Lab 3: DT Filters - Non-recursive and Recursive

Problem 1: Non-recursive filters for image deblurring

In this exercise you will learn how to express DT convolution in a matrix/vector format and how to use this format for deblurring images.

The blurring is implemented using a simple non-recursive DT filter and you will compute the frequency response of the blur filter.

To do this you will need to use the `freqz` command (Tutorial 3.2 in *Computer Explorations*).

In the following set of tasks, enter and execute your code in the "Code" sections, and answer the underlined questions in in a "Text " section following your code section.

Background: Digital Image Representation

Digital images are made up of many "dots," or pixels (picture elements). By placing these pixels close enough to each other, the images viewed on a computer display or hardcopy appear to be continuous. On a digital computer, the brightness and color information for each pixel is encoded by a number in an array, or, equivalently, an element in a matrix. The location of each value within the array is indexed by two integers, e.g., *X(*3, 4) identifies the pixel value located in the matrix *X* in the third row, fourth column.

Usually the values in the arrays are integers from 0 to $2^n - 1$, where *n* is the number of bits used to represent the brightness of each pixel. For example, consider a black-and-white picture where the luminance, or brightness, of each pixel is stored using 8 bits. In this case, the relative brightness of each pixel can be represented as one of 256 possible levels, called gray levels. Usually black is encoded by a value of 0 and white by a value of 255.

The color information for pixels in the image also is usually encoded as integers stored in matrix arrays. In MATLAB, images are stored as integers in arrays and the color information---how each pixel value maps to a certain color--- is stored separately. In this lab exercise, we use a grayscale image.

Background: Modeling Linear Blurs for Image Deblurring

A horizontal motion blur represents a linear system. Horizontal blurring causes each pixel in an image to contain some information from *N* previous pixels of the same row. A simple model of horizontal blurring using DT convolution is

$$y[l,n] = \sum_{k=\max(0,n-N+1)}^{n} x[l,k]h[n-k] \qquad (1)$$

where $h[n]$ is the unit impulse response

$$h[n] = 1/N \text{ for } n = 0, \ldots, N-1$$

and $x[m,n]$ and $y[m,n]$ are arrays of numerical values that represent values in the original and blurred pictures, respectively. Note that this "blurring" operation corresponds to replacing a pixel values by an average of the surrounding pixel values. The first index denotes the mth row, the second index indicates the nth column, and the image is $K \times L$. For each row l, the DT convolution above in Eq. (1) can be expressed in matrix form as $(\mathbf{y}_l)^\top = H(\mathbf{x}_l)^\top$, where $\mathbf{x}_l$ is the lth row of the original image, $\mathbf{y}_l$ is the lth row of the blurred image and H is a suitably defined $L \times L$ matrix.

Specifically, Eq. (1) can be written in matrix form as

$$\begin{bmatrix} y[l,0] \\ y[l,1] \\ \vdots \\ y[l,L-1] \end{bmatrix} = \begin{bmatrix} h[0] & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ h[1] & h[0] & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ h[N-1] & h[N-2] & h[N-3] & \ldots & h[0] & 0 & \ldots & \ldots & 0 \\ 0 & h[N-1] & h[N-2] & \cdots & \cdots & h[0] & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & h[N-1] & \cdots & \cdots & h[1] & h[0] \end{bmatrix} \begin{bmatrix} x[l,0] \\ x[l,1] \\ \vdots \\ x[l,L-1] \end{bmatrix}.$$

The $L \times L$ matrix H has a **banded** structure and is called a Toeplitz matrix. You can use the **`toeplitz`** command in MATLAB to quickly construct H.

If you transpose both sides of the matrix equation above for each row and then form image matrices from the x's and y's, the blurring operation can be represented by

$$\begin{bmatrix} y[0,0] & \cdots & y[0,L-1] \\ \vdots & \vdots & \vdots \\ y[K-1,0] & \cdots & y[K-1,L-1] \end{bmatrix} = \begin{bmatrix} x[0,0] & \cdots & x[0,L-1] \\ \vdots & \vdots & \vdots \\ x[K-1,0] & \cdots & x[K-1,L-1] \end{bmatrix} \cdot H^\top$$

yielding $Y = XH^\top$. Since H is square, deblurring can be accomplished by simply inverting the matrix $H^\top$ to solve for $X = Y(H^\top)^{-1}$. Note that the matrix ordering is important in the matrix multiplication.

Task 1. Load the MATLAB file `Lab3_S25.mat`. This will load two variables, blur_S25 and `map_S25`.

The first contains a blurred image of a vehicle and the second is a colormap.

Then show the blurred image blur_S25 by entering the MATLAB commands:

`imshow(blur_S25)`

`colormap(map_S25)`

```
load('Lab3_S25.mat');
imshow(blur_S25);
colormap(map_S25);
```

Task 2. Figure out how many of the previous pixel values are affecting each pixel in the same row of the blurred image (i.e., find the length N of the blur filter) by trial and error.

To do this, write an M-file with a function called `deblur` to deblur the image for different values of N. The value of N will be between 1 and the number of columns in the image. When you can see the vehicle and identify the license plate number of the car (a vanity plate), you have found the correct value of N. With exactly the correct value of N, you will see the image with no distortion artifacts, such as vertical bands, shifted parts of the image, black-white inversion, or faint ghosts (even on Halloween). If you are off by even just 1, you will see some of these effects.

Determine the size of the image. Run your function with the correct value of N and show the resulting deblurred image.

Identify the exact length N of the blur filter.

Identify the state, license plate number, the registration date (month and year), and the additional message on the license plate.

Do you know the name of the font used for actual license plate numbers in California?

The length of the blur is $N = 1290$.

The California license plate number is "I 'heart' ECE 101". The registration date is May 2025. The additional message is "Do The Conv-*-lution!".

The font used for the numbers on California license plates is called **Penitentiary Gothic.** Seriously.

```
size(blur_S25)
```

```
ans = 1×2
        2589        1975
```

```
output=deblur(blur_S25,1290);
imshow(output);
```

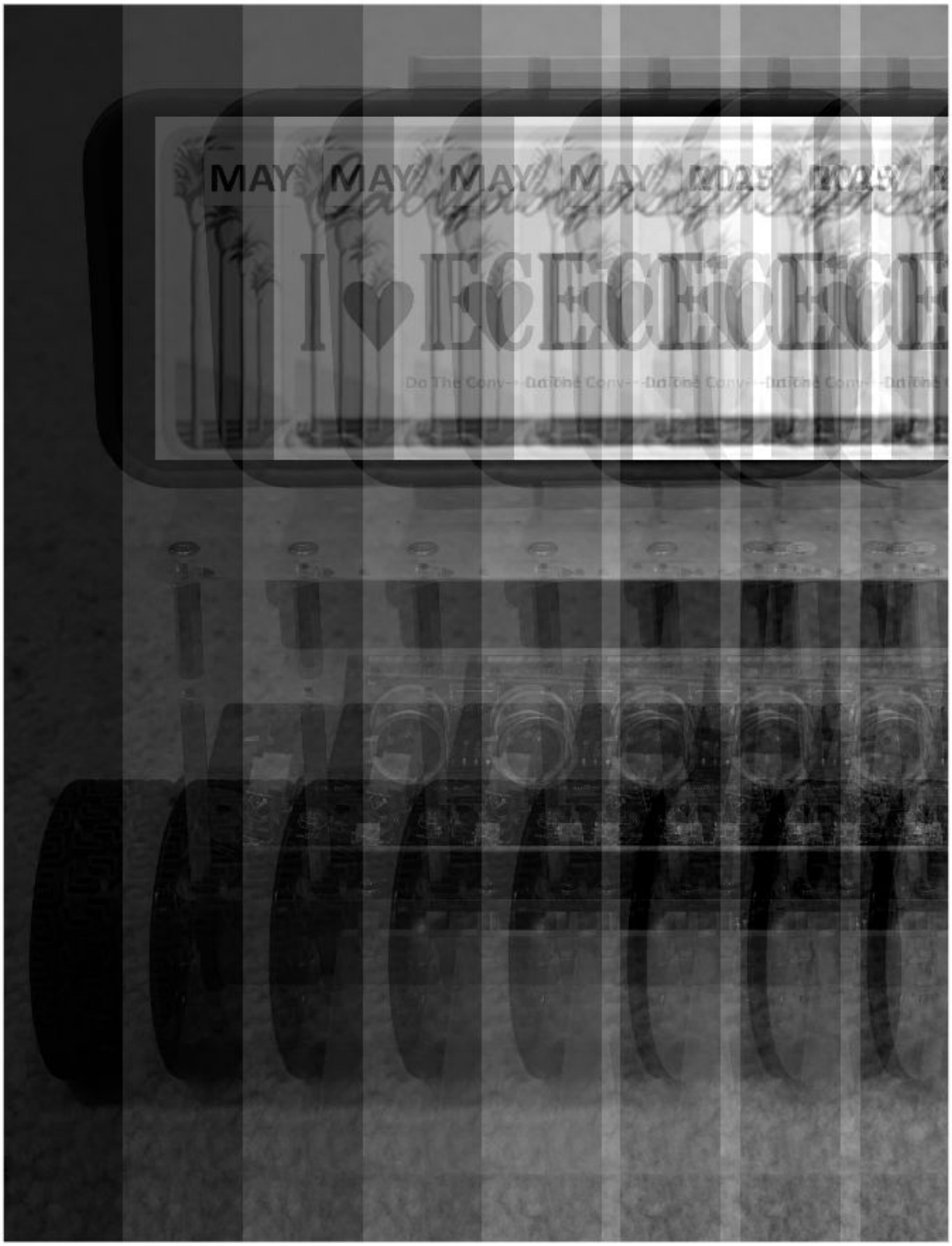

MAY
California
2025
I ♥ ECE101
Do The Conv-*-lution!

