Discrete Time Signals and Systems

Elec 342

Lab Experiment #2

Andre Hei Wang Law

4017 5600

Section MK-X

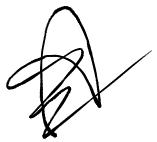
TA: Ahmed Salameh

TA Email: ahmed.salameh@mail.concordia.ca

Performed on February 6, 2023

Due on February 20, 2023

“I certify that this submission is my original work and meets the Faculty’s Expectations of

Originality.”

**1) Objectives**

This second experiment will be divided into two parts with their own goal. The objective of part 1 is to continue with introducing more MATLAB commands and features. Mainly, students will explore topics such as looping, conditional selections, array processing features and the various properties of signal and systems. For part 2, the goal is to practice using MATLAB to determine a system’s response to an input signal using convolution functions.

**2) Theory**

For Loop:

**for** (variable = index¬¬\_expression)  
 statement1;  
 statement2;  
 etc;  
 **end**

Conditional If Statement:

**if** (condition)  
 statement1;  
 statement2;  
 etc;  
 **end**

The *condition* statement respects the usage of operators. Below is a list of relational operators:

< less than

<= less than or equal

> greater than

>= greater than or equal

== equal

~= not equal

& Logical AND

| Logical OR

Conditional If-Else Statement:

**if** (condition)  
 statement1;  
 **else**  
 statement2;  
 **end**

Linearity Property of Signals and Systems:

In order to satisfy the property of a linear system, it needs to prove that the input signal x3[n]=x1[n]+x2[n] and the output response is y3[n]=y1[n]+y2[n] given that the input signal of x1[n] produces an output response of y1[n] and that the input signal of x2[n] produces an output response of y2[n].

Saving Plot Outputs:

It is recommended to save plots using the command *print*. The syntax of this command is

*Print -formatType filename*

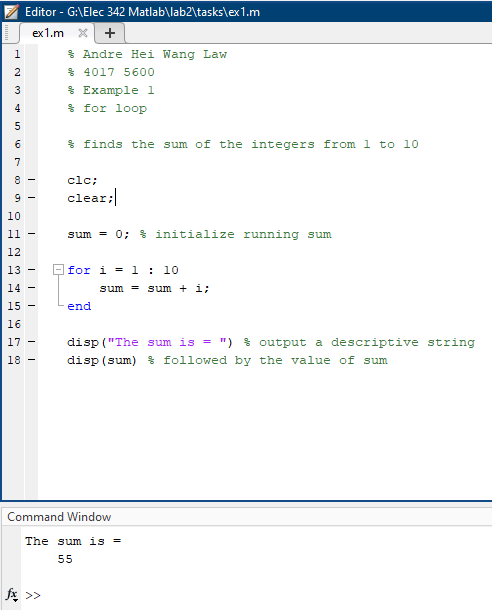
For example, the command *print -dpsc even\_odd\_components.ps* prints the plot as a PostScript file under the name of even\_odd\_components.

**3) Tasks, Results and Discussion**

Task 1: For Loop Summation

% Andre Hei Wang Law  
% 4017 5600  
% Example 1  
% for loop  
  
% finds the sum of the integers from 1 to 10  
  
clc;  
clear;  
  
sum = 0; % initialize running sum  
  
**for** i = 1 : 10  
 sum = sum + i;  
**end**  
  
disp("The sum is = ") % output a descriptive string  
disp(sum) % followed by the value of sum

Results 1:



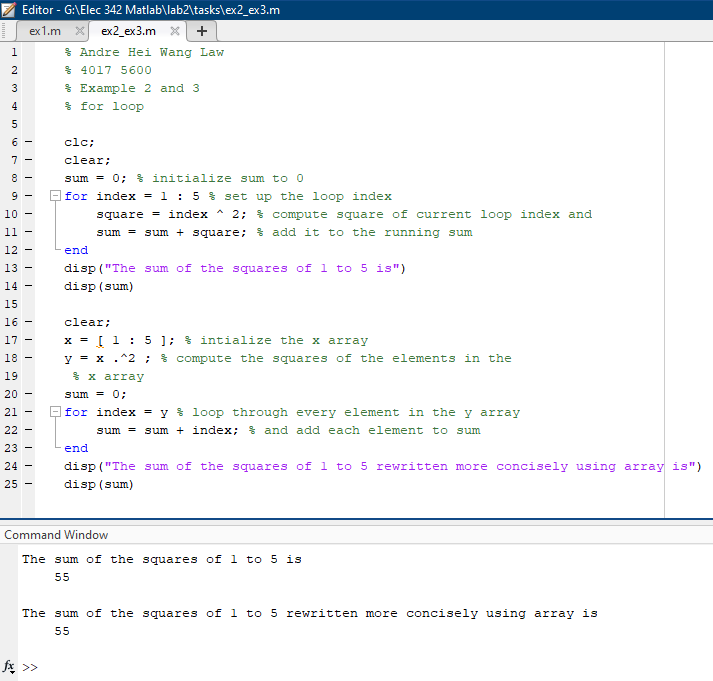
Discussion 1:

In MATLAB, a for-loop’s structure starts with the looping statement, then everything below it was be the lines being looped until the last line where “end” resides. In this case, the looping statement is “i” from 1 to 10. The looping segment is sum equal to itself increment by one. This loop can be proven by simply performing the addition manually. (1+2+3+4+5+6+7+8+9+10 = 55).

