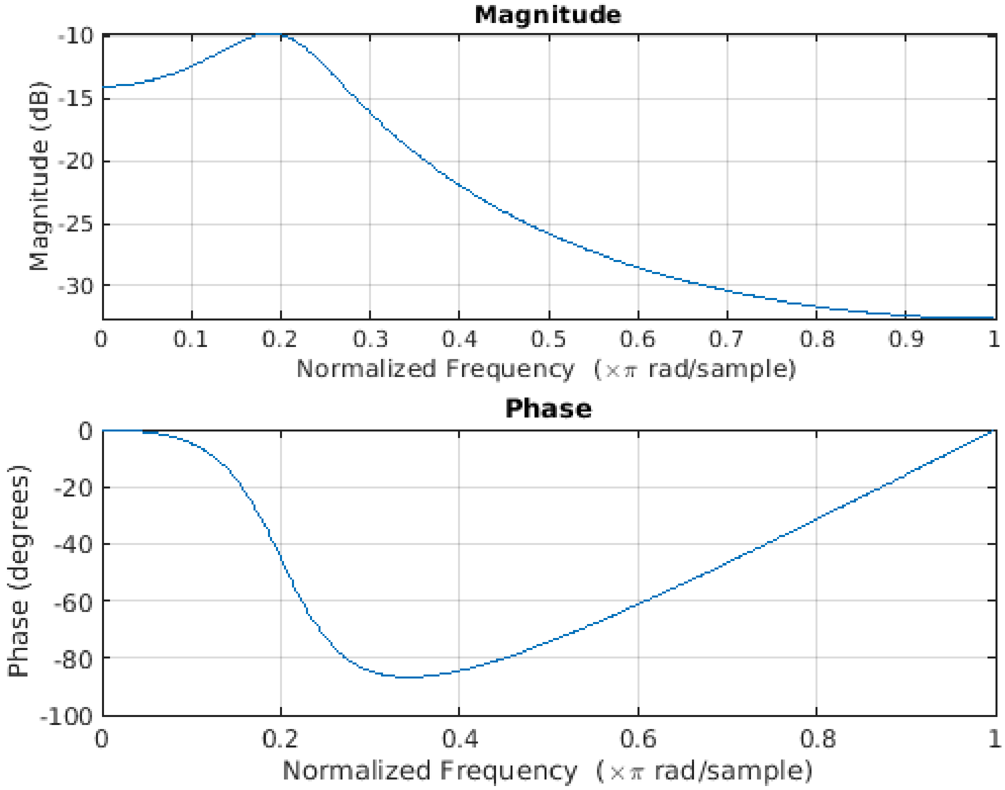
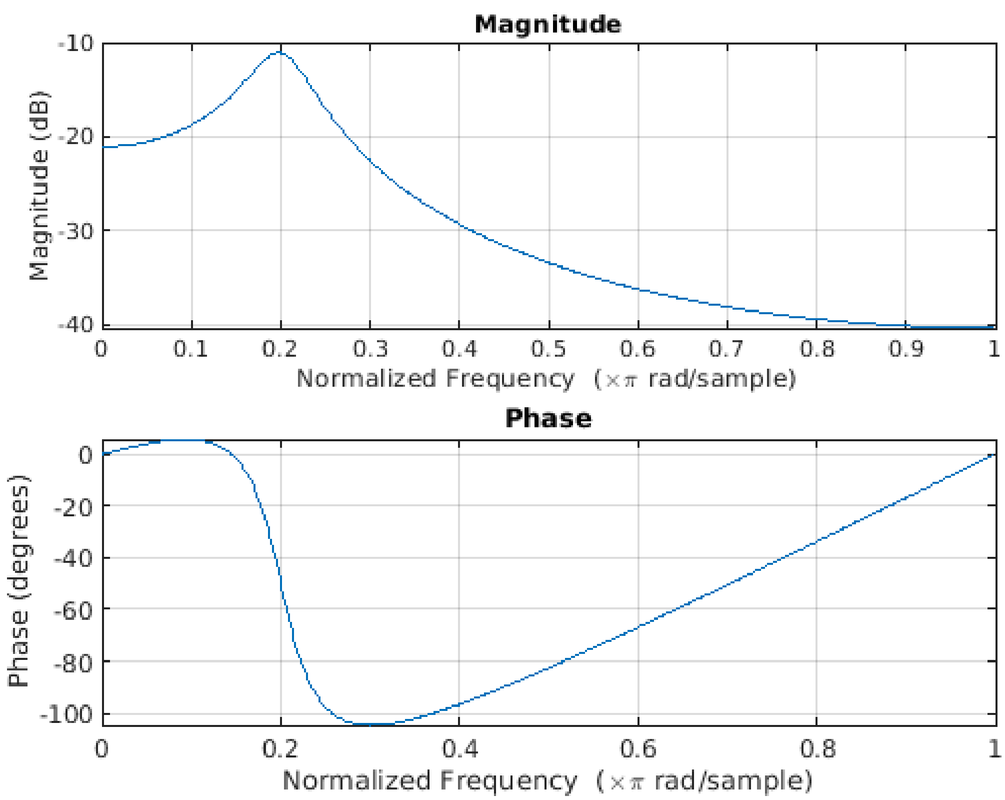
Table of Contents

