EE-384: Digital Signal Processing **Spring 2023**

Laboratory 1: Discrete-time Signals in MATLAB

Instructor: Mr Ammar Naseer *EE UET New Campus*

Aims

We begin with the concepts of signals in discrete time. A number of important types of signals and their operations are introduced. The emphasis in this chapter is on the representations and implementation of signals using MATLAB.

Pre-Lab:

Discrete time Signals

Signals are broadly classified into analog and discrete signals. An analog signal will be denoted by x(t), in which the variable t can represent any physical quantity, but we will assume that it represents time in seconds. A discrete signal will be denoted by x(n), in which the variable n is integer-valued and represents discrete instances in time. Therefore it is also called a discrete-time signal, which is a number sequence and will be denoted by one of the following notations.

$$x(n) = \{x(n)\} = \{\dots, x(1), \underset{\uparrow}{x(0)}, x(1), \dots\}$$

where the up-arrow indicates the sample at n = 0.

In MATLAB we can represent a finite-duration sequence by a row vector of appropriate values. However, such a vector does not have any information about sample position n. Therefore a correct representation of x(n) would require two vectors, one each for x and n. For example, a sequence x(n) = $[2\ 1\ -1\ \underset{\uparrow}{0}\ 1\ 4\ 3\ 7]$ can be represented in MATLAB by

```
>> n=[-3,-2,-1,0,1,2,3,4]; x=[2,1,-1,0,1,4,3,7];
```

Generally, we will use the x-vector representation alone when the sample position information is not required or when such information is trivial (e.g. when the sequence begins at n = 0). An arbitrary infinite-duration sequence cannot be represented in MATLAB due to the finite memory limitations.

Type of Sequences

We use several elementary sequences in digital signal processing for analysis purposes. Their definitions and MATLAB representations follow.

Unit Sample Sequence

$$\delta(n) = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases} = \{\ldots, 0, 0, \underset{\uparrow}{1}, 0, 0, \ldots\}$$

In MATLAB the function zeros (1,N) generates a row vector of N zeros, which can be used to implement δ(n) over a finite interval. However, the logical relation n==0 is an elegant way of implementing δ(n).

To implement,

$$\delta(n - n_0) = \begin{cases} 1 & n = n_0 \\ 0 & n \neq n_0 \end{cases}$$

over the $n_1 < n_0 < n_2$ interval, we will use the following MATLAB function

```
function [x,n] = impseq(n1,n2,n0)
% -------------------------------
% [x,n] = impseq(n1,n2,n0)
% Generates x(n) = impulse(n-n0); n1 < n < n2
% Generate an impulse with a length of n1 < n < n2
% The impulse would have a delay of n0

n = n1:n2;
x = [(n-n0) == 0];
```

For example, to generate $x(n) = \delta[n-2]$; $-5 \leq n \leq 5$, we will need the following MATLAB script:

```
>> [x,n] = impseq(-5,5,2);
>> stem(n,x)
```

Unit Step Sequence

$$u(n) = \begin{cases} 1, & n \geq 0 \\ 0, & n < 0 \end{cases} = \left\{\ldots, 0, 0, \underset{\uparrow}{1}, 1, 1, \ldots\right\}$$

In MATLAB the function ones(1,N) generates a row vector of N ones. It can be used to generate u(n) over a finite interval. Once again an elegant approach is to use the logical relation n>=0. To implement

$$u(n - n_0) = \begin{cases} 1, & n \geq n_0 \\ 0, & n < n_0 \end{cases}$$

over the $n_1 < n_0 < n_2$ interval, we will use the following MATLAB function

```
function [x,n] = stepseq(n1,n2,n0)
% -----------------------------------------
% [x,n] = stepseq(n0,n1,n2)
% Generates x(n) = u(n-n0); n1 < n < n2
% Generate a plot with a length of n1 < n < n2
% The step function would have a delay of n0
n = [n1:n2];
x = [(n-n0) >= 0];
```

For example, to generate $x(n) = u[n+1]$; -5 ≤ n ≤ 5, we will need the following MATLAB script:

```
>> [x,n] = stepseq(-5,5,-1);
>> stem(n,x)
```

Real-valued exponential sequence

$$x(n) = a^n, \quad \forall n; \quad a \in \mathbb{R}$$

In MATLAB an array operator "." is required to implement a real exponential sequence.

For example, to generate x(n) = $(0.9)^n$; 0 ≤ n ≤ 10, we will need the following MATLAB script:

```
>> n = [0:10]; x = (0.9).^n;
>> stem(n,x)
```

Complex-valued exponential sequence

$$x(n) = e^{(\sigma + j\omega_0)n}, \quad \forall n$$

Where σ produces an attenuation (if < 0) or amplification (if > 0) and ω_0 is the frequency in radians. A MATLAB function **exp** is used to generate exponential sequences.

