ECE 340 Lab 1

Omar Mahmoud
1753607
Section D21

9/25/2024

1 Signal Generation and Plotting

Consider the following discrete functions:

$$x_1[k] = -5.1\sin(0.1\pi k - 3\pi/4) \quad -10 \leq k \leq 10 \tag{1}$$

$$x_2[k] = -(0.9)^k e^{(j\pi k/10)} \quad 0 \leq k \leq 100 \tag{2}$$

To start, $x_1[k]$ can be plotted on MATLAB through the following script:

```
subplot(2, 2, [1 2]);
%% Define x_1
k = -10:40;
x1 = -5.1*sin(0.1*pi.*k - 3*pi/4) + 1.1*cos(0.4*pi.*k);
%% Plot x_1
stem(k, x1, 'LineWidth', 1, 'Color', 'b');
title("Plot of x1[k].");
xlabel("k");
ylabel("x1[k]");
```

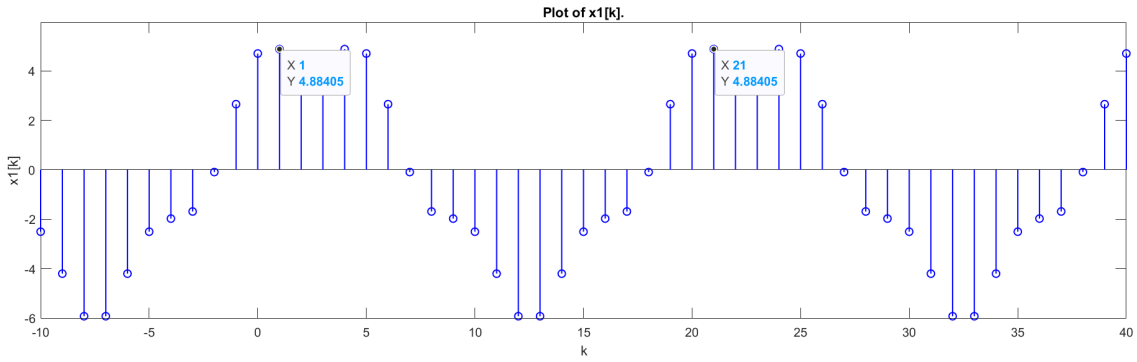

Figure 1: Plot of $x_1[k]$ with two points labeled to find the period.

By examining the graph, it is clear that $x_1[k]$ is a periodic sequence. To find the period, subtract the X of the two selected points:

$$T_{x_1} = X_2 - X_1 = 21 - 1 = 20$$

When plotting x_2 an imaginary and the real component will be produced. These can be plotted seperately through the following script:

```
%% Define x_2
k = 0:100;
x2 = (-0.9).^k .* exp(1).^(1j*pi.*k./10);
%% Plot the real component of the x_2
subplot(2, 2, 3);
stem(k, real(x2),'LineWidth', 1, 'Color', 'r');
title("Real Plot of x2[k].");
xlabel("k");
ylabel("Re(x2[k])");
%% Plot the img component of x_2
subplot(2, 2, 4);
stem(k, imag(x2),'LineWidth', 1, 'Color', 'r');
title("Imaginary Plot of x2[k].");
xlabel("k");
ylabel("Im(x2[k])");
```

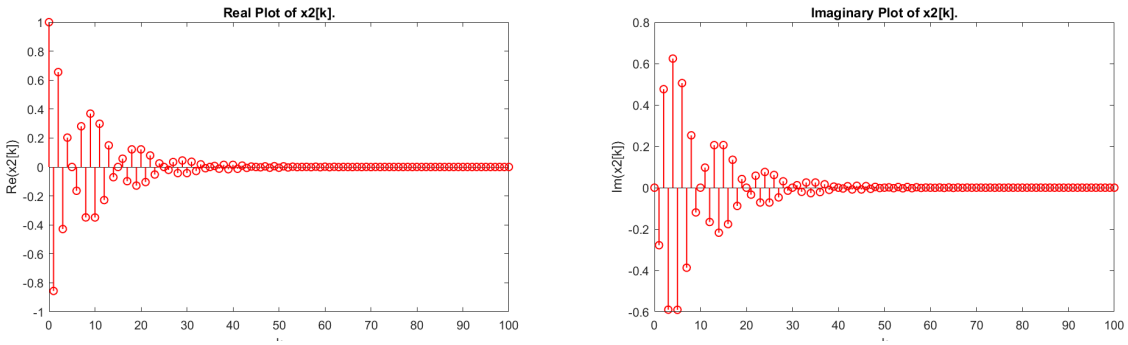

Figure 2: Plots of $Re(x_2[k])$ and $Im(x_2[k])$.