Task 3: Print out the deblurring results for at least 4 incorrect values of N that you tried. (If you got the correct N in fewer tries, try a variety of at least 4 incorrect values anyway.) Include at least one value that is very close, but not equal the true value of N.

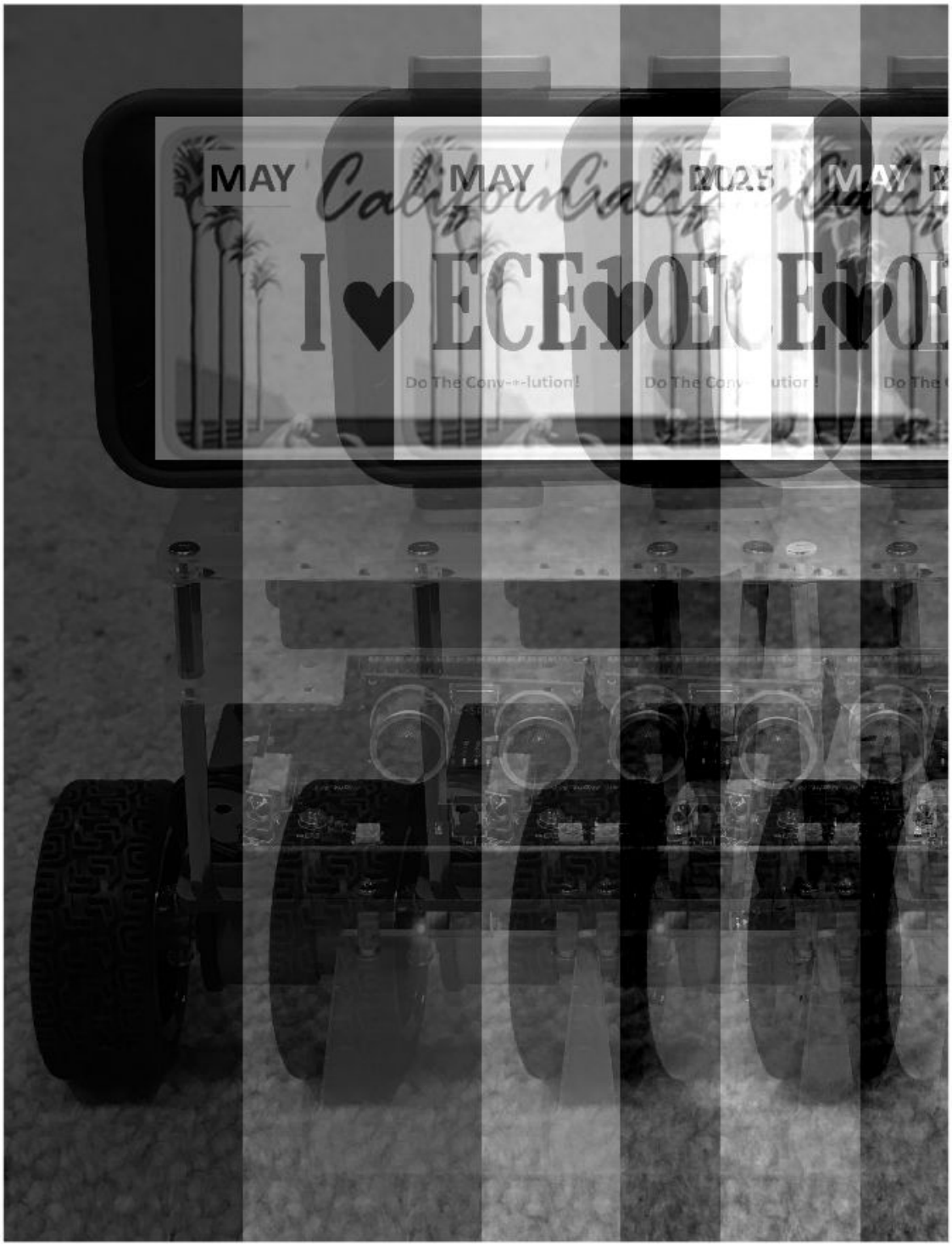
Based upon what you see with a few different incorrect values of N, how would you characterize an improperly deblurred image? (Your answer can be qualitative.)

There are many artifacts, including vertical bands, shifted parts of the image, black-white inversion, and faint ghost images. Some of these are still visible even when the guessed blur filter length is pretty close to the actual value of $N = 1290$. Zooming in also reveals image pixelation effects not related to the blurring. Notice that even when your incorrect value of N is off by only 1 (i.e., 1289), there are noticeable distortions in the reconstructed image.

```
output=deblur(blur_S25,250);
imshow(output)
```




```
output=deblur(blur_S25,500);
imshow(output)
```




```
output=deblur(blur_S25,1000);
imshow(output)
```


```
output=deblur(blur_S25,1500);
imshow(output)
```


```
output=deblur(blur_S25,1300);
imshow(output)
```


```
output=deblur(blur_S25,1289);
imshow(output)
```

Task 4. Write a difference equation for the length-N blur filter. Write the impulse response $h[n]$.

Difference equation:

$$y[n] = \frac{1}{N}\sum_{k=0}^{N-1} x[n-k]$$

Impulse response:

$$h[n] = \frac{1}{N}\sum_{k=0}^{N-1} \delta[n-k]$$

Task 5. Now, consider blur filters of length N=4 and N=7.

Part 1. Analytically determined the frequency responses $H(e^{j\omega})$ of each of these two blur filters.

Plot the magnitudes $|H(e^{j\omega})|$.

The eigenfunction property of LTI systems and signals $x[n] = e^{j\omega n}$ gives:

$$\begin{aligned} H(e^{j\omega})\ e^{j\omega n} &= (1/N)\sum_{k=0}^{N-1} e^{j\omega(n-k)} \\ &= (1/N)\left(\sum_{k=0}^{N-1} (e^{-j\omega})^k\right) e^{j\omega n} \end{aligned}$$

Solving for $H(e^{j\omega})$ and evaluating the geometric series:

$$\begin{aligned} H(e^{j\omega}) &= (1/N)\frac{1-e^{-j\omega N}}{1-e^{-j\omega}} \\ &= (1/N)\frac{e^{-j\omega N/2}}{e^{-j\omega/2}}\left(\frac{e^{j\omega N/2}-e^{-j\omega N/2}}{e^{j\omega/2}-e^{-j\omega/2}}\right) \\ &= (1/N)e^{-j\omega(N-1)/2}\ \frac{\sin(\omega N/2)}{\sin(\omega/2)} \end{aligned}$$

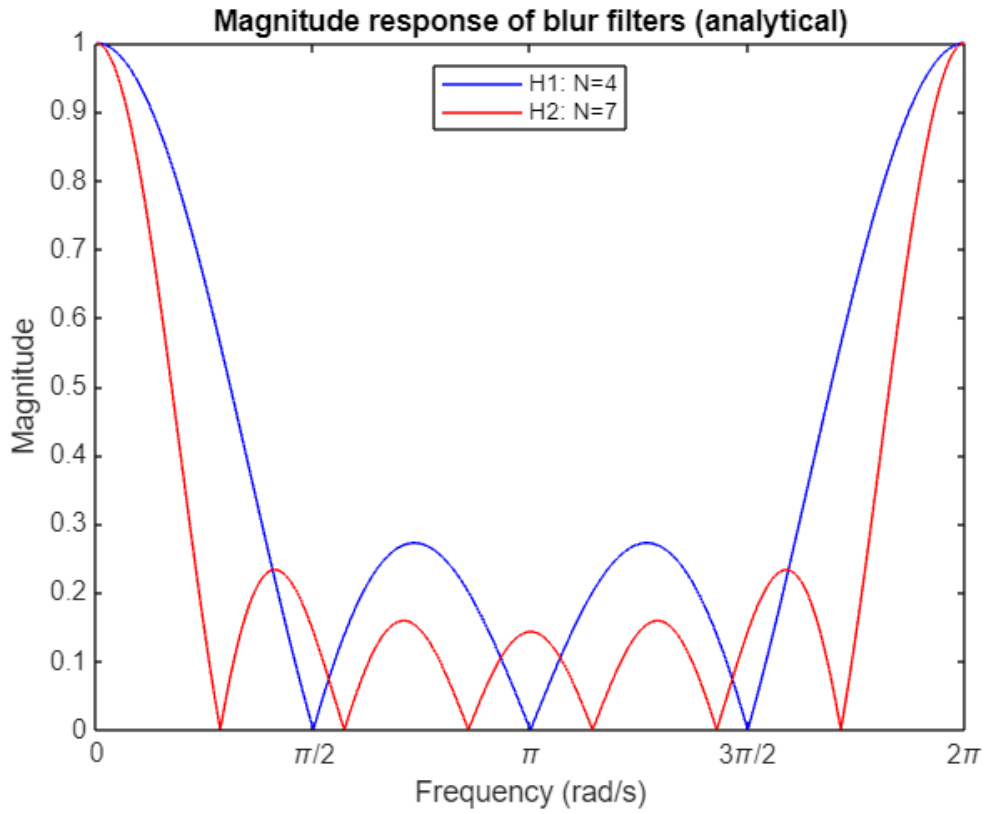
So

$$H_1(e^{j\omega}) = (1/4)e^{-j3\omega/2}\ \frac{\sin(2\omega)}{\sin(\omega/2)}$$