Task 2: For Loop Square

% Andre Hei Wang Law  
% 4017 5600  
% Example 2 and 3  
% for loop  
  
clc;  
clear;  
sum = 0; % initialize sum to 0  
**for** index = 1 : 5 % set up the loop index  
 square = index ^ 2; % compute square of current loop index and  
 sum = sum + square; % add it to the running sum  
**end**  
disp("The sum of the squares of 1 to 5 is")  
disp(sum)  
  
clear;  
x = [ 1 : 5 ]; % intialize the x array  
y = x .^2 ; % compute the squares of the elements in the  
 % x array  
sum = 0;  
**for** index = y % loop through every element in the y array  
 sum = sum + index; % and add each element to sum  
**end**  
disp("The sum of the squares of 1 to 5 rewritten more concisely using array is")  
disp(sum)

Results 2:



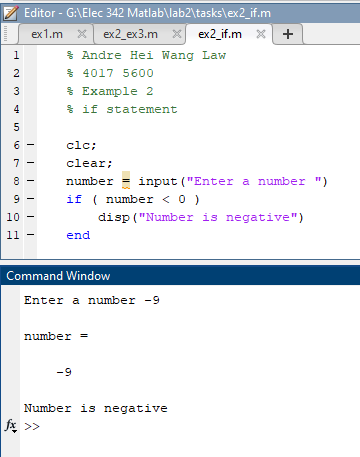
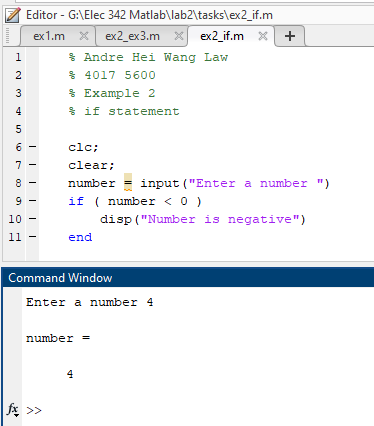
Discussion 2:

Task 2 is composed of two different codes with the same functionality. In both cases, it performs the sum of the squares. For the first example, it loops from index 1 to 5 and performs a square which is then added to the overall sum. However, for the second example, the loops go through every element of the array y which directly computes the square on the array x. All this is then added to the overall sum, similarly to the first example.

Task 3: If Statement Positive and Negative

% Andre Hei Wang Law  
% 4017 5600  
% Example 2  
% if statement  
  
clc;  
clear;  
number = input("Enter a number ")  
**if** ( number < 0 )  
 disp("Number is negative")  
**end**

Results 3:



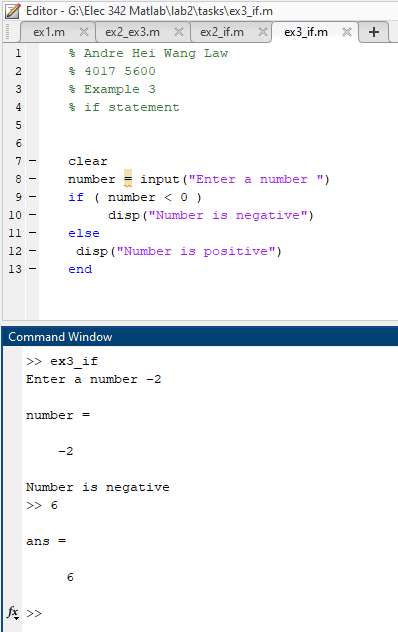
Discussion 3:

For this task, students are introduced to the “input” statement. In this case, the user input will be stored into the “number” variable which enters the if-statement. This simple test displays a message whenever the user input number is negative. On the left image, it can be seen that 4 doesn’t prompt the message. On the other hand, the input -9 does.

Task 4: If-Else Statement Positive and Negative

% Andre Hei Wang Law  
% 4017 5600  
% Example 3  
% if-else statement  
  
  
clear  
number = input("Enter a number ")  
**if** ( number < 0 )  
 disp("Number is negative")  
**else**  
 disp("Number is positive")  
**end**

Results 4:



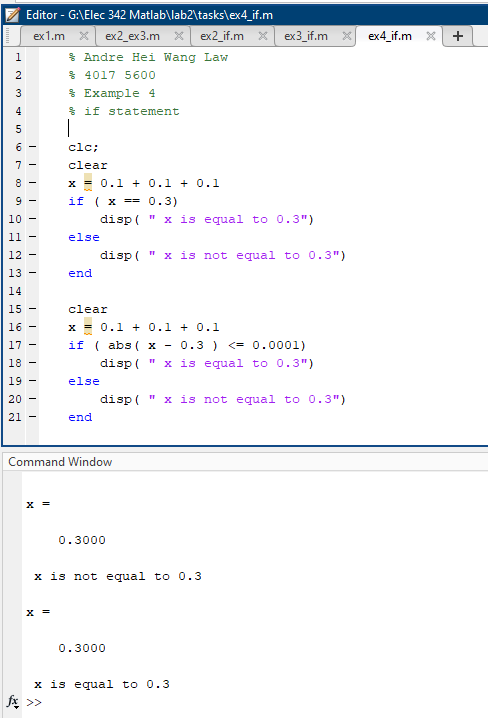
Discussion 4:

Similarly to task 3, students are now introduced to if-else statements. Using the previously learned input statement, it is possible to further the previous code with an else statement which displays whenever a number is positive. The image above proves that the inputs and if-else statement functions accordingly.