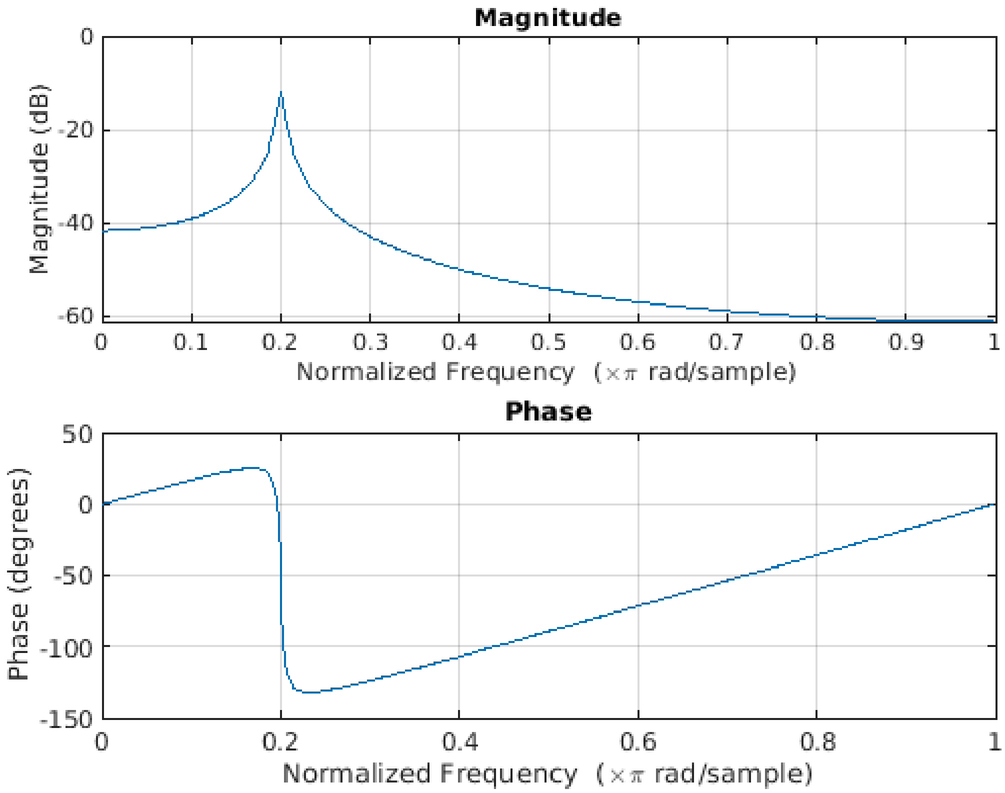

```
clear all;
close all;
clc;
```

3-1-a:

```
clear all;
close all;
clc;

n = 500;
f0 = 500;
fs = 10^4;

w = linspace(0, pi, n);
R = [0.8, 0.9, 0.99];

w0 = f0 * 2 * pi / fs;

for r=R
    % Defining filter params
    G = (1-r)*(1-2*r*cos(w0)+r^2)^.5;
    b = [G];
    a = [1, -2*r*cos(2*w0), r^2];

    % Plotting frequency response
    figure('Name', sprintf("R=%d", r));
    freqz(b, a);
end
```


