Discrete-Time Signal

```
% Code By: Ka Ming Lui
% Date: 10/11/2021
clear; close; clc;
```

Important Equations:

Discretized Signal

$$x[n] = \cos(\Omega n)$$

Period of the Discretized Signal:

$$T = \frac{F_s}{f}$$

OR

$$T = \frac{2\pi}{\Omega}$$

(In **samples per period**)

```
f_CT = 250;
Fs = 500;
T_spp_Fs_f = Fs/f_CT;
fprintf('T = %g samples/period (Using Frequency of CT Signal and Sampling Rate) \n', T_spp_Fs_f
```

```
T = 2 samples/period (Using Frequency of CT Signal and Sampling Rate)
```

```
Omega_T = 2*pi;
T_spp_Omega = 2*pi/Omega_T;
fprintf('T = %g samples/period (Using Ω) \n', T_spp_Omega);
```

```
T = 1 samples/period (Using Ω)
```

Corresponding Sampling Frequency

$$F_s = \frac{2\pi f}{\Omega}$$

```
Fs_CSF = 2*pi*f_CT/Omega_T;
fprintf('Fs = %g Hz \n', Fs_CSF);
```

```
Fs = 250 Hz
```

Duration of Continuous-Time Signal (Depends on f, Fs, Ω) [?]

First Zero Crossing:

$$f_{\text{first zero crossing}} = \frac{F_s}{2\pi} - f$$

```
first_0_cross = Fs_CSF/(2*pi) - f_CT;
fprintf('First Zero Crossing = %g Hz \n', first_0_cross);
```

```
First Zero Crossing = -210.211 Hz
```

$$\omega_{\text{first zero crossing}} = 2\pi f_{\text{first zero crossing}} = \frac{\pi}{a}$$

a is Half-Duration of the Signal **(In radians per second)**

```
a = pi/(2*pi*first_0_cross);
fprintf('The Signal Duration (2a) is = %g seconds. \n', 2*a);
```

```
The Signal Duration (2a) is = -0.00475712 seconds.
```

Discrete-Time Frequency:

$$\Omega = \frac{2\pi f}{F_s}$$

f is **Frequency of Continuous Time Signal (In Hz)**

F_s is **Sampling Rate (In Hz)**

OR

$$\Omega = \frac{2\pi}{T}$$

```
Omega = 2*pi/T_spp_Fs_f;
if Omega <= 2*pi
    fprintf('Ω = %gπ radians/sample \n', Omega/pi);
else
    fprintf('Ω = %gπ radians/sample \n', Omega/pi - 2*floor(Omega/(2*pi)));
end
```

```
Ω = 1π radians/sample
```

Nyquist Theorem:

$$F_s \geq 2f$$

Or else the result will be **aliasing**.

```
if Fs >= 2*f_CT
```

```
    fprintf('No Aliasing \n');
else
    fprintf('Aliasing \n');
end
```

```
No Aliasing
```

Z-Transform:

$$H(z) = \frac{\text{Multiplication of ALL zero polynomials}}{\text{Multiplication of ALL pole polynomials}}$$

$$1 - 2\alpha z^{-1} + (\alpha^2 + \beta^2)z^{-2} = 0$$

(In the case of 2 poles and 2 zeros, with each pole has a conjugate)

```
% Defining the poles and zeros observed from the figure
pole_1 = 0.6 + 0.4j;
pole_2 = -0.7 + 0.5j;
zero_1 = 0.9 + 0j;
zero_2 = -0.9 + 0j;

pole_1 = [pole_1 conj(pole_1)];
pole_2 = [pole_2 conj(pole_2)];
pole = [pole_1; pole_2];
zero = [zero_1; zero_2];

% Finding Poles and Zeros Polynomial
pp1 = poly(pole(1, :));
pp2 = poly(pole(2, :));
pz1 = poly(zero(1, :));
pz2 = poly(zero(2, :));
syms z
H_z_numerator = conv(pz1, pz2);
H_z_denominator = conv(pp1, pp2);
H_z = poly2sym(H_z_numerator, z)/poly2sym(H_z_denominator, z);
fprintf('H(z) = ');
```

```
H(z) =
```

```
disp(H_z);
```

$$\frac{z^2 - \frac{81}{100}}{z^4 + \frac{z^3}{5} - \frac{21\,z^2}{50} - \frac{4\,z}{25} + \frac{481}{1250}}$$

When Ω, Gain (From the Z-Transform):

