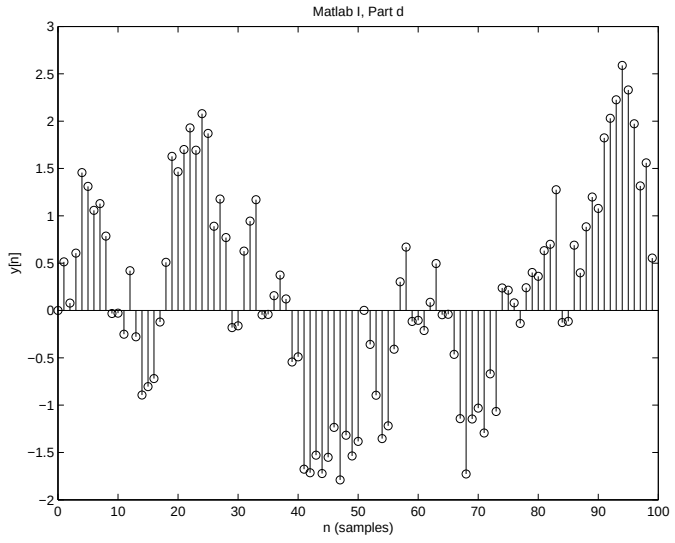

Figure 4: Ouptut signal $y[n]$ for Project 2.7, Part (d), doing the convolution for the entire signal at once.

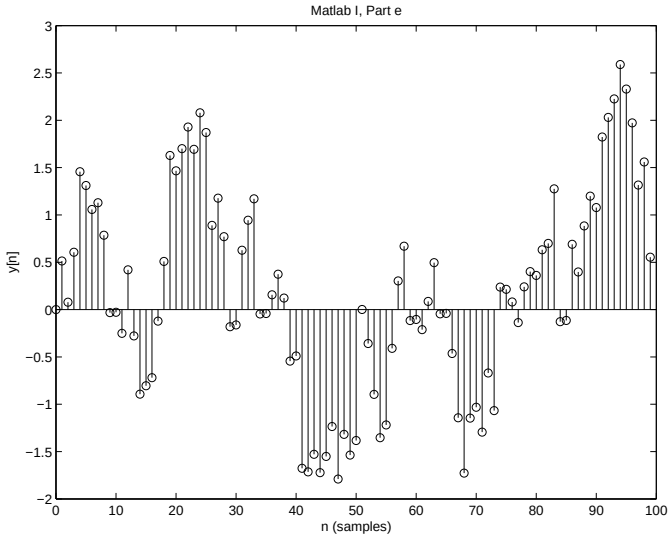

Figure 5: Ouptut signal $y[n]$ for Project 2.7, Part (e) built from combining the two blocks separately convolved.

Project 2.10: Echo Cancellation via Inverse Filtering

This project demonstrates that you can remove an echo from a signal using a filter which is the inverse system to the impulse response of the echo which distorted the speech. The inverse system is actually an IIR system, but we will see that a relatively long finite approximation removes almost all of the echo, so that it is hard to hear what remains.

Before we begin working, we need to load the data for this project: `load lineup.mat`

(a) To find the impulse response, set $x[n] = \delta[n]$, and the resulting output $y[n] = h[n]$. From Eq. (2.21), we get $h[n] = \delta[n] + \alpha\delta[n-N]$. So, the impulse response is an impulse at $n = 0$ with height 1 and another one with height $\alpha = 0.5$ at $n = N = 1000$.

The following commands define the impulse response for $0 \leq n \leq 1000$ and plot it, giving Figure 6. We see the expected values at $n = 0$ and $n = 1000$ for the echo.

```
N = 1000;
alpha = 0.5;
he = [1 zeros(1,N-1) alpha];
nhe = [0:N];

figure(1)
stem(nhe,he)
xlabel('Time (samples)')
ylabel('he[n]')
title('Project 2.10, Part (a): Impulse response for echo system')

if FINALPLOTS
  print -deps proj210a.eps
end
```

