**Task 1:**

Basically it talks about Signal representation in analog and discrete. Talked about various matlab functions like ones, stepseq, exp() e.t.c. It talked about operations on sequences ranging from signal addition, signal multiplication, scaling, shifting, folding, sample summation, sample products and signal energy.

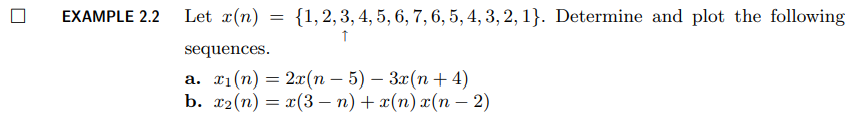
We utilize most of the concepts in Task 2.

**Task 2**

**Example 2.1**

| **a)**    **Code:**  **%Example 2.1**  **%Syed Asghar Abbas Zaidi**  **n = [-5:5];**  **x1 = 2\*impseq(-2,-5,5)**  **x2 = impseq(4,-5,5);**  **[x,n2] = sigadd(x1,n,-x2,n)**  **subplot(2,2,1);**  **stem(n,x);**  **title('Sequence in Problem 2.1a, Asghar')**  **xlabel('n'); ylabel('x(n)');**  **Output:**    **Analysis:**  We define a function impseq(n0,n1,n2) that generates a discrete-time unit impulse sequence x(n) = delta(n-n0) for the range n1 <= n <= n2. The function returns two variables x and n, where x is the generated sequence and n is the range of n values.  The code that we are using follows the logic, such that it uses the function to generate sequence x for the range -5 <= n <= 5. The sequence is defined as 2\*impseq(-2,-5,5) - impseq(4,-5,5).  The 2\*impseq(-2,-5,5) makes an impulse at n = -2 of amplitude 2.  While, -impseq(4,-5,5). Makes an impulse at n=4 of amplitude -1.  We add the two functions using the **sigadd** function! It adds the two signals, x1 and x2, with their respective time indices n1 and n2. It determines the time indices of the resulting signal y by considering the common duration of both input signals. The function initialises arrays y1 and y2 with zeros, maps the values of x1 and x2 to the corresponding durations of y1 and y2, and finally adds these two sequences element-wise to obtain the resulting signal y. The resulting signal is valid for the duration that includes both n1 and n2.  **I won’t be repeating sigadd functionality in later parts, I will be using it to add two functions together.**  In summary, the code defines a function to generate a discrete-time unit impulse sequence and uses it to generate and plot a sequence for a specific range of n values.  Aside of that, the output is correct cause equation tells us of 2 impulses, first with the amplitude of 2 at -2, and second at 4, with the amplitude of -1 |
| --- |
| **b)**    **Code:**  **%Example 2.1b**  **%Syed Asghar Abbas Zaidi**  **n = [0:20];**  **x1 = n.\*(stepseq(0,0,20)-stepseq(10,0,20));**  **x2 = 10\*exp(-0.3\*(n-10)).\*(stepseq(10,0,20)-stepseq(20,0,20));**  **[x,n3] = sigadd(x1,n,x2,n);**  **subplot(2,2,2);**  **stem(n,x);**  **title('Sequence in Problem 2.1b, Asghar')**  **xlabel('n');**  **ylabel('x(n)');**  **Figure:**    **Analysis:**  We use a function stepseq(n0,n1,n2) that generates a discrete-time unit step sequence x(n) = u(n-n0) for the range n1 <= n <= n2. The function returns two variables x and n, where x is the generated sequence and n is the range of n values.  The code then uses this function to generate two sequences x1 and x2 for the range 0 <= n <= 20. The first sequence x1 is defined as n.