$$H_2(e^{j\omega}) = (1/7)e^{-j3\omega}\ \frac{\sin(7\omega/2)}{\sin(\omega/2)}$$

```
figure
w=linspace(0,2*pi,1024);
H1=(1/4)*exp(-j*3*w/2).*sin(2*w)./sin(w/2);xlim([0 2*pi]);
H2=(1/7)*exp(-j*3*w).*sin(7*w/2)./sin(w/2);
hold on
plot(w,abs(H1),'b');
plot(w,abs(H2),'r');
box on;
```

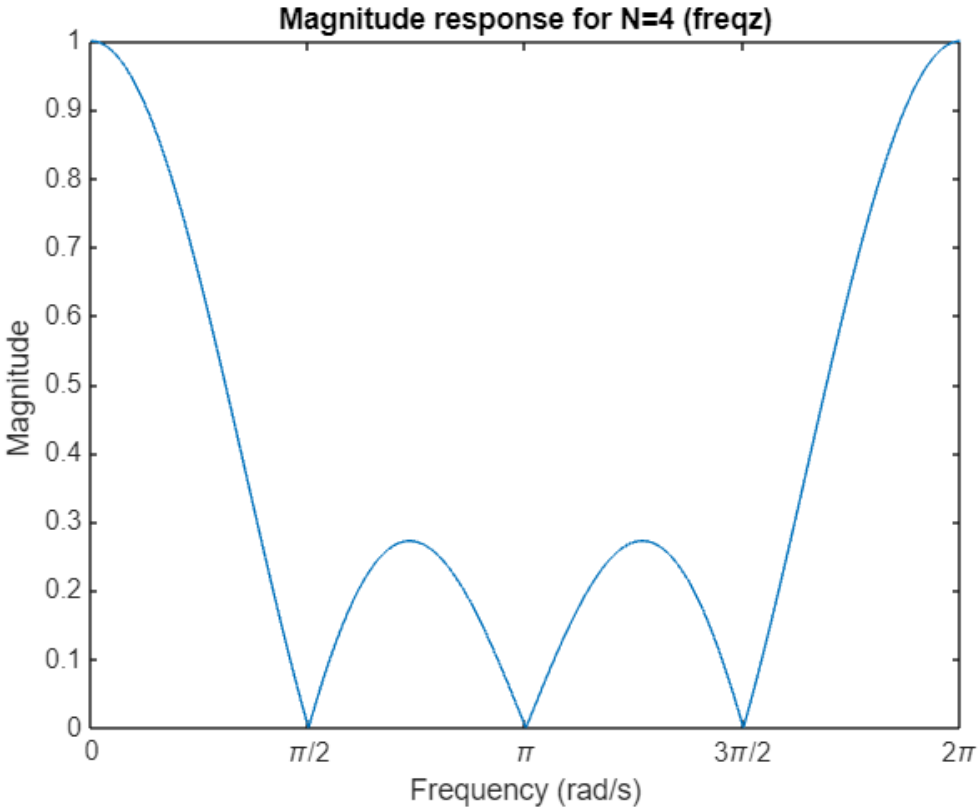
```
xticks([0 256*pi/512 512*pi/512 768*pi/512 1024*pi/512]);
xticklabels({'0','\pi/2','\pi','3\pi/2','2\pi'});
ylabel('Magnitude');xlabel('Frequency (rad/s)');
legend('H1: N=4','H2: N=7','Location','north');
title('Magnitude response of blur filters (analytical)')
```



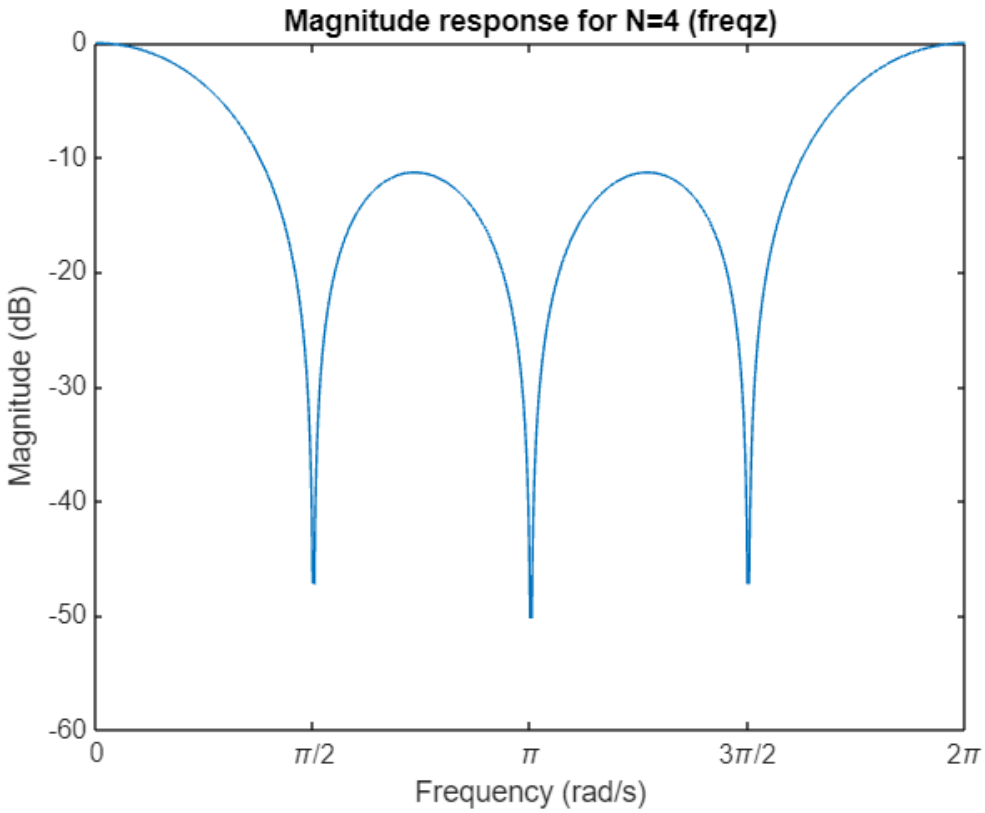
Part 2. Use the `freqz` command to evaluate the frequency response $H(e^{j\omega})$ of each of these filters at 1024 points between 0 and 2π. You should use the '`whole`' option with `freqz` to do this.

Use `plot` and `abs` to generate appropriately labeled graphs of the magnitude of the frequency response for these systems. Include plots where the magnitude is shown on a linear and on a decibel (dB) scale, with magnitude `x converted to 20*log10(abs(x))dB.`