It is clear from the plots that $x_2[k]$ is not periodic, therefore it is not necessary to find a period.

To find the total energy of the discrete-time signals $x_1[k]$ and $x_2[k]$ the following equation is used:

$$E_x = \sum_{n=-N}^{N} |x[n]|^2 \tag{3}$$

where E_x represents the total energy of the signal over $[-N, N]$. The energy can be computed using the MATLAB script provided below:

```
%% Get energy of x_1
fprintf('E_x1 = %.4f\n', sum(abs(x1).^2));
%% Get energy of x_2
fprintf('E_x2 = %.4f\n', sum(abs(x2).^2));
```

The output of the script is as follows:

```
E_x1 = 698.5335
E_x2 = 5.2632
```

2 Digital Audio

MATLAB provides features that facilitate the reading and writing of audio signals from and to .wav files. The following script reads the audio file **baila.wav**, and stores its audio signal into the matrix, x_3 along with its sampling rate in F_s. In this context, the rows of x_3 represents the samples, while the columns represent the channels of the audio signal:

```
%% Read the audio file baila.wav
[x3, Fs] = audioread('baila.wav');
[num_samples, num_channels] = size(x3);
disp(['Number of samples: ', num2str(num_samples)]);
disp(['Number of channels: ', num2str(num_channels)]);
```

The output of the script is as follows:

```
Number of samples: 1000000
Number of channels: 1
```

From the output, it is evident **baila.wav** contains a mono-channel signal with 1,000,000 samples. To analyze this signal, the next step involves creating an apporpriate time vector and plotting the mono signal against it using the following script:

```
%% Create time vector
t = (0:num_samples-1) / Fs;
%% Plot the audio signal against it
figure(1);
plot(t, x3);
ylabel('Amplitude');
xlabel('Time (seconds)');
title('Audio Signal over Time');
```

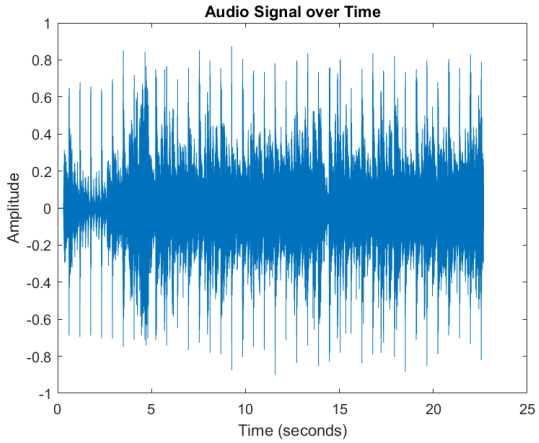

Figure 3: Plot of the mono audio signal x_3 extracted from **baila.wav**.

To assess the total energy of x_3 the same method as Section 1 is employed:

```
%% Get total energy
fprintf('E_x3 = %.4f\n', sum(abs(x3).^2));
```

The value generated by the script is as follows:

```
E_x3 = 20422.7731
```

To showcase MATLAB's audio writing features, the following script samples the first half of the matrix x_3 and creates a new matrix x_{3s}. This matrix is then used to play half of the **baila.wav** file using MATLAB's *sound* function and generates a new **baila_half.wav** file using the *audiowrite* function:

```
%% Get x3s, play it, and create baila_half.wav
x3s = x3(1:floor(num_samples/2), :);
sound(x3s, Fs);
audiowrite('baila_half.wav', x3s, Fs);
```

3 Digital Image

MATLAB provides features that facilitate the reading and writing from and to JPEG image files. The following script reads the JPEG file **vase.jpg**, and stores its value into the matrix *vase*:

```
%% read the jpg file and get metrics
vase = imread('vase.jpg');
[rows, cols] = size(vase);
max_pixel = max(vase(:));

disp(['Number of rows: ', num2str(rows)]);
disp(['Number of cols: ',  num2str(cols)]);
disp(['Max pixel value: ',  num2str(max_pixel)]);
```

The output of script returns the number of rows, columns, and the maximum pixel value of the image:

```
Number of rows: 512
Number of cols: 1083
Max pixel value: 255
```

MATLAB also facilitates the creation of new JPEG files. Below *imwrite* is utilized to create a new image **vase_bright.jpg**, by adding 30 to all the pixel values of the original image. This results in a visually brighter version of the original **vase.jpg**:

```
%% create the new bright jpg
vase_bright = vase + 30;
imwrite(vase_bright, 'vase_bright.jpg', 'jpg', 'Quality', 100);
```

Figure 4: Comparison of the original **vase.jpg** (right) and the brightened **vase_bright.jpg** (left).

