Report of Signals and Systems Lab Assignment 3

Written by HUANG Guanchao, SID 11912309 and GONG Xinrui, SID 11911233.

3.5

Basic Problem a

We expected $x[n]$ to be purely real, for the coefficients is $a_k = a^*_{-k}$, and for real number, $x = x^*$.

Basic Problem b

For N = 5 and x[n] is real, we can find that $a_1 = a_{-4} = 2e^{-j\pi/3}, a_3 = a_{-2} = e^{-j\pi/4}$.

The MATLAB script is shown below.

```
N = 5;
a0 = 1;
a1 = 2 * exp(-j * pi / 3);
a2 = exp(j * pi / 4);
a3 = exp(-j * pi / 4);
a4 = 2 * exp(j * pi / 3);
a = [a0 a1 a2 a3 a4];
```

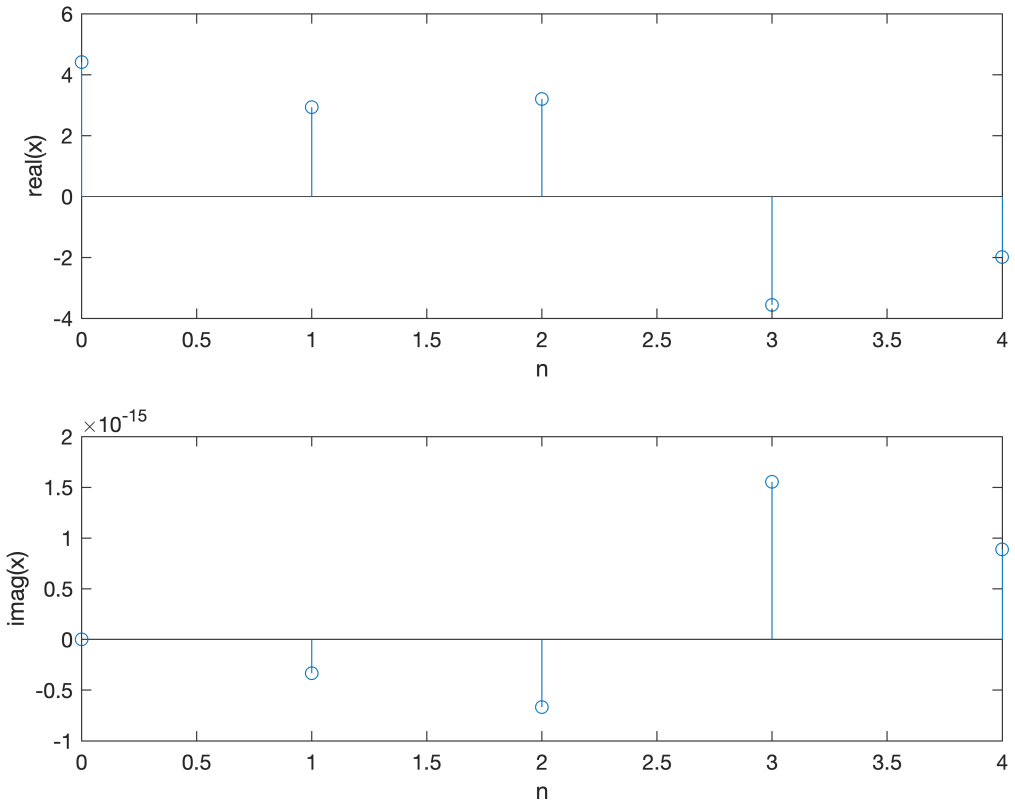
Basic Problem c

For this question, we use `for` loop to first find the exponent components for `x`, then use another `for` loop to calculate `x`.

The MATLAB script for the method is shown below.

```
for t = 0:4
    e1(t + 1) = exp(j * t * (2 * pi / 5));
end

for t = 0:4
    x(t + 1) = sum(a .* (e1.^t));
end
```

Finally, make sure the plot is divided into real component part and imaginary component part.The plot is shown below.



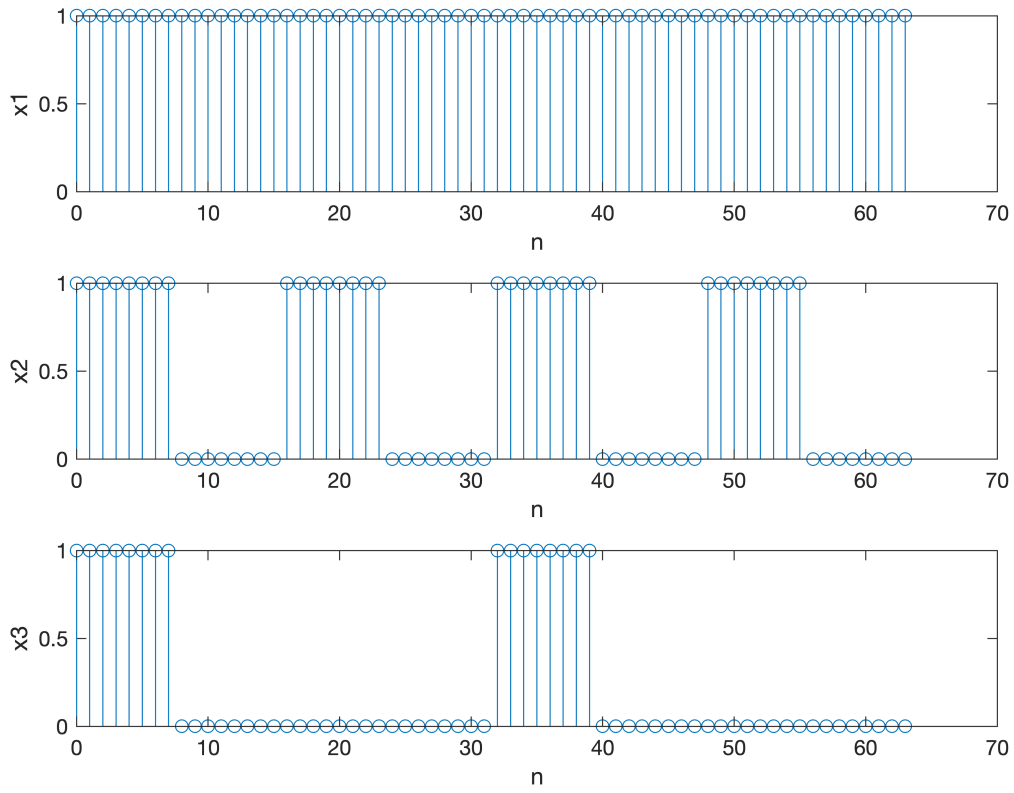
The MATLAB script is shown below.

```
n = 0:4;
e1 = zeros(1, 5);
x = zeros(1, 5);

for t = 0:4
    e1(t + 1) = exp(j * t * (2 * pi / 5));
end

for t = 0:4
    x(t + 1) = sum(a .* (e1.^t));
end

subplot(2, 1, 1)
stem(n, real(x))
xlabel('n')
ylabel('real(x)')

subplot(2, 1, 2)
stem(n, imag(x))
xlabel('n')
ylabel('imag(x)')
```

Intermediate Problem d

Write out `x1`, `x2`, `x3` then simply use another three vectors to contain the repeated form for these three vectors.

The plot is shown below.



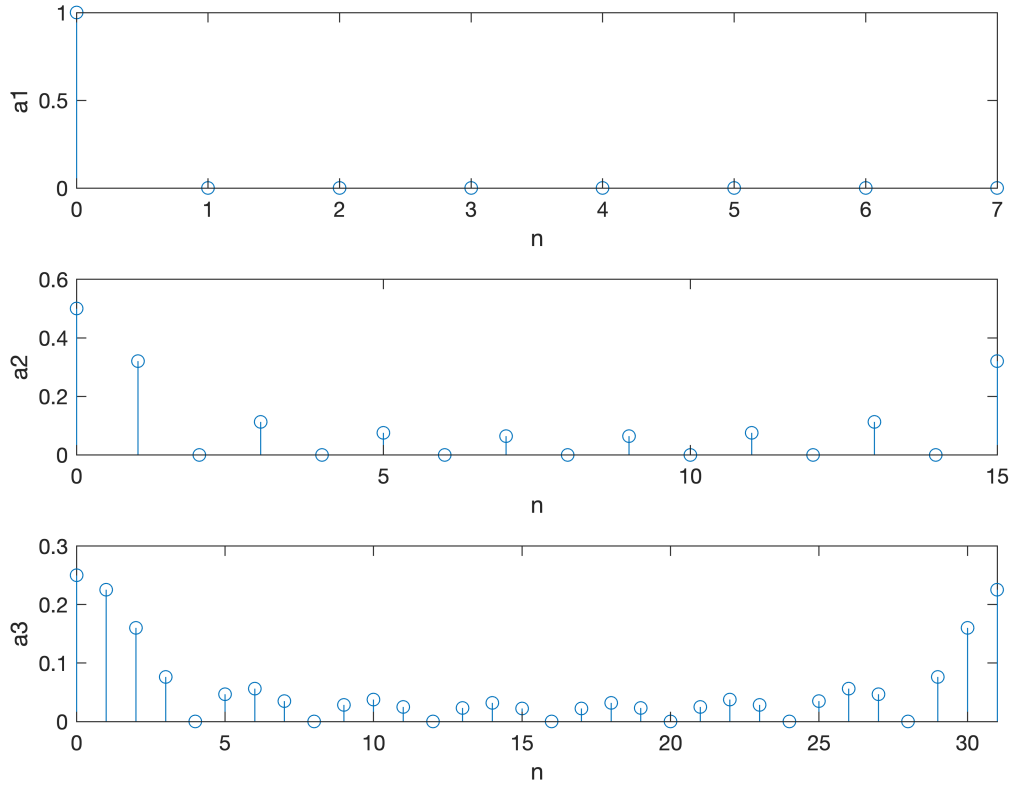
The MATLAB script is shown below.

```
n1 = 0:7;
x1 = ones(1, 8);
n2 = 0:15;
x2 = [ones(1, 8) zeros(1, 8)];
n3 = 0:31;
x3 = [ones(1, 8) zeros(1, 24)];

n = 0:63;
x1_n = [x1 x1 x1 x1 x1 x1 x1 x1];
x2_n = [x2 x2 x2 x2];
x3_n = [x3 x3];

subplot(3, 1, 1)
stem(n, x1_n)
xlabel('n')
ylabel('x1')

subplot(3, 1, 2)
stem(n, x2_n)
xlabel('n')
ylabel('x2')

subplot(3, 1, 3)
stem(n, x3_n)
xlabel('n')
ylabel('x3')
```

Intermediate Problem e

Using the function `fft()` to calculate `a1`, `a2`, `a3` and be careful the three signals have different periods.