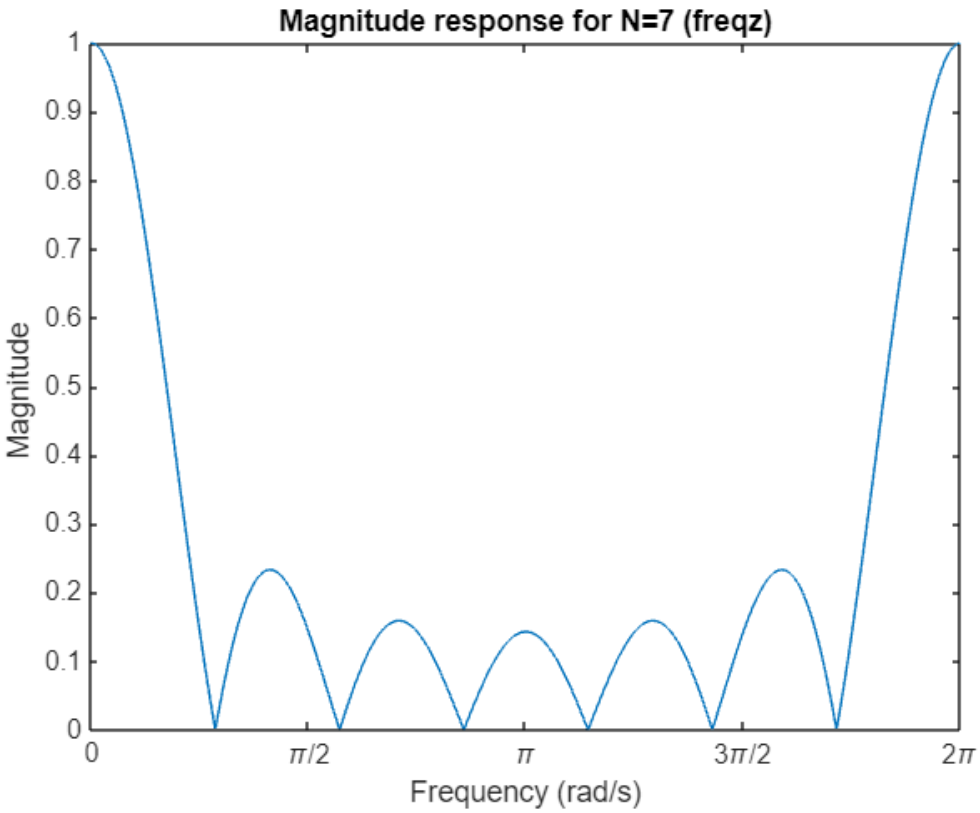
```
figure
%N=4
a4 = 1;
b4 = 1/4*ones(1,4);
f4 = freqz(b4,a4,1024,'whole');
% linear scale
plot(0:2*pi/1023:2*pi,abs(f4)); xlim([0 2*pi]);
title('Magnitude response for N=4 (freqz)');
xticks([0 256*pi/512 512*pi/512 768*pi/512 1024*pi/512]);
xticklabels({'0','\pi/2','\pi','3\pi/2','2\pi'});
ylabel('Magnitude ');xlabel('Frequency (rad/s)');a3 = 1;
```



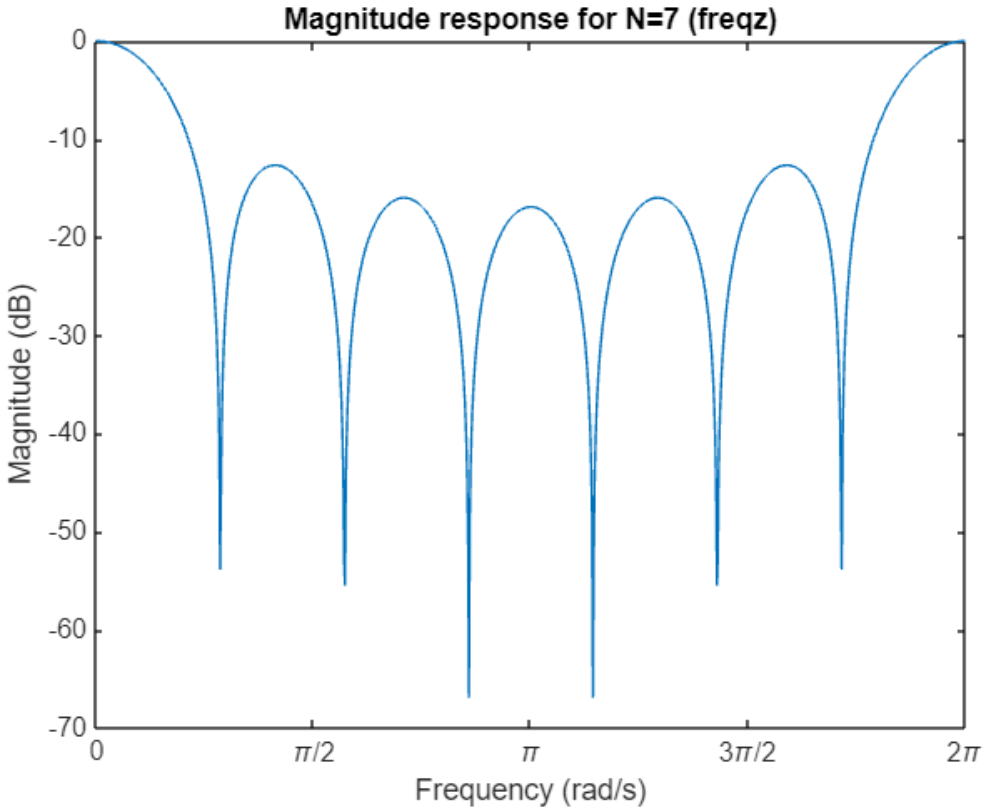

```
% b4 = 1/4*ones(1,4);
% f4 = freqz(b4,a4,1024,'whole');
% log scale
plot(0:2*pi/1023:2*pi,20*log10(abs(f4))); xlim([0 2*pi]);
title('Magnitude response for N=4 (freqz)');
xticks([0 256*pi/512 512*pi/512 768*pi/512 1024*pi/512]);
xticklabels({'0','\pi/2','\pi','3\pi/2','2\pi'});
ylabel('Magnitude (dB)');xlabel('Frequency (rad/s)');
```




```
% N=7
a7 = 1;
b7 = 1/7*ones(1,7);
f7 = freqz(b7,a7,1024,'whole');
% linear scale
plot(0:2*pi/1023:2*pi,abs(f7)); xlim([0 2*pi]);
title('Magnitude response for N=7 (freqz)');
xticks([0 256*pi/512 512*pi/512 768*pi/512 1024*pi/512]);
xticklabels({'0','\pi/2','\pi','3\pi/2','2\pi'});
ylabel('Magnitude ');xlabel('Frequency (rad/s)');
```




```
% log scale
plot(0:2*pi/1023:2*pi,20*log10(abs(f7))); xlim([0 2*pi]);
title('Magnitude response for N=7 (freqz)');
xticks([0 256*pi/512 512*pi/512 768*pi/512 1024*pi/512]);
xticklabels({'0','\pi/2','\pi','3\pi/2','2\pi'});
ylabel('Magnitude (dB)');xlabel('Frequency (rad/s)');
```



Part 3: Looking at your results of Parts 1 and 2, would you say that blur filters have low-pass or high-pass filter characteristics? How does the filter behavior depend on the filter length$N$?

The blur filters have low-pass characteristics. The filters have $N-1$ zeroes in the frequency response. For larger N, the range of frequencies attenuated is larger.

Problem 2: First-order recursive DT filters

This exercise is taken from Section 3.8, "First-Order Recursive Discrete-Time Filters" in *Computer Explorations in SIGNALS AND SYSTEMS [2nd edition]* by Buck, Daniel, Singer. In this exercise, you will explore the effect of first-order *recursive* discrete-time filters on discrete-time periodic signals. You will need to use the `fft` and `ifft` commands (Tutorial 3.1), as well as the `filter` command (Tutorial 2.2).

The exercise focuses on two causal LTI systems described by first-order recursive difference equations:

System 1: $y[n]+0.9y[n-1]=x[n]$

System 2: $y[n]-0.9y[n-1]=x[n]$

The input signal $x[n]$ you will apply to both systems will be the periodic signal with period $N=20$ described by the DTFS coefficients

$$a_k = \begin{cases} -0.7, & k = \pm 1, \\ 0.4, & k = \pm 9, \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

Task 1. Analytically determined the frequency response $H(e^{j\omega})$ of each of these systems. Plot the magnitudes $|H(e^{j\omega})|$.

$$y[n] - a\ y[n-1] = x[n]$$

The eigenfunction property of LTI systems and signals $x[n] = e^{j\omega n}$ gives:

$$H(e^{j\omega})e^{j\omega n} - a\ H(e^{j\omega})e^{j\omega(n-1)} = e^{j\omega n}$$
$$H(e^{j\omega})e^{j\omega n}(1 - a\ e^{-j\omega}) = e^{j\omega n}$$