Task 5: If-Else Statement using Absolute Value

% Andre Hei Wang Law  
% 4017 5600  
% Example 4  
% if-else and absolute value  
  
clc;  
clear  
x = 0.1 + 0.1 + 0.1  
**if** ( x == 0.3)  
 disp( " x is equal to 0.3")  
**else**  
 disp( " x is not equal to 0.3")  
**end**  
  
clear  
x = 0.1 + 0.1 + 0.1  
**if** ( abs( x - 0.3 ) <= 0.0001)  
 disp( " x is equal to 0.3")  
**else**  
 disp( " x is not equal to 0.3")  
**end**

Results 5:



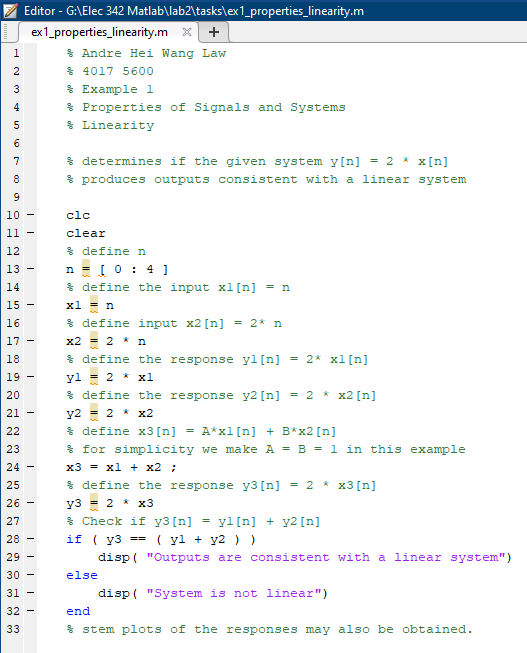
Discussion 5:

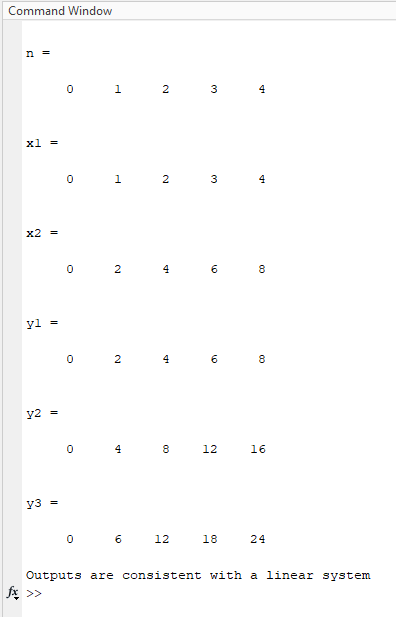
In this activity, students learned that MATLAB doesn’t perform exact calculations. Given the above code, it should be obvious that x = 0.1+0.1+0.1 = 0.3. However, MATLAB displays that x isn’t equal to 0.3 unless we put an absolute value statement. This is due to the fact that MATLAB have very small decimal integers after zero that makes it inequal. Therefore, it is best practice to insert an absolute value with a small threshold value to provide MATLAB with a margin of precision.

Task 6: Linearity

% Andre Hei Wang Law  
% 4017 5600  
% Example 1  
% Properties of Signals and Systems  
% Linearity  
  
% determines if the given system y[n] = 2 \* x[n]   
% produces outputs consistent with a linear system  
  
clc  
clear  
% define n  
n = [ 0 : 4 ]  
% define the input x1[n] = n  
x1 = n  
% define input x2[n] = 2\* n  
x2 = 2 \* n  
% define the response y1[n] = 2\* x1[n]  
y1 = 2 \* x1  
% define the response y2[n] = 2 \* x2[n]  
y2 = 2 \* x2  
% define x3[n] = A\*x1[n] + B\*x2[n]  
% for simplicity we make A = B = 1 in this example  
x3 = x1 + x2 ;  
% define the response y3[n] = 2 \* x3[n]  
y3 = 2 \* x3  
% Check if y3[n] = y1[n] + y2[n]  
**if** ( y3 == ( y1 + y2 ) )  
 disp( "Outputs are consistent with a linear system")  
**else**  
 disp( "System is not linear")  
**end**  
% stem plots of the responses may also be obtained.