\*(stepseq(0,0,20)-stepseq(10,0,20)) and the second sequence x2 is defined as 10\*exp(-0.3\*(n-10)).\*(stepseq(10,0,20)-stepseq(20,0,20)). The final sequence x is obtained by adding x1 and x2. The sequence is then plotted using the stem function.  In summary, the code defines a function to generate a discrete-time unit step sequence and uses it to generate and plot two sequences for a specific range of n values. |
| **c)**    **Code:**  **%Example 2.1c**  **%Syed Asghar Abbas Zaidi**  **n = [0:50];**  **x1 = cos(0.04\*pi\*n)**  **x2 = 0.2\*randn(size(n));**  **[x,n4] = sigadd(x1,n,x2,n);**  **subplot(2,2,3);**  **stem(n,x);**  **title('Sequence in Problem 2.1c, Asghar')**  **xlabel('n');**  **ylabel('x(n)');**  **Figure:**    **Analysis:** The given code generates a sequence “x(n)” for the range 0 <= n <= 50.  The sequence is defined as “cos(0.04\*pi\*n)+0.2\*randn(size(n))”.  The ‘cos’ function generates a cosine wave with a frequency of ‘0.04\*pi’ and the ‘randn’ function generates random noise with a mean of ‘0’ and a standard deviation of ‘1’. The ‘size’ function returns the size of the input array ‘n’. The sequence is then plotted using the `stem` function.  In summary, the code generates a sequence using a cosine wave and random noise and plots it for a specific range of “n” values. It generates a Cosine wave on which, a noise is added onto it. |
| **d)**    **Code:**  **%Example 2.1d**  **%Syed Asghar Abbas Zaidi**  **n = [-10:9]; x = [5,4,3,2,1];**  **xtilde = x' \* ones(1,4);**  **xtilde = (xtilde(:))';**  **stem(n,xtilde); title('Sequence in Problem 2.1d, Asghar')**  **xlabel('n'); ylabel('xtilde(n)');**  **Output:**    **Analyse:**  The sequence is defined as xtilde(n), where ‘n’ is a vector ranging from -10 to 9 and xtilde is a matrix of size 5x4.  The matrix xtilde is created by multiplying the transpose of x with a matrix of ones of size 1x4. The resulting matrix is then reshaped into a vector using the colon operator (:) |
| **Final Code:**  **%Syed Asghar Abbas Zaidi**  **n = [-5:5];**  **x1 = 2\*impseq(-2,-5,5)**  **x2 = impseq(4,-5,5);**  **[x,n2] = sigadd(x1,n,-x2,n)**  **subplot(2,2,1);**  **stem(n,x);**  **title('Sequence in Problem 2.1a, Asghar')**  **xlabel('n'); ylabel('x(n)');**  **n = [0:20];**  **x1 = n.\*(stepseq(0,0,20)-stepseq(10,0,20));**  **x2 = 10\*exp(-0.3\*(n-10)).\*(stepseq(10,0,20)-stepseq(20,0,20));**  **[x,n3] = sigadd(x1,n,x2,n);**  **subplot(2,2,2);**  **stem(n,x);**  **title('Sequence in Problem 2.1b, Asghar')**  **xlabel('n');**  **ylabel('x(n)');**  **n = [0:50];**  **x1 = cos(0.04\*pi\*n)**  **x2 = 0.2\*randn(size(n));**  **[x,n4] = sigadd(x1,n,x2,n);**  **subplot(2,2,3);**  **stem(n,x);**  **title('Sequence in Problem 2.1c, Asghar')**  **xlabel('n');**  **ylabel('x(n)');**  **n = [-10:9]; x = [5,4,3,2,1];**  **xtilde = x' \* ones(1,4);**  **xtilde = (xtilde(:))';**  **subplot(2,2,4);**  **stem(n,xtilde);**  **title('Sequence in Problem 2.1d, Asghar')**  **xlabel('n'); ylabel('xtilde(n)');**  **Final Output:** |