The plot is shown below.



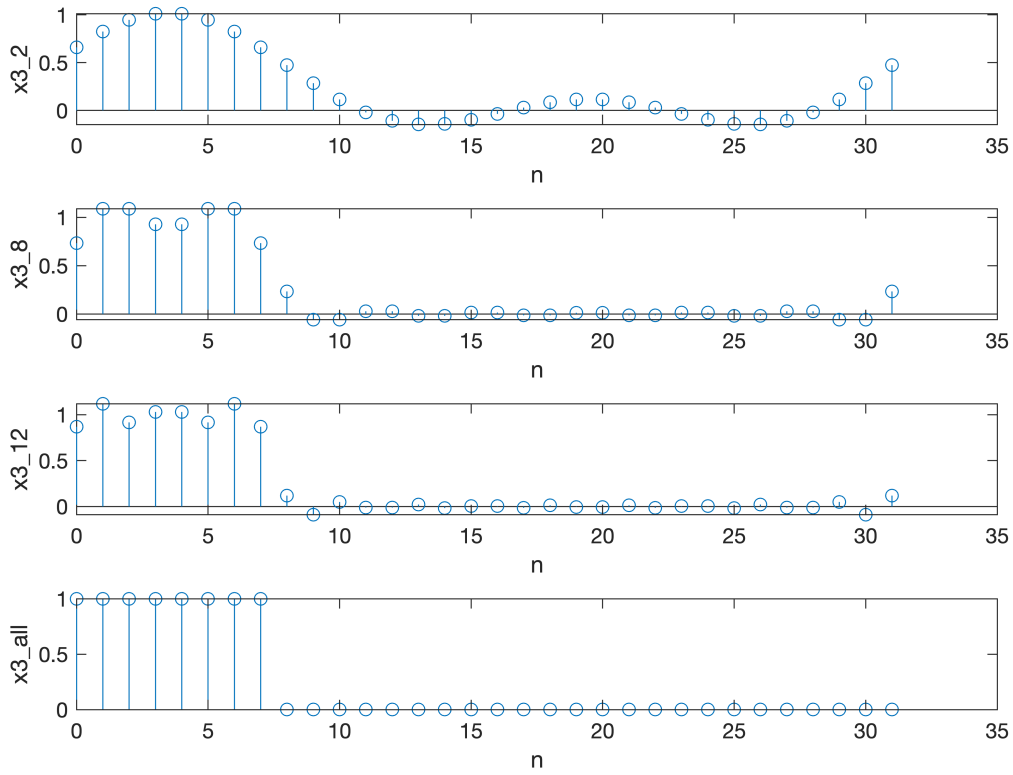
The MATLAB script is shown below.

```
N1 = 8;
N2 = 16;
N3 = 32;

a1 = 1/N1*fft(x1);
a2 = 1/N2*fft(x2);
a3 = 1/N3*fft(x3);

subplot(3, 1, 1)
stem(n1,abs(a1))
xlabel ('n')
ylabel('a1')

subplot(3, 1, 2)
stem(n2,abs(a2))
xlabel ('n')
ylabel('a2')

subplot(3, 1, 3)
stem(n3,abs(a3))
xlim([0 31])
xlabel ('n')
ylabel('a3')
```

Intermediate Problem f

This problem need us to produce the required four signals which are come from `x3`. So we use the similar methods as in problem c to calculate each signal. However, since the order of indices in `a3` when using function `fft()` are different from the type we write in hand, so now we count the components of `x3` by using `a3` directly instead of using the component sections of `a3` like in problem c.

The plot is shown below.



The MATLAB script is shown below.

```
N3 = 32
x3 = [ones(1, 8), zeros(1, 24)];
a3 = 1 / N3 * fft(x3);

a3_2 = [a3(1:3) zeros(1, 27) a3(31:32)];
a3_8 = [a3(1:9) zeros(1, 15) a3(25:32)];
a3_12 = [a3(1:13) zeros(1, 7) a3(21:32)];

for n = 1:32
    x3_2(n) = a3(1);

    for t = 1:2
        x3_2(n) = x3_2(n) + a3(t + 1) * exp(j * t * 2 * pi * (n - 1) / 32) +
conj(a3(t + 1)) * exp((-j) * t * 2 * pi * (n - 1) / 32);
    end

end

for n = 1:32
```

```
    x3_8(n) = a3(1);

    for t = 1:8
        x3_8(n) = x3_8(n) + a3(t + 1) * exp(j * t * 2 * pi * (n - 1) / 32) +
conj(a3(t + 1)) * exp(-j * t * 2 * pi * (n - 1) / 32);
    end

end

for n = 1:32
    x3_12(n) = a3(1);

    for t = 1:12
        x3_12(n) = x3_12(n) + a3(t + 1) * exp(j * t * 2 * pi * (n - 1) / 32) +
conj(a3(t + 1)) * exp(-j * t * 2 * pi * (n - 1) / 32);
    end

end

for n = 1:32
    x3_all(n) = a3(1);

    for t = 1:15
        x3_all(n) = x3_all(n) + a3(t + 1) * exp(j * t * 2 * pi * (n - 1) / 32) +
conj(a3(t + 1)) * exp(-j * t * 2 * pi * (n - 1) / 32);
    end

    x3_all(32) = x3_all(n) + a3(17) * exp(j * 17 * 2 * pi * (n - 1) / 32)

end

r = 0:31;
subplot(4, 1, 1)
stem([0:31], real(x3_2))
xlabel('n')
ylabel('x3_2', 'Interpreter', 'none')

subplot(4, 1, 2)
stem(r, real(x3_8))
xlabel('n')
ylabel('x3_8', 'Interpreter', 'none')

subplot(4, 1, 3)
stem(r, real(x3_12))
xlabel('n')
ylabel('x3_12', 'Interpreter', 'none')

subplot(4, 1, 4)
stem(r, real(x3_all))
xlabel('n')
ylabel('x3_all', 'Interpreter', 'none')
```

Intermediate Problem g

Because $a_k = a^*_{-k}$, $x_3[n] = x_3[-n]^*$, and according to the property of DTFS, signals which has these qualities is real.

Intermediate Problem h

The plots have already been plotted in problem f, and we can easily notice that they do not display the Gibb's phenomenon.

Advanced Problem

The file `dtfs.m` is shown in the code block below.

```
function a = dtfs(x, n_init)
    N = size(x, 2);
    a = zeros(1, N);

    for k = 1:N

        for n = 1:N
            a(k) = a(k) + 1 / N * x(n) * exp(-1j * (k - 1) * 2 * pi / N * (n +
n_init - 1));
        end

    end

end
```

The running results of `dtfs()` function we write is identical with the results given in the question.

3.8

Intermediate Problems a

For this problem, we can find out the vector easily. Notice that the two systems are all DT systems, so the order for index should be $[a_0, a_1, \ldots, a_k]$, and a_1 and a_2 should be the index vectors for outputs while b_1 and b_2 is for inputs.

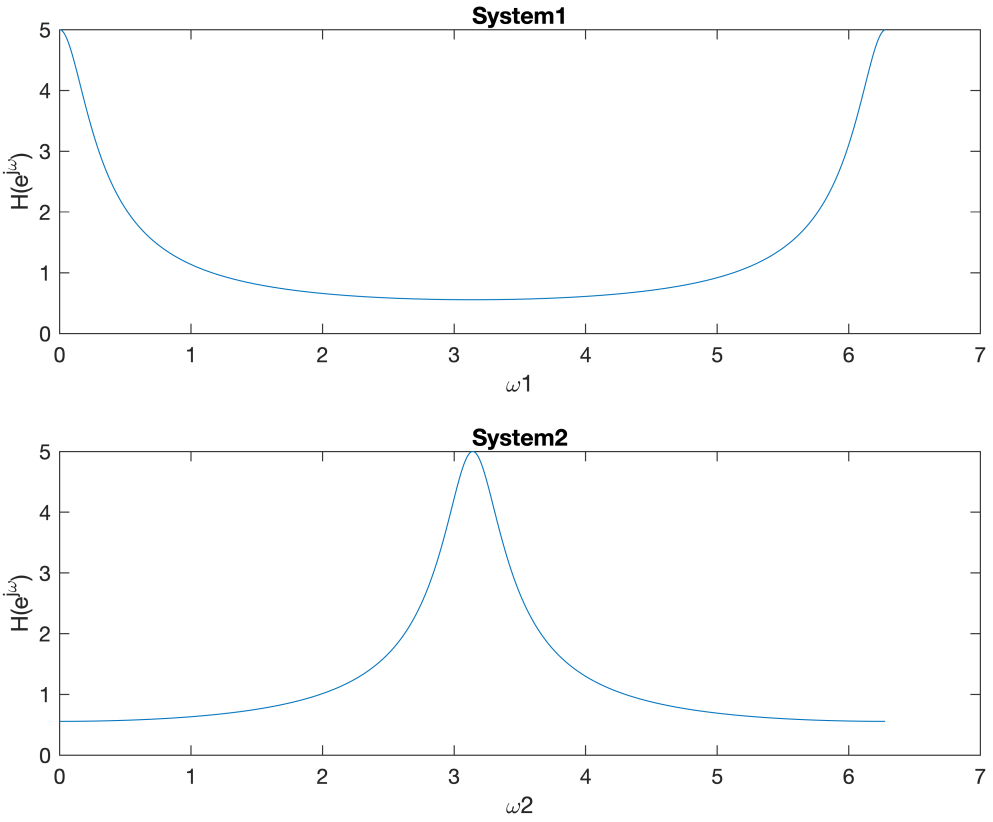
The MATLAB script is shown below.

```
% System1
a1 = [1 -0.8];
b1 = 1;

% System2
a2 = [1 0.8];
b2 = 1;
```

Intermediate Problem b

In this problem we need yo find out the frequency response for the two systems. To do this we need to use the function `freqz()`. All we need to do is to check the index vectors we need to use and the number of points we want to collect. The index vectors have already been found out in problem a and the number of points has been mentioned in the question stem, which is 1024. Then we can easily plot the plots.

The plots are shown below.



We can easily conclude from the plots that the system1 is a highpass filter and system2 is a lowpass filter.

The MATLAB script is shown below.