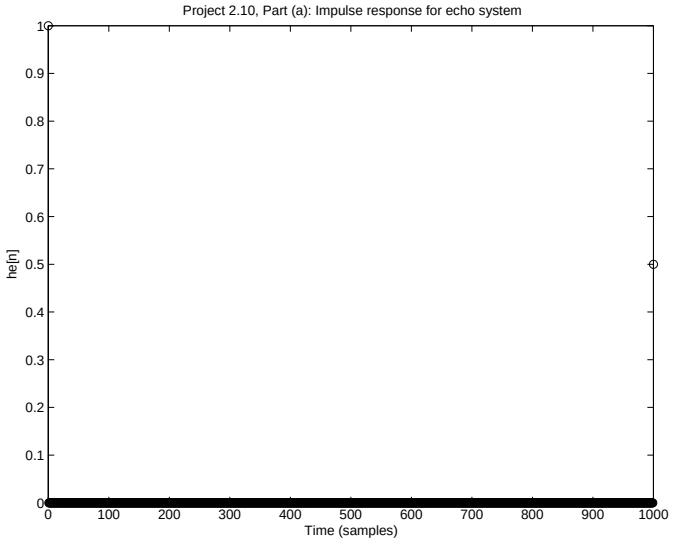

Figure 6: Impulse response for echo system from Eq. (2.21)

(b) To solve this part, we set $y[n]$ equal between Eq. (2.21) and (2.22)

$$x[n] + \alpha x[n-N] = z[n] + \alpha z[n-N].$$

Clearly, if $z[n] = x[n]$, both sides will obviously be the same, so $z[n] = x[n]$ is a valid solution to this difference equation.

(c) For the echo removal system, remember that $y[n]$, the echo signal, is the input, so its coefficients in the LCCDE set the `binv` vector. Similarly, $z[n]$, the "de-echoed" signal, is the output, so its coefficients in the LCCDE set the `ainv` vector. The following commands define the impulse used as the input to filter, and the filter coefficients `ainv` and `binv` for the inverse system, then compute the impulse response, $h_{er}[n]$. Figure 7 plots the impulse response – which was not required – but I thought folks might be curious what it looked like.

```
% Define a signal to use as delta[n] as the system input
d = [1 zeros(1,4000)];
nd = [0:4000];
% Define vectors a and b for the inverse system using the
% coefficients in Eq. (2.22).  Note that in this equation, y[n] is
% now the input (and thus determines the b coefficients), while
% z[n], the "de-echoed" is the output, (and thus determines the a
% coefficients).
binv = 1;
ainv = [1 zeros(1,N-1) alpha];
her = filter(binv,ainv,d);
nher = nd; % Output from a filter has the same indices as the input
           % vector (here d)

figure(2)
clf
stem(nher,her)
xlabel('Time (samples)')
ylabel('her[n]')
title(['Project 2.10, Part (c): Impulse response of echo removal' ...
       ' system']);

% Assignment didn't ask for a plot of the impulse response system,
% but it is interesting to see what it looks like
if FINALPLOTS
  print -deps proj210c.eps
end
```

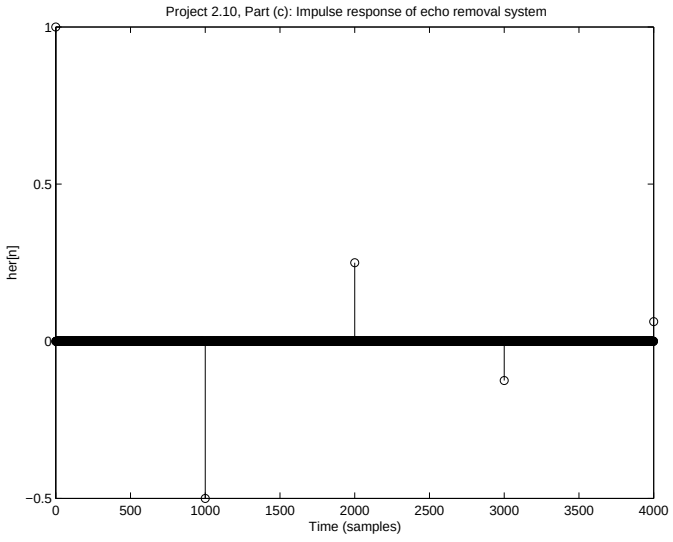

Figure 7: Impulse response for echo removal system from Eq. (2.22)

