

Digital Signal Processing using MATLAB

2. DISCRETE-TIME SIGNALS & SYSTEMS

Brooks/Cole
Vinay K. Ingle
John G. Proakis

DISCRETE-TIME SIGNALS

Signals	analog $x_a(t)$	Variable t is can represent any physical quantity. represents time in second.
	discrete $x(n)$	Variable n is integer valued and represents discrete instance in time

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

↑

↑: *Sample at $n = 0$*

Using MATLAB

- Represent a finite_duration sequence by a row vector of appropriate values.
- Ex)

math	$x(n) = \{2, 1, -1, 0, 1, 4, 3, 7\}$ \uparrow
MATLAB	<pre>>> n = [-3, -2, -1, 0, 1, 2, 3]; >> x = [2, 1, -1, 0, 1, 4, 3, 7];</pre>

TYPE OF SEQUENCE I

- 1. Unit sample sequence

$$\delta(n) = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases} = \left\{ \dots, 0, 0, 1, 0, 0, \dots \right\} \begin{matrix} \uparrow \\ \text{일반화} \end{matrix} \rightarrow \delta(n - n_0) = \begin{cases} 1, & n = n_0 \\ 0, & n \neq n_0 \end{cases} \text{ for } n_1 < n_0 \leq n_2$$

```
function [x,n] = impseq(n0,n1,n2)
% Generates x(n) = delta(n-n0); n1 <= n,n0 <= n2
% -----
% [x,n] = impseq(n0,n1,n2)
%
% if ((n0 < n1) | (n0 > n2) | (n1 > n2))
%     error('arguments must satisfy n1 <= n0 <= n2')
% end
n = [n1:n2]; x = [(n-n0) == 0];
```

참고)
zeros(1,N)
 Generates a row vector of N zeros

TYPE OF SEQUENCE II

2. Unit step sequence

$$u(n) = \begin{cases} 1, & n \geq 0 \\ 0, & n < 0 \end{cases} = \begin{cases} \dots, 0, 0, 1, 1, 1, \dots \\ \uparrow \\ \text{일반화} \end{cases} \quad \rightarrow \quad u(n-n_0) = \begin{cases} 1, & n \geq n_0 \\ 0, & n < n_0 \end{cases}$$

for $n_1 < n_0 \leq n_2$

```
function [x,n] = stepseq(n0,n1,n2)
% Generates x(n) = u(n-n0); n1 <= n, n0 <= n2
% -----
% [x,n] = stepseq(n0,n1,n2)
%
if ((n0 < n1) | (n0 > n2) | (n1 > n2))
    error('arguments must satisfy n1 <= n0 <= n2')
end
n = [n1:n2]; x = [(n-n0) >= 0];
```

참고)
ones(1,N)
Generates a row vector of N ones

TYPE OF SEQUENCE III

3. Real-valued exponential sequence :

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

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

○ “.”[^] : implement a real exponential sequence

4. Complex-valued exponential sequence :

$$x(n) = e^{(\sigma + jw_0)n}, \forall n \quad \text{ex) } x(n) = \exp[(2 + j3)n], 0 \leq n \leq 10$$

σ : attenuation w_0 : frequency in radians

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

○ **exp** is used to generate exponential sequence

TYPE OF SEQUENCE IV

- 5. *Sinusoidal sequence* :

$$x(n) = \cos(\omega_0 n + \theta), \forall n$$

$$\text{ex) } x(n) = 3 \cos(0.1\pi n + \pi/3) + 2 \sin(0.5\pi n), 0 \leq n \leq 10$$

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

- cos(or sin) : generate sinusoidal sequence

- 6. *Random sequence* :

- **rand(1,N)** : generates a length N random sequence whose elements uniformly distributed between [0,1]

- **randn(1,N)** : generates a length N Gaussian random sequence with mean 0 and variance 1

TYPE OF SEQUENCE V

- 7. *Periodic sequence* :

$$\text{if) } x(n) = x(n + N), \forall n$$

then) x(n) is periodic

- **Smallest integer N** that satisfies the above relation is called ***fundamental period***.