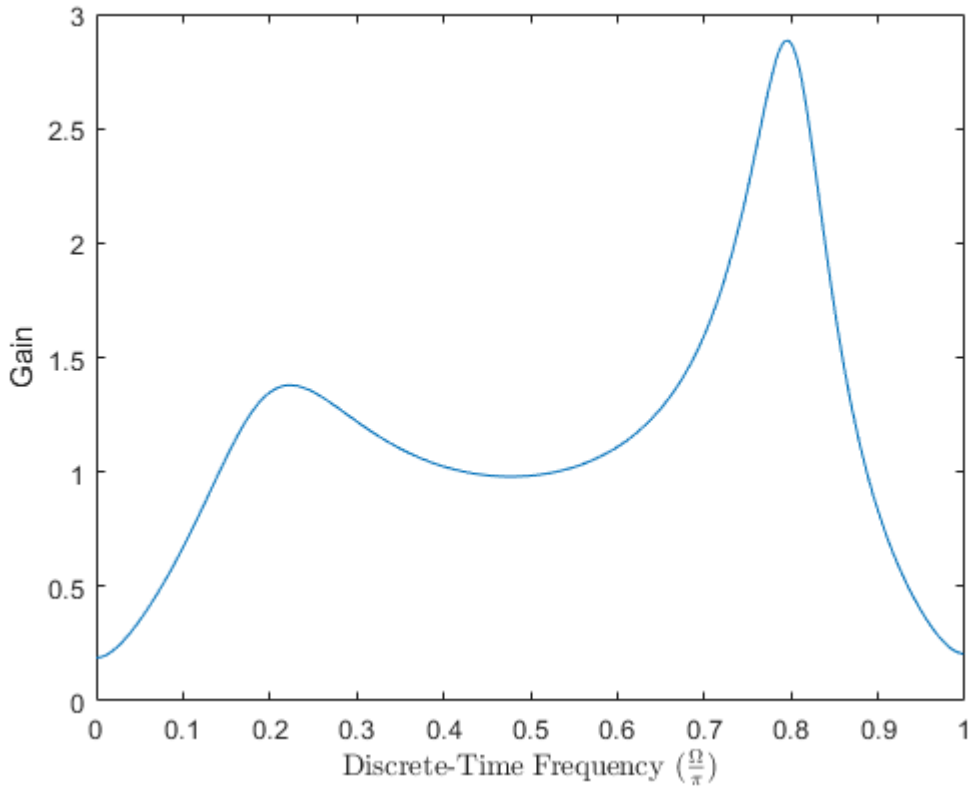
```
Omega_gain = 1;

pz = poly(zero(:));
```

```
pp = poly(pole(:));
[H_z_plot, Omega_plot] = freqz(pz, pp);
gain_plot = abs(H_z_plot);

position_gain = round(1 + Omega_gain/(length(gain_plot) - 1)^-1);
gain_Omega = gain_plot(position_gain);
fprintf('When Ω/π = %g, the gain would be %g (%gdB)', Omega_gain, gain_Omega, 20*log10(gain_Ome
```

```
When Ω/π = 1, the gain would be 0.205816 (-13.7304dB)
```

Bode Plot of the System:

```
plot(Omega_plot/pi, gain_plot);
ylabel('Gain');
xlabel('Discrete-Time Frequency ($\frac{\Omega}{\pi}$)', 'interpreter', 'latex');
```



IIR Circuit:

Poles coefficient **[back]**, Zeros coefficient **[front]**, the coefficient point straight toward y[n] **[next]**.

$y[n] = x[n] + \mathrm{bx}[n-1] + \mathrm{ay}[n-1]\ldots$

a is back, b is front

```
p_size_array = [length(pp1), length(pp2), length(pz1), length(pz2)];
max_size_array = max(p_size_array);
```

```
% Only for display purpose
pp1_u = zeros(1, max_size_array); pp1_u(1:length(pp1)) = pp1(:)';
pp2_u = zeros(1, max_size_array); pp2_u(1:length(pp2)) = pp2(:)';
pz1_u = zeros(1, max_size_array); pz1_u(1:length(pz1)) = pz1(:)';
pz2_u = zeros(1, max_size_array); pz2_u(1:length(pz2)) = pz2(:)';

fprintf('First Module (From First Set of Zero and Pole): \n(1st Level) \t\t\tTo Next: %g \n(2nd
```

```
First Module (From First Set of Zero and Pole):
(1st Level)    To Next: 1
(2nd Level) Back: 1.2   Front: -0.9
(3rd Level) Back: -0.52  Front: 0
```

```
fprintf('Second Module (From Second Set of Zero and Pole): \n(1st Level) \t\t\tTo Next: %g \n(2
```

```
Second Module (From Second Set of Zero and Pole):
(1st Level)    To Next: 1
(2nd Level) Back: -1.4   Front: 0.9
(3rd Level) Back: -0.74  Front: 0
```