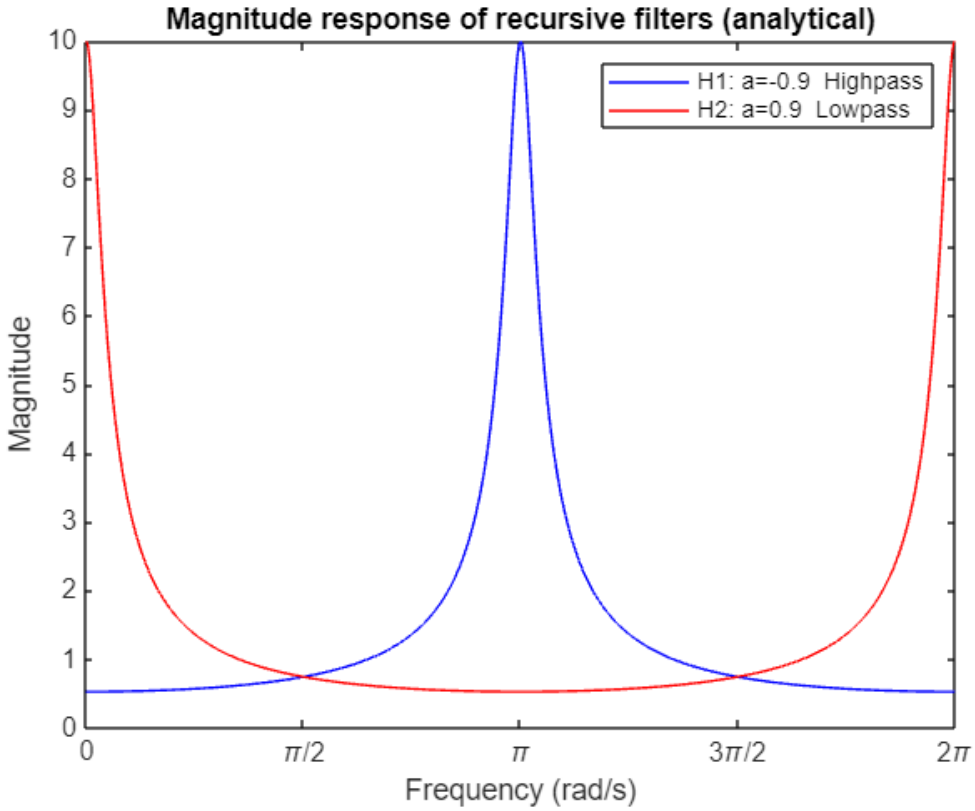
Therefore

$$H(e^{j\omega}) = \frac{1}{1 - a\ e^{-j\omega}}.$$

So,

$$H_1(e^{j\omega}) = \frac{1}{1 + 0.9\ e^{-j\omega}}$$
$$H_2(e^{j\omega}) = \frac{1}{1 - 0.9\ e^{-j\omega}}$$

```
figure
w=linspace(0,2*pi,1024);
H1=1./(1+0.9*exp(-j*w));xlim([0 2*pi]);
H2=1./(1-0.9*exp(-j*w));xlim([0 2*pi]);
hold on
plot(w,abs(H1),'b');
plot(w,abs(H2),'r');
box on;
xticks([0 256*pi/512 512*pi/512 768*pi/512 1024*pi/512]);
xticklabels({'0','\pi/2','\pi','3\pi/2','2\pi'});
ylabel('Magnitude');xlabel('Frequency (rad/s)');
legend('H1: a=-0.9  Highpass','H2: a=0.9  Lowpass','Location','northeast')
title('Magnitude response of recursive filters (analytical)')
```



Task 2. Define vectors `a1` and `b1` for the difference equation describing System 1 in the format specified by `filter` and `freqz`. Similarly, define `a2` and `b2` to describe System 2.

```
 %%%  Define a1, b1, a2, b2
a1 = [1 0.9];
b1 = 1;
a2 = [1 -0.9];
b2 = 1;
```

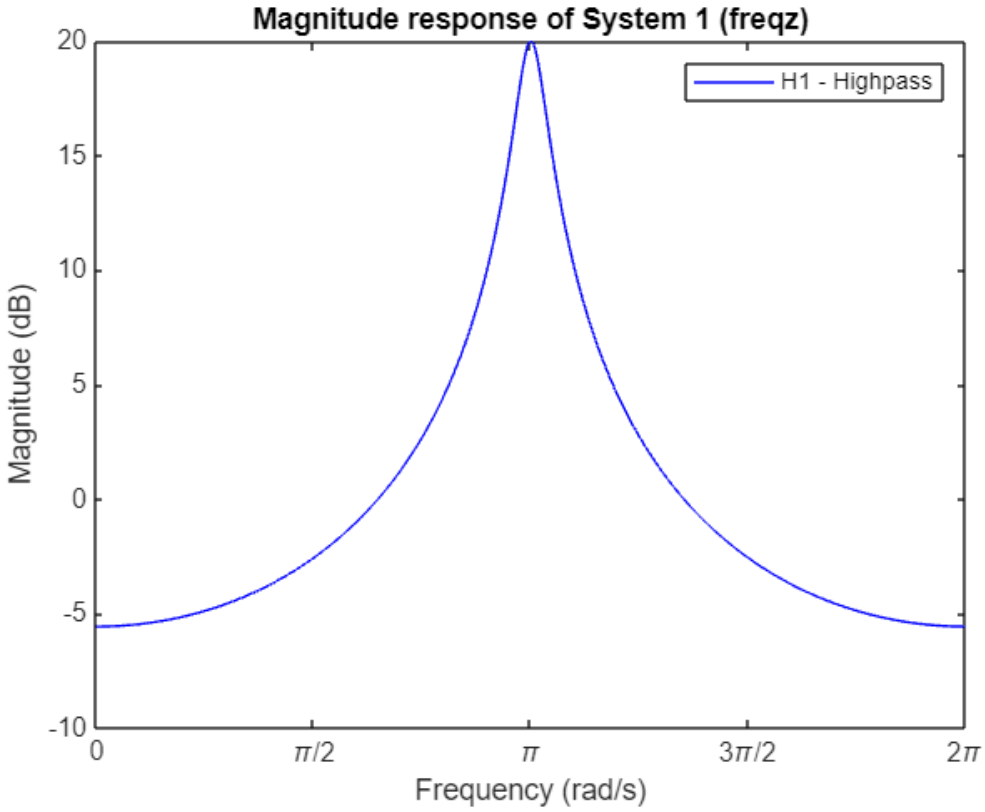
Task 3. Use `freqz` to evaluat the frequency response of Systems 1 and 2 at 1024 points between 0 and 2π. Note that you will again have to use the '`whole`' option with freqz to do this. Use `plot` and `abs` to generate appropriately labeled graphs of the magnitude of the frequency response for these systems (dB scale only). Based on the frequency response plots, specify whether each system in a highpass, lowpass, or bandpass filter.

System 1 is highpass.

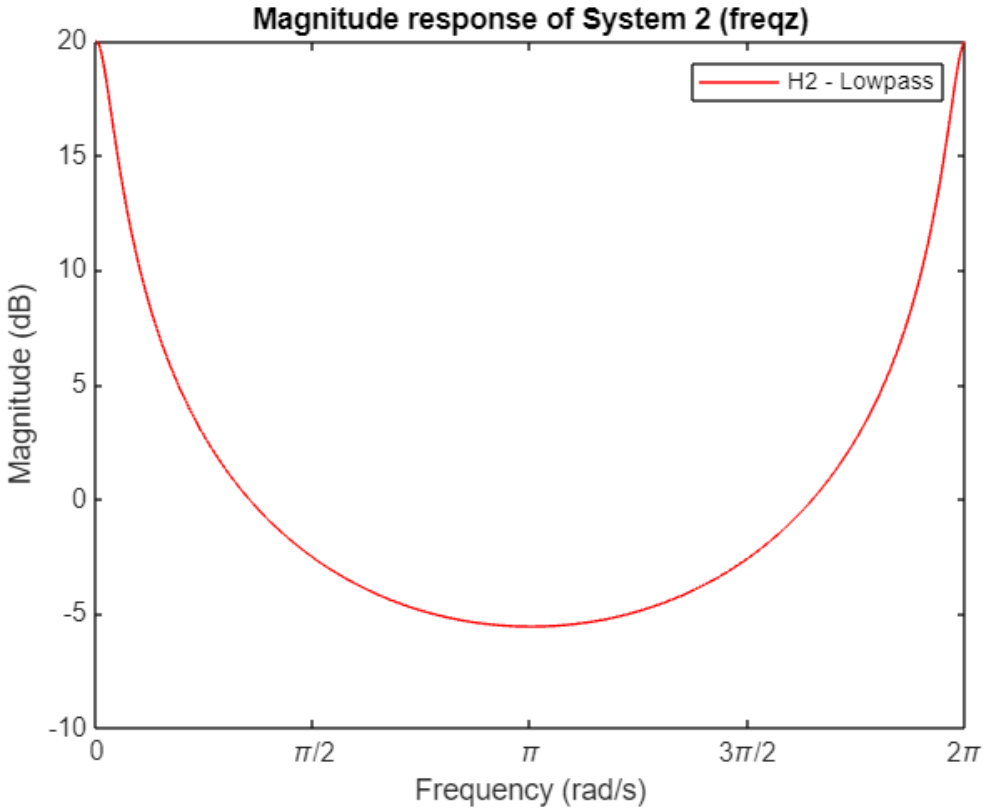
System 2 is lowpass.

```
%%%  Plot frequency response of system 1
figure
f1 = freqz(b1,a1,1024,'whole');
```

```
plot(0:2*pi/1023:2*pi,20*log10(abs(f1)),'b'); xlim([0 2*pi]);
title('Magnitude response of System 1 (freqz)');
xticks([0 256*pi/512 512*pi/512 768*pi/512 1024*pi/512]);
xticklabels({'0','\pi/2','\pi','3\pi/2','2\pi'});
ylabel('Magnitude (dB)');xlabel('Frequency (rad/s)');
legend('H1 - Highpass')
```




```
%%%  Plot frequency response of system 2
figure
f2 = freqz(b2,a2,1024,'whole');
plot(0:2*pi/1023:2*pi,20*log10(abs(f2)),'r'); xlim([0 2*pi]);
title('Magnitude response of System 2 (freqz)');
xticks([0 256*pi/512 512*pi/512 768*pi/512 1024*pi/512]);
xticklabels({'0','\pi/2','\pi','3\pi/2','2\pi'});
ylabel('Magnitude (dB)');xlabel('Frequency (rad/s)');
legend('H2 - Lowpass')
```