Magnitude
Magnitude (dB)
-10
-15
-20
-25
-30
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Normalized Frequency (×π rad/sample)
Phase
Phase (degrees)
0
-20
-40
-60
-80
-100
0
0.2
0.4
0.6
0.8
1
Normalized Frequency (×π rad/sample)

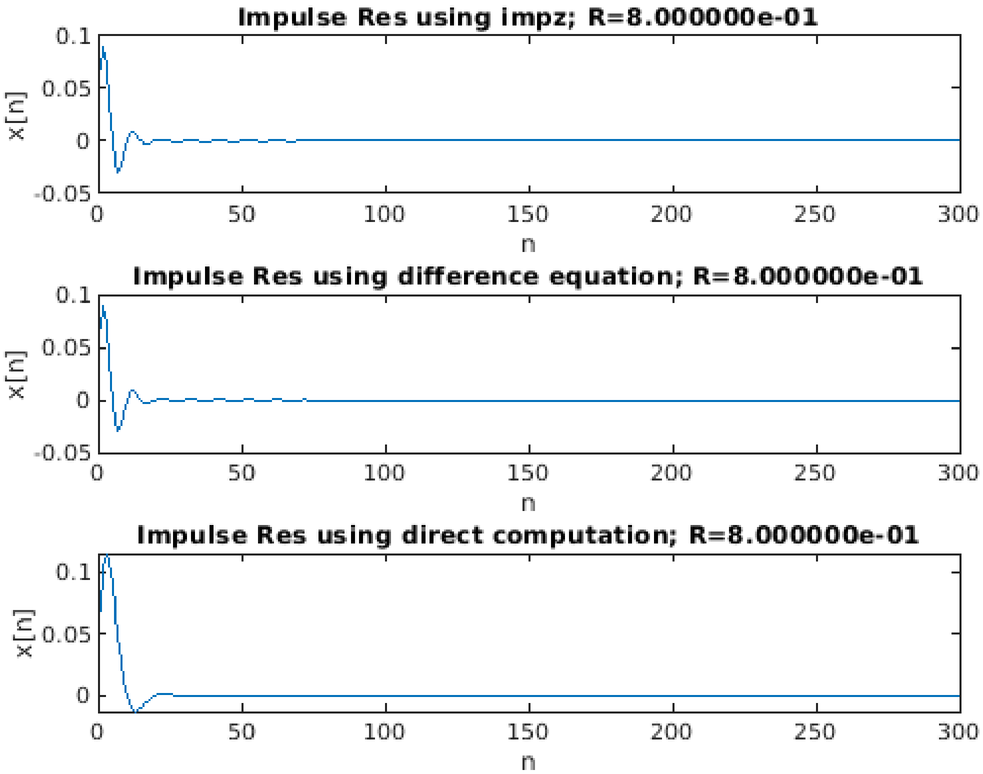

Magnitude
Magnitude (dB)
-10
-20
-30
-40
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Normalized Frequency (×π rad/sample)
Phase
Phase (degrees)
0
-20
-40
-60
-80
-100
0
0.2
0.4
0.6
0.8
1
Normalized Frequency (×π rad/sample)