For example, to generate x(n) = exp[(2+j3)n], 0 ≤ n ≤ 10, we will need the following MATLAB script:

```
>> n = [0:10];x = exp((2+3j)*n);
>> stem(n,x)
```

Sinusoidal sequence

$$x(n) = a\cos(\omega_0 n + \theta_0), \quad \forall n$$

where A is an amplitude and θ_0 is the phase in radians. A MATLAB function cos (or sin) is used to generate sinusoidal sequences. For example, to generate $x(n) = 3\cos\left(0.1\pi n + \frac{\pi}{3}\right) + 2\sin\,(0.5\pi n)$. 0<n<10, we will need the following MATLAB script:

```
>> n = [0:10]; x = 3*cos(0.1*pi*n+pi/3) + 2*sin(0.5*pi*n);
>> stem(n,x)
```

Periodic Sequence

A sequence x(n) is periodic if x(n) = x(n+N); ∀ n. The smallest integer N that satisfies this relation is called the fundamental period. We will use $\tilde{x}$(n) to denote a periodic sequence. To generate P periods of $\tilde{x}$ (n) from one period {x(n), 0<n<N1}, we can copy x(n) P times:

```
>> xtilde = [x,x,...,x];
```

But an elegant approach is to use MATLABs powerful indexing capabilities. First we generate a matrix containing P rows of x(n) values. Then we can concatenate P rows into a long row vector using the construct (:). However, this construct works only on columns. Hence we will have to use the matrix transposition operator ' to provide the same effect on rows.

```
>> xtilde = x' * ones(1,P); % P columns of x; x is a row vector
>> xtilde = xtilde(:); % long column vector
>> xtilde = xtilde'; % long row vector
```

Note that the last two lines can be combined into one for compact coding.

Operation on sequence

Here we briefly describe basic sequence operations and their MATLAB equivalents.

1. **Signal addition**: This is a sample-by-sample addition given by
$$\{x_1[n]\} + \{x_2[n]\} = \{x_1[n] + x_2[n]\}$$
It is implemented in MATLAB by the arithmetic operator +. However, the lengths of x_1(n) and x_2(n) must be the same. If sequences are of unequal lengths, or if the sample positions are different for equal-length sequences, then we cannot directly use the operator +. We have to first augment x_1(n) and x_2(n) so that they have the same position vector n (and hence the same length). This requires careful attention to MATLABs indexing operations. In particular, logical operation of intersection **&**, relational operations like ;= and ==, and the **find** function are required to make x_1(n) and x_2(n) of equal length. The following function, called the sigadd function, demonstrates these operations.

```
function [y,n] = sigadd(x1,n1,x2,n2)
% ------------------------------------
% [y,n] = sigadd(x1,n1,x2,n2)
% implements y(n) = x1(n)+x2(n)
% y = sum sequence over n, which includes n1 and n2
% x1 = first sequence over n1
% x2 = second sequence over n2 (n2 can be different from n1)

n = min(min(n1),min(n2)):max(max(n1),max(n2)); % duration of y(n)
y1 = zeros(1,length(n));
y2 = y1; % initialization
y1(find((n>=min(n1))&(n<=max(n1))==1))=x1; % x1 with duration of y
y2(find((n>=min(n2))&(n<=max(n2))==1))=x2; % x2 with duration of y
y = y1+y2; % sequence addition
```

For example, to generate $x(n) = \delta[n-2] + \delta[n+4]$; $-5 \leq n \leq 5$, we will need the following MATLAB script:

```
>> [x1,n1] = impseq(-5,5,4);
>> [x2,n2] = impseq(-5,5,-2);
>> [x,n] = sigadd(x1,n1,x2,n2);
>> stem(n,x)
```

2. **Signal Multiplications:** This is a sample-by-sample (or "dot") multiplication) given by

$$\{x_1[n]\}\,.\{x_2[n]\} = \{x_1[n]x_2[n]\}$$

It is implemented in MATLAB by the array operator ” * ”. Once again, the similar restrictions apply for the “ .* operator as for the + operator”. Therefore we have developed the sigmult function, which is similar to the sigadd function.

```
function [y,n] = sigmult(x1,n1,x2,n2)
% --------------------------------------------
% implements y(n) = x1(n)*x2(n)
% [y,n] = sigmult(x1,n1,x2,n2)
% y = product sequence over n, which includes n1 and n2
% x1 = first sequence over n1
% x2 = second sequence over n2 (n2 can be different from n1)

n = min(min(n1),min(n2)):max(max(n1),max(n2)); % duration of y(n)
y1 = zeros(1,length(n));
y2 = y1; %
y1(find((n>=min(n1))&(n<=max(n1))==1))=x1; % x1 with duration of y
y2(find((n>=min(n2))&(n<=max(n2))==1))=x2; % x2 with duration of y
y = y1 .* y2; % sequence multiplication
```

For example, to generate $x(n) = \delta[n-2] * \delta[n+4]$; $-5 \leq n \leq 5$, we will need the following MATLAB script:

```
>> [x1,n1] = impseq(-5,5,4);
>> [x2,n2] = impseq(-5,5,-2);
>> [x,n] = sigmult(x1,n1,x2,n2);
```

```
>> stem(n,x)
```