**Example 2.2**

****

**Answer:**

| **a)**    **Code:**  **n = -2:10;**  **x = [1:7,6:-1:1];**  **[x11,n11] = sigshift(x,n,5);**  **[x12,n12] = sigshift(x,n,-4);**  **[x1,n1] = sigadd(2\*x11,n11,-3\*x12,n12);**  **%subplot(2,1,1);**  **stem(n1,x1); title('Sequence in Example 2.2a, Asghar')**  **xlabel('n'); ylabel('x1(n)');**  **Figure:**    **Analysis:**  Signal Shifting Function (sigshift):  Takes an input signal x and its corresponding time indices m.  Shifts the signal by an amount specified by k.  Outputs the shifted signal y and the new time indices n.  Signal Generation and Operations:  Generates a signal x with time indices n ranging from -2 to 10.  Shifts the signal x by 5 units forward (x11) and 4 units backward (x12) using the sigshift function.  Combines the shifted signals and forms a new signal x1. |
| --- |
| **b)**    **Code:**  **n = -2:10;**  **x = [1:7,6:-1:1];**  **[x21,n21] = sigfold(x,n); [x21,n21] = sigshift(x21,n21,3);**  **[x22,n22] = sigshift(x,n,2); [x22,n22] = sigmult(x,n,x22,n22);**  **[x2,n2] = sigadd(x21,n21,x22,n22);**  **%subplot(2,1,2);**  **stem(n2,x2); title('Sequence in Example 2.2b, Asghar')**  **xlabel('n');**  **ylabel('x2(n)'); Figure:**    **Analysis:**  **Here's a summary of the provided code:**  SIGSHIFT Function:  (explained in (a) already)  SIGFOLD Function:  - Implements the operation y(n) = x(-n), where y is the folded signal.  - Takes two inputs: x (input signal) and n(time indices).  - Outputs the folded signal y and the new time indices n.  MAIN CODE ANALYSIS:  Creates a time vector **n** ranging from -2 to 10 and a signal x consisting of a sequence  Calls the SIGFOLD function to fold the signal x and then **shifts it by 3 units forward.**  Calls the SIGSHIFT function to **shift the original signal x by 2 units** and then **multiplies it element-wise with the shifted signal.**  **Adds the two resulting signals to form a new signal x2.**  In simpler terms, the code manipulates signals by folding, shifting, and combining them. |
| **Final Code:**  **n = -2:10;**  **x = [1:7,6:-1:1];**  **[x11,n11] = sigshift(x,n,5);**  **[x12,n12] = sigshift(x,n,-4);**  **[x1,n1] = sigadd(2\*x11,n11,-3\*x12,n12);**  **subplot(2,1,1);**  **stem(n1,x1); title('Sequence in Example 2.2a, Asghar')**  **xlabel('n'); ylabel('x1(n)');**  **[x21,n21] = sigfold(x,n); [x21,n21] = sigshift(x21,n21,3);**  **[x22,n22] = sigshift(x,n,2); [x22,n22] = sigmult(x,n,x22,n22);**  **[x2,n2] = sigadd(x21,n21,x22,n22);**  **subplot(2,1,2);**  **stem(n2,x2); title('Sequence in Example 2.2b, Asghar')**  **xlabel('n');**  **ylabel('x2(n)');**  **Final Figure:** |

**Example 2.3**

| **Code:**  **n = [-10:1:10]; alpha = -0.1+0.3j;**  **x = exp(alpha\*n);**  **subplot(2,2,1); stem(n,real(x));title('real part');xlabel('n')**  **subplot(2,2,2); stem(n,imag(x));title('imaginary part');xlabel('n')**  **subplot(2,2,3); stem(n,abs(x));title('magnitude part');xlabel('n')**  **subplot(2,2,4); stem(n,(180/pi)\*angle(x));title('phase part');xlabel('n')**  **Figure:**    **Analysis:**  The provided MATLAB code generates and plots the characteristics of a complex exponential signal x with time indices n:  Time Indices and Complex Number:  - Defines time indices n from -10 to 10 with a step of 1.  - Defines a complex number alpha = -0.1 + 0.3j.  Complex Exponential Signal x:  - Generates a complex exponential signal using the formula (x = e^{alpha n}).  Plotting Real and Imaginary Parts:  - Subplot 1: Plots the real part of the signal against n.  - Subplot 2: Plots the imaginary part of the signal against n.  Plotting Magnitude and Phase:  - Subplot 3: Plots the magnitude of the signal against n.  - Subplot 4: Plots the phase of the signal (in degrees) against n.  In summary, the code provides a visual representation of the real part, imaginary part, magnitude, and phase of a complex exponential signal over the specified range of time indices. Each subplot offers insights into different aspects of the signal. |
| --- |