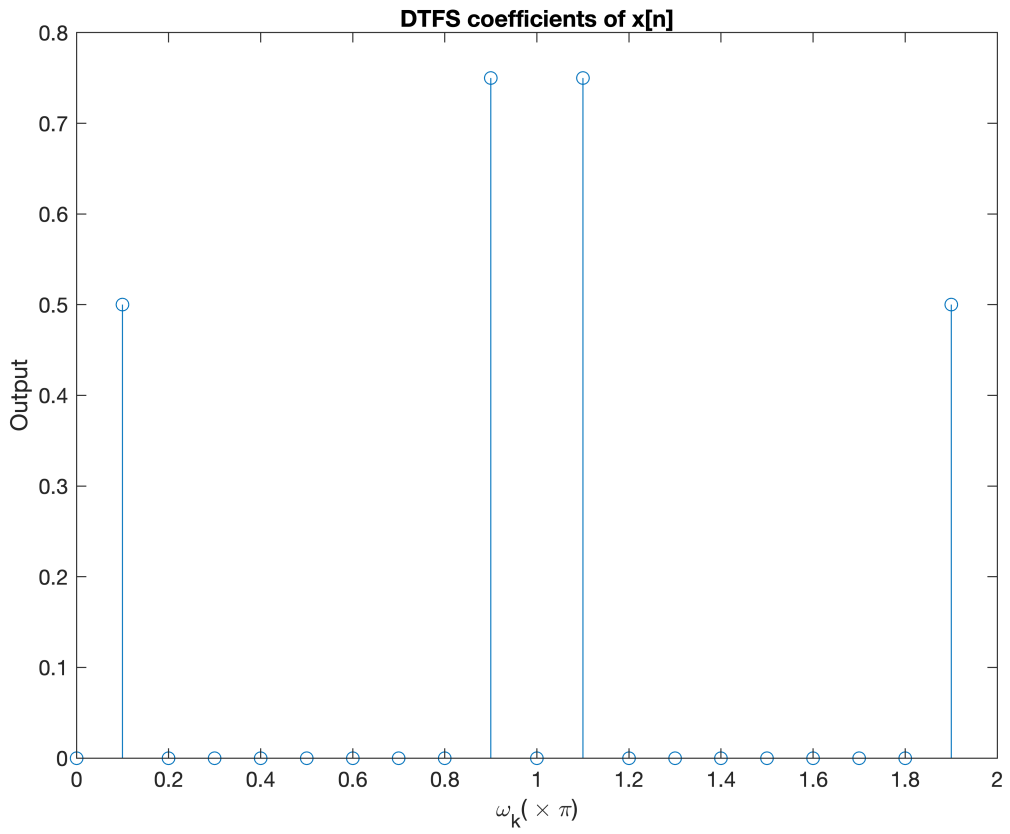
```
% System1
a1 = [1 -0.8];
b1 = 1;

% System2
a2 = [1 0.8];
b2 = 1;

N = 1024;
[H1, omega1] = freqz(b1, a1, N, 'whole');
[H2, omega2] = freqz(b2, a2, N, 'whole');

% Plot the plots
subplot(2, 1, 1)
plot(omega1, abs(H1))
xlabel ('\omega1')
ylabel('H(e^{j\omega})')
```

```
title('System1')

subplot(2, 1, 2)
plot(omega2, abs(H2))
xlabel ('\omega2')
ylabel('H(e^{j\omega})')
title('System2')
```

Intermediate Problem c

In this question, the DTFS coefficients a_k we need to draw has already been written out, so we only need to write it in the form of vector and plot the figure using function `stem()`.

The plot is shown below.

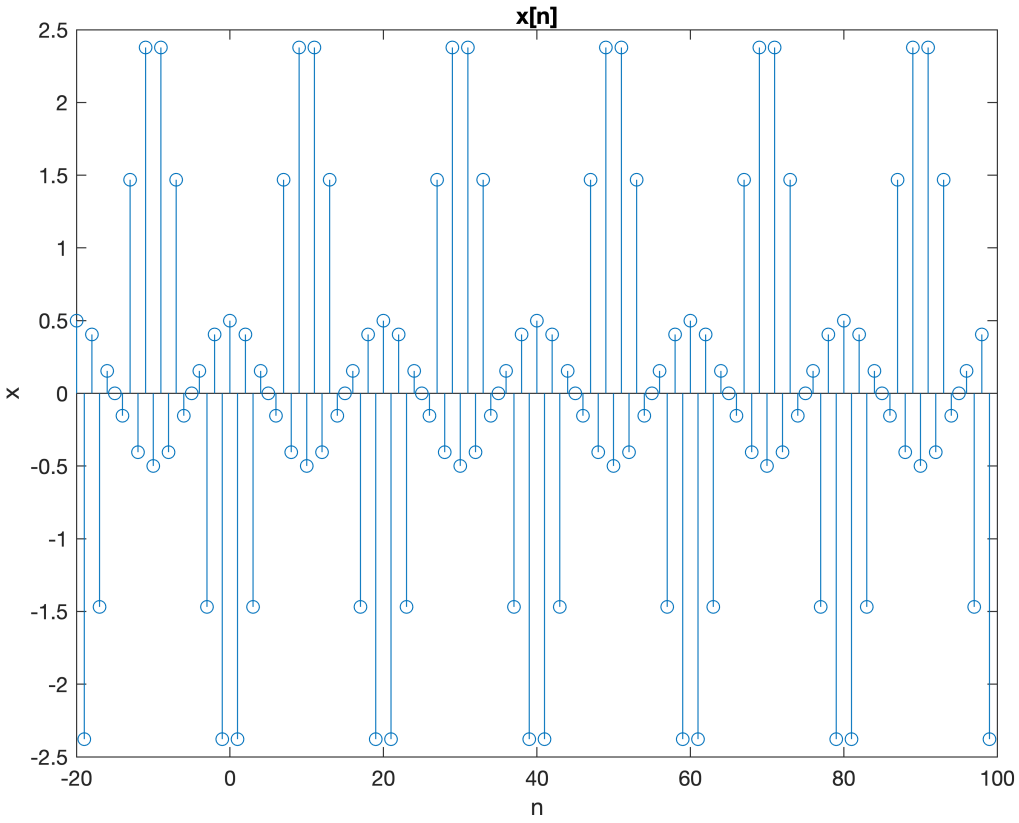


Comparing this plot with the plots for frequency response, we can find that for System1, since it is highpass, frequency components $k = \pm 9$ will be amplified and $k = \pm 1$ will be attenuated. For system2, since it is lowpass, frequency components $k = \pm 1$ will be amplified and $k = \pm 9$ will be attenuated.

Intermediate Problem d

To generate `x`, we should first generate `x_20`. The coefficient vector `a_k` has already been generated and the period is $N = 20$, so now we use the function `ifft()` to generate x_20, which should be `x_20 = 20*ifft(a_x)`.

The plot is shown below.



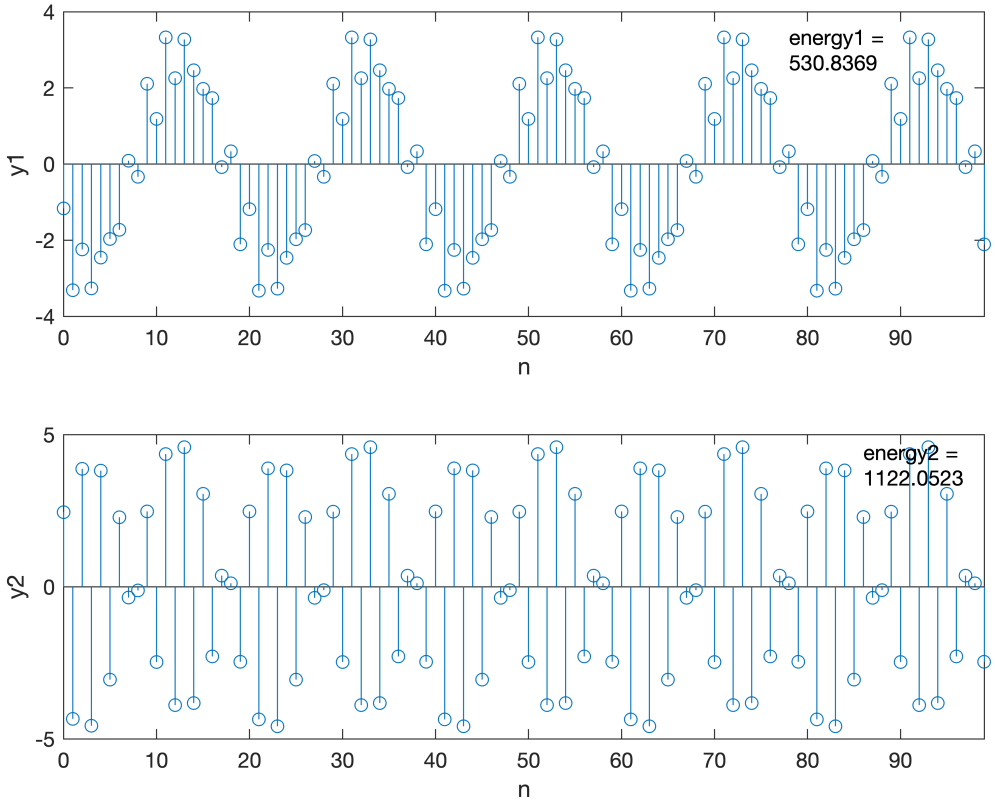
The MATLAB script is shown below.

```
k = 0:19;
n = -20:99;
a_x = [-1/2 zeros(1, 7) 3/4 0 3/4 zeros(1, 7) -1/2 0];
x_20 = 20 * ifft(a_x);
x = [x_20 x_20 x_20 x_20 x_20 x_20];

stem(n, x)
xlabel('n')
ylabel('x')
title('x[n]')
```

Intermediate Problem e

In this question, we have already find out `x` in problem d, and the index vectors for the two systems have also been found in problem a. So now we use the function `filter()` to find the two outputs `y1` and `y2`. Since it is required to plot the outputs for $0 \leq x \leq 99$, so we use a function `xlim()` to limit the range.

The plot is shown below.



We can conclude from the plots that system1 has more high frequency energy and system2 has more low frequency energy, which can confirm answers in problem c.

The MATLAB script is shown below.