Results 6:





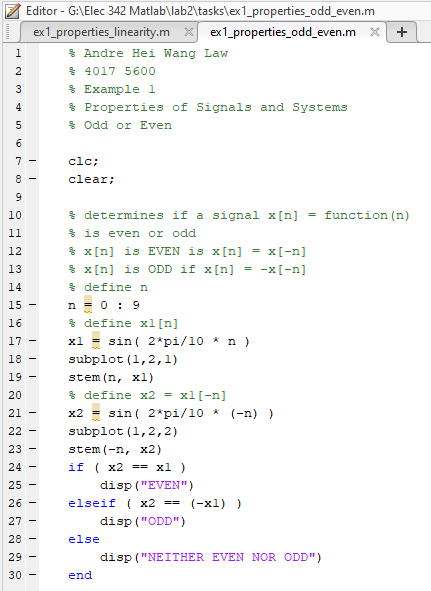
Discussion 6:

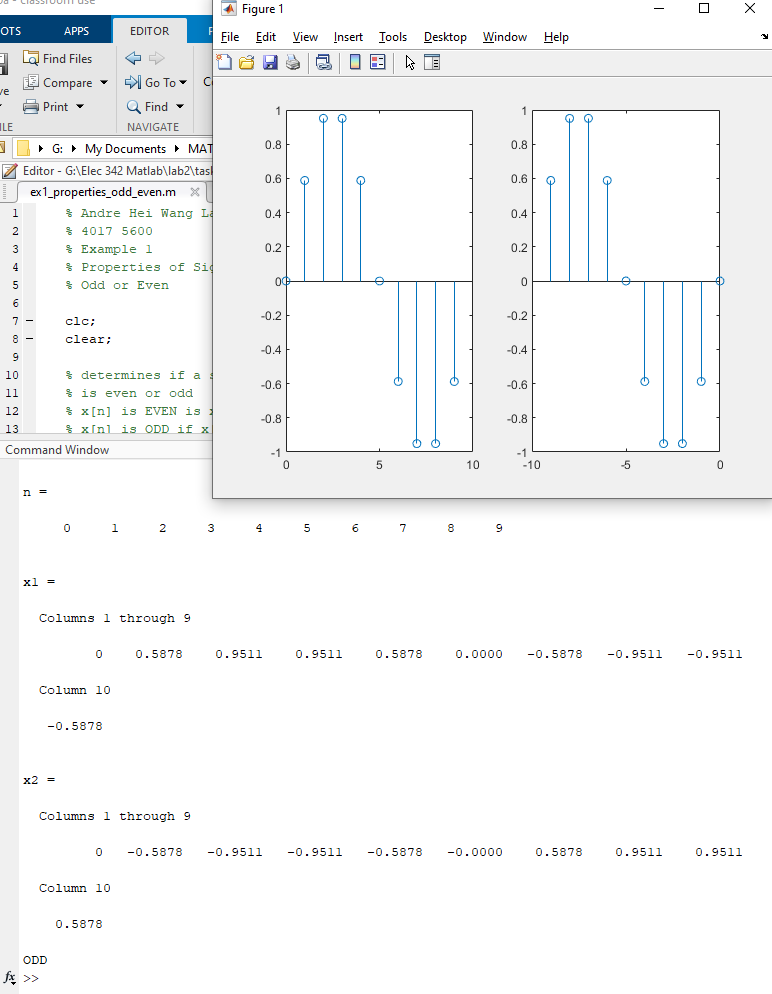
To demonstrate the linearity property of signals and systems, we can perform the additive property of linear system test. In the above example, the range n is from 0 to 4 and the output y is 2x. As such, this property can be proven by comparing the values of y3 and y1+y2. If this holds true, it means that this system is linear.

Task 7: Odd or Even

% Andre Hei Wang Law  
% 4017 5600  
% Example 1  
% Properties of Signals and Systems  
% Odd or Even  
  
clc;  
clear;  
  
% determines if a signal x[n] = function(n)  
% is even or odd  
% x[n] is EVEN is x[n] = x[-n]  
% x[n] is ODD if x[n] = -x[-n]  
% define n  
n = 0 : 9  
% define x1[n]  
x1 = sin( 2\*pi/10 \* n )  
subplot(1,2,1)  
stem(n, x1)  
% define x2 = x1[-n]  
x2 = sin( 2\*pi/10 \* (-n) )  
subplot(1,2,2)  
stem(-n, x2)  
**if** ( x2 == x1 )  
 disp("EVEN")  
**elseif** ( x2 == (-x1) )  
 disp("ODD")  
**else**  
 disp("NEITHER EVEN NOR ODD")  
**end**

Results 7:





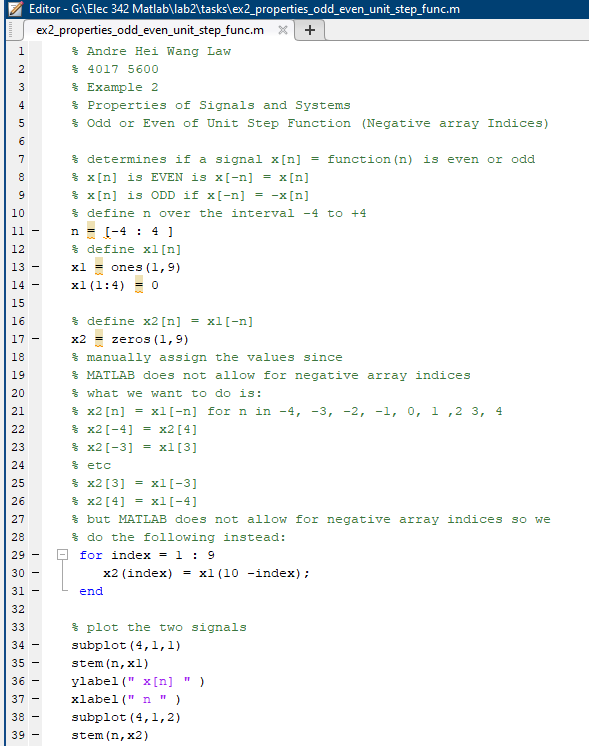
Discussion 7:

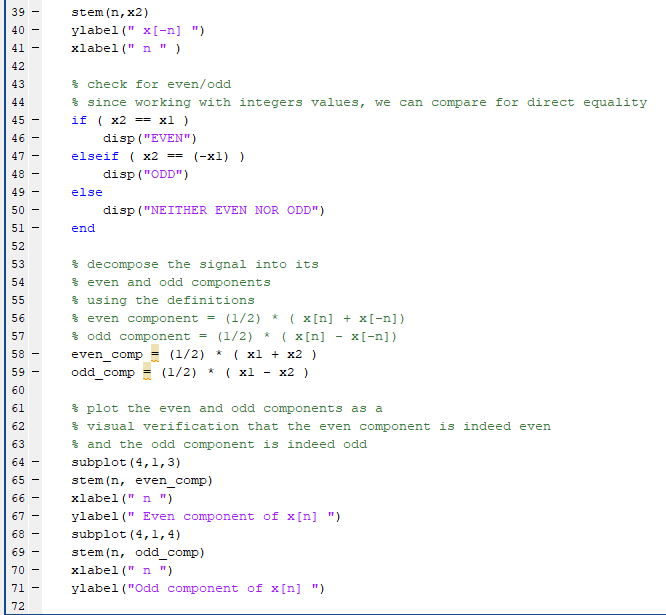
The above task works on testing odd and even signals which are said to be; even signals if x[n] = x[-n] and odd signal if x[n] = -x[-n]. Knowing this, the above example tried to determine whether the signal sin(2pi/10\*n) is off or even. The results are then displayed by an if-else statement. In this case, the sin signal is an odd signal.

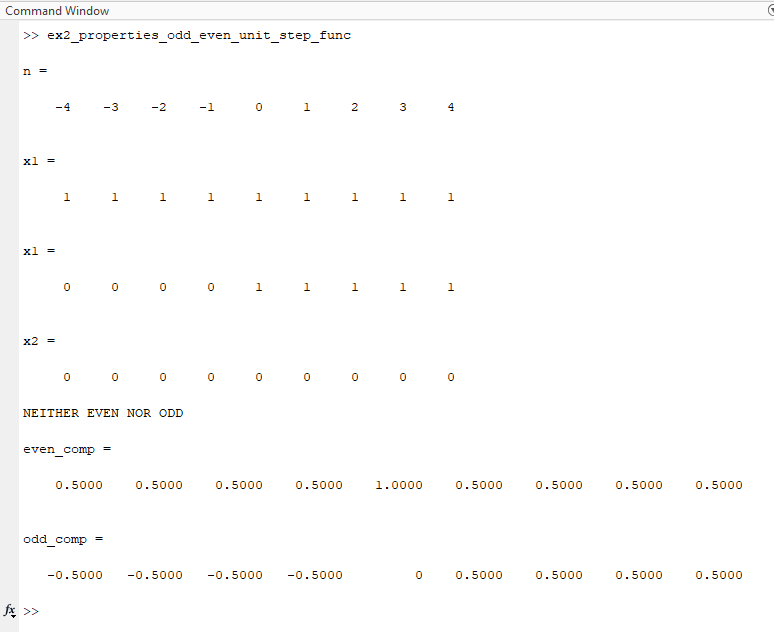
Task 8: Odd or Even of Unit Step Function (Negative Array Indices)

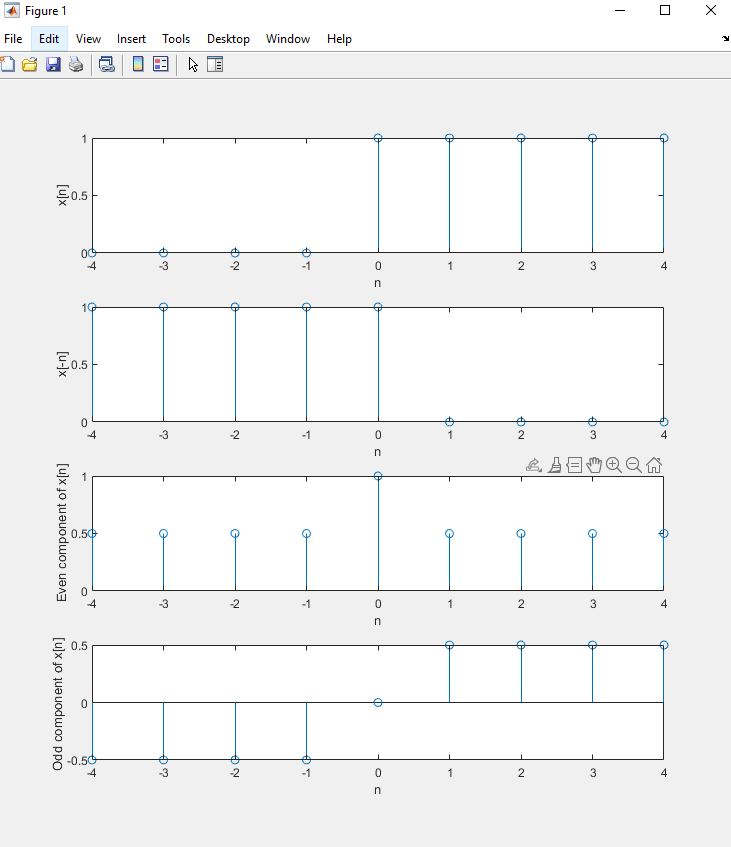
% Andre Hei Wang Law  
% 4017 5600  
% Example 2  
% Properties of Signals and Systems  
% Odd or Even of Unit Step Function (Negative array Indices)  
  