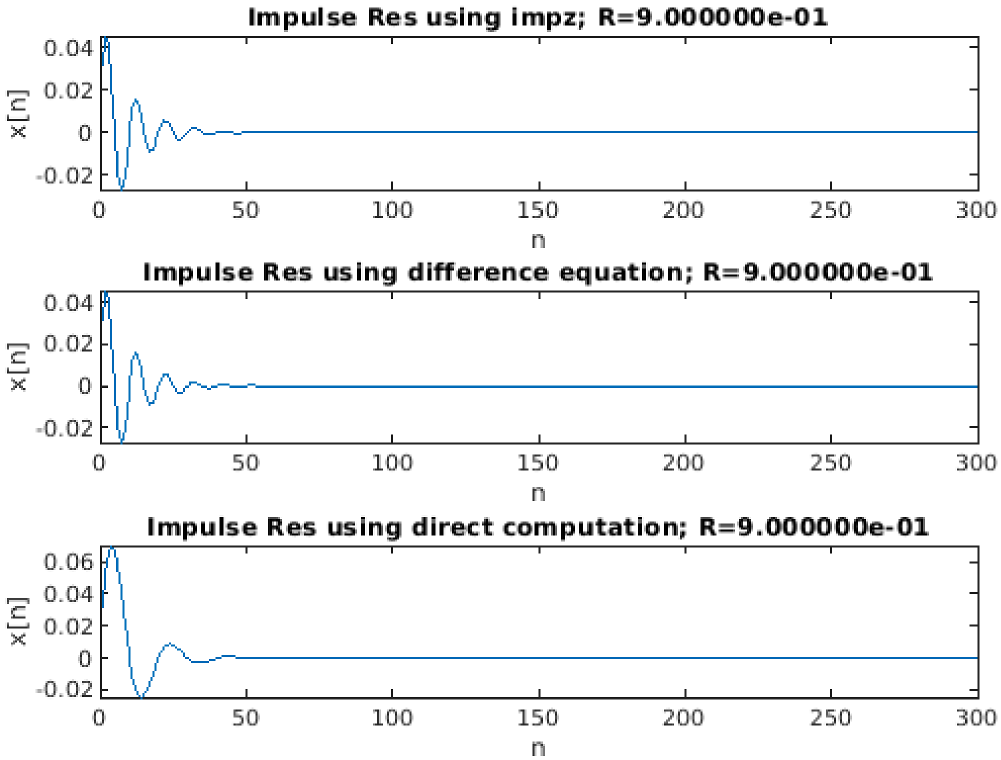


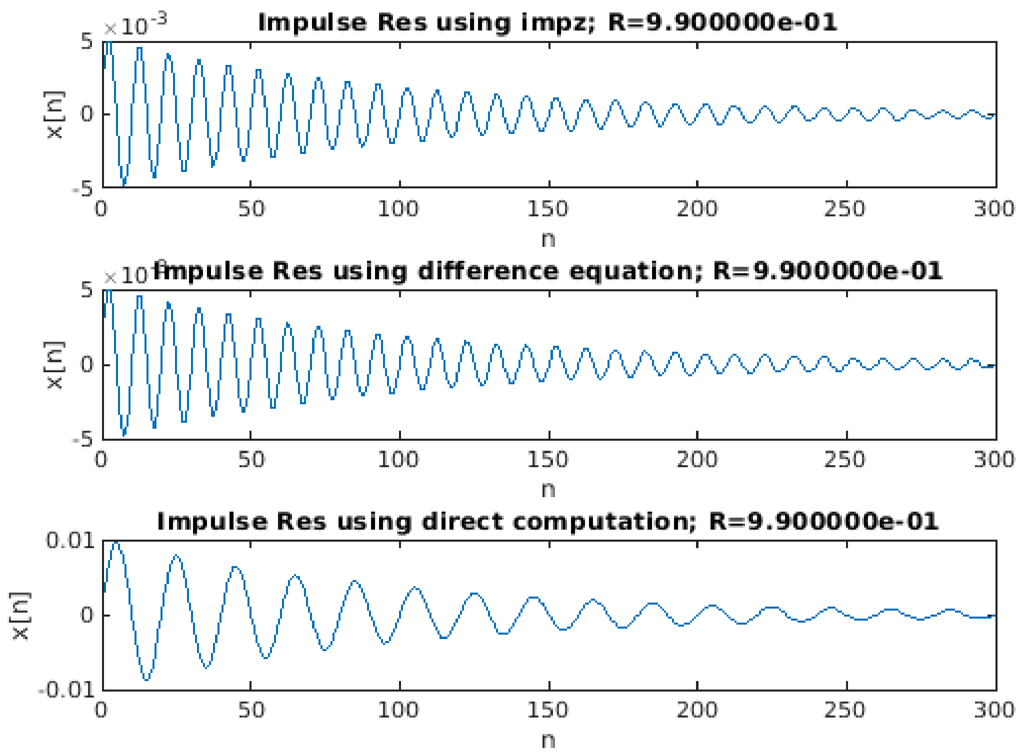
3-1-b:

```
clear all;
close all;
clc;

n = 300;
f0 = 500;
fs = 10^4;

w = linspace(0, pi, n);
R = [0.8, 0.9, 0.99];

w0 = f0 * 2 * pi / fs;
y = zeros(1,n);

for r=R
    % Defining filter params
    G = (1-r)*(1-2*r*cos(w0)+r^2)^0.5;
    b = [G];
    a = [1, -2*r*cos(2*w0), r^2];

    % Calculation directly

    % hn = zeros(1, n);
    % for k = 1:n
```

```
    %     hn(k) = (G/(sin(w0)))*(r^k)*sin(w0*(k + 1));
    % end
    t = 0:n-1;
    hn = (G/(sin(w0))).*r.^t.*sin(w0.*(t + 1));

    % Calculating using diff method
    y(1) = G;
    y(2) = -a(2)*y(1);
    for k = 3:n
        y(k) = -a(2)*y(k-1) - a(3)*y(k-2);
    end

    % Calculating using impz
    imp_res = impz(b, a, n);

    % Plotting results
    figure('Name', sprintf("R=%d", r));
    subplot(3,1,1);

    plot(imp_res);
    xlabel('n');
    ylabel('x[n]');
    title(sprintf("Impulse Res using impz; R=%d", r));

    subplot(3,1,2);
    plot(y);
    xlabel('n');
    ylabel('x[n]');
    title(sprintf("Impulse Res using difference equation; R=%d", r));

    subplot(3,1,3);
    plot(hn);
    xlabel('n');
    ylabel('x[n]');
    title(sprintf("Impulse Res using direct computation; R=%d", r));
end
```