(d) To remove the echo, we use the same filter as in part (c), but instead of using $\delta[n]$ as the input, we use the echo signal $y[n]$. Figure 8 is a plot of the signal $z[n]$ with the echo removed. Since there are too many values, we use `plot` instead of `stem` to give a sense of the envelope. Comparing against a plot of $y[n]$, we see that the signal has been shortened since the echo was removed. Listening to $z[n]$ in comparison to $y[n]$, it is clear that the echo was removed, or at least greatly reduced.

```
% Now use the same filter coefficients to process the echoed speech
% as input and get z[n] out
z = filter(binv,ainv,y);
nz = 0:6999;

figure(3)
clf
plot(nz,z);
title('Project 2.10, Part (d), Echo-Removed Speech z[n]')
xlabel('Time (samples)')
ylabel('z[n]')

if FINALPLOTS
  print -deps proj210d.eps
end

% sound(z)
% the echo-cancelled version sounds much cleaner, and also appears
% cleaner (and shorter) on the plots
```

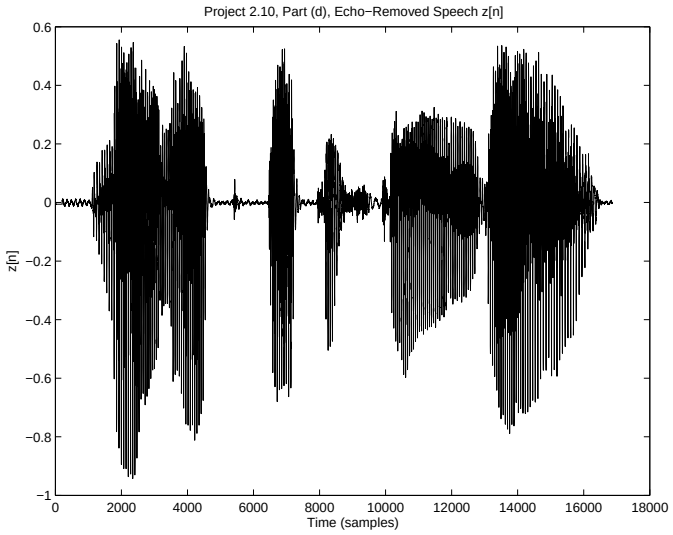

Figure 8: Signal $z[n]$ output from the echo removal system for Project 2.10 part (d)

(e) To find the impulse response of two cascaded systems, we convolve their impulse response. As we saw in Project 2.7, we define the time indices for the output of the convolution using the start and end indices for each individual impulse response $h_e[n]$ and $h_{er}[n]$. Figure 9 shows the overall impulse response for the cascade of the echo system and the echo removal system used in part (d). As expected for an inverse, the result has height 1 at $n = 0$, and is zero for almost all of the rest of the interval. Looking carefully, there is a small sample at $n = 5000$ with height of 0.05. This residual "echo" occurs because the true echo removal system in Eq. (2.22) is IIR, but we are approximating it with a FIR system in `her` that is only 4000 points long. This produces a small error at the end of the convolution, but it is 10 times smaller than the original echo, and thus not obvious in $z[n]$ when you listen to it.

```
% part e
hoa = conv(he,her);
nhoa = [(nhe(1)+nher(1)):(nhe(end)+nher(end))];
figure(4)
clf
stem(nhoa,hoa)
title('Project 2.10, Part (e), Overall Impulse Response h_{oa}[n]')
ylabel('h_{oa}[n]')
xlabel('Time (samples)')
```

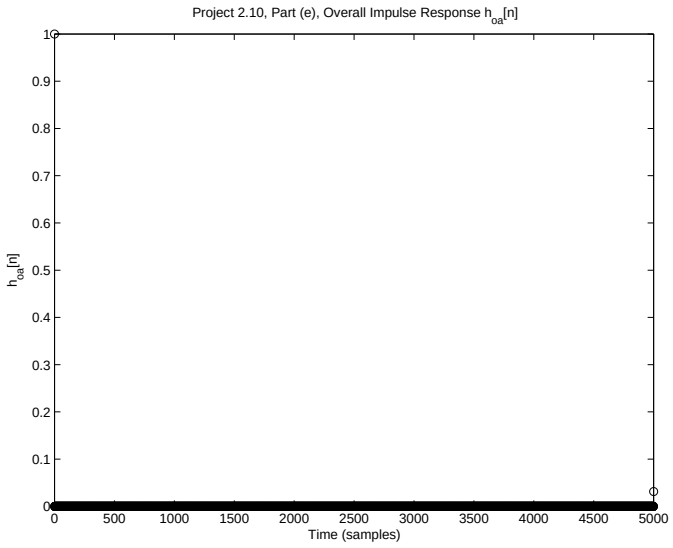

Figure 9: Impulse response for cascade of echo and echo removal system

Project 3.4: Eigenfunctions of Discrete-Time LTI Systems

This project examines four different input signals to an LTI system to see which of them are eigenfunctions. Remember than if $x[n]$ is an eigenfunction, the output of any LTI system will be $y[n] = Ax[n]$, i.e., a scaled version of the input. We can check this in Matlab by dividing the output by the input at each sample to see if this is a constant vs. time.