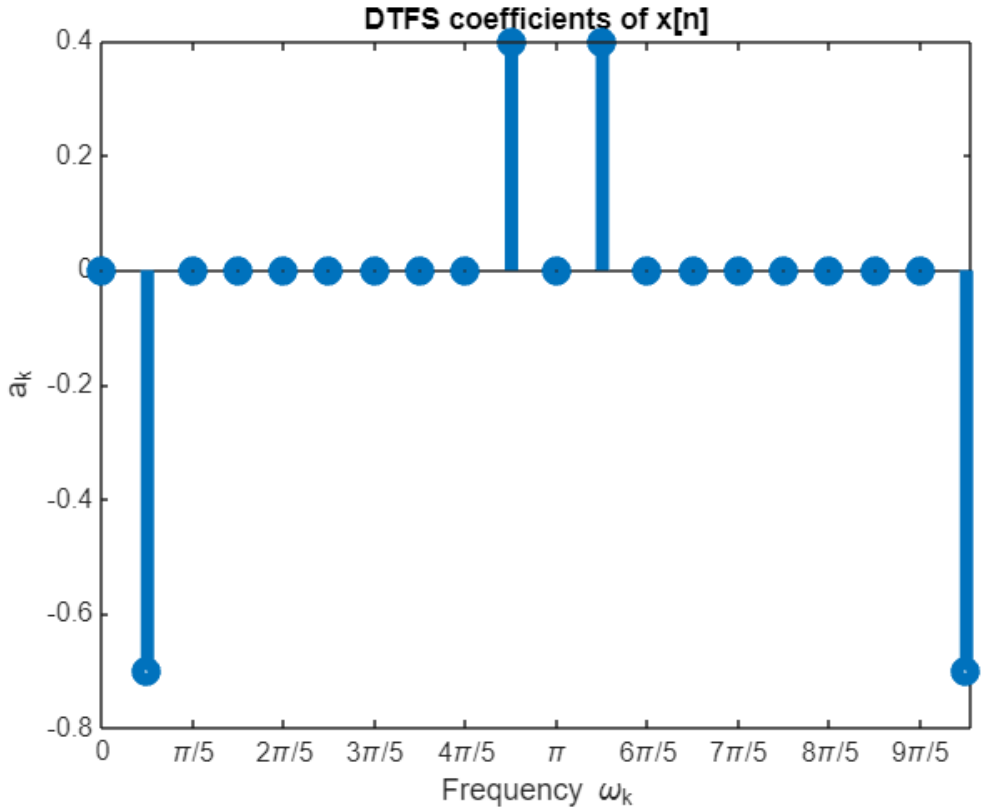


Task 4. Use Eq. (1) to define the vector `a_x to be the DTFS coefficients of` $x[n]$ for $0 \leq k \leq 19$. Generate a plot of the DTFS coefficients using `stem where the` x-axis is labeled with frequency $\omega_k = (2\pi/20)k$. Compare the plot with the frequency responses you generated in Task 1 and Task 3, and for each system state which frequency components willl be amplified and which will be attenuated when $x[n]$ is the input to the system.

System 1 will **amplify** high frequency components ($\pm\frac{2\pi \cdot 9}{20}$) and *attenuate* low frequency components ($\pm\frac{2\pi}{20}$).

System 2 will **amplify** low frequency components ($\pm\frac{2\pi}{20}$) and *attenuate* high frequency components ($\pm\frac{2\pi \cdot 9}{20}$).

```
%%% Plot DTFS coefficients
N = 20;
k = 0:N-1;
a_x = [0 -0.7 0 0 0 0 0 0 0 0.4 0 0.4 0 0 0 0 0 0 0 -0.7 ];
w_x = (2*pi/N)*k;
figure; stem(w_x,a_x,'LineWidth',5)
xlabel('Frequency \omega_k');
ylabel('a_k');
title('DTFS coefficients of x[n]');
xticks([0:2*2*pi/20:2*pi*19/20])
xticklabels({'0','\pi/5','2\pi/5','3\pi/5','4\pi/5','\pi','6\pi/5','7\pi/5','8\pi/
5','9\pi/5'})
```



Task 5. Use the results of Task 1 to analytically determine the coefficients $b_{1,k}$ and $b_{2,k}$ of the output signals produced by System 1 and System 2.

Use `stem` and `abs` to generate plots of the **magnitudes** of the DTFS coefficients for both sequences. Do your results agree with your conclusions in Task 4?

$$b_k = H(e^{jk\omega_0})a_k, \text{ where } \omega_0 = \frac{2\pi}{20}$$

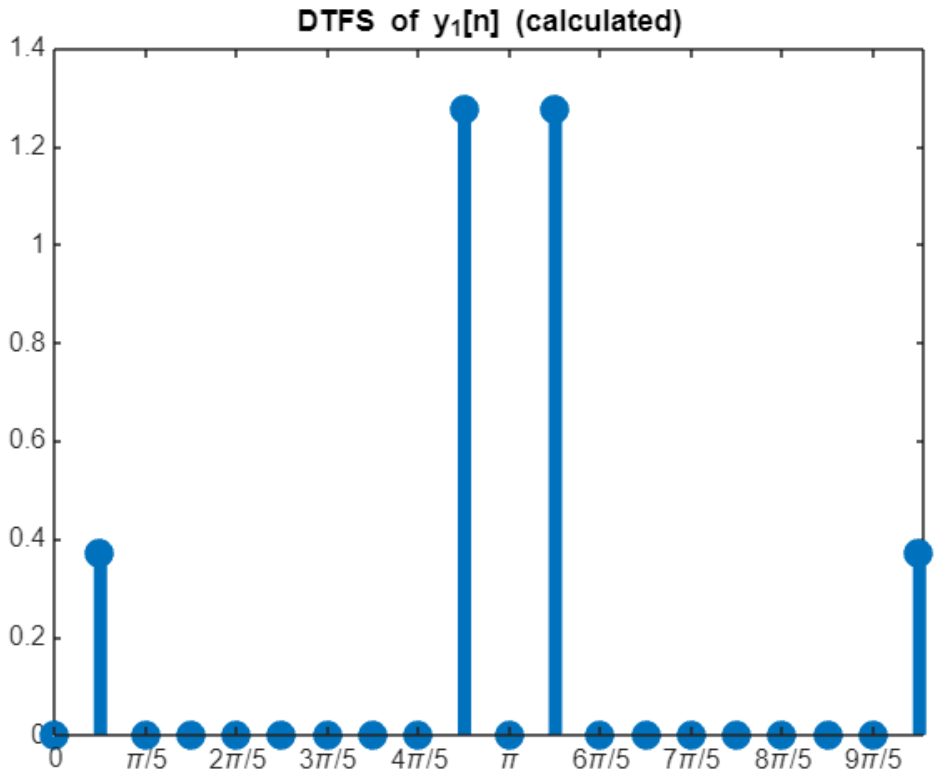
Output $y_1[n]$ from System1 has more low frequency content.

Output $y_2[n]$ from System 2 has more high frequency content.

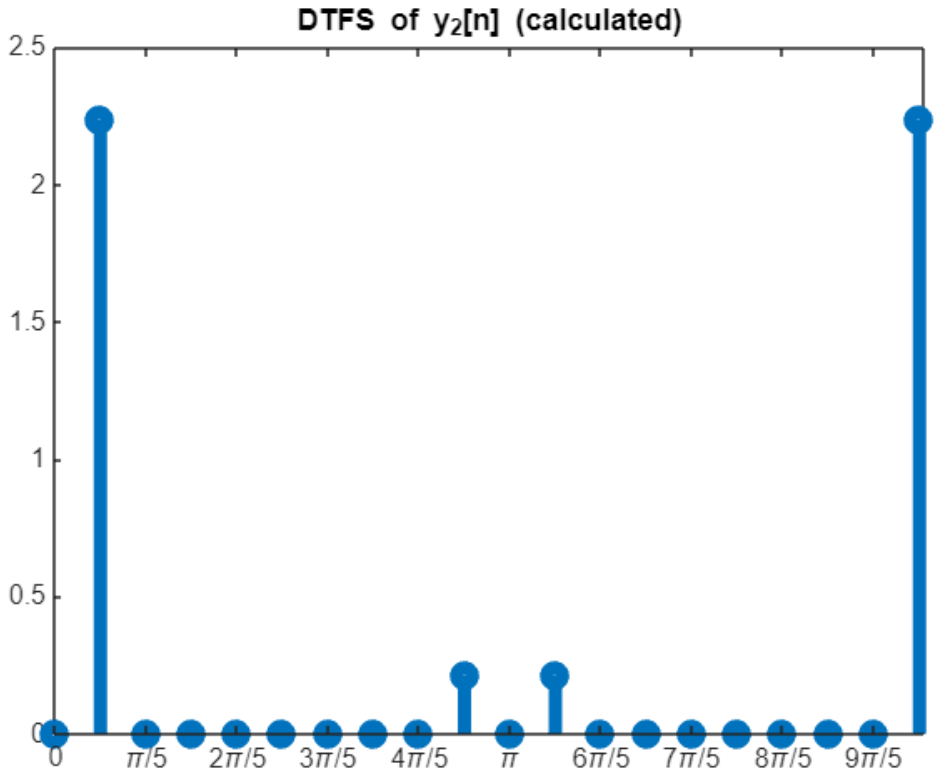
This agrees with conclusions in Task 4.

```
figure
k=[0:19];
wk=((2*pi)/20)*k;
a_x = [0 -0.7 0 0 0 0 0 0 0 0.4 0 0.4 0 0 0 0 0 0 0 -0.7 ];
H1_k=1./(1+0.9*exp(-j*wk));xlim([0 2*pi]);
b_y1 = a_x.*H1_k;
%
stem(w_x,abs(b_y1),'LineWidth',5)
xticks([0:2*2*pi/20:2*pi*19/20])
xticklabels({'0','\pi/5','2\pi/5','3\pi/5','4\pi/5','\pi','6\pi/5','7\pi/5','8\pi/
5','9\pi/5'})
```

```
title('DTFS of y_1[n] (calculated)');
```