**TASK 3:**

**Example 2.4**

| **Code:**  **n = [0:10]; x = stepseq(0,0,10)-stepseq(10,0,10);**  **[xe,xo,m] = evenodd(x,n);**  **subplot(2,2,1); stem(n,x); title('Rectangular pulse, Asghar')**  **xlabel('n'); ylabel('x(n)'); axis([-10,10,0,1.2])**  **subplot(2,2,2); stem(m,xe); title('Even Part, Asghar')**  **xlabel('n'); ylabel('xe(n)'); axis([-10,10,0,1.2])**  **subplot(2,2,4); stem(m,xo); title('Odd Part, Asghar')**  **xlabel('n'); ylabel('xe(n)'); axis([-10,10,-0.6,0.6])**  **Figure:**    **Analysis:**  The code generates and visualises a rectangular pulse signal (x(n) and it's even xe(n) and odd xo(n) components. The rectangular pulse is created using step sequences, and the signal is decomposed into even and odd parts using a function called “evenodd”. The resulting signals are then plotted in different subplots:  Rectangular Pulse Signal:  Subplot 1: Displays the rectangular pulse signal x(n) with time indices from 0 to 10.  Even Part of the Signal:  Subplot 2: Shows the even part xe(n) of the signal obtained through decomposition.  Odd Part of the Signal:  Subplot 4: Displays the odd part xo(n) of the signal obtained through decomposition.  Axis limits are adjusted for better visualisation in each subplot. |
| --- |

**EXERCISES**

1. **Exercise P2.1 Part, 1,3,4,5**

| **Code:**  **%Syed Asghar Abbas Zaidi**  **n = [-5:15];**  **x1 = 3\*impseq(-2,-5,15)**  **x2 = 2\*impseq(0,-5,15)**  **x3 = -impseq(3,-5,15)**  **x4 = 5\*impseq(7,-5,15)**  **x = x1 + x2 + x3 + x4;**  **stem(n,x);**  **title('P2.1 Part(a), Asghar')**  **xlabel('n'); ylabel('x(n)');**  **Figure:**    **Analysis:**  As can clearly be observed, the output correctly showcases the equation where there are 4 different impulses on n = -2,0,3,7 with amplitude 3,2,-1,5 respectively. This was achieved using impseq function    **Code:**  **%Syed Asghar Abbas Zaidi**  **n = [0:30];**  **x1 = 10\*stepseq(0,0,30)**  **x2 = -5\*stepseq(5,0,30)**  **x3 = -10\*stepseq(10,0,30)**  **x4 = +5\*stepseq(15,0,30)**  **x = x1 + x2 + x3 + x4;**  **stem(n,x);**  **title('P2.1 Part(c), Asghar')**  **xlabel('n'); ylabel('x(n)');**  **Figure:**    **Analysis:**  Equations have multiple unit step functions added onto each other so their amplitudes add or cancel each other out. Amplitude remains 10 for all values below n = 5, from n = 5 to n = 9, amplitude is 5, while from n = 10 to n =14, amplitude is -5. Ahead, due to multiple unit steps cancelling each other, they always result with 0.    **Code:**  **%Syed Asghar Abbas Zaidi**  **n = [0:30];**  **x = exp(0.1\*(n)).\*(stepseq(-20,0,30)-stepseq(10,0,30));**  **stem(n,x);**  **title('P2.1 Part D, Asghar')**  **xlabel('n'); ylabel('x(n)');**  **Figure:**    **Analysis:**  The MATLAB code generates a stem plot which is the product of an exponentially increasing term exp(0.1n) and a rectangular pulse that starts at -20 and ends at 0, subtracted by a pulse starting at 10 and ending at 30. This combination results in a sequence that emphasizes the exponential growth while excluding certain intervals.    **Code:**  **%Syed Asghar Abbas Zaidi**  **n = [-200:200];**  **x1 = cos(0.049\*pi\*n)**  **x2 = cos(0.051\*pi\*n)**  **x = x1 + x2**  **stem(n,x);**  **title('P2.1 Part E, Asghar')**  **xlabel('n');**  **ylabel('x(n)');**  **Figure:**    **Analysis:**  The code generates a stem plot of x(n), where x(n) is the **sum** of two cosine functions: **cos(0.049\*pi\*n) and cos(0.051\*pi\*n).** |
| --- |

All **functions** were provided in the book by default so I didn’t wanted to be redundant by including them here. Aside of that, all of my code including functions is uploaded on canvas through zip file if you want to check.

**THANK YOU!**