(a) For this part, we define the time index used for all four inputs and define vectors representing $x[n]$ over this interval. Figures 10–13 show the results. For $x_1[n]$, we plot the real and imaginary parts separately.

```
% Part (a)
% Define the time index vector used by all of the signals
n = -20:100;

% Define the four input signals
x1 = exp(j*(pi/4)*n);
x2 = sin((pi*n/8)+(pi/16));
x3 = (9/10).^n;
x4 = n+1;

% Plot each of the four input signals
figure(1)
subplot(211)
stem(n,real(x1))
xlabel('Time (samples)')
ylabel('Real(x_1[n])')
title('Project 3.4, Part (a): x_1[n]');
subplot(212)
stem(n,imag(x1))
xlabel('Time (samples)')
ylabel('Imag(x_1[n])')

if FINALPLOTS
  print -deps proj34x1.eps
end

figure(2)
stem(n,x2);
xlabel('Time (samples)')
ylabel('x_2[n])')
title('Project 3.4, Part (a): x_2[n]');

if FINALPLOTS
  print -deps proj34x2.eps
end

figure(3)
stem(n,x3);
xlabel('Time (samples)')
ylabel('x_3[n])')
title('Project 3.4, Part (a): x_3[n]');

if FINALPLOTS
  print -deps proj34x3.eps
end

figure(4)
stem(n,x4);
```

```
xlabel('Time (samples)')
ylabel('x_4[n])')
title('Project 3.4, Part (a): x_4[n]');

if FINALPLOTS
  print -deps proj34x4.eps
end
```

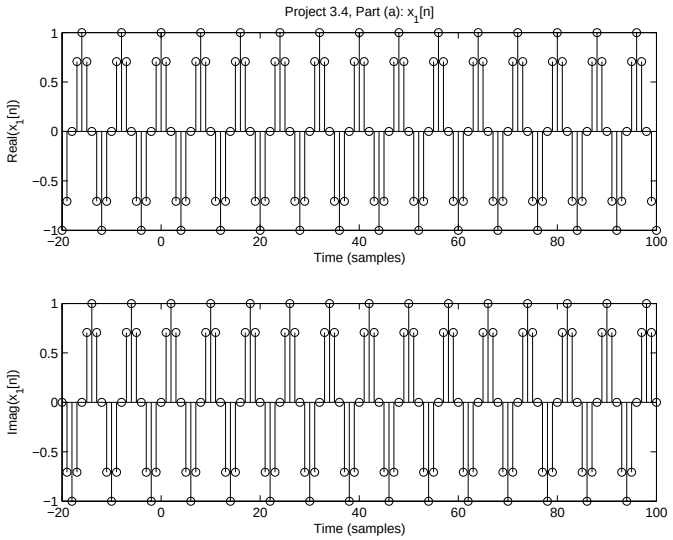

Figure 10: Input signal $x_1[n]$ for $-20 \leq n \leq 100$

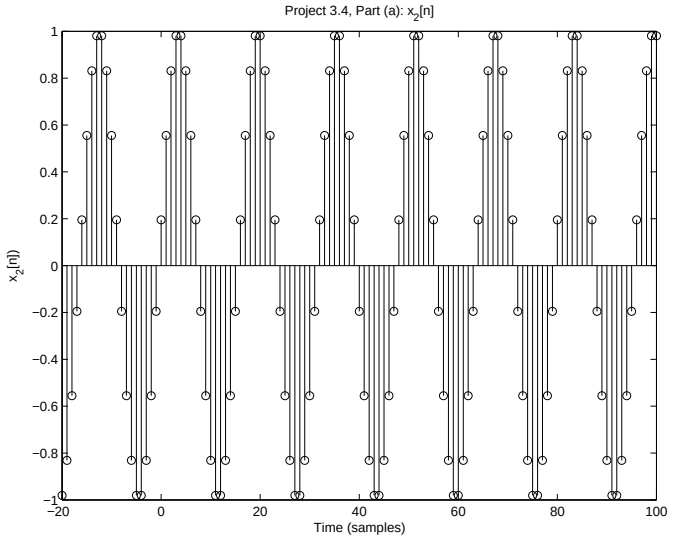

Figure 11: Input signal $x_2[n]$ for $-20 \leq n \leq 100$

(b) For this part, we use our old friend `filter` to implement the causal LTI system satisfying the LCCDE given by