Impulse Res using impz; R=8.000000e-01
x[n]
n
Impulse Res using difference equation; R=8.000000e-01
x[n]
n
Impulse Res using direct computation; R=8.000000e-01
x[n]
n


Impulse Res using impz; R=9.000000e-01
x[n]
0.04
0.02
0
-0.02
0 50 100 150 200 250 300
n
Impulse Res using difference equation; R=9.000000e-01
x[n]
0.04
0.02
0
-0.02
0 50 100 150 200 250 300
n
Impulse Res using direct computation; R=9.000000e-01
x[n]
0.06
0.04
0.02
0
-0.02
0 50 100 150 200 250 300
n

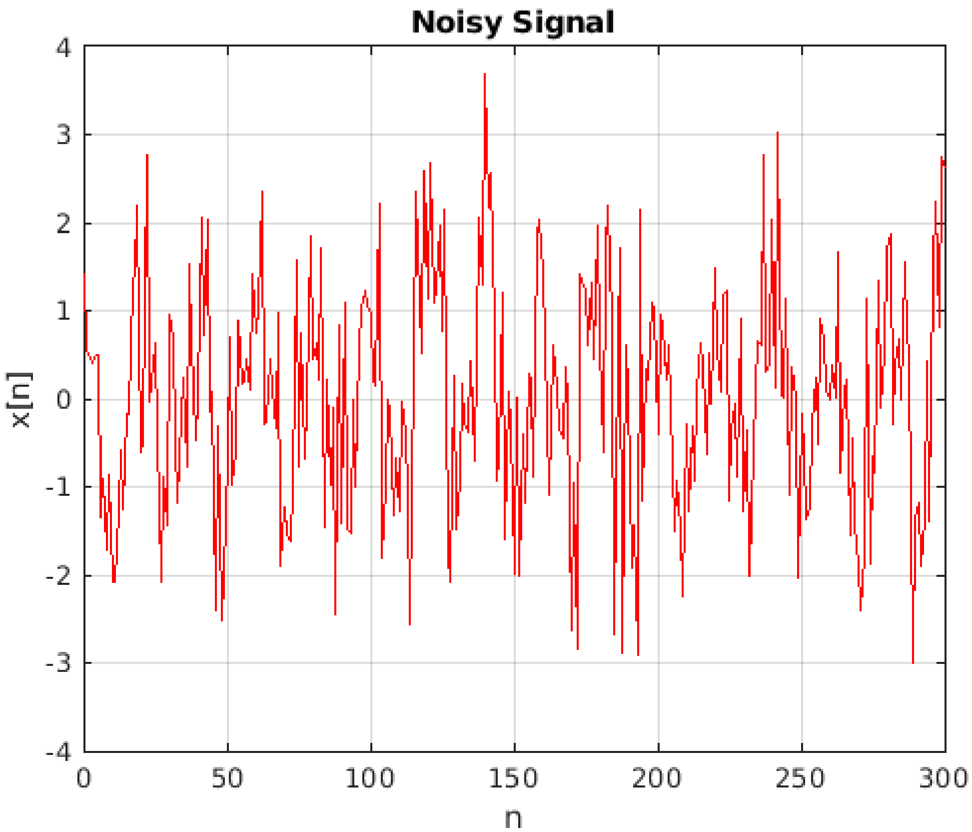

Impulse Res using impz; R=9.900000e-01
×10^-3
x[n]
5
0
-5
0 50 100 150 200 250 300
n
Impulse Res using difference equation; R=9.900000e-01
x[n]
5
0
-5
0 50 100 150 200 250 300
n
Impulse Res using direct computation; R=9.900000e-01
x[n]
0.01
0
-0.01
0 50 100 150 200 250 300
n

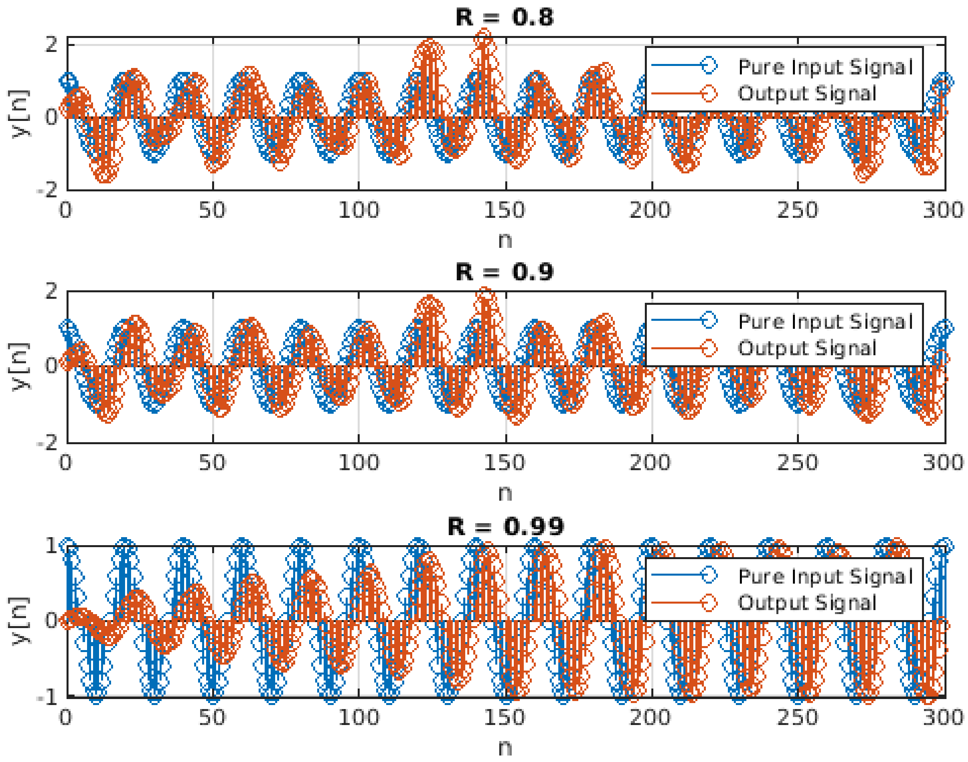
3-1-c:

```
clear all;
close all;
clc;

f0 = 500; % Hz
fs = 10000; % Hz
w  = 2 * pi * f0 / fs;
R = [0.80, 0.90, 0.99];

N = 300;
n  = 0:1:N;
s  = cos (w*n);