3. Shifting In this operation, each sample of x(n) is shifted by an amount k to obtain a shifted sequence y(n).

$$y[n] = \{x[nk]\}$$

If we let m = nk, then n = m + k and the above operation is given by

$$y[m + k] = \{x[m]\}$$

Hence this operation has no effect on the vector x, but the vector n is changed by adding k to each element. This is shown in the function **sigshift**.

```
function [y,n] = sigshift(x,m,k)
% -------------------------------------
% [y,n] = sigshift(x,m,k)
% implements y(n) = x(n-k)
%delays the function by k sequences

n = m+k;
y = x;
```

For example, to generate $x(n) = \delta[n - 2]$; -3 ≤ n ≤ 7, we can also write the following MATLAB script:

```
>> [x,n] = impseq(-5,5,0);
>> stem(n,x)
>> [y,n] = sigshift(x,n,2);
>> stem(n,y)
```

4. Folding In this operation each sample of x(n) is flipped around n = 0 to obtain a folded sequence y(n).

$$y[n] = \{x[n]\}$$

In MATLAB this operation is implemented by **fliplr(x)** function for sample values and by **-fliplr(n)** function for sample positions as shown in the **sigfold** function.

```
function [y,n] = sigfold(x,n)
% ------------------------------
% [y,n] = sigfold(x,n)
% implements y(n) = x(-n)

y = fliplr(x);
n = -fliplr(n);
```

For example, to generate $x(n) = \delta[n + 2]$; -5 ≤ n ≤ 5, we can also write the following MATLAB script:

```
>> [x,n] = impseq(-5,5,2);
>> stem(n,x)
>> [y,n] = sigfold(x,n);
>> stem(n,y)
```

5. **Sample summation**. This operation differs from signal addition operation. It adds all sample values of x(n) between n_1 and n_2.

$$\sum_{n=n_1}^{n_2} x(n) = x(n_1) + ... + x(n_2)$$

It is implemented by the **sum(x(n1:n2))** function.

6. **Sample Product:** This operation also differs from signal multiplication operation. It multiplies all sample values of x(n) between n_1 and n_2.

$$\prod_{n=n_1}^{n_2} x(n) = x(n_1) \times ... \times x(n_2)$$

It is implemented by the **prod(x(n1:n2))** function.

7. **Signal energy.** The energy of a sequence x(n) is given by

$$\mathcal{E}_x = \sum_{-\infty}^{\infty} x(n)x^*(n) = \sum_{-\infty}^{\infty} |x(n)|^2$$

where superscript * denotes the operation of complex conjugation. The energy of a finite-duration sequence x(n) can be computed in MATLAB using

```
>> Ex = sum(x .* conj(x)); % one approach
>> Ex = sum(abs(x) .^ 2); % another approach
```

Main Lab

2A: Generate and plot each of the following sequences over the indicated interval.

a) $\mathrm{x[n]} = 3\delta[\mathrm{n}-3] - \delta[\mathrm{n}-6], \quad 0 \le n \le 10$

b) $\mathrm{x[n]} = \mathrm{n}\{\mathrm{u[n]} - \mathrm{u[n}-10]\} + 10e^{-0.3(n-10)}\{u[n-10] - u[n-20]\}, \; 0 \le n \le 20$

c) $\tilde{x}[n] = \{... 5, 4, 3, 2, 1, \underset{\uparrow}{5}, 4, 3, 2, 1, 5, 4, 3, 2, 1, ...\} \qquad -10 \le n \le 9$

2B: Let x(n) = $\{-1, 2, \underset{\uparrow}{3}, 4, 5, 6, 7, 6, 5, 4, 3, -2, 1\}$ Determine and plot the following sequences

a) $x_1[n] = 2x[n-5] - 3x[n+4]$

b) $x_2[n] = 4x[5-n] - 3x[n + last\ digit\ of\ your\ student\ ID]$

2C: Generate the complex-valued signal

$$x[n] = e^{(0.1+0.3j)n} \qquad -10 \le n \le 10$$

and plot its magnitude, phase, the real part, and the imaginary part in four separate subplots.

Rubrics

Rubrics	No participation (0 points)	Average (1 point)	Excellent (2 points)
R1: Attendance and Ethics	T> 15min Or Absent	07<T<15 min	T< 7min
R3: Results	Absent or no manual submitted	Axis label or title of graph not defined	printscreen or snippet tool used (Code not visible)

	No participation (0 points)	Unsatisfactory (1 point)	Poor (2 points)	Fair (3 points)	Excellent (4 points)
R2: Code Function	Absent or Matlab Code not implemented	Most of the lab task not implemented	Major flaws in code function	Fair amount of code implemented	Most of the code implemented with minor flaws
	Good (5 points)	**Excellent (6 points)**			
R2: Code Function	All task implemented but after rectification from instructor	All task implemented with proper understanding			

Rule violation To be observed each time	Any rule violation would be given -1 mark each time and would be deducted from the total marks earned.

R1 and R2 would be evaluated individually for each student

R3 would be evaluated in terms of group