$\tilde{x}(n)$: *periodic sequence*

P periodics of $\tilde{x}(n)$: one period $\{x(n)\}$ 을 P번 복사

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

TYPE OF SEQUENCE VI

- (:) : construct
- ' : matrix transposition operator

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

OPERATIONS ON SEQUENCES I

• 1. Signal addition :

$$\{x_1(n)\} + \{x_2(n)\} = \{x_1(n) + x_2(n)\}$$

- Sample-by-sample addition
- “+”
- The lengths of $x_1(n)$ and $x_2(n)$ must be the same
- Required to make $x_1(n)$ and $x_2(n)$ of equal length
 - &, <=, ==, find
- Implement “sigadd”



- Implement “sigadd”

```
function [y,n] = sigadd(x1,n1,x2,n2)
% implements y(n) = x1(n)+x2(n)
% -----
% [y,n] = sigadd(x1,n1,x2,n2)
% 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
```

OPERATIONS ON SEQUENCES II

- 2. *Signal multiplication* :

$$\{x_1(n)\} \cdot \{x_2(n)\} = \{x_1(n) \cdot x_2(n)\}$$

- Sample-by-sample multiplication (or “dot” multiplication)
- “**.***”
- Similar restrictions apply for the “**.***” as for the “**+**”



- Implement “sigmult”

```
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 = ones(1,length(n)); y2 = y1; 가 아닐까?
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
```

OPERATIONS ON SEQUENCES III

- 3. *Scaling* :

$$\alpha \{x(n)\} = \{\alpha x(n)\}$$

- Each sample is multiplied by a scalar α
- “*”

- 4. *shifting* :

$$y(n) = \{x(n - k)\}$$

- Each sample of $x(n)$ is shifted by an amount k to obtain a shifted sequence $y(n)$

$$\text{if let } m = n - k$$

$$n = m + k$$

$$\text{then } y(m + k) = \{x(m)\}$$



- Implement “sigshift”

```
function [y,n] = sigshift(x,m,n0)
% implements y(n) = x(n-n0)
% -----
% [y,n] = sigshift(x,m,n0)
%
n = m+n0; y = x;
```

OPERATIONS ON SEQUENCES IV

- 5. *Folding* :

$$y(n) = \{-x(n)\}$$

- Each sample of $x(n)$ is flipped around $n=0$ to obtain a folded sequence $y(n)$
- Implement “sigfold” by `fliplr(x)` & `-fliplr(x)`

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

OPERATIONS ON SEQUENCES V

- 6. *sample summation* :

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

- Adds all sample values of $x(n)$ between n_1 and n_2 .
- Implement by “ **sum(x(n21:n2))** ”

- 7. *sample products* :

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

- Multiplies all sample values of $x(n)$ between n_1 and n_2 .
- Implement by “ **prod(x(n21:n2))** ”

OPERATIONS ON SEQUENCES VI

- 8. *Signal energy* :

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

- Superscript * denotes the operation of complex conjugation
- Implement by

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

- 9. *signal power* :

$$P_x = \frac{1}{N} \sum_0^{N-1} |x(n)|^2$$

- Average power of a periodic sequence with fundamental period N

SOME USEFUL RESULTS I

- Unit sample synthesis

$$x(n) = \sum_{k=n_1}^{n_2} x(k)\delta(n-k)$$

- Arbitrary sequence $x(n)$ can be synthesized as a weighted sum of delayed and scaled unit sample sequences.

- Even and odd synthesis

$$\left\{ \begin{array}{l} \text{if } x_e(-n) = x_e(n) \\ \text{then } x_e(n) \text{ is called even (symmetric)} \end{array} \right. \quad \left\{ \begin{array}{l} \text{if } x_o(-n) = -x_o(n) \\ \text{then } x_o(n) \text{ is called odd (antisymmetric)} \end{array} \right.$$

- Any arbitrary real-valued sequence $x(n)$ can be decomposed into its **even** and **odd** component

$$x(n) = x_e(n) + x_o(n)$$

$$x_e(n) = \frac{1}{2}[x(n) + x(-n)], \quad x_o(n) = \frac{1}{2}[x(n) - x(-n)]$$

- Implement “**evenodd**”

```
function [xe, xo, m] = evenodd(x,n)
% Real signal decomposition into even and odd parts
% -----
% [xe, xo, m] = evenodd(x,n)
%
if any(imag(x) ~= 0)
    error('x is not a real sequence')
end
m = -fliplr(n);
m1 = min([m,n]); m2 = max([m,n]); m = m1:m2;
nm = n(1)-m(1); n1 = 1:length(n);
x1 = zeros(1,length(m));
x1(n1+nm) = x; x = x1;
xe = 0.5*(x + fliplr(x));
xo = 0.5*(x - fliplr(x));
```

SOME USEFUL RESULTS II

- The geometric series

- A one-sided exponential sequence of the form

$$\{\alpha^n, n \geq 0\}, \quad \alpha \text{ is arbitrary constant} \quad \sum_{n=0}^{\infty} \alpha^n \rightarrow \frac{1}{1-\alpha}, \quad \text{for } |\alpha| < 1$$

- Correlation of sequences

$$\sum_{n=0}^{N-1} \alpha^n \rightarrow \frac{1-\alpha^N}{1-\alpha}, \quad \text{for } \forall \alpha$$