v  = randn(1,length(n));
x  = s + v;

figure('Name', 'Noisy Signal');
plot(n, x, 'r');
grid on;
xlabel('n');
ylabel('x[n]');
title('Noisy Signal');

figure("Name", "Filtering Noisy Signal");
for ll = 1:length(R)
    % Defining filter parameters
    G = (1 - R(ll)) * (1 - 2*R(ll)*cos(2*w) + R(ll)^2)^0.5;
    b = G;
    a = [1, -2*R(ll)*cos(w), R(ll)^2];

    % Filtering process
    y = zeros(1,N);
    for mm = 1:N
        if (mm == 1)
            y(mm) = G*x(mm) ;
            w1 = y(1) ;
        elseif(mm == 2)
            y(mm) = -a(2)*w1 + G*x(mm) ;
            w2 = w1;
            w1 = y(2) ;
        else
            y(mm) = -a(2)*w1 - a(3)*w2 + G*x(mm) ;
            w2 = w1;
            w1 = y(mm);
        end
    end
    % Plotting results
    subplot(length(R), 1, ll);
    stem(n, s);
    hold on;
    stem(n(1:end-1), y);
    grid on;
```

```
    xlabel('n');
    ylabel('y[n]');
    title("R = " + num2str(R(ll)));
    legend("Pure Input Signal", "Output Signal")
end
```