Find y[n] From Multi-Levels FIR/IIR Single Module:

Make sure the array sizes of a and b are the same.

```
n_output = 4;
next = 1;
a = [-0.2 0.5];
b = [0 0];
level = 3 - 1;
x = [1];
x = [zeros(1, level), x, zeros(1, n_output-length(x))];
y = zeros(1, n_output+level);
for n = 1+level:n_output+level
    y(n) = next*x(n) + b(1)*x(n-1) + a(1)*y(n-1);
    if level >= 2
        for m = 2:level
            y(n) = y(n) + b(m)*x(n-m) + a(m)*y(n-m);
        end
    end
end
y = y(1+level:n_output+level);
fprintf(['y[n] = [', num2str(y(:).'), '] \n']);
```

```
y[n] = [1       -0.2       0.54      -0.208]
```

When Ω, Gain (From FIR/IIR):

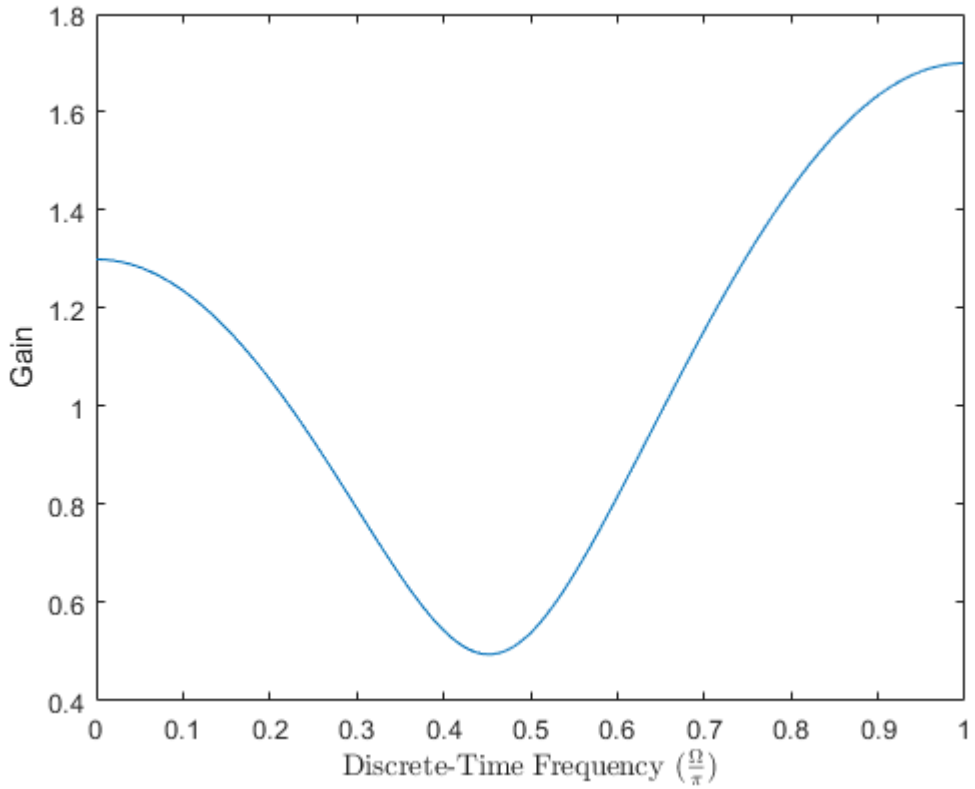
```
Omega_gain_IIR = 0.8;

pz_new_IIR = [next, a];
pp_new_IIR = [next, -b];
[H_z_plot_new_IIR, Omega_plot_new_IIR] = freqz(pz_new_IIR, pp_new_IIR);
```

```
H_mag_new_IIR = abs(H_z_plot_new_IIR);
H_ang_new_IIR = rad2deg(angle(H_z_plot_new_IIR));

position_gain = round(1 + Omega_gain_IIR/(length(H_mag_new_IIR) - 1)^-1);
gain_H_IIR = H_mag_new_IIR(position_gain);
phase_H_IIR = H_ang_new_IIR(position_gain);
fprintf('Gain at Ω/π = %g is <strong>%.3f∠%.2f° (%.2fdB)<strong>', Omega_gain_IIR, gain_H_IIR,
```