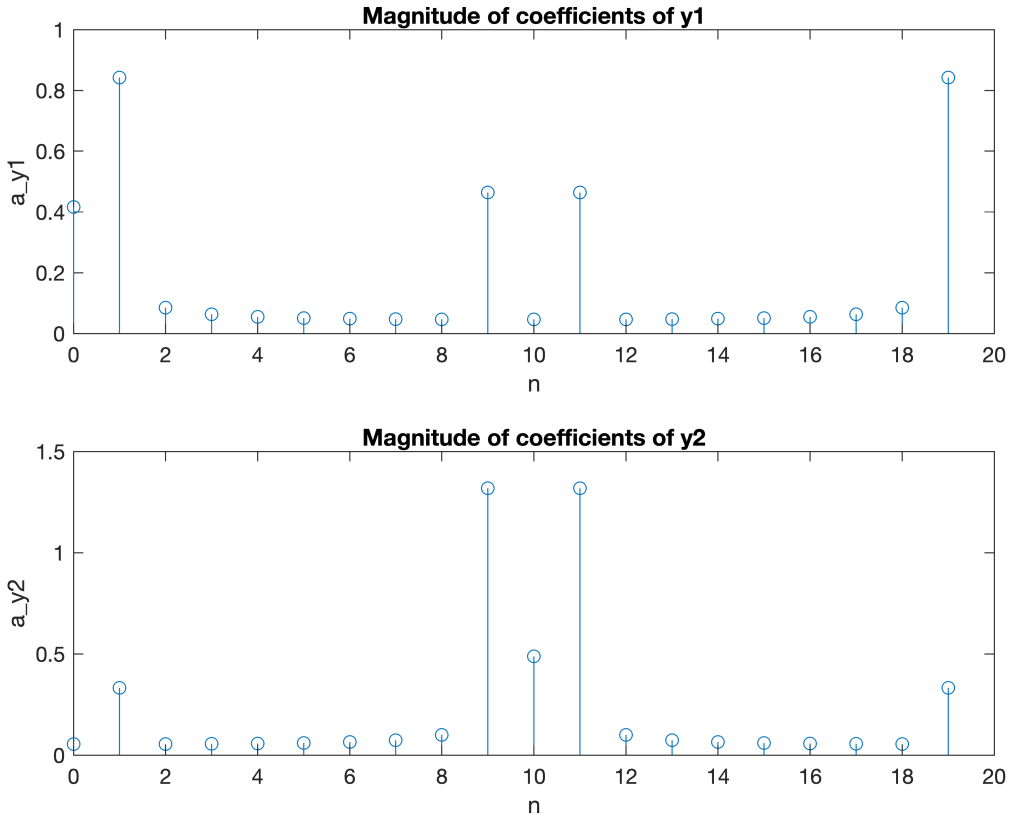
```
k = 0:19;
% range for input
n = -20:99;
a_x = [0 -1/2 zeros(1, 7) 3/4 0 3/4 zeros(1, 7) -1/2];
x_20 = 20 * ifft(a_x);
x = [x_20 x_20 x_20 x_20 x_20 x_20];

% System1
a1 = [1 -0.8];
b1 = 1;
y1 = filter(b1, a1, x);
E1 = sum(y1.^2);
% System2
a2 = [1 0.8];
b2 = 1;
y2 = filter(b2, a2, x);
E2 = sum(y2.^2);

% plot
subplot(2, 1, 1)
stem(n, y1);
xlim([0, 99])
xlabel('n')
ylabel('y1')
text(78, 3, {'energy1 = ', num2str(E1)})

subplot(2, 1, 2)
```

```
stem(n, y2);
xlim([0, 99])
xlabel('n')
ylabel('y2')
text(86, 4, {'energy2 = ', num2str(E2)})
```

Intermediate Problem f

This question needs us to find out coefficient for y1 and y2. Since we have already found y1 and y2, now we need to choose a period of these two signals as the input signals for function `fft()` , notice that the period $N = 20$, so we can calculate them then.

The plot is shown below.



We can now conclude that with the same input, output of system1 has higher high frequency energy and lower low frequency energy, system2 has higher low frequency energy and lower high frequency energy. So we confirm the results in problem e.

The MATLAB script is shown below.

```
k = 0:19;
% range for input
n = -20:99;
a_x = [0 -1/2 zeros(1, 7) 3/4 0 3/4 zeros(1, 7) -1/2];
x_20 = 20 * ifft(a_x);
x = [x_20 x_20 x_20 x_20 x_20 x_20];

% System1
a1 = [1 -0.8];
b1 = 1;
y1 = filter(b1, a1, x);
```

```
y1_20 = y1(1:20);
% System2
a2 = [1 0.8];
b2 = 1;
y2 = filter(b2, a2, x);
y2_20 = y2(1:20);

a_y1 = (1/20) * fft(y1_20);
a_y2 = (1/20) * fft(y2_20);

subplot(2, 1, 1)
stem(k, abs(real(a_y1)))
xlabel('n')
ylabel('a_y1', 'Interpreter', 'none')
title('Magnitude of coefficients of y1')

subplot(2, 1, 2)
stem(k, abs(real(a_y2)))
xlabel('n')
ylabel('a_y2', 'Interpreter', 'none')
title('Magnitude of coefficients of y2')
```

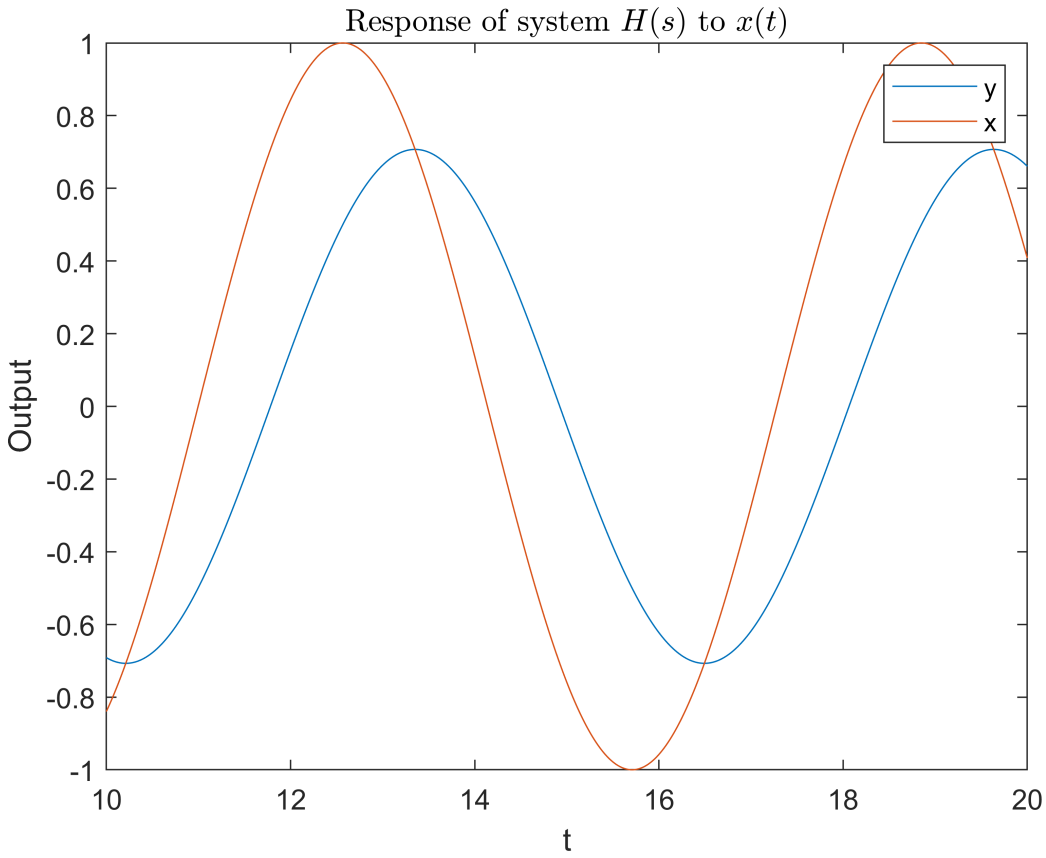
3.9

Intermediate Problem a

This problem needs us to find the output of a system of which the frequency response and input are given.

According to the system function given, the index vector is `b = [1]` and `a = [1 1]`. The input signal is $x(t) = cos(t)$, and the region for t is $0 \leq t \leq 20$. Now we have figure out all the parameters needed for calculating the output, function `lsim()` can be used to calculate the output.

The plot is shown below.



We can see that the amplitude and phase of $x(t)$ and $y(t)$ are different. The reason can be found in the system function, from which we can derive that

$$y(t) + \frac{dy(t)}{dx(t)} = x(t) \Rightarrow y(t) = \frac{\cos t + \sin t}{2} = y(t) = \frac{\sqrt{2}}{2}\cos\left(t - \frac{\pi}{4}\right)$$

So in this form of $y(t)$ we can clearly see the difference in amplitude and phase.

The MATLAB script is shown below.

```
t = linspace(0, 20, 1000);
x = cos(t);
b = [1];
a = [1 1];

y = lsim(b, a, x, t);

plot(t, y)
xlim([10 20])
hold on
plot(t, x)
xlim([10 20])
xlabel('t')
ylabel('Output')
title('Response of system $H(s)$ to $x(t)$', 'Interpreter', 'latex')
legend('y', 'x');
```