% determines if a signal x[n] = function(n) is even or odd  
% x[n] is EVEN is x[-n] = x[n]  
% x[n] is ODD if x[-n] = -x[n]  
% define n over the interval -4 to +4  
n = [-4 : 4 ]  
% define x1[n]  
x1 = ones(1,9)  
x1(1:4) = 0  
  
% define x2[n] = x1[-n]  
x2 = zeros(1,9)  
% manually assign the values since  
% MATLAB does not allow for negative array indices  
% what we want to do is:  
% x2[n] = x1[-n] for n in -4, -3, -2, -1, 0, 1 ,2 3, 4  
% x2[-4] = x2[4]  
% x2[-3] = x1[3]  
% etc  
% x2[3] = x1[-3]  
% x2[4] = x1[-4]  
% but MATLAB does not allow for negative array indices so we  
% do the following instead:  
 **for** index = 1 : 9  
 x2(index) = x1(10 -index);  
 **end**  
   
% plot the two signals  
subplot(4,1,1)  
stem(n,x1)  
ylabel(" x[n] " )  
xlabel(" n " )  
subplot(4,1,2)  
stem(n,x2)  
ylabel(" x[-n] ")  
xlabel(" n " )  
  
% check for even/odd  
% since working with integers values, we can compare for direct equality  
**if** ( x2 == x1 )  
 disp("EVEN")  
**elseif** ( x2 == (-x1) )  
 disp("ODD")  
**else**  
 disp("NEITHER EVEN NOR ODD")  
**end**  
  
% decompose the signal into its  
% even and odd components  
% using the definitions  
% even component = (1/2) \* ( x[n] + x[-n])  
% odd component = (1/2) \* ( x[n] - x[-n])  
even\_comp = (1/2) \* ( x1 + x2 )  
odd\_comp = (1/2) \* ( x1 - x2 )  
  
% plot the even and odd components as a  
% visual verification that the even component is indeed even  
% and the odd component is indeed odd  
subplot(4,1,3)  
stem(n, even\_comp)  
xlabel(" n ")  
ylabel(" Even component of x[n] ")  
subplot(4,1,4)  
stem(n, odd\_comp)  
xlabel(" n ")  
ylabel("Odd component of x[n] ")

Results 8:





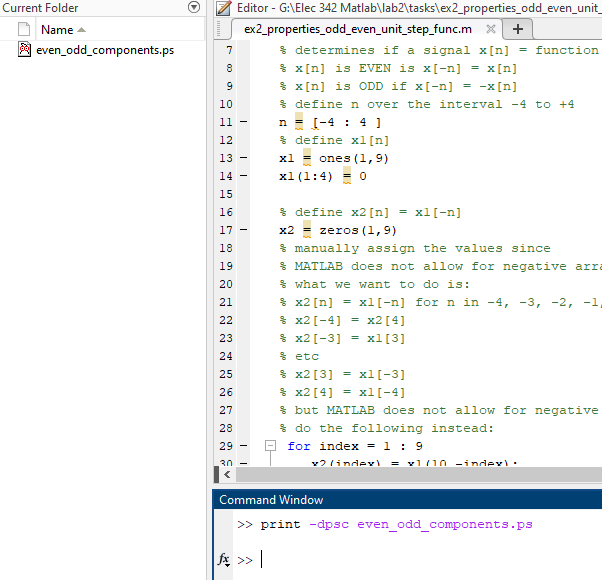




Discussion 8:

The last exercise was to practice using all the tool we’ve learned so far and use to graph even and odd signals while also understanding MATLAD special case of not being able to perform negative indices. A work around is to utilize a for-loop that reverse the order by changing the index order such were done for x2. In the following step, students used subplot and stem to graph the results in which were also put into an if-else statement to display whether the signal is odd, even or neither. Lastly, the signal was decomposed and its graph were put as subplots. The final results are four graphs, the original, its inverse, its even component and off component.

BONUS TASK:



In MATLAB, there is a print function that allows users to save charts and graphs in specific file format. In the above case, it is a PostScript file. The general syntax for this command is print(filename, formatType, formatOptions).