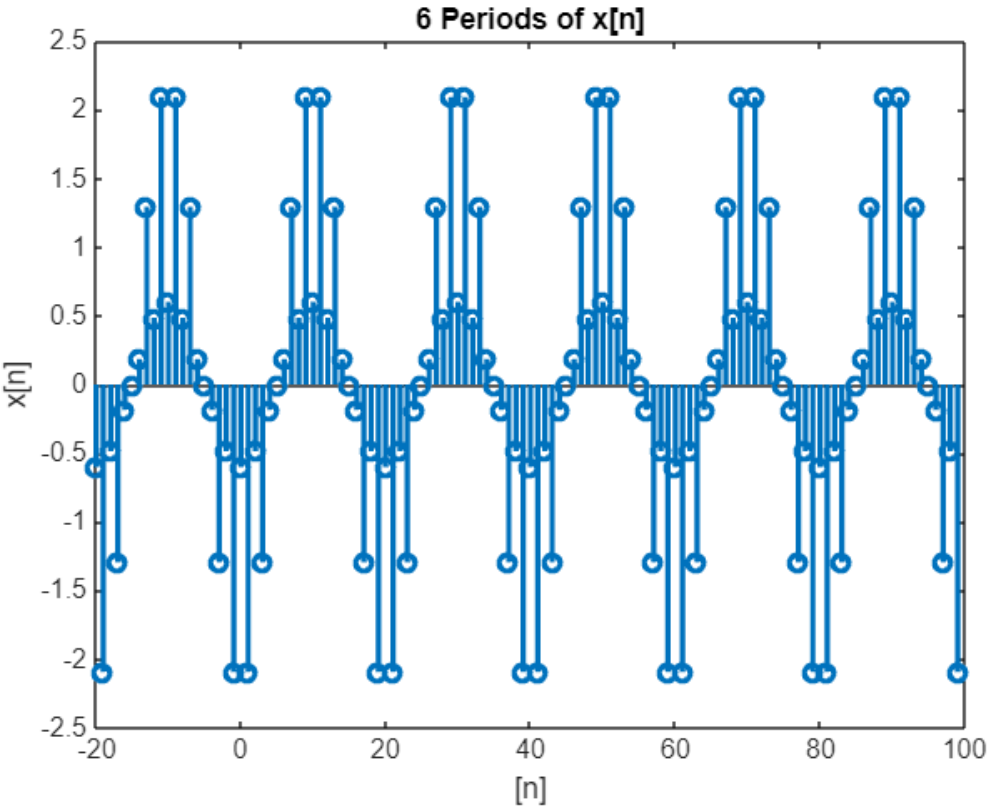



```
%
figure
H2_k=1./(1-0.9*exp(-j*wk));xlim([0 2*pi]);
b_y2 = a_x.*H2_k;
stem(w_x,abs(b_y2),'LineWidth',5)
xticks([0:2*2*pi/20:2*pi*19/20])
xticklabels({'0','\pi/5','2\pi/5','3\pi/5','4\pi/5','\pi','6\pi/5','7\pi/5','8\pi/
5','9\pi/5'})
title('DTFS of y_2[n] (calculated)');
```



Task 6. Define `x_20` to be one period of $x[n]$ for $0 \leq n \leq 19$ using `ifft` with `a_x` as specified in Tutorial 3.1. Use `x_20` to create `x`, consisting of 6 periods of $x[n]$ for $-20 \leq n \leq 99$. Also define `n` to be this range of sample indices, and generate a plot of $x[n]$ on this interval using `stem`.

```
%%% Inverse FFT
x_20 = N*ifft(a_x);
n = -20:99;
x = repmat(x_20,1,6);
figure; stem(n,x,'LineWidth',2);
xlabel('[n]');
ylabel('x[n]');
title('6 Periods of x[n]');
```

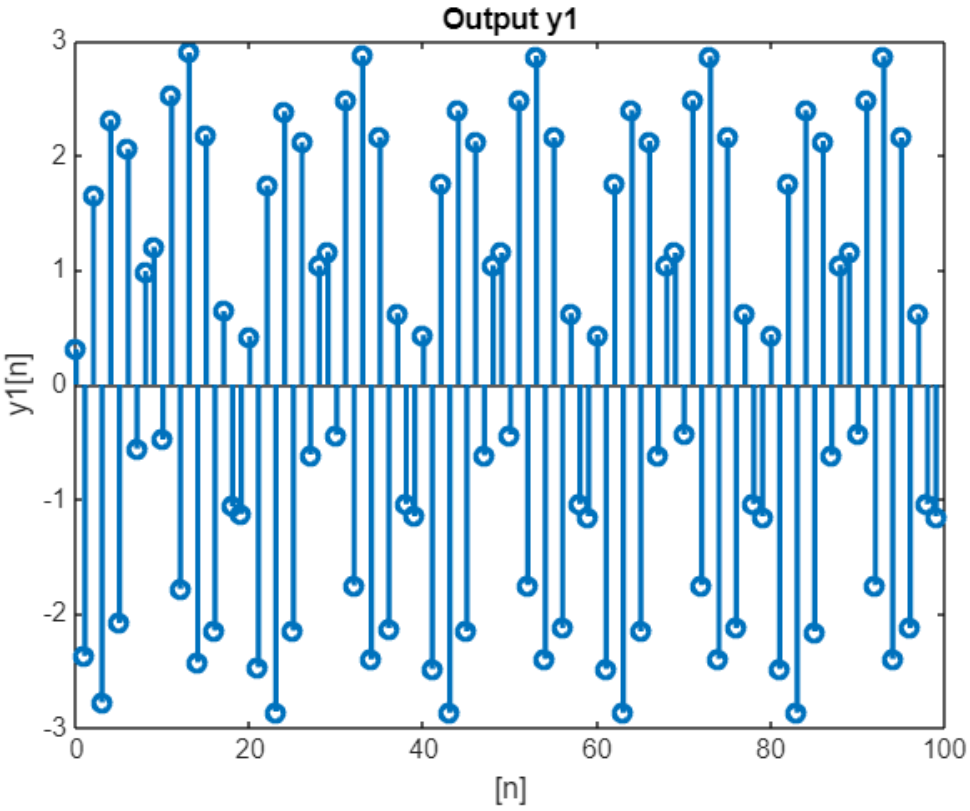


Task 7. Use `filter` to compute `y1` and `y2`, the outputs of Systems 1 and 2 when $x[n]$ is the input. Plot both outputs for $0 \leq n \leq 99$ using `stem`. Based on these plots, state which output contains more high frequency energy and which has more low frequency energy. Do the plots confirm your answers in Task 4 and Task 5?