Intermediate Problem b

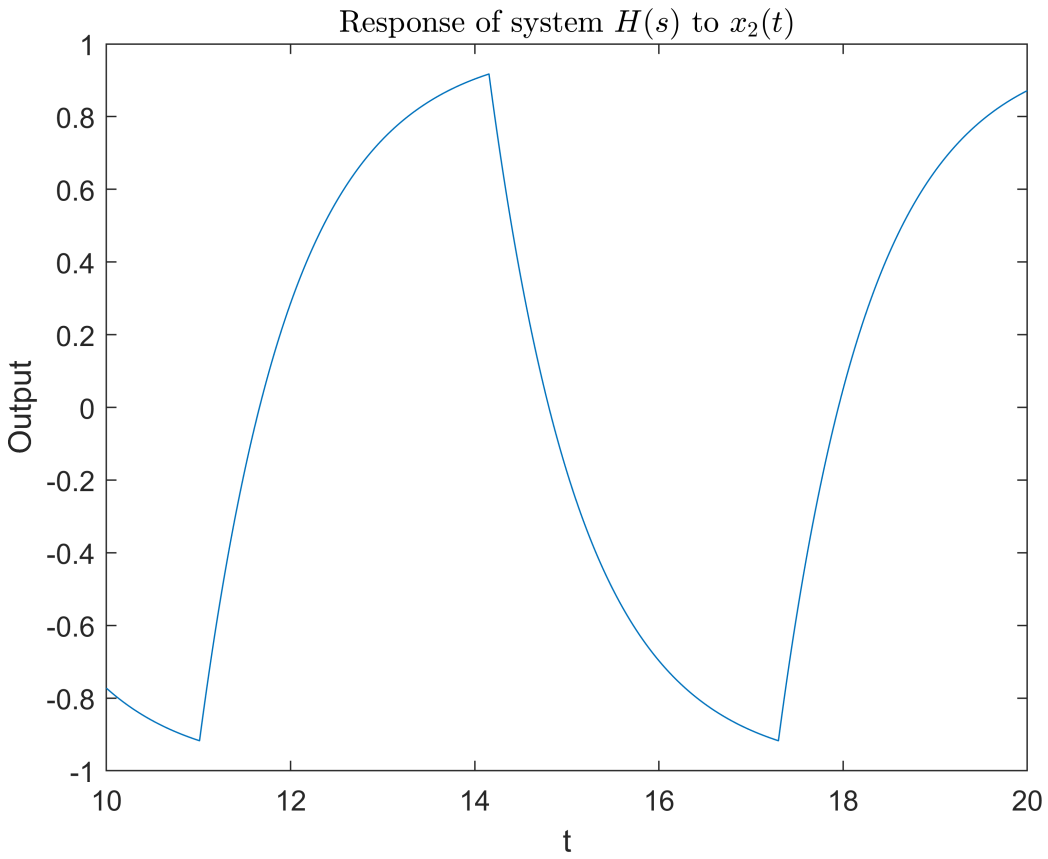
This problem needs us to find the output for the same system but change the input into a square wave `x2`.

The way to produce a square wave has been given in the question stem and is shown below.

```
>> x2 = cos(t);
>> x2(x2 > 0) = ones(size(x2(x2 > 0)));
>> 2(x2 < 0) = -ones(size(x2(x2 < 0)));
```

Now we can use the same way to find out the output as in problem a.

The plot is shown below.



The MATLAB script is shown below.

```
t = linspace(0, 20, 1000);

% square wave
x2 = cos(t);
x2(x2 > 0) = ones(size(x2(x2 > 0)));
x2(x2 < 0) = -ones(size(x2(x2 < 0)));

b = [1 0];
a = [1 1];
y2 = lsim(b, a, x2, t);

plot(t, y2)
xlim([10 20])
xlabel('t')
ylabel('Output')
```

```
title('Response of H to x2')
```

Advanced Problem c

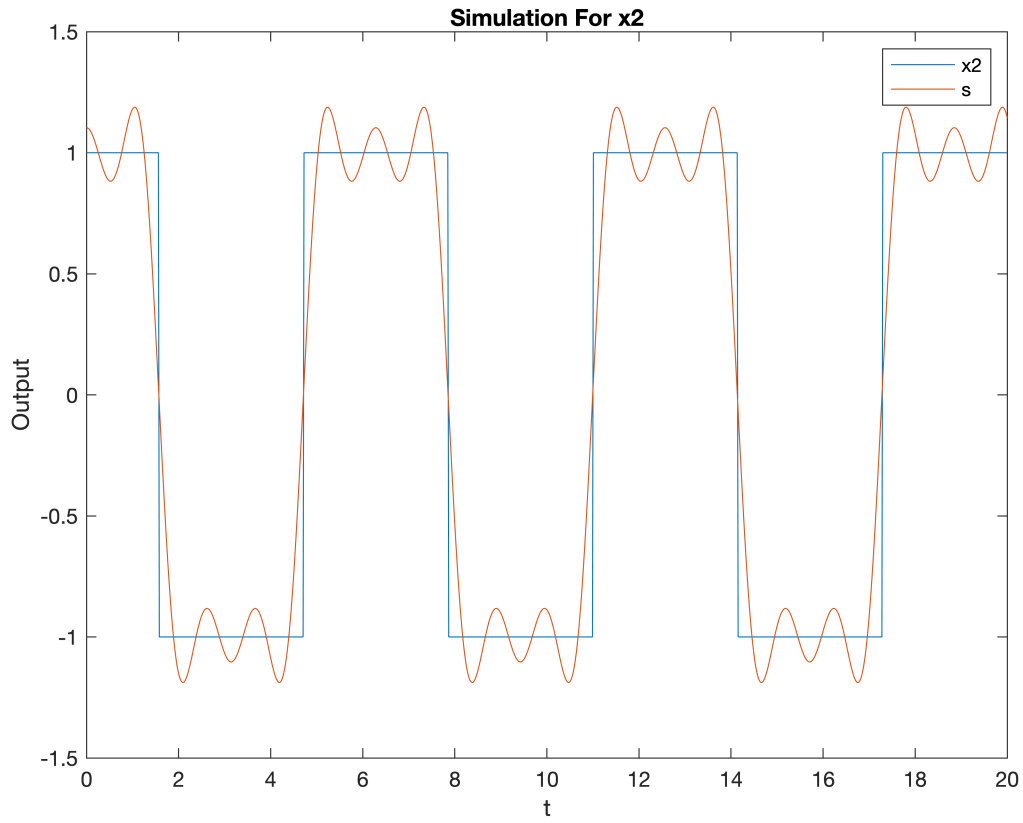
The question needs us to simulate `x2` using the first ten nonzero numbers of the CTFS coefficients. Because we have already noticed `x2`, and we can noticed that the signal $s(t)$ and $x_2(t)$ has the relationship of time invariance. Because we know that $a_k = \frac{sin(\pi k/2)}{\pi k}$ for $s(t)$, and $s(t)$ has the same a_k, we can calculate the value of a_k easily. Then we place the positive coefficients in vector `apos_k` and the negative coefficients in vector `aneg_k`. After that we create the harmonic components using the method given by the question stem.

So we have five harmonic components as shown below.

```
s1 = apos_k(1) * exp(j * t) + aneg_k(1) * exp(-j * t);
s2 = apos_k(2) * exp(j * 2 * t) + aneg_k(2) * exp(-j * 2 * t);
s3 = apos_k(3) * exp(j * 3 * t) + aneg_k(3) * exp(-j * 3 * t);
s4 = apos_k(4) * exp(j * 4 * t) + aneg_k(4) * exp(-j * 4 * t);
s5 = apos_k(5) * exp(j * 5 * t) + aneg_k(5) * exp(-j * 5 * t);
```

Then we sum them up to get the simulation results for x2.

The plot is shown below.



The MATLAB script is shown below.

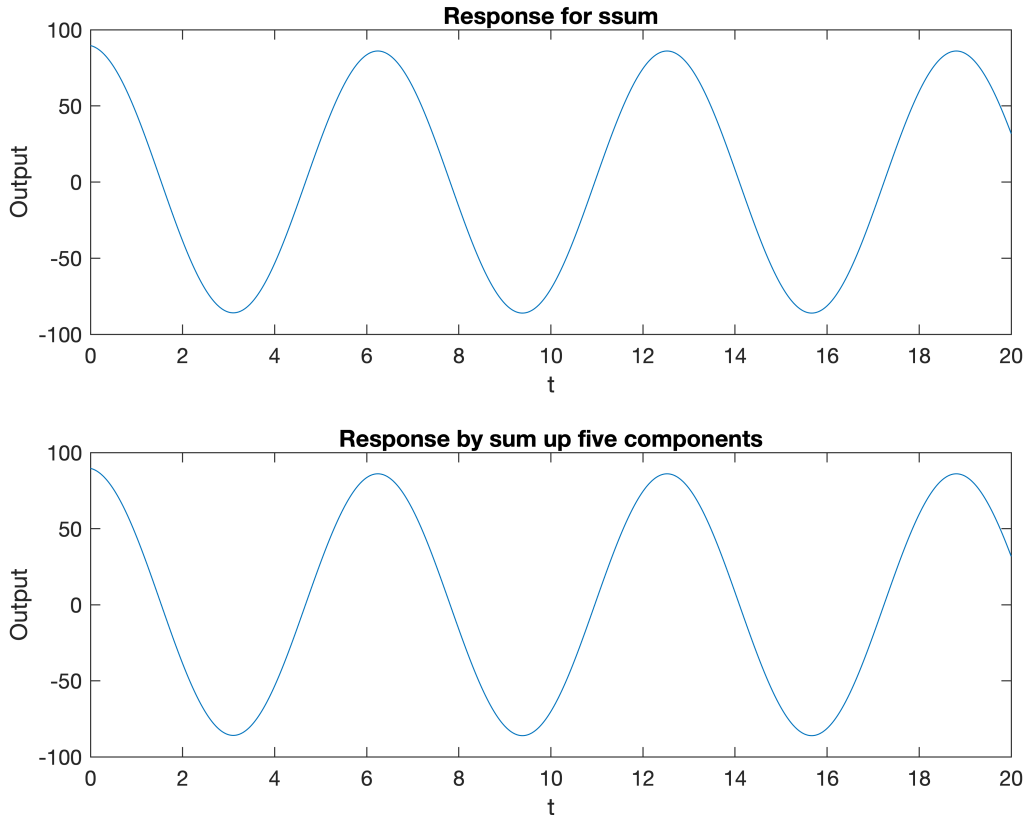
```
t = linspace(0, 20, 1000);

% square wave
x2 = cos(t);
x2(x2 > 0) = ones(size(x2(x2 > 0)));
```

```
x2(x2 < 0) = -ones(size(x2(x2 < 0)));

% coefficients for x2 CTFS
for i = 1:5
    apos_k(i) = 2 * sin(pi * i / 2) / (pi * i);
    aneg_k(i) = 2 * sin(-pi * i / 2) / (-pi * i);
end

s1 = apos_k(1) * exp(j * t) + aneg_k(1) * exp(-j * t);
s2 = apos_k(2) * exp(j * 2 * t) + aneg_k(2) * exp(-j * 2 * t);
s3 = apos_k(3) * exp(j * 3 * t) + aneg_k(3) * exp(-j * 3 * t);
s4 = apos_k(4) * exp(j * 4 * t) + aneg_k(4) * exp(-j * 4 * t);
s5 = apos_k(5) * exp(j * 5 * t) + aneg_k(5) * exp(-j * 5 * t);
s = real(s1 + s2 + s3 + s4 + s5);

% plot
plot(t, x2)
hold on
plot(t, s)
xlabel('t')
ylabel('Output')
title('Simulation For x2')
legend('x2', 's')
```

Advanced Problem d

This question need us to find out the response for `s1`, `s2`, `s3`, `s4`, `s5` and `ssum`, and in problem c we have already find out those inputs. And because the $H(s)$ hasn't changed, the two index vectors `a` and `b` wil be the same as problem a. Then, we use the function `lsim()` to produce Outputs.