Gain at Ω/π = 0.8 is $1.441\angle 24.38^\circ$ **(3.17dB)**

Bode Plot of the System:

```
plot(Omega_plot_new_IIR/pi, H_mag_new_IIR);
ylabel('Gain');
xlabel('Discrete-Time Frequency ($\frac{\Omega}{\pi}$)', 'interpreter', 'latex');
```



From the Given H(z):

Determine whether the filter is stable or unstable:

```
H_z_denominator_new = [1 -2 0.5];
poles = roots(H_z_denominator_new);
stable = sum(abs(poles) < 1) == length(poles);

if stable
    fprintf('Stable \n');
```

```
else
    fprintf('Unstable \n');
end
```

```
Unstable
```

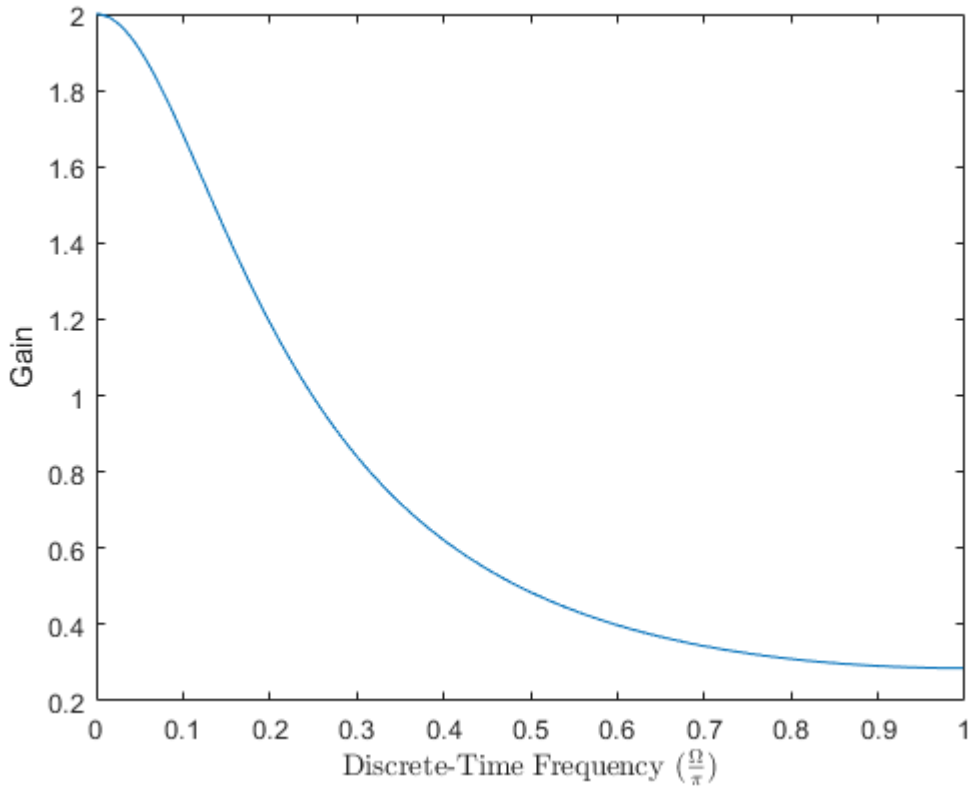
When Ω, Gain (From the Given H(z)):

```
H_z_numerator_new = [1];
Omega_gain_H_z = 1;

pz_new = H_z_numerator_new;
pp_new = H_z_denominator_new;
[H_z_plot_new_H_z, Omega_plot_new_H_z] = freqz(pz_new, pp_new);
gain_plot_new_H_z = abs(H_z_plot_new_H_z);

position_gain = round(1 + Omega_gain_H_z/(length(gain_plot_new_H_z) - 1)^-1);
gain_Omega = gain_plot_new_H_z(position_gain);
fprintf('When Ω/π = %g, the gain would be %g (%gdB)', Omega_gain_H_z, gain_Omega, 20*log10(gain
```

```
When Ω/π = 1, the gain would be 0.285716 (-10.8813dB)
```