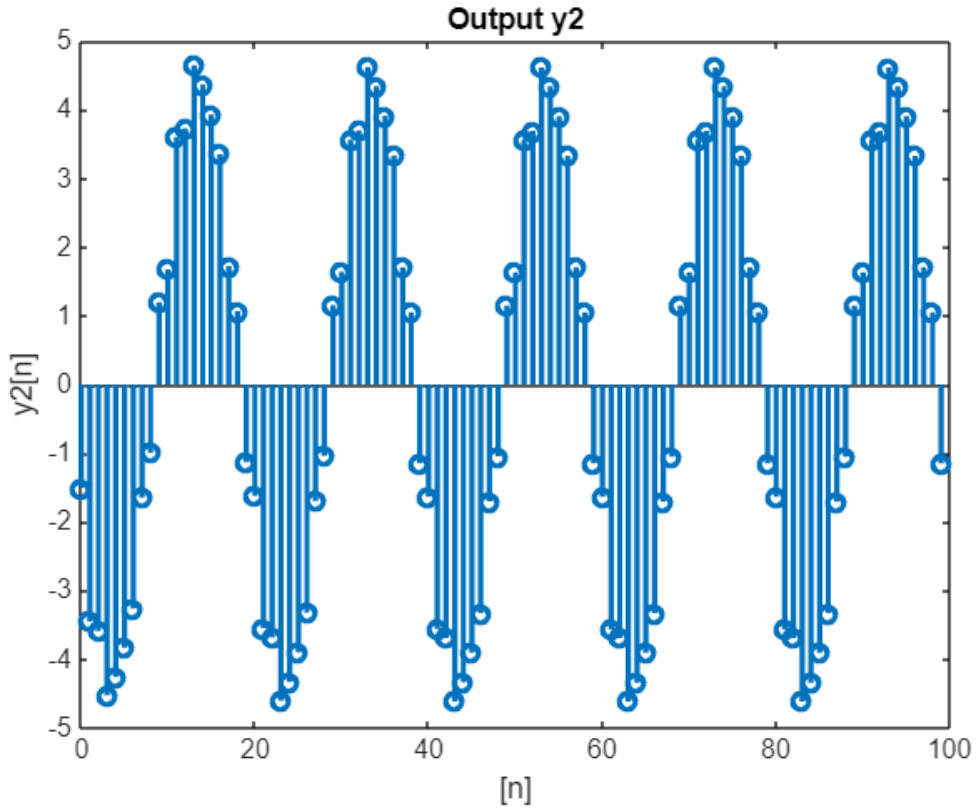
Output y1 contains more high-frequency energy, as expected from Task 3.

Output y2 contains more low-frequency energy, as expected from Task 3.

```
y1 = filter(b1,a1,x);
stem(n(21:end),y1(21:end),'LineWidth',2);
xlabel('[n]');
ylabel('y1[n]');
title('Output y1');
```




```
y2 = filter(b2,a2,x);
stem(n(21:end),y2(21:end),'LineWidth',2);
xlabel('[n]');
ylabel('y2[n]');
title('Output y2');
```



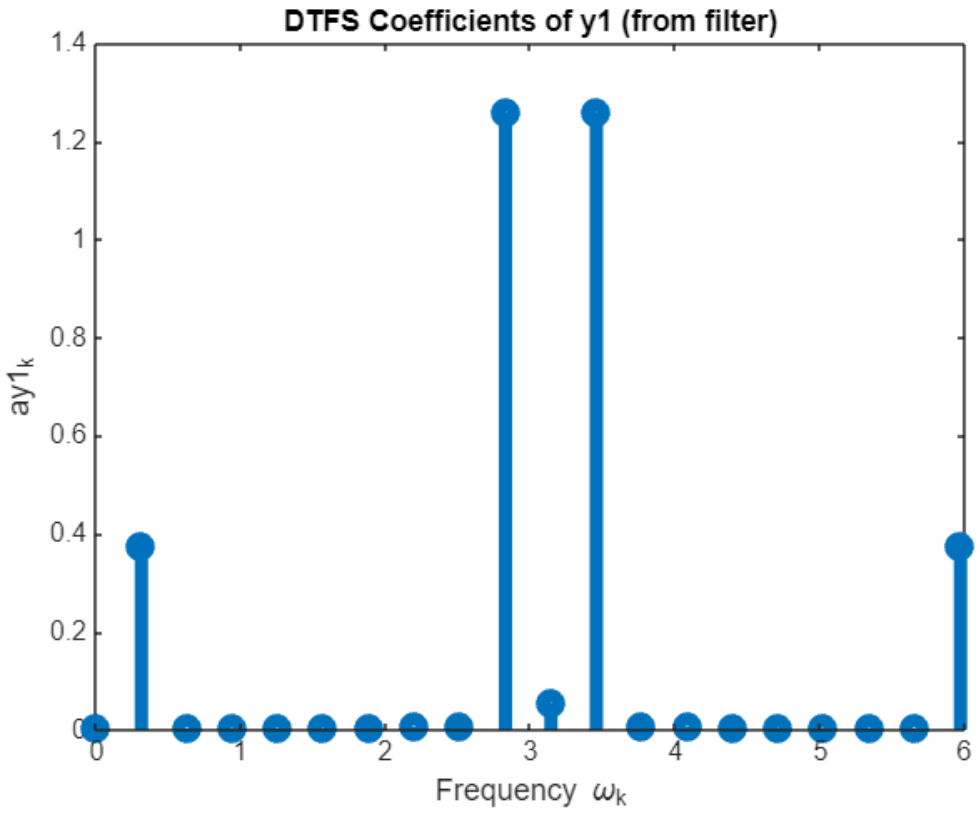
Task 8. Define `y1_20` and `y2_20` to be the segments of `y1` and `y2` corresponding to $y_1[n]$ and $y_2[n]$ for $0 \leq n \leq 19$.Use these vectors and `fft` to compute `a_y1` and `a_y2`, the DTFS coefficients of `y1` and `y2`. Use `stem` and `abs` to generate plots of the magnitudes of the DTFS coefficeints for both sequences.

Are these plots consistent with your answers in Task 7?

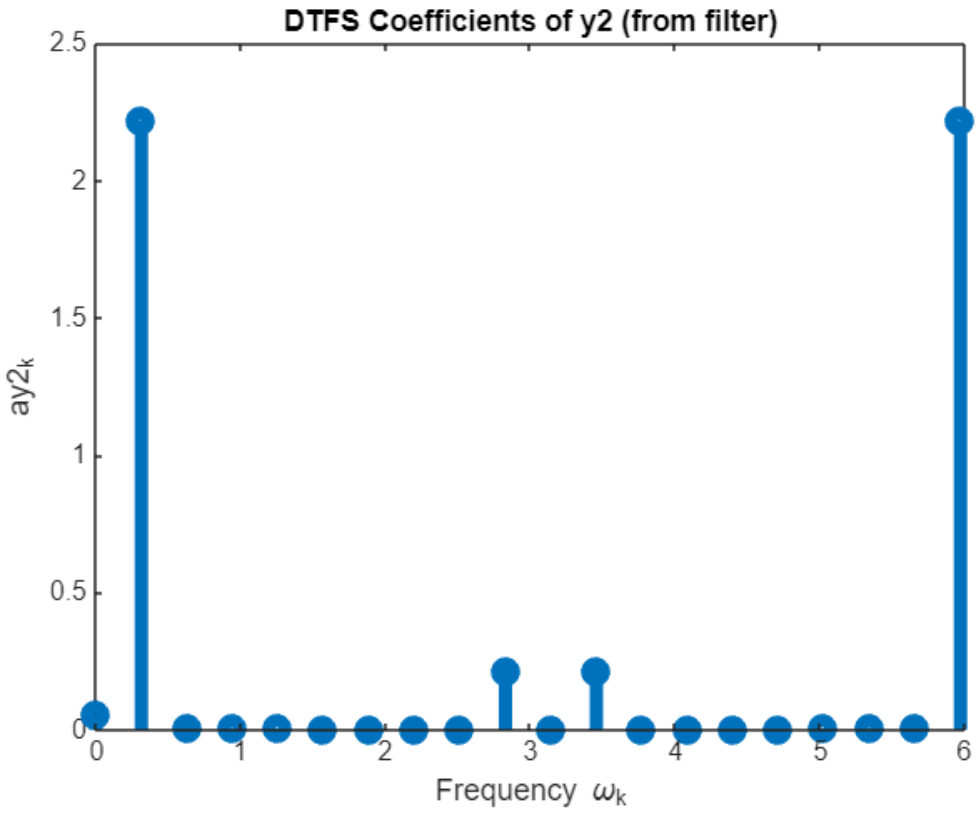
Do they differ from your analytical results in Part 5? If so, try to explain why.

Yes, the plots are consistent. They look almost identical. Output y1 has larger high-frequency Fourier series coefficients. Output y2 has larger low-frequency Fourier series coefficients.

```
%%% Output DTFS for y1
y1_20 = y1(21:40);
ay1 = (1/N)*fft(y1_20);
stem(w_x,abs(ay1),'LineWidth',5);
xlabel('Frequency \omega_k');
ylabel('ay1_k');
title('DTFS Coefficients of y1 (from filter)');
```




```
%%% Output DTFS for y2
y2_20 = y2(21:40);
ay2 = (1/N)*fft(y2_20);
stem(w_x,abs(ay2),'LineWidth',5);
xlabel('Frequency \omega_k');
ylabel('ay2_k');
title('DTFS Coefficients of y2 (from filter)');
```



Functions

```
0;
```

```
function [output] = blur(image,N)
L = size(image,2);
H = toeplitz([ones(1,N) zeros(1,L-N)], [1 zeros(1,L-1)]);
H = (1/N)* H;
output = (H*(image'))';
end

function [output] = deblur(image,N)
L = size(image,2);
H = toeplitz([ones(1,N) zeros(1,L-N)], [1 zeros(1,L-1)]);
H = (1/N)* H;
output = image * inv(H');
end
```