- Measure of the degree to which two sequences are similar

- Crosscorrelation

- If) $x(n), y(n)$ is finite energy

- Then) crosscorrelation of $x(n), y(n)$

$$r_{x,y}(l) = \sum_{n=-\infty}^{\infty} x(n)y(n-l), \quad l: \text{shift or lag parameter}$$

- Autocorrelation

- Special case of equation(2.6) when $y(n)=x(n)$

$$r_{x,x}(l) = \sum_{n=-\infty}^{\infty} x(n)x(n-l)$$

DISCRETE SYSTEMS I

- LINER SYSTEMS

$$L[a_1x_1(n) + a_2x_2(n)] = L[a_1x_1(n)] + L[a_2x_2(n)], \quad \forall a_1, a_2, x_1(n), x_2(n)$$

If $L[.]$ satisfies the principle of superposition

$$y(n) = L[x(n)] = L\left[\sum_{k=-\infty}^{\infty} x(k)\delta(n-k)\right] = \sum_{k=-\infty}^{\infty} x(k)L[\delta(n-k)]$$

$L[\delta(n-k)]$ is impulse response

Denoted by $h(n, k)$

$$\therefore y(n) = \sum_{k=-\infty}^{\infty} x(k)h(n, k)$$

- Linear time-invariant (LTI) system

- A linear system in which an input-output pair, $x(n)$ and $y(n)$, is invariant to a shift n in time

$$x(n) \rightarrow \boxed{} \rightarrow y(n) \rightarrow \boxed{} \rightarrow y(n-k)$$

$$\bullet x(n) \rightarrow \boxed{} \rightarrow x(n-k) \rightarrow \boxed{} \rightarrow y(n-k)$$

DISCRETE SYSTEMS II

- LTI system : use the LTI[.] operator

$$y(n) = LTI[x(n)] = \sum_{k=-\infty}^{\infty} x(k)h(n-k)$$

$$y(n) \cong x(n) * h(n)$$

- Impulse response of an LTI system is given by $h(n)$

$$x(n) \rightarrow \boxed{} \rightarrow y(n) = x(n) * h(n)$$

- *Stability*

- BIBO stable : bounded – input bounded – output stable

$$|x(n)| < \infty \Rightarrow |y(n)| < \infty, \quad \forall x, y$$

- *Causality*

- If the output at index n_0 depends only on the input up to and including the index n_0
- Output does not depend on the future values of the input

CONVOLUTION I

- EXAMPLE 2.5 (식에 의한 해석)
 - Rectangular pulse $x(n)=u(n)-u(n-10)$
 - Impulse response $h(n)=(0.9)^n u(n)$
 - $y(n)=x(n)*h(n)$?
- Sol)

$$y(n) = \sum_{k=0}^9 (1)(0.9)^{(n-k)} u(n-k) = (0.9)^n \sum_{k=0}^9 (0.9)^{-k} u(n-k)$$

i) $n < 0$: then $u(n-k) = 0, \quad 0 \leq k \leq 9$

$$y(n) = 0$$

ii) $0 \leq n \leq 9$: then $u(n-k) = 1, \quad 0 \leq k \leq n$

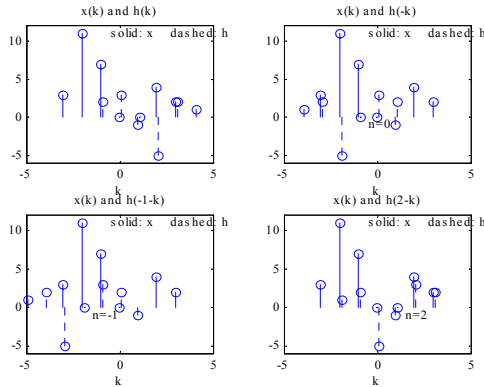
$$y(n) = (0.9)^n \sum_{k=0}^n (0.9)^{-k} = (0.9)^n \sum_{k=0}^n [(0.9)^{-1}]^k = (0.9)^n \frac{1 - (0.9)^{-(n+1)}}{1 - (0.9)^{-1}} = 10[1 - (0.9)^{n+1}], \quad 0 \leq n \leq 9$$

iii) $n \geq 9$: then $u(n-k) = 1, \quad 0 \leq k \leq 9$

$$y(n) = (0.9)^n \sum_{k=0}^9 (0.9)^{-k} = (0.9)^n \frac{1 - (0.9)^{-10}}{1 - (0.9)^{-1}} = 10(0.9)^{n-9}[1 - (0.9)^{10}], \quad n \geq 9$$

CONVOLUTION II

- EXAMPLE 2.6 (그림에 의한 해석)
 - $x(n)=[3,11,7,0,-1,4,2]$, $-3 \leq n \leq 3$
 - $h(n)=[2,3,0,-5,2,1]$, $-1 \leq n \leq 4$
 - $y(n)=x(n)*h(n)$?
- Sol)