$$y[n] - 0.25y[n-1] = x[n] + 0.9x[n-1].$$

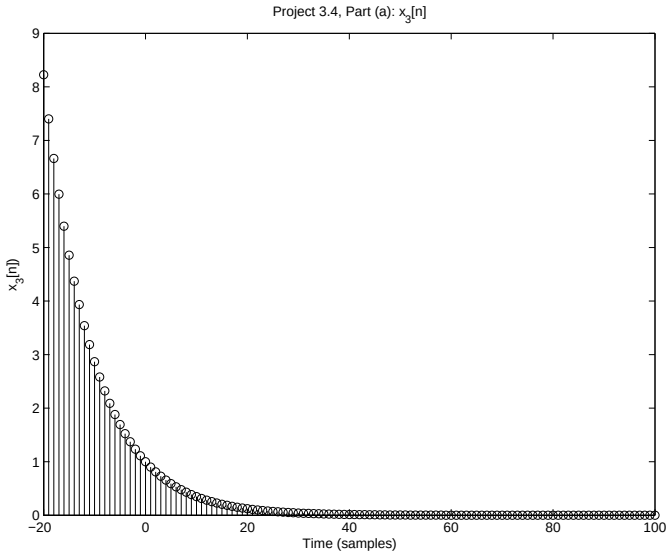

Figure 12: Input signal $x_3[n]$ for $-20 \leq n \leq 100$

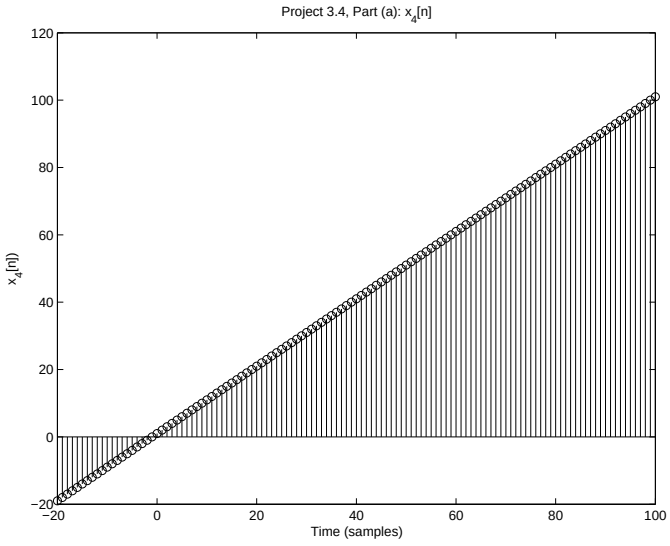

Figure 13: Input signal $x_4[n]$ for $-20 \leq n \leq 100$

We compute the output $y[n]$ corresponding to each of the four inputs defined in part (a). The plots ignore $-20 \leq n \leq -1$ since `filter` assumes initial rest, which means that the input signal turns on at the first sample given, rather than continuing infinitely far into the past. This results in a transient at the start that does not represent what happens with a true eigenfunction response. This transient has vanished by $n = -1$, so the results from $n = 0$ forward is what is expected if the eigenfunction had been the input for the infinite past. Figures 14–17 represent the output obtained.

```
% Define coefficient vectors for the filter from Eq. (3.3)
a = [1 -0.25];
b = [1 0.9];

% Find the four output from each input
y1 = filter(b,a,x1);
y2 = filter(b,a,x2);
y3 = filter(b,a,x3);
y4 = filter(b,a,x4);

figure(5)
subplot(211)
stem(n,real(y1))
a = axis;
axis([0 100 -2.5 2.5]);
title('Project 3.4, Part (b): y_1[n]')
xlabel('Time (samples)')
ylabel('Real(y_1[n])')
subplot(212)
stem(n,imag(y1))
a = axis;
axis([0 100 -2.5 2.5]);
xlabel('Time (samples)')
ylabel('Imag(y_1[n])')

if FINALPLOTS
  print -deps proj34y1.eps
end

figure(6)
stem(n,y2);
xlabel('Time (samples)')
ylabel('y_2[n])')
title('Project 3.4, Part (b): y_2[n]');
axis([0 100 -2.5 2.5])

if FINALPLOTS
  print -deps proj34y2.eps
end

figure(7)
stem(n,y3);
xlabel('Time (samples)')
ylabel('y_3[n])')
title('Project 3.4, Part (b): y_3[n]');
axis([0 100 0 20])

if FINALPLOTS
  print -deps proj34y3.eps
end
```

```
figure(8)
stem(n,y4);
xlabel('Time (samples)')
ylabel('y_4[n])')
title('Project 3.4, Part (b): y_4[n]');
axis([0 100 0 300])

if FINALPLOTS
  print -deps proj34y4.eps
end
```