**4) Questions**

Part I - Question 1 a):

% Andre Hei Wang Law  
% 4017 5600  
% lab 2, Part 1, Question 1, Part a  
  
% Part a)  
clear  
clc  
  
% define n  
n = [0 : 9];  
% define the input x1[n]   
x = n;  
% define the response y1[n]  
y = x.^2;  
  
**for** i = 0 : 9  
 x = x + x.^2;  
 y = y + y.^2;  
**end**  
  
disp(x)  
disp(y)

Part I - Question 1 a): Results

Part I - Question 1 b):

% Andre Hei Wang Law  
% 4017 5600  
% lab 2, Part 1, Question 1, part b  
  
% Part b)  
clear

clc

% define n

n = [0 : 9];

% define the input x1[n] = n

x = sin((2\*pi)./(10\*n));

% define the response y1[n] = 2\* x1[n]

y = x.^2;

for i = 0 : 9

x = x + x.^2;

y = y + y.^2;

end

disp(x)

disp(y)

Part I - Question 1 b): Results

Part I - Question 2 a):

% Andre Hei Wang Law  
% 4017 5600  
% lab 2, Part 1, Question 2  
  
% part a)  
clc  
clear  
  
% define n  
n = [ 0 : 10 ]  
% define the input x1[n]   
x1 = sin(((2\*pi)./10)\*n);  
% define input x2[n]   
x2 = sin(((2\*pi)./10)\*n);  
% define the response y1[n]   
y1 = 2 \* x1;  
% define the response y2[n]   
y2 = 2 \* x2;  
  
% define x3[n] = A\*x1[n] + B\*x2[n] where A = B = 1   
x3 = x1 + x2;  
% define the response y3[n]   
y3 = 2 \* x3  
% define the response y4[n]   
y4 = y1 + y2  
  
% Check if y3[n] = y1[n] + y2[n]  
**if** ( abs(y4 - y3) <= 0.0001 )  
 disp( "Outputs are consistent with a linear system")  
**else**  
 disp( "System is not linear")  
**end**  
   
hold on % command so plot remains on same graph  
stem(n,y3) % output signal plot  
stem(n,x3) % input signal plot  
  
title("Stem Plots of all input and output signals")  
xlabel("Interval from 0 to 10")  
ylabel("Input and Output Signals")  
hold off

Part I - Question 2 a): Results

Part I - Question 2 b) i):

% Andre Hei Wang Law  
% 4017 5600  
% lab 2, Part 1, Question 1  
  
% Part b)   
% i) y[n] = x^2[n]  
% small set of values  
  
clc  
clear  
  
% define the input x1[n]   
x1 = [0,1];  
% define input x2[n]   
x2 = [0,1];  
% define the response y1[n]   
y1 = x1.^2;  
% define the response y2[n]   
y2 = x2.^2;  
  
% define x3[n] = A\*x1[n] + B\*x2[n] where A = B = 1   
x3 = x1 + x2;  
% define the response y3[n]   
y3 = x3.^2  
% define the response y4[n]   
y4 = y1 + y2  
  
% Check if y3[n] = y1[n] + y2[n]  
  
**if** ( abs(y4 - y3) <= 0.0001 )  
 disp( "Outputs are consistent with a linear system")  
**else**  
 disp( "System is not linear")  
**end**  
   
  
% large set of values  
  
% define the input x1[n]   
x1 = [0:10];  
% define input x2[n]   
x2 = [0:10];  
% define the response y1[n]   
y1 = x1.^2;  
% define the response y2[n]   
y2 = x2.^2;  
  
% define x3[n] = A\*x1[n] + B\*x2[n] where A = B = 1   
x3 = x1 + x2;  
% define the response y3[n]   
y3 = x3.^2  
% define the response y4[n]   
y4 = y1 + y2  
  
% Check if y3[n] = y1[n] + y2[n]  
**if** ( abs(y4 - y3) <= 0.0001 )  
 disp( "Outputs are consistent with a linear system")  
**else**  
 disp( "System is not linear")  
**end**

Part I - Question 2 b) i): Results

Part I - Question 2 b) ii):

% Andre Hei Wang Law  
% 4017 5600  
% lab 2, Part 1, Question 1  
  
% Part b)   
% ii) y[n] = 2x[n] + 5δ[n]  
% small set of values  
  
clc  
clear  
  
% define the input x1[n]   
x1 = [0,1];  
% define input x2[n]   
x2 = [0,1];  
% define the response y1[n]   
y1 = 2\*x1 + 5\*x1;  
% define the response y2[n]   
y2 = 2\*x2 + 5\*x2;  
  
% define x3[n] = A\*x1[n] + B\*x2[n] where A = B = 1   
x3 = x1 + x2;  
% define the response y3[n]   
y3 = x3.^2  
% define the response y4[n]   
y4 = y1 + y2  
  
% Check if y3[n] = y1[n] + y2[n]  
  
**if** ( abs(y4 - y3) <= 0.0001 )  
 disp( "Outputs are consistent with a linear system")  
**else**  
 disp( "System is not linear")  
**end**  
   
  
% large set of values  
  
% define the input x1[n]   
x1 = [0:10];  
% define input x2[n]   
x2 = [0:10];  
% define the response y1[n]   
y1 = 2\*x1 + 5\*x1;  
% define the response y2[n]   
y2 = 2\*x2 + 5\*x2;  
  
% define x3[n] = A\*x1[n] + B\*x2[n] where A = B = 1   
x3 = x1 + x2;  
% define the response y3[n]   
y3 = x3.^2  
% define the response y4[n]   
y4 = y1 + y2  
  
% Check if y3[n] = y1[n] + y2[n]  
**if** ( abs(y4 - y3) <= 0.0001 )  
 disp( "Outputs are consistent with a linear system")  
**else**  
 disp( "System is not linear")  
**end**