The plot is shown below.



The MATLAB script is shown below.

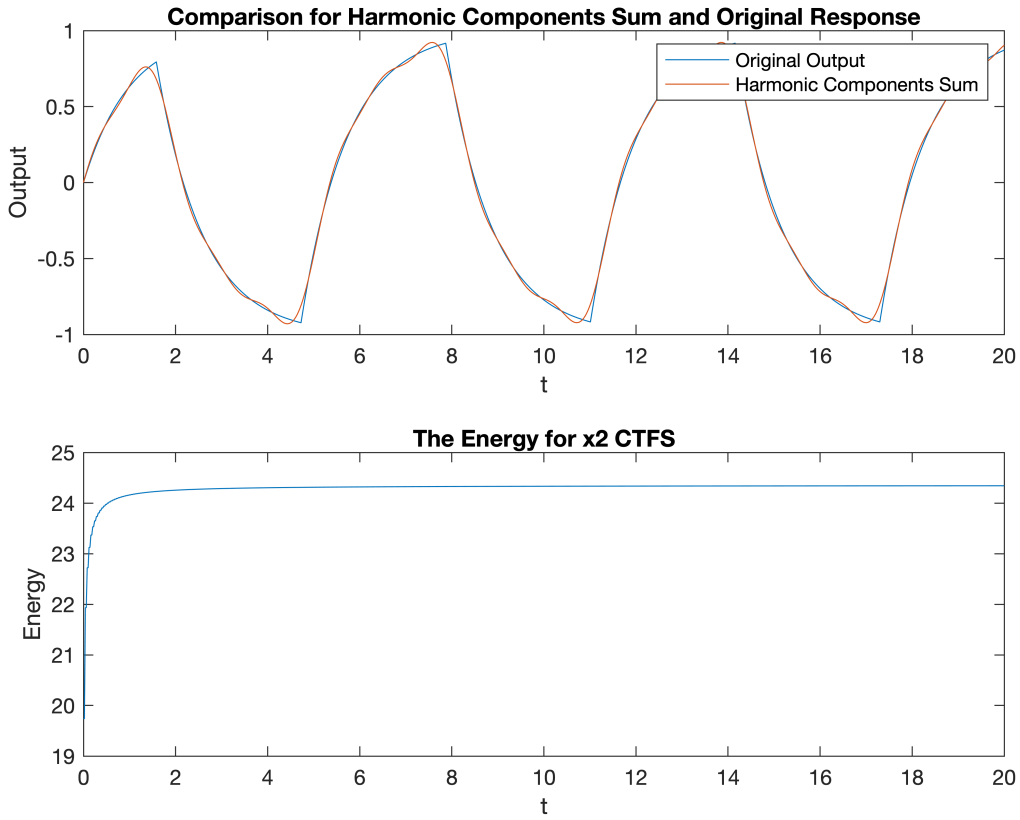
```
t=linspace (0,20,1000) ;
x2 = sign(cos(t));

t = linspace(0, 20, 1000);

% coefficients for x2 CTFS
for i = 1:5
    apos_k(i) = 2 * sin(pi*i/2)/(pi*i);
    aneg_k(i) = 2 * sin(-pi*i/2)/(-pi*i);
end

s1 = apos_k(1) * exp(j * t) + aneg_k(1) * exp(-j * t);
s2 = apos_k(2) * exp(j * 2 * t) + aneg_k(2) * exp(-j * 2 * t);
s3 = apos_k(3) * exp(j * 3 * t) + aneg_k(3) * exp(-j * 3 * t);
s4 = apos_k(4) * exp(j * 4 * t) + aneg_k(4) * exp(-j * 4 * t);
s5 = apos_k(5) * exp(j * 5 * t) + aneg_k(5) * exp(-j * 5 * t);
ssum = real(s1 + s2 + s3 + s4 + s5);

% Output
b = 1;
a = [1 1];

y1 = lsim(b, a, s1, t);
y2 = lsim(b, a, s2, t);
y3 = lsim(b, a, s3, t);
y4 = lsim(b, a, s4, t);
y5 = lsim(b, a, s5, t);
ysum1 = lsim(b, a, ssum, t);
ysum2 = y1 + y2 + y3 + y4 + y5;
```

```
% plot
subplot(2, 1,1)
plot(t, ysum1)
xlabel('t')
ylabel('Output')
title('Response for ssum')

subplot(2, 1,2)
plot(t, ysum2)
xlabel('t')
ylabel('Output')
title('Response by sum up five components')
```

Advanced Problem e

We first place the original output and the simulation output together to make a comparison.

The plot for two output and the energy for `x2` is shown below.



We can easily conclude from the plots that the two response are similar, the reason can be figure out in the second plot that the energy for `x2` is mostly determined by the low frequency part, which is the part we choose to use for simulation.

The MATLAB script is shown below.

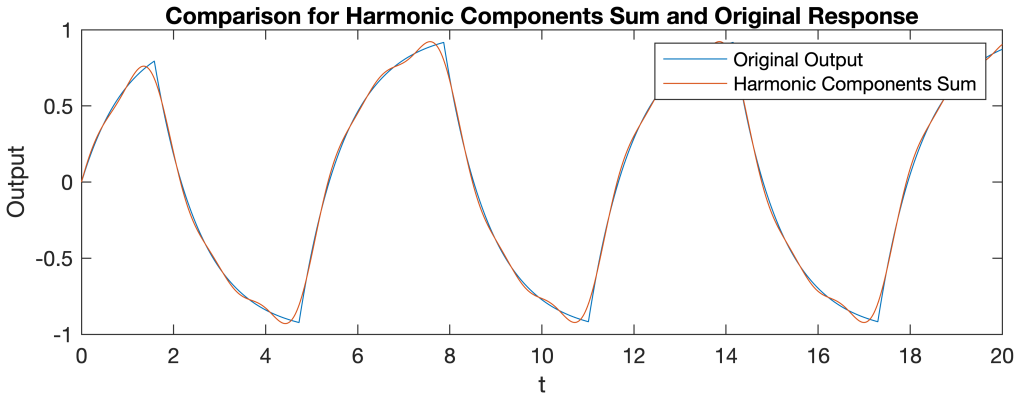
```
t = linspace(0, 20, 1000);
% square wave
x2 = cos(t);
x2(x2 > 0) = ones(size(x2(x2 > 0)));
x2(x2 < 0) = -ones(size(x2(x2 < 0)));
```

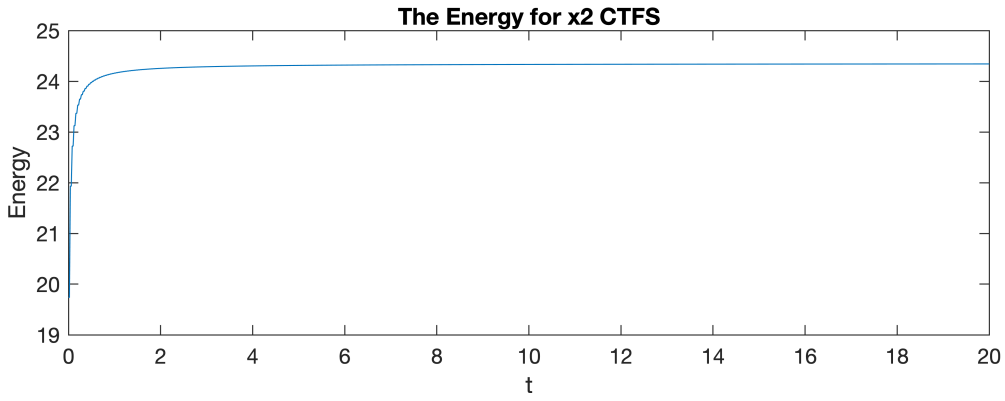
```
apos_k = [];
aneg_k = [];

% coefficients for x2 CTFS
for i = 1:5
    apos_k(i) = 2 * sin(pi * i / 2) / (pi * i);
    aneg_k(i) = 2 * sin(-pi * i / 2) / (-pi * i);
end

s1 = apos_k(1) * exp(j * t) + aneg_k(1) * exp(-j * t);
s2 = apos_k(2) * exp(j * 2 * t) + aneg_k(2) * exp(-j * 2 * t);
s3 = apos_k(3) * exp(j * 3 * t) + aneg_k(3) * exp(-j * 3 * t);
s4 = apos_k(4) * exp(j * 4 * t) + aneg_k(4) * exp(-j * 4 * t);
s5 = apos_k(5) * exp(j * 5 * t) + aneg_k(5) * exp(-j * 5 * t);
ssum = real(s1 + s2 + s3 + s4 + s5);

% The two outputs
b = 1;
a = [1 1];
ysum = lsim(b, a, ssum, t);
y = lsim(b, a, x2, t);

%plot
subplot(2, 1, 1), plot(t, y)
hold on
plot(t, ysum)
xlabel('t')
ylabel('Output')
title('Comparison for Harmonic Components Sum and Original Response')
legend('Original Output', 'Harmonic Components Sum')

% energy
k = 1000;
E = zeros(1, 1000);
e = 0;

for i = 1:k
    e = e + (sin(i * pi / 2) / i * pi)^2 + (sin(i * (-pi) / 2) / i * (-pi))^2
    E(i) = e;
end

subplot(2, 1, 2)
plot(t, E)
xlabel('t')
ylabel('Energy')
title('The Energy for x2 CTFS')
```

Advanced Problem f

We use the same method as produce components of `x2` to find out the components of `y2`.

The plot is shown below.





Because some components are too small, and when using different type of way to compute the outputs will have some small difference, so the plots are not exactly the same, but they can also prove the correctness for our original results.