Bode Plot of the System:

```
plot(Omega_plot_new_H_z/pi, gain_plot_new_H_z);
ylabel('Gain');
xlabel('Discrete-Time Frequency ($\frac{\Omega}{\pi}$)', 'interpreter', 'latex');
```



Discrete-Time Fourier Transform:

$$H(j\Omega) = \sum_{n} h[n] e^{-j\Omega n}$$

```
h_n = [1 -0.5 2.2];
Omega_DTFT = 1/3*pi;

fprintf(['h[n] = [' num2str(h_n(:).') '] \n']);
```

```
h[n] = [1       -0.5        2.2]
```

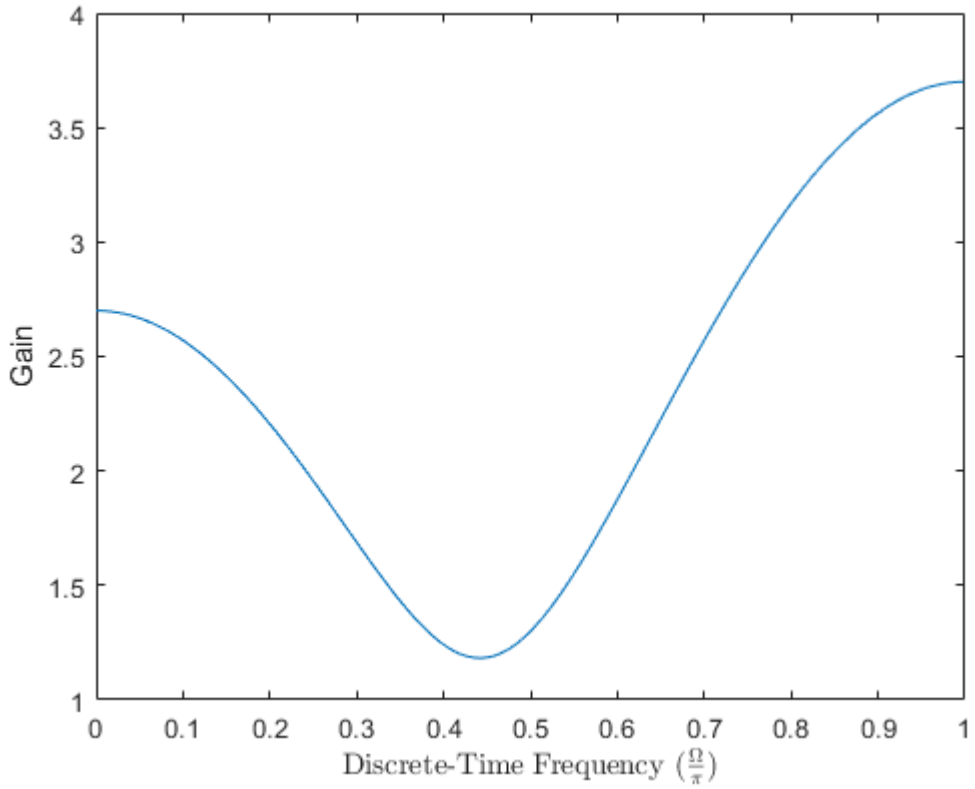
```
fprintf('Ω = %gπ radians/sample\n', Omega_DTFT/pi);
```

```
Ω = 0.333333π radians/sample
```

```
n_h_n = length(h_n);
H_Omega = 0; % Initial Value
for count_DTFT = 1:n_h_n
    H_Omega = H_Omega + h_n(count_DTFT)*exp(-1j*Omega_DTFT*(count_DTFT-1));
end
H_mag = abs(H_Omega);
H_ang = rad2deg(angle(H_Omega));
fprintf('Gain at Ω = %gπ is <strong>%.2f∠%.2f°</strong> \n', Omega_DTFT/pi, H_mag, H_ang);
```

```
Gain at Ω = 0.333333π is 1.51∠-103.37°
```