4 System Response

The schematic of a discrete-time signal is given below:

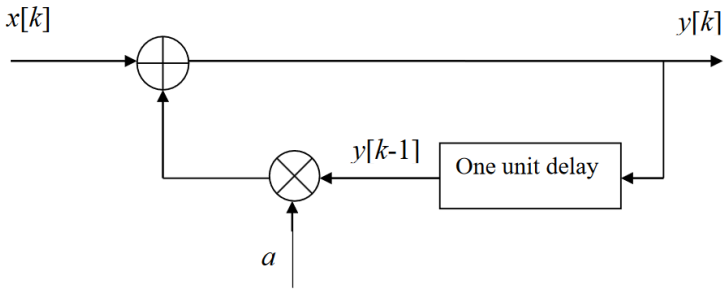

Figure 5: Schematic of the discrete-time signal for this section.

The relationship between $x[k]$ and $y[k]$ can be expressed as:

$$y[k] = x[k] + ay[k-1] \tag{4}$$

Assuming that x[k] is a unit-step function and $y[k] = 0$ for $k < 0$, the $y[k]$ of the system for $0 \leq k < 5$ for (i) $a = 0.5$ and (ii) $a = 5$ can be computed manually:

i) when $a = 0.5$:

$$\begin{gathered} y[0] = x[0] + 0.5y[-1] = (1) + 0.5(0) = 1 \\ y[1] = x[1] + 0.5y[0] = (1) + 0.5(1) = 1.5 \\ y[2] = x[2] + 0.5y[1] = (1) + 0.5(1.5) = 1.75 \\ y[3] = x[3] + 0.5y[2] = (1) + 0.5(1.75) = 1.875 \\ y[4] = x[4] + 0.5y[3] = (1) + 0.5(1.875) = 1.9375 \end{gathered}$$

ii) when $a = 5$

$$\begin{gathered} y[0] = x[0] + 5y[-1] = (1) + 5(0) = 1 \\ y[1] = x[1] + 5y[0] = (1) + 5(1) = 6 \\ y[2] = x[2] + 5y[1] = (1) + 5(6) = 31 \\ y[3] = x[3] + 5y[2] = (1) + 5(31) = 156 \\ y[4] = x[4] + 5y[3] = (1) + 5(156) = 781 \end{gathered}$$

To get y[k] for $0 \leq k < 50$, it is more time efficient to use MATLAB. Below is the function *sysresp*, which calculates and plots the apporpriate $y[k]$ for a given input $x[k]$ and system parameter a:

```
function y=sysresp(x, a)
%
% Computes the output in response to an arbitrary input x[n], n=0, ...N-1.
% Assume that the system has 0 initial conditions.
% Input:
%   x: the input signal,
%   a: the system parameter
% Output:
%   y: the output signal

N = length(x); % length of the vector
y = zeros(1, N);

for k = 1:N+1
  if k == 1
    y(k) = 1;
  else
    y(k) = 1 + a * y(k-1);
  end
end

figure(1)
stem(1:N+1, y,'LineWidth', 1, 'Color', 'b');
title(['Plot of y[k] when a = ', num2str(a), '.']);
xlabel('k');
ylabel('y[k]');
return
```

When we use this function with the system parameters $a = 0.5$ and $a = 5$, we get the following plots:

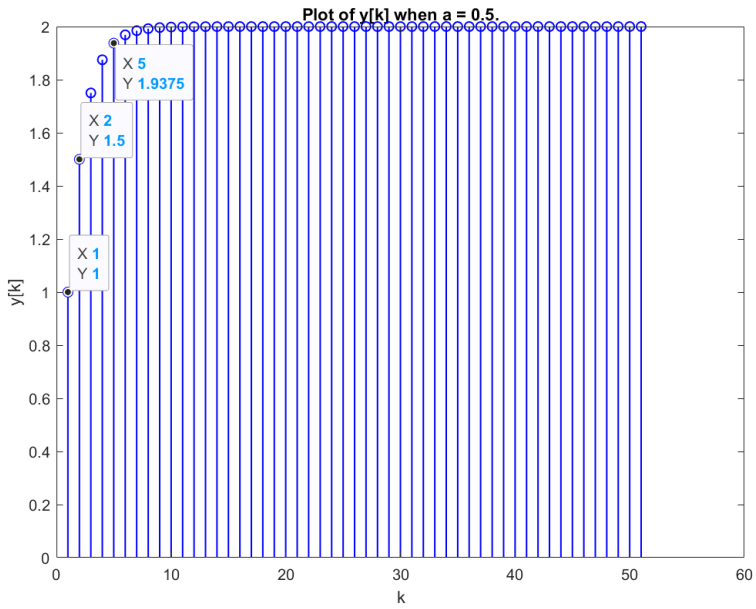

Figure 6: Plot of $y[k]$ when $a = 0.5$

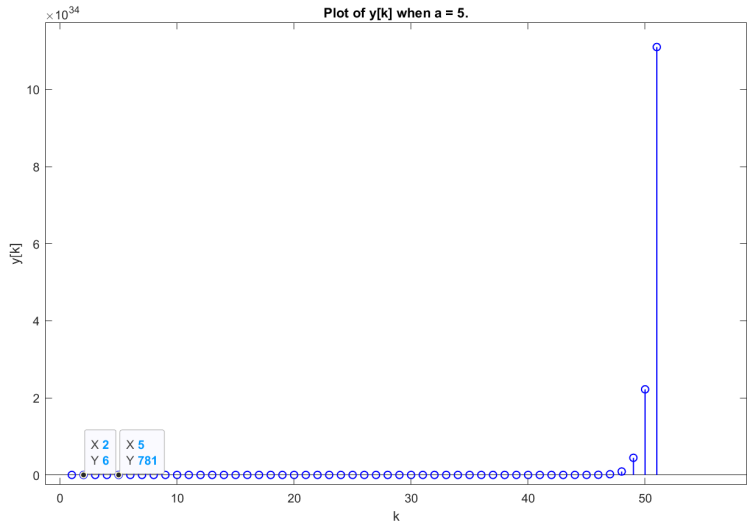

Figure 7: Plot of $y[k]$ when $a = 5$

Upon examining the graphs, it is evident that the results from *sysresp* match the initial hand calculations. It is also evident that $y[k]$ is bounded showing a BIBO stable signal for $a = 0.5$, but is not bounded $a = 5$ showing a BIBO unstable signal.

Another a value that would allow for a BIBO stable signal is 0.2. This can be confirmed using the *sysresp* function:

```
x = ones(1, 50);
sysresp(x, 0.2);
```

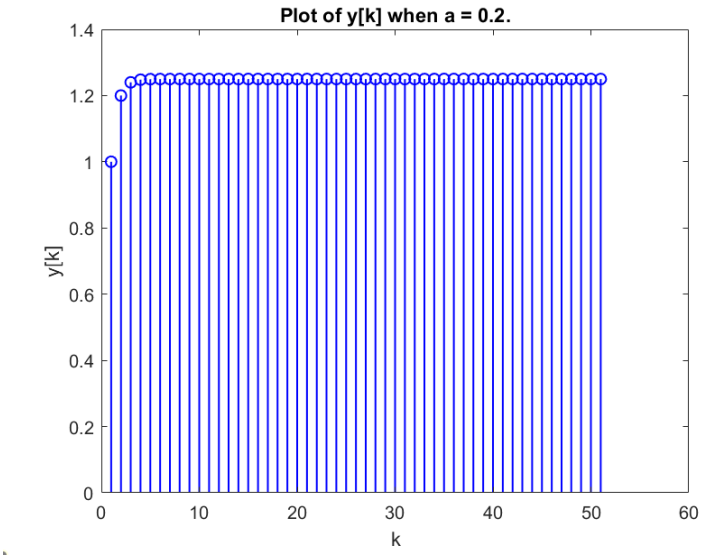

Figure 8: Plot of $y[k]$ when $a = 0.2$

Upon examining the graph, it is clear that $y[k]$ is bounded showing a BIBO stable signal for $a = 0.2$.