A larger R value leads to improved results (output closer to the ideal), but the settling time increases as R becomes larger.



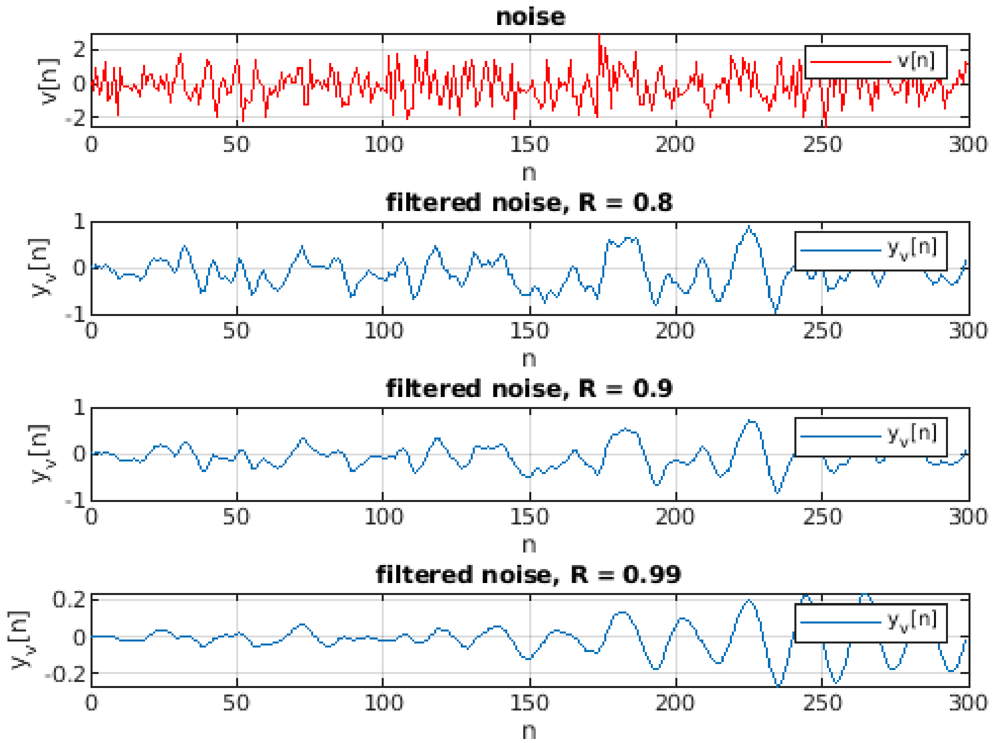


3-1-d:

```
clear all;
close all;
clc;

f0 = 500; % Hz
fs = 10000; % Hz
w  = 2 * pi * f0 / fs;
R = [0.80, 0.90, 0.99];
N = 300;
n  = 0:1:N;
s  = cos (w*n);
v  = randn(1,length(n));
x  = s + v;

figure("Name", "Nosie vs Filtered Noise")
subplot(length(R)+1, 1, 1);
plot(n, v,  "r");
hold on;
grid on;
xlabel('n');
ylabel('v[n]');
title('noise');
legend('v[n]');
```

```
for c = 1:length(R)
    % Defining filter parameters
    G = (1 - R(c)) * (1 - 2*R(c)*cos(2*w) + R(c)^2)^0.5;
    b = G;
    a = [1, -2*R(c)*cos(w), R(c)^2];

    % Filtering process (difference method)
    y_v = zeros(1,N); % Filter Output
    for mm = 1:N
        if (mm == 1)
            y_v(mm) = G*v(mm) ;
            w1 = y_v(1) ;
        elseif(mm == 2)
            y_v(mm) = -a(2)*w1 + G*v(mm) ;
            w2 = w1;
            w1 = y_v(2) ;
        else
            y_v(mm) = -a(2)*w1 - a(3)*w2 + G*v(mm) ;
            w2 = w1;
            w1 = y_v(mm);
        end
    end

    % Plotting results
    subplot(length(R)+1, 1, c+1);
    plot(n(1:end-1), y_v);
    grid on;
    xlabel('n');
    ylabel('y_v[n]');
    title("filtered noise, R = " + num2str(R(c)));
    legend('y_v[n]');
end
```

Applying white noise to a bandpass filter capitalizes on the filter's ability to select a specific frequency band. White noise contains power across all frequencies. The bandpass filter isolates and emphasizes a single frequency from the noise, resulting in a clean sine wave output.



3-1-e:

```
clear all;
close all;