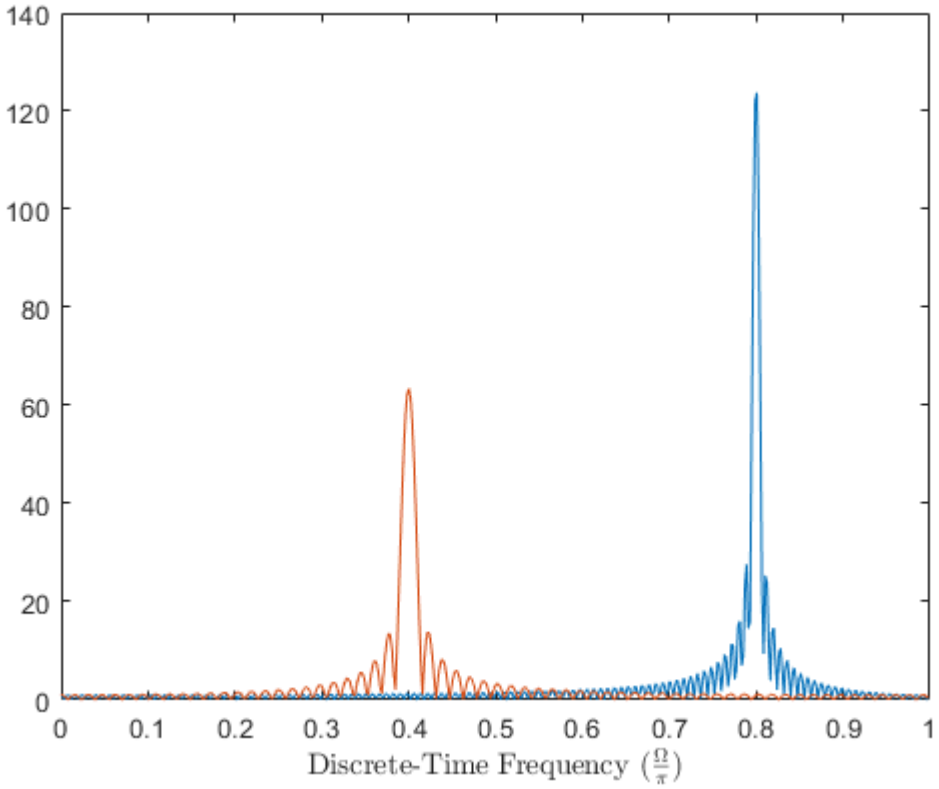
```
[X_plot_new_h_n, Omega_plot_new_h_n] = freqz(h_n);
plot(Omega_plot_new_h_n/pi, abs(X_plot_new_h_n));
ylabel('Gain');
xlabel('Discrete-Time Frequency ($\frac{\Omega}{\pi}$)', 'interpreter', 'latex');
```



Graph (From Ω):

Blue is x[n], Orange is y[n]

```
Omega_DTFT_plot = 0.8*pi;
k = 0:250;
x_for_DTFT = cos((Omega_DTFT_plot)*k); % Define x[n]
y_for_DTFT = x_for_DTFT(1:2:end); % Define y[n] by dropping every other sample
[X_DTFT, Omega_x] = freqz(x_for_DTFT); % Compute the DTFTs
[Y_DTFT, Omega_y] = freqz(y_for_DTFT);
plot_X = abs(X_DTFT);
plot_Y = abs(Y_DTFT);
plot(Omega_x/pi, plot_X, Omega_y/pi, plot_Y); % Plot the results
xlabel('Discrete-Time Frequency ($\frac{\Omega}{\pi}$)', 'interpreter', 'latex');
```




```
[~, I_X] = max(plot_X, [], 'linear');
[~, I_Y] = max(plot_Y, [], 'linear');
fprintf('Ω of x[n]: %.2fπ \n', I_X/length(X_DTFT));
```

```
Ω of x[n]: 0.80π
```

```
fprintf('Ω of y[n]: %.2fπ \n', I_Y/length(Y_DTFT));
```

```
Ω of y[n]: 0.40π
```

Discrete-Time Convolution:

$y[n] = x[n] * h[n]$

```
x_n = [1 -3 2];
y_n = conv(x_n, h_n);
fprintf(['h[n] = [', num2str(h_n(:).'), '] \n']);
```

```
h[n] = [1       -0.5        2.2]
```

```
fprintf(['y[n] = [', num2str(y_n(:).'), '] \n']);
```

```
y[n] = [1       -3.5        5.7       -7.6        4.4]
```

Discrete-Time Deconvolution (From x[n] and y[n], Find h[n]):

Long Division: y[n] over x[n]

```
x_n_deconvolution = [1 -3 2];
y_n_deconvolution = [1 -1 -4 4];
h_n_deconvolution = deconv(y_n_deconvolution, x_n_deconvolution);
fprintf(['h[n] = [', num2str(h_n_deconvolution(:).'), '] \n']);
```

```
h[n] = [1  2]
```