Part I - Question 2 b) ii): Results

Part I - Question 3 a):

% Andre Hei Wang Law  
% 4017 5600  
% lab 2, Part 1, Question 3  
  
% part a)  
clc  
clear  
% define n  
n = [0:10]  
% define the input x1[n]   
format long  
x1 = exp(-2\*abs(n)).\*sin((2.\*pi./36).\*n);  
% define the input x2[n]   
x2 = x1;  
  
% x[-n] input  
**for** index = 1 : 11  
 x2(index) = x1(12 -index);  
**end**  
  
subplot(4,1,1)  
stem(n,x1) % first plot  
ylabel(" x[n] " )  
xlabel(" n ")  
  
subplot(4,1,2)  
stem(n,x2) % second plot  
ylabel(" x[-n]")  
xlabel(" n ")  
  
**if** ( x2 == x1 )  
 disp("EVEN")  
**elseif** ( x2 == (-x1) )  
 disp("ODD")  
**else**  
 disp("NEITHER EVEN NOR ODD")  
**end**  
  
% decompose the signal into its even and odd components  
% using the definitions even component = (1/2) \* ( x[n] + x[-n])  
% odd component = (1/2) \* ( x[n] - x[-n])  
even\_comp = (1/2) \* ( x1 + x2 );  
odd\_comp = (1/2) \* ( x1 - x2 );  
% plot the even and odd components as a visual verification that the   
% even component is indeed even and the odd component is indeed odd  
subplot(4,1,3)  
stem(n, even\_comp) % third plot  
xlabel(" n " )  
ylabel(" Even component of x[n] ")  
  
subplot(4,1,4)  
stem(n, odd\_comp) % fourth plot  
xlabel(" n " )  
ylabel(" Odd component of x[n] ")

Part I - Question 3 a): Results

Part I - Question 3 b):

% Andre Hei Wang Law  
% 4017 5600  
% lab 2, Part 1, Question 3  
  
% part b)  
clc  
clear  
% define n  
n = [-5:5]  
% define the input x1[n]   
format long  
x1 = (-1).^(n-1);  
% define the input x2[n]   
x2 = x1;  
  
% x[-n] input  
**for** index = 1 : 11  
 x2(index) = x1(12 -index);  
**end**  
  
subplot(4,1,1)  
stem(n,x1) % first plot  
ylabel(" x[n] " )  
xlabel(" n ")  
  
subplot(4,1,2)  
stem(n,x2) % second plot  
ylabel(" x[-n]")  
xlabel(" n ")  
  
**if** ( x2 == x1 )  
 disp("EVEN")  
**elseif** ( x2 == (-x1) )  
 disp("ODD")  
**else**  
 disp("NEITHER EVEN NOR ODD")  
**end**  
  
% decompose the signal into its even and odd components  
% using the definitions even component = (1/2) \* ( x[n] + x[-n])  
% odd component = (1/2) \* ( x[n] - x[-n])  
even\_comp = (1/2) \* ( x1 + x2 );  
odd\_comp = (1/2) \* ( x1 - x2 );  
% plot the even and odd components as a visual verification that the   
% even component is indeed even and the odd component is indeed odd  
subplot(4,1,3)  
stem(n, even\_comp) % third plot  
xlabel(" n " )  
ylabel(" Even component of x[n] ")  
  
subplot(4,1,4)  
stem(n, odd\_comp) % fourth plot  
xlabel(" n " )  
ylabel(" Odd component of x[n] ")

Part I - Question 3 b): Results

Part I - Question 3 c):

% Andre Hei Wang Law  
% 4017 5600  
% lab 2, Part 1, Question 3  
  
% part c)  
clc  
clear  
n = [1 : 20 ]  
x1 = sin((2\*pi/40) \* n) .\* cos((2\*pi/40) \* n)  
**for** index = 1 : 20  
% Note: In MATLAB, no need to pre-allocate the array,  
% unlike C++ and other high-level programming languages.  
 x2(index) = sin((2\*pi/40) \* index) \* cos((2\*pi/40) \* index)  
**end**  
subplot(2,1,1)  
stem(n, x1)  
title("Elegant method making full use of MATLABs array capabilities")  
xlabel("n")  
ylabel("x[n]")  
subplot(2,1,2)  
stem(n, x2)  
title("Gets the job done, but it is a lot of work and we are not in the MATLAB mindset")  
xlabel("n")  
ylabel("x[n]")

Part I - Question 3 c): Results

Part II - Question 1:

Part II - Question 2:

Part II - Question 3 a):

Part II - Question 3 b):

**5) Conclusions**

In conclusion, for the second experiment, students were introduced further into additional MATLAB features and commands such as using absolute value, print, for loops, etc. Then, we tested properties of signals and systems such as proving if a system is linear, time invariant, etc. Lastly, students worked on even and odd signals with follow-ups on subplots and stems which were introduced in experiment 1. Overall, by working on these aspects, students were able to practice a system’s response to an input signal.

**6) Appendix**

There will be no appendix for this experiment. All source code were placed accordingly in their section above.

**7) References**

ELEC 342 Lab 2: Additional MATLAB features, Properties of Signals and Systems, Convolution and System Response

<https://users.encs.concordia.ca/~realtime/elec342/manuals/lab2.pdf>