clc;

f0 = 500; % Hz
fs = 10000; % Hz
w0  = 2 * pi * f0 / fs;
R = [0.80, 0.90, 0.99];
N = 300;
n  = 0:1:N;
s  = cos (w0*n);
v  = randn(1,length(n));
x  = s + v;
for c = 1:length(R)
    % Defining filter parameters
    G = (1 - R(c)) * (1 - 2*R(c)*cos(2*w0) + R(c)^2)^0.5;
    b = G;
    a = [1, -2*R(c)*cos(w0), R(c)^2];
    y_v = zeros(1,N);

    % Filtering process (using difference method)
    for mm = 1:N
        if (mm == 1)
```

```
            y_v(mm) = G*v(mm) ;
            w1 = y_v(1) ;
        elseif(mm == 2)
            y_v(mm) = -a(2)*w1 + G*v(mm) ;
            w2 = w1;
            w1 = y_v(2) ;
        else
            y_v(mm) = -a(2)*w1 - a(3)*w2 + G*v(mm) ;
            w2 = w1;
            w1 = y_v(mm);
        end
    end

    % Noise Reduction Ratio Calculation
    NRR = std(y_v)^2/std(v)^2;
    % NRR_wrong = (1+R(c)^2)/((1+R(c))*(1+2*R(c)*cos(w0)+R(c)^2));
    NRR_tr = (G^2/(2*sin(w0)^2)) * ((1/(1-R(c)^2)) - ...
        ((cos(2*w0)-R(c)^2) / (1-2*R(c)^2*cos(2*w0)+R(c)^4))); %Theoretical
 value

    NRR_energy = sum(impz(b, a, 10^4).^2);

    % Displaying NRR results
    fprintf("NRR R=%d:\nUsing variance: %d\nUsing formula: %d\nUsing energy of
 impulse response: %d\n\n", ...
        R(c), NRR, NRR_tr, NRR_energy);
end
```

```
NRR R=8.000000e-01:
Using variance: 1.576903e-01
Using formula: 1.683456e-01
Using energy of impulse response: 1.683456e-01

NRR R=9.000000e-01:
Using variance: 8.505200e-02
Using formula: 9.754537e-02
Using energy of impulse response: 9.754537e-02

NRR R=9.900000e-01:
Using variance: 3.413506e-03
Using formula: 1.004279e-02
Using energy of impulse response: 1.004279e-02
```

Published with MATLAB® R2023a