The MATLAB script is shown below.

```
t = linspace(0, 20, 1000);

% square wave
x2 = sign(cos(t));
y2 = lsim(1, [1 1], x2, t);

ak = 1 / (2 * pi) * fft(x2);
bk = 1 / (2 * pi) * fft(y2);

% coefficients for x2 CTFS
for k = 1:10
    val = ak(k);

    if ak(k) < 0
        apos_k = horzcat(apos_k, val);
    else
        aneg_k = horzcat(aneg_k, val);
    end

end

s1 = apos_k(1) * exp(j * t) + aneg_k(1) * exp(-j * t);
s2 = apos_k(2) * exp(j * 2 * t) + aneg_k(2) * exp(-j * 2 * t);
s3 = apos_k(3) * exp(j * 3 * t) + aneg_k(3) * exp(-j * 3 * t);
s4 = apos_k(4) * exp(j * 4 * t) + aneg_k(4) * exp(-j * 4 * t);
s5 = apos_k(5) * exp(j * 5 * t) + aneg_k(5) * exp(-j * 5 * t);
```

```
ssum = real(s1 + s2 + s3 + s4 + s5);

% coefficients for y CTFS
for k = 1:10
    val = bk(k);

    if bk(k) < 0
        bpos_k = horzcat(bpos_k, val);
    else
        bneg_k = horzcat(bneg_k, val);
    end

end

ys1 = real(bpos_k(1) * exp(j * t) + bneg_k(1) * exp(-j * t));
ys2 = real(bpos_k(2) * exp(j * 2 * t) + bneg_k(2) * exp(-j * 2 * t));
ys3 = real(bpos_k(3) * exp(j * 3 * t) + bneg_k(3) * exp(-j * 3 * t));
ys4 = real(bpos_k(4) * exp(j * 4 * t) + bneg_k(4) * exp(-j * 4 * t));
ys5 = real(bpos_k(5) * exp(j * 5 * t) + bneg_k(5) * exp(-j * 5 * t));

% Output
b = 1;
a = [1 1];

ya1 = lsim(b, a, s1, t);
ya2 = lsim(b, a, s2, t);
ya3 = lsim(b, a, s3, t);
ya4 = lsim(b, a, s4, t);
ya5 = lsim(b, a, s5, t);
y = lsim(b, a, x2, t);

% plot
subplot(5, 1, 1)
plot(t, ya1)
hold on
plot(t, ys1)
xlim([10 20])
xlabel('t')
ylabel('Output')
legend('y1', 'ys1')

subplot(5, 1, 2)
plot(t, ya2)
hold on
plot(t, ys2)
xlim([10 20])
xlabel('t')
ylabel('Output')
legend('y2', 'ys2')

subplot(5, 1, 3)
plot(t, ya3)
hold on
plot(t, ys3)
xlim([10 20])
xlabel('t')
ylabel('Output')
```

```
legend('y3', 'ys3')

subplot(5, 1, 4)
plot(t, ya4)
hold on
plot(t, ys4)
xlim([10 20])
xlabel('t')
ylabel('Output')
legend('y4', 'ys4')

subplot(5, 1, 5)
plot(t, ya5)
hold on
plot(t, ys5)
xlim([10 20])
xlabel('t')
ylabel('Output')
legend('y5', 'ys5')
```

3.10

The DTFS for a periodic discrete-time signal with fundamental period N is given as

$$a_k = \frac{1}{N} \sum_{n=0}^{N-1} x[n] e^{-jk(2\pi/N)n}$$

Advanced Problem a

In total, there are N Fourier series coefficients needed to be calculated, hence the the total number of operations for calculating the DTFS for a periodic discrete-time signal with fundamental period N is

$$n = N(N + 1 + N - 1) = 2N^2.$$

Advanced Problem b

The MATLAB script is shown in the code block below.

```
dtfscomps = zeros(1, 5);
N = [8 32 64 128 256] .* 10;

for index = 1:size(N, 2)
    x = (0.9).^(0:N(index) - 1);
    clear X
    tic
    X = dtfs(x, 0);
    dtfscomps(index) = toc;
end

disp(dtfscomps)
```

> For better visualization, the period is increased by 10 times.

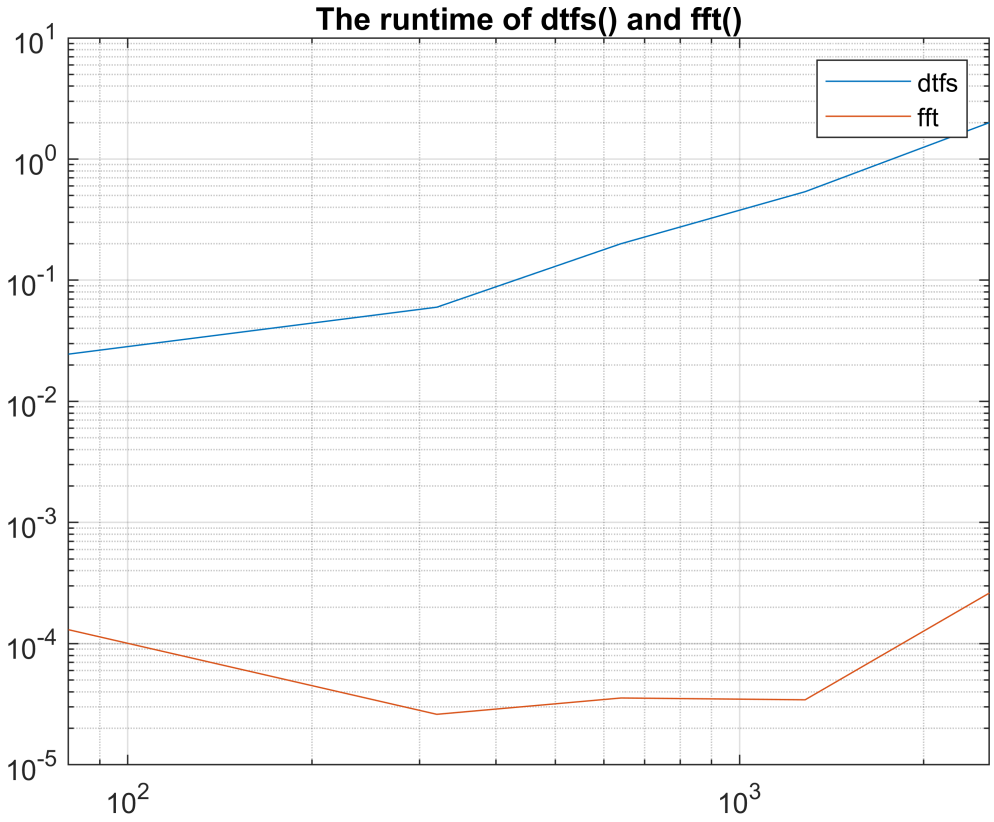
Advanced Problem c

The MATLAB script is shown in the code block below.

```
fftcomps = zeros(1, 5);
N = [8 32 64 128 256] .* 10;

for index = 1:size(N, 2)
    x = (0.9).^(0:N(index) - 1);
    clear X
    tic
    X = fft(x, 0);
    fftcomps(index) = toc;
end

disp(fftcomps)

A3_3_10_b
loglog(N, dtfscomps)
hold on
loglog(N, fftcomps)
legend('dtfs', 'fft')
title('The runtime of dtfs() and fft()')
grid on
```

> For better visualization, the period is increased by 10 times.

The plot is shown below.



According to the plot, we can see that the runtime of `fft()` is always lower than that of `dtfs()`, and increases with period N slower than `dtfs()`, which indicates savings are given by FFT when N is large.

Advanced Problem d

The period of $y[n]$ should also be $N_y = N$. In each shift, there should be i times of multiplication and $i-1$ times of addition, hence the total number of operations is

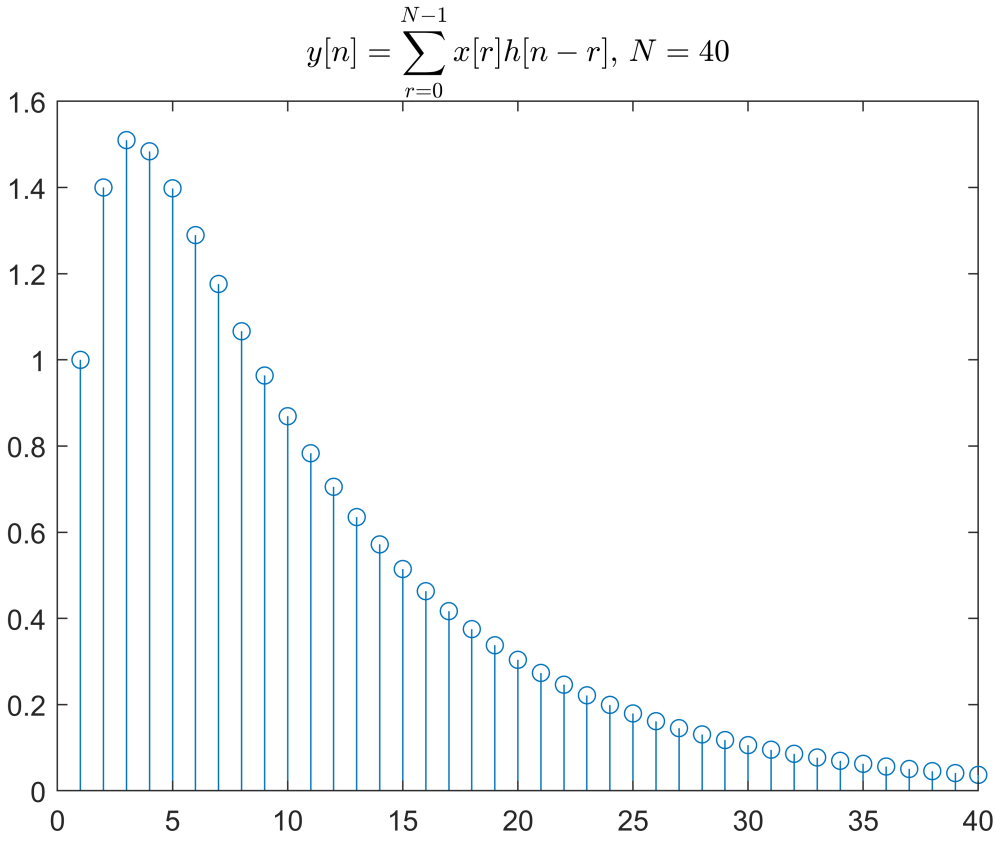
$$\sum_{1}^{N}(i+i-1)+\sum_{N}^{1}(i+i-1)-(N+N-1)=2n^2-2N+1=O(N^2)$$

Advanced Problem e

The MATLAB script of this part is shown in the code block below.