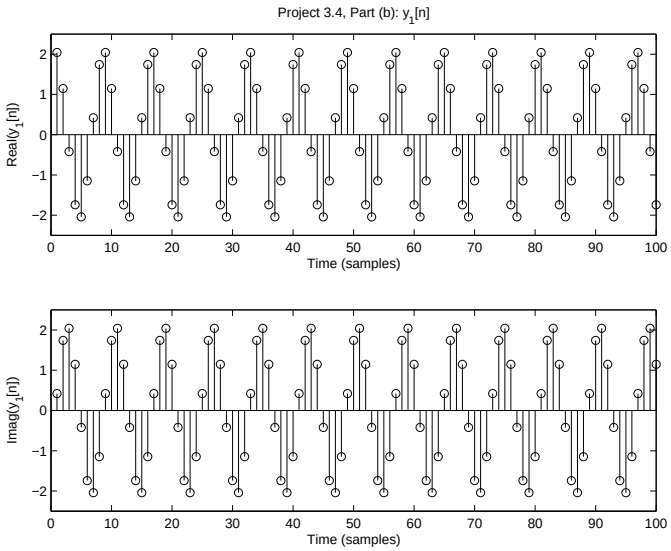

Figure 14: Output signal $y_1[n]$ for $0 \leq n \leq 100$

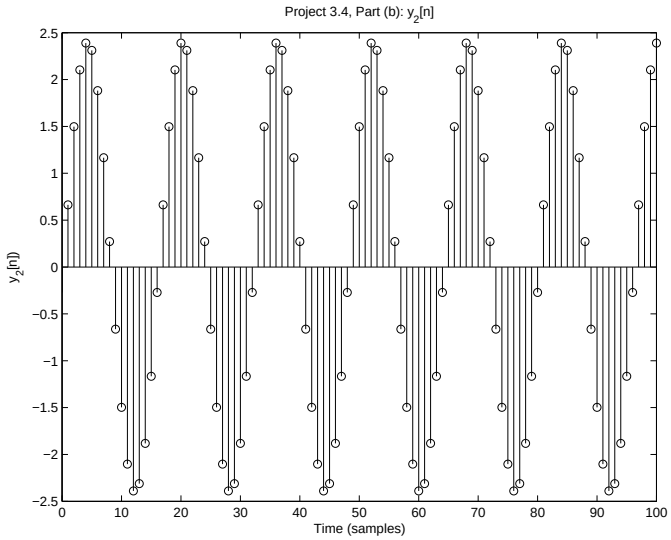

Figure 15: Output signal $y_2[n]$ for $0 \leq n \leq 100$

(c) Lastly, we compute the ratio $y[n]/x[n]$ for each value of n and plot the results for the