CONVOLUTION III

MATLAB IMPLEMENTATION

```
>> x=[3,11,7,0,-1,4,2];
>> h=[2,3,0,-5,2,1];
>> y=conv(x,h)
y={6,31,47,6,-51,-5,41,18,-22,-3,8,2}
```

- Make the “conv_m”
 - [conv + (beginning point, end point)]
 - If) given finite duration

$$\{x(n) : n_{xb} \leq n \leq n_{xe}\} \text{ and } \{h(n) : n_{hb} \leq n \leq n_{he}\}$$
 - Then)

$$n_{yb} = n_{xb} + n_{hb}, \quad n_{ye} = n_{xe} + n_{he}$$



○ Implement “conv_m”

```
function [y,ny] = conv_m(x,nx,h,nh)
% Modified convolution routine for signal processing
% -----
% [y,ny] = conv_m(x,nx,h,nh)
% y = convolution result
% ny = support of y
% x = first signal on support nx
% nx = support of x
% h = second signal on support nh
% nh = support of h
%
nyb = nx(1)+nh(1); nye = nx(length(x)) + nh(length(h));
ny = [nyb:nye];
y = conv(x,h);
```

CONVOLUTION IV

- SEQUENCE CORRELATIONS REVISITED
 - Crosscorrelation $r_{yx}(l)$
 - $r_{yx}(l)=y(l)*x(-l)$ cf)P20 식(2.6)
 - autocorrelation $r_{xx}(l)$
 - $r_{xx}(l)=x(l)*x(-l)$ cf)P20 식(2.7)
- EXAMPLE 2.8
 - **Let** $x(n)=[3,11,7,0,-1,4,2]$, $y(n)=x(n-2)+w(n)$
 - $w(n)$ is Gaussian sequence, mean 0, variance 1
 - Crosscorrelation $r_{yx}(l)$?
 - **Sol**
 - P28 참고
 - Crosscorrelation indeed peaks at $l=2$
 - $y(n)$ is similar to $x(n)$ shifted by 2

DIFFERENCE EQUATIONS I

- Linear constant coefficient difference equation

$$\sum_{k=0}^N a_k y(n-k) = \sum_{m=0}^M b_m x(n-m), \quad \forall n$$

$$y(n) = \sum_{m=0}^M b_m x(n-m) - \sum_{k=1}^N a_k y(n-k)$$

- Solution to this equation

$$y(n) = y_H(n) + y_P(n)$$

- $y_H(n)$: homogeneous part of the solution

$$y_H(n) = \sum_{k=1}^N c_k z_k^n \quad z_k, k=1,2,\dots,N \text{ are } N \text{ root(natural frequency)}$$

$$\sum_{k=0}^N a_k z^k = 0$$

- This characteristic equation is important in determining the stability of system

DIFFERENCE EQUATIONS II

- If the roots z_k satisfy the condition

$$|z_k| < 1, \quad k=1,\dots,N$$

- Then a causal system described by (2.19) is stable

- $y_P(n)$: particular part of the solution determined from the right-hand side of (2.18)

- MATLAB IMPLEMENTATION

- Filter

- Filter is available to solve difference equations numerically, given the input and the difference equation coefficients

$$y = \text{filter}(b,a,x)$$

where

$$b = [b_0, b_1, \dots, b_M]; \quad a = [a_0, a_1, \dots, a_N]; \quad \text{length}(y) = \text{length}(x)$$

a, b : coefficient array in (2.18)

x : input sequence array

$a_0 \neq 0$



● EXAMPLE 2.9

$$y(n] - y(n-1) + 0.9y(n-2) = x(n) ; \forall n$$

- A. calculate and plot the impulse response $h(n)$ at $n=-20, \dots, 100$.
- B. calculate and plot the unit step response $h(n)$ at $n=-20, \dots, 100$.
- C. Is the system specified by $h(n)$ stable?

DIFFERENCE EQUATIONS III

● DIGITAL FILTER

- FIR filter (finite-duration Impulse response) filter

- If the unit impulse response of an LTI system is of finite duration
- $h(n) = 0$ for $n < n_1$ $n > n_2$
- Causal FIR filter

$$y(n) = \sum_{m=0}^M b_m x(n-m) , h(0) = b_0, \dots, h(M) = b_M \text{ all other } h(n) = 0$$

- **Nonrecursive** filter or **moving average** filter

- IIR filter (infinite-duration impulse response) filter

- the unit impulse response of an LTI system is of infinite duration
- Part of equation (2.18) is recursive filter

$$\sum_{k=0}^K a_k y(n-k) = x(n)$$