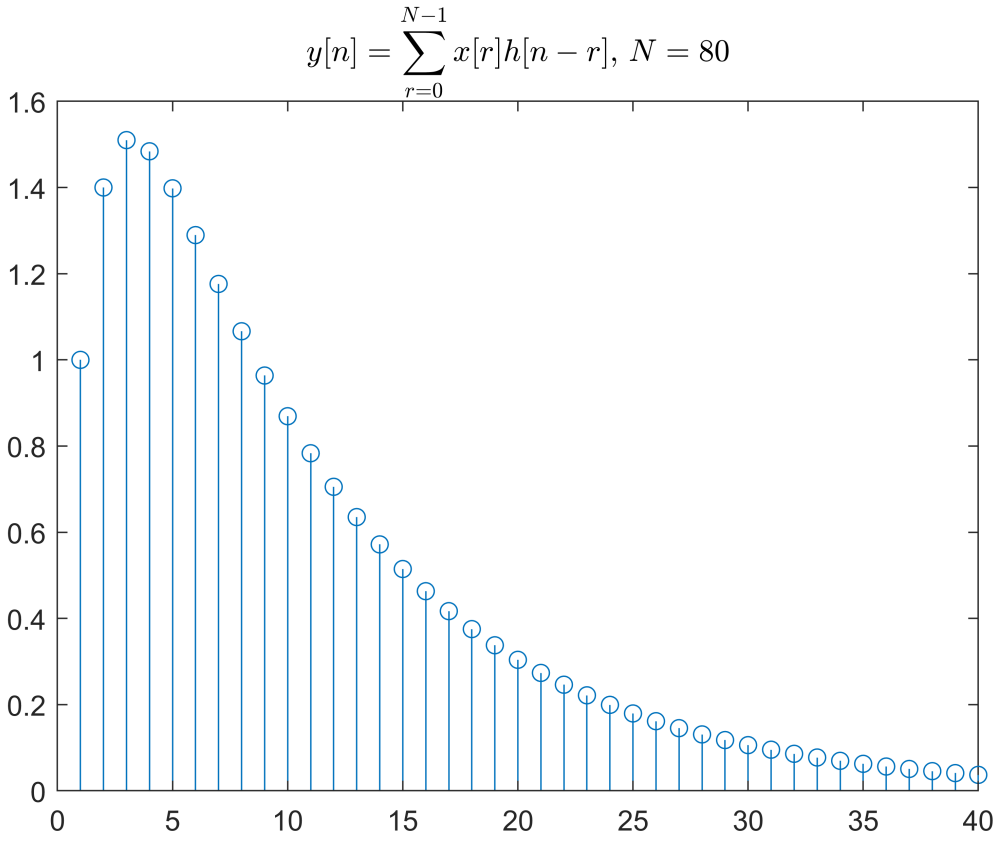
```
%% Problem e
N = 40;
n = 0:N - 1;
x = 0.9.^n;
h = 0.5.^n;
ny = 0:N - 1;

tic
y = conv([x x], h);
y_f40c = y(1, 1:N);
f40c = toc;

figure
stem(ny, y_f40c)
title('$$y[n]=\sum_{r=0}^{N-1}x[r]h[n-r]$$, $$N=40$$', 'Interpreter', 'latex')
```

The plot is shown below.



Advanced Problem f

The MATLAB script is identical with that in Problem e, but replacing the period with $N = 80$. The plot is shown below.

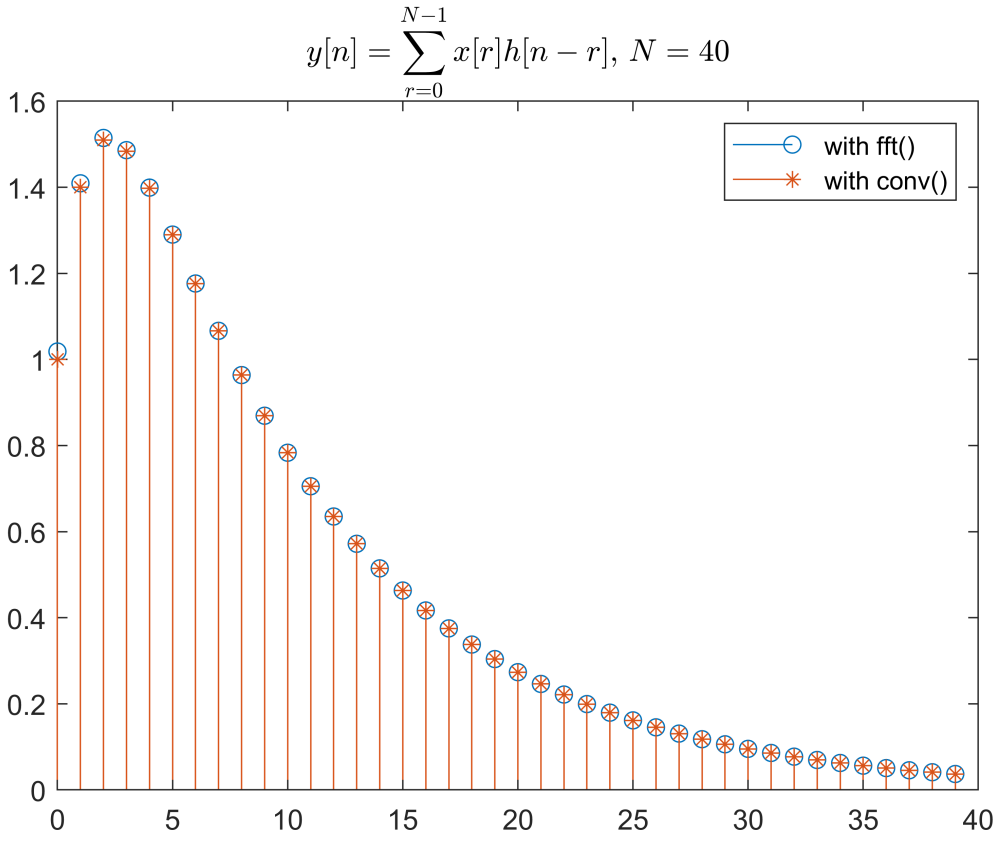


Advanced Problem g

The MATLAB script of this part is shown in the code block below.

```
%% Problem g
N = 40;
n = 0:N - 1;
x = 0.9.^n;
h = 0.5.^n;
ny = 0:N - 1;

tic
ax = fft(x, N);
ah = fft(h, N);
ay = ax .* ah;
y_f40f = ifft(ay, N);
f40f = toc;

figure
stem(ny, y_f40f)
hold on
stem(ny, y_f40c, '*')
legend('with fft()', 'with conv()')
title('$$y[n]=\sum_{r=0}^{N-1}x[r]h[n-r]$$, $$N=40$$', 'Interpreter', 'latex')
```

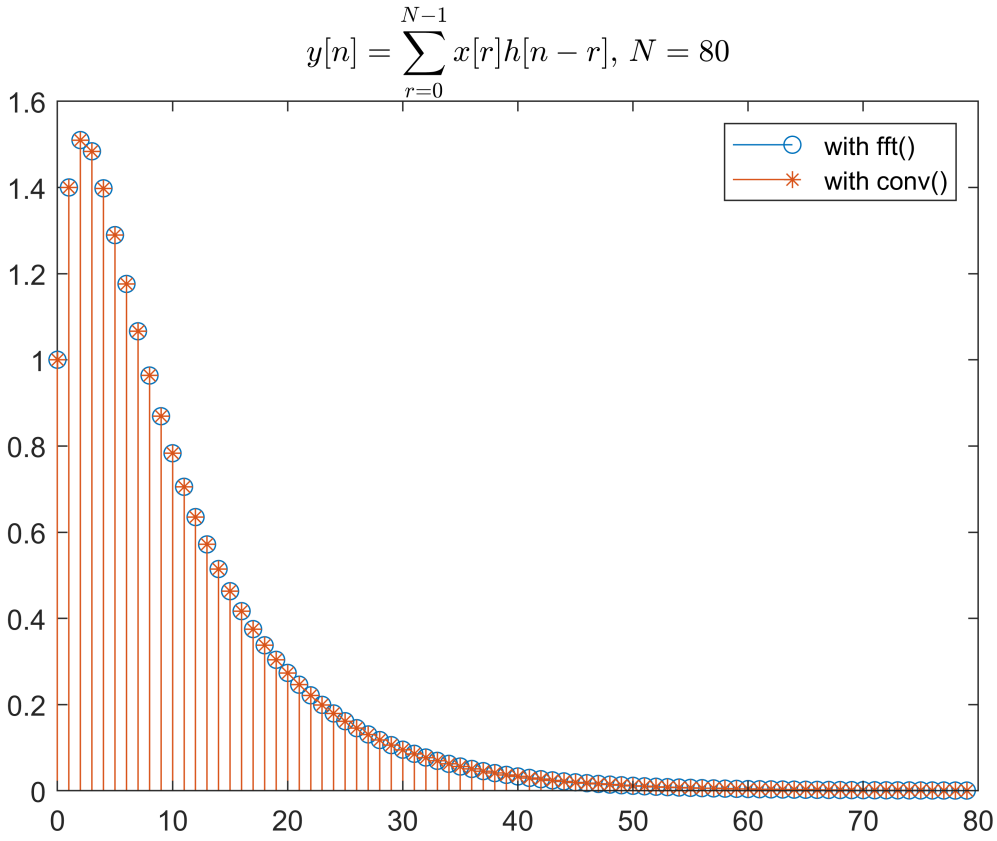
The plot is shown below.



According to the plot, the result obtained from `conv()` and `fft()` with `ifft()` is almost identical.

Advanced Problem h

The MATLAB script is identical with that in Problem g, but replacing the period with $N = 80$. The plot is shown below.



According to the plot, the result obtained from `conv()` and `fft()` with `ifft()` is again almost identical.

Advanced Problem i

The ratio is `f40c / f40f = 1.4660`, `f80c / f80f = 1.7016`. For larger N, the ratio is larger. Calculate period convolution with FFT would be more efficient. Consider a even larger period $N = 160$, the MATLAB script is shown in the code block below.

```
N = 160;
n = 0:N - 1;
x = 0.9.^n;
h = 0.5.^n;
ny = 0:N - 1;

tic
y = conv([x x], h);
y_f160c = y(1, 1:N);
f160c = toc;

tic
ax = fft(x, N);
ah = fft(h, N);
ay = ax .* ah;
y_f160f = ifft(ay, N);
```

```
f160f = toc;

display(f160c / f160f)
```

The result is `f160c / f160f = 1.9639`, the ratio is even larger. The FFT method is exactly what we would choose for $N > 80$ and any other larger Ns.