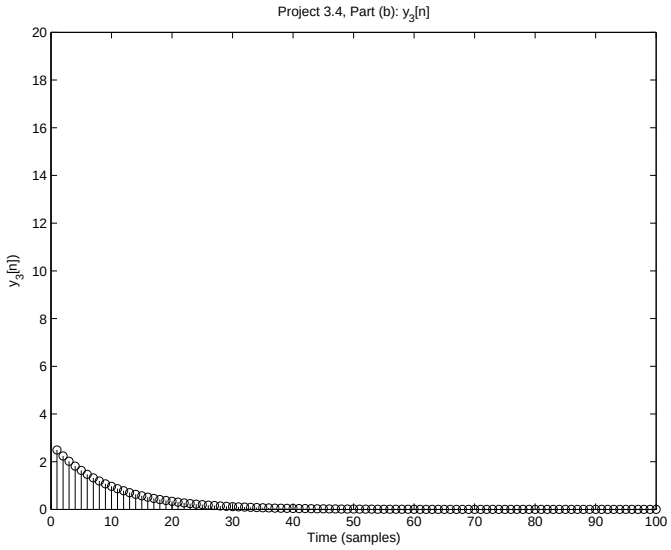

Figure 16: Output signal $y_3[n]$ for $0 \leq n \leq 100$

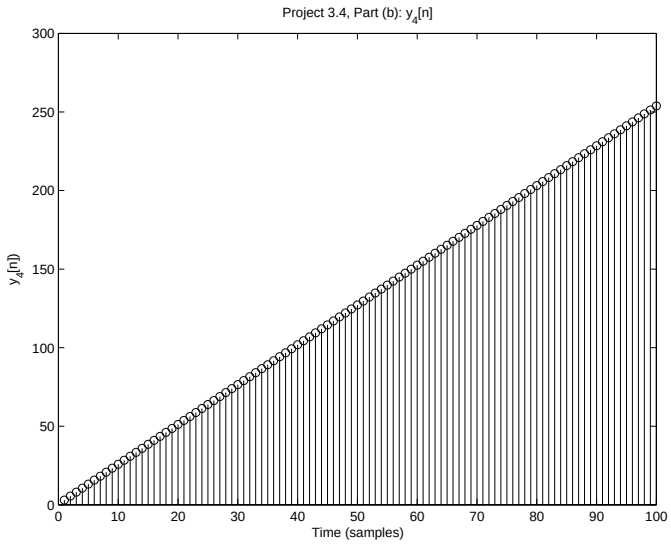

Figure 17: Output signal $y_4[n]$ for $0 \leq n \leq 100$

interval $0 \leq n \leq 100$. If the input is an eigenfunction, this ratio should be a constant, or nearly so within numerical error that is undetectable on the plot. From Figures 18–21, we see that `H1` and `H3` are constants, and that `H2` and `H4` vary with n. This means that $x_1[n]$ and $x_3[n]$ are eigenfunctions, but $x_2[n]$ and $x_4[n]$ are not.

```
% Part (c)
% Compute the ratio of input to output at each time for each pair
% to see if it is a constant scalar
H1 = y1./x1;
H2 = y2./x2;
H3 = y3./x3;
H4 = y4./x4;

% Now plot these values for the region of interest.

figure(9)
subplot(211)
stem(n,real(H1))
axis([0 100 0 2])
xlabel('Time (samples)')
ylabel('Real(H_1)')
title('Project 3.4, Part (c): H_1')
subplot(212)
stem(n,imag(H1))
axis([0 100 -1.5 0])
xlabel('Time (samples)')
ylabel('Imag(H_1)')

if FINALPLOTS
  print -deps proj34H1.eps
end

figure(10)
stem(n,H2)
axis([0 100 -2 6])
xlabel('Time (samples)')
ylabel('H_2')
title('Project 3.4, Part (c): H_2')

if FINALPLOTS
  print -deps proj34H2.eps
end

figure(11)
stem(n,H3)
axis([0 100 0 3])
xlabel('Time (samples)')
ylabel('H_3')
title('Project 3.4, Part (c): H_3')

if FINALPLOTS
  print -deps proj34H3.eps
end

figure(12)
stem(n,H4)
axis([0 100 0 3])
xlabel('Time (samples)')
ylabel('H_4')
```

```
title('Project 3.4, Part (c): H_4')

if FINALPLOTS
  print -deps proj34H4.eps
end

>> mean(H1(n>=0))
ans =
   1.7415 - 1.1470i
>> mean(H3(n>=0))
ans =
    2.7692
```

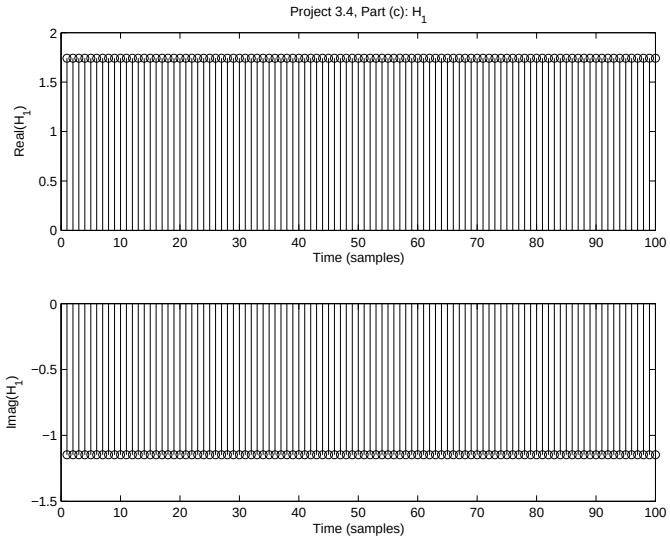

Figure 18: Ratio $H_1[n]$ for $0 \leq n \leq 100$

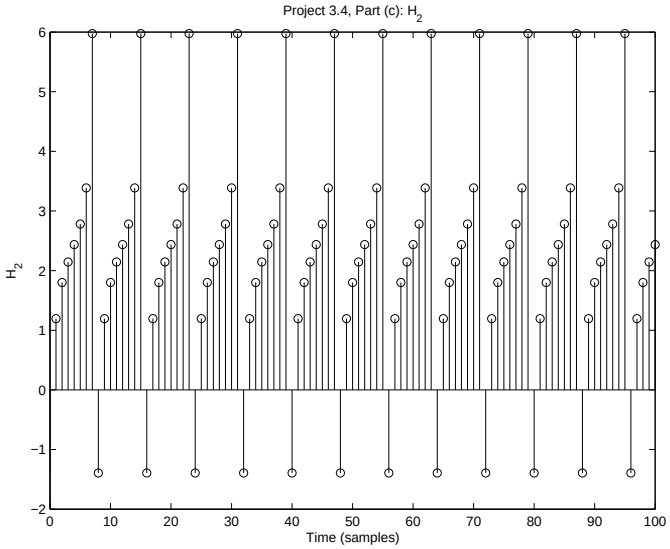

Figure 19: Ratio $H_2[n]$ for $0 \leq n \leq 100$

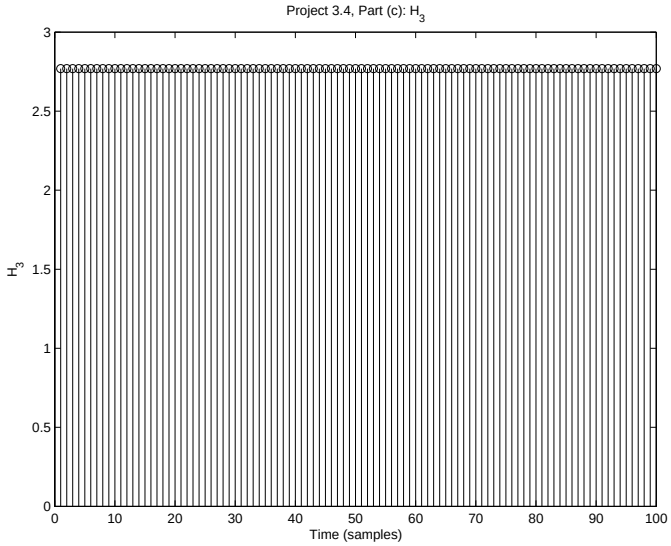

Figure 20: Ratio $H_3[n]$ for $0 \leq n \leq 100$

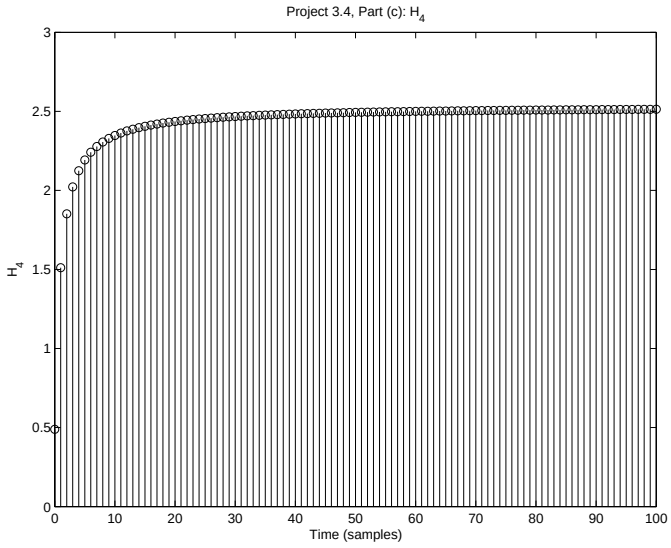

Figure 21: Ratio $H_4[n]$ for $0 \leq n \leq 100$

We can estimate the eigenfunctions for $x_1[n]$ and $x_3[n]$ by computing the average value on the region for $n \geq 0$. The following commands to this, and find $H_1 = 1.74 - j1.15$ and $H_3 = 2.77$. You can also estimate these values from the plots of `H1` and `H3`.