LAB3 MATLAB Programming

涂峻绫(12213010) and 欧阳安男(12211831)

Contents

Introduction

MATLAB is a powerful tool.

In this lab, firstly, we are going to explore the properties of systems, such as highpass, lowpass or bandpass by using ff and iff.

Secondly, we are going to explore the FFT algorithm for the computing the DTFS by using dtfs and fft

Results and Analysis

3.8

(a)

■ 3.8 First-Order Recursive Discrete-Time Filters

This exercise demonstrates the effect of first-order recursive discrete-time filters on periodic signals. You will examine the frequency responses of two different systems and also construct a periodic signal to use as input for these systems. This exercise assumes you are familiar with using **fft** and **ifft** to compute the DTFS of a periodic signal as described in Tutorial 3.1. In addition, it is also assumed you are proficient with the **filter** and **freqz** commands described in Tutorials 2.2 and 3.2. Several parts of this exercise require you to generate vectors which should be purely real, but have very small imaginary parts due to roundoff errors. Use **real** to remove these residual imaginary parts from these vectors.

This exercise focuses on two causal LTI systems described by first-order recursive difference equations:

$$\begin{aligned} \text{System 1:} \quad y[n] - 0.8y[n-1] &= x[n], \\ \text{System 2:} \quad y[n] + 0.8y[n-1] &= x[n]. \end{aligned}$$

The input signal $x[n]$ will be the periodic signal with period $N = 20$ described by the DTFS coefficients

$$a_k = \begin{cases} 3/4\,, & k = \pm 1\,, \\ -1/2\,, & k = \pm 9\,, \\ 0\,, & \text{otherwise}\,. \end{cases} \tag{3.10}$$

Intermediate Problems

(a). Define vectors **a1** and **b1** for the difference equation describing System 1 in the format specified by **filter** and **freqz**. Similarly, define **a2** and **b2** to describe System 2.

Initialize.

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

Create vectors a_1, b_1 and a_2, b_2.

```
a1 = [1 -0.8]
```

```
a1 = 1×2
    1.0000   -0.8000
```

```
b1 = 1
```

```
b1 = 1
```

```
a2 = [1 0.8]
```

```
a2 = 1×2
    1.0000    0.8000
```

```
b2 = 1
```

```
b2 = 1
```

(b)

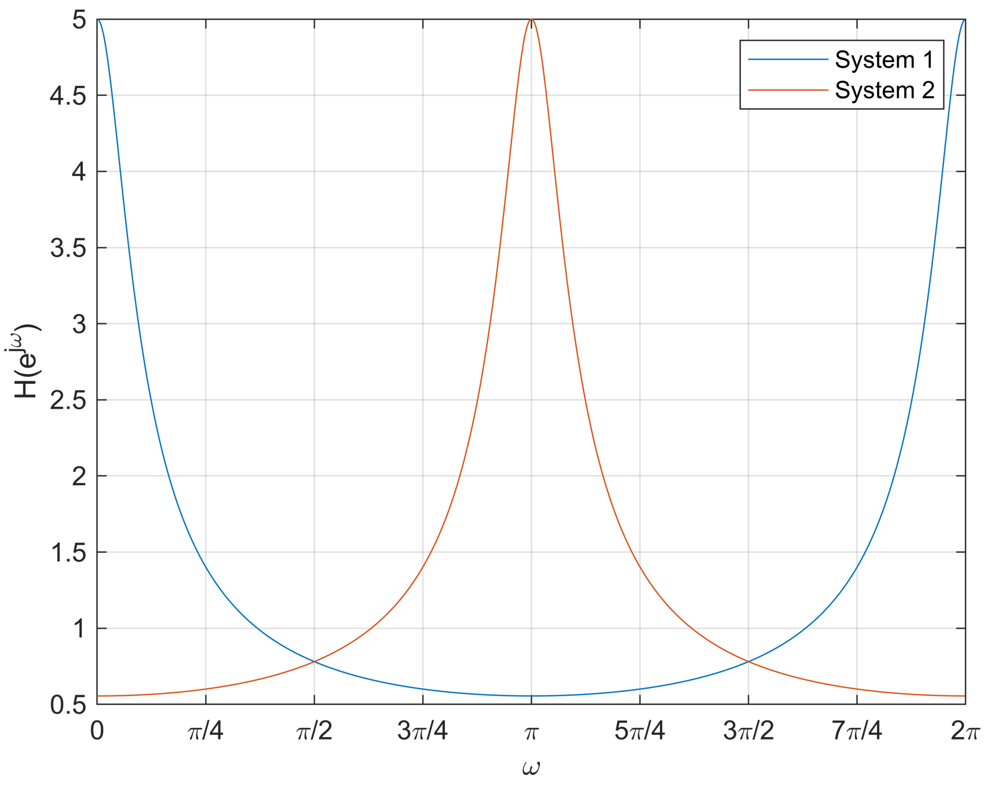
(b). Use **freqz** to evaluate the frequency responses of Systems 1 and 2 at 1024 points between 0 and 2π. Note that you will have to use the **'whole'** option with **freqz** to do this. Use **plot** and **abs** to generate appropriately labeled graphs of the magnitude of the frequency response for both systems. Based on the frequency response plots, specify whether each system is a highpass, lowpass, or bandpass filter.

Calculate $H_1, \omega_1, H_2, \omega_2$.

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

Plot frequency responese for both systems.

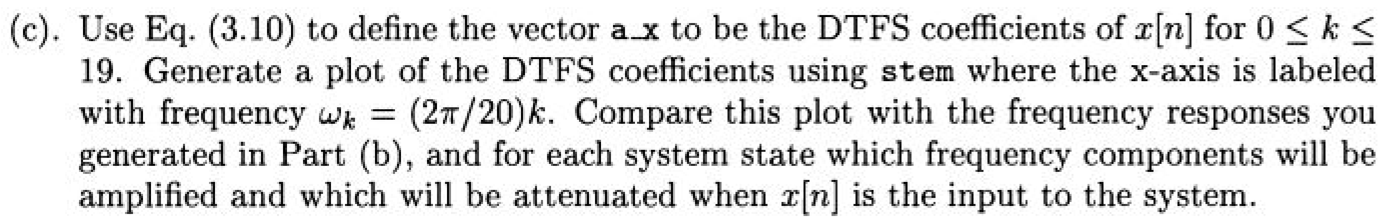
```
figure;
plot(omega1,abs(H1));
hold on;
plot(omega2,abs(H2));
grid on;
axis([0 2*pi ylim]);
xlabel('\omega');
ylabel('H(e^{j\omega})');
xticks(0:pi/4:2*pi);
xticklabels({'0' '\pi/4' '\pi/2' '3\pi/4' '\pi' '5\pi/4' '3\pi/2' '7\pi/4' '2\pi'});
legend('System 1','System 2');
```



The frequency responese for system 1 is high when ω close to 0 so system 1 is lowpass filter.

The frequency responese for system 2 is high when ω close to π so system 2 is highpass filter.

(c)



(c). Use Eq. (3.10) to define the vector **a_x** to be the DTFS coefficients of $x[n]$ for $0 \leq k \leq 19$. Generate a plot of the DTFS coefficients using **stem** where the x-axis is labeled with frequency $\omega_k = (2\pi/20)k$. Compare this plot with the frequency responses you generated in Part (b), and for each system state which frequency components will be amplified and which will be attenuated when $x[n]$ is the input to the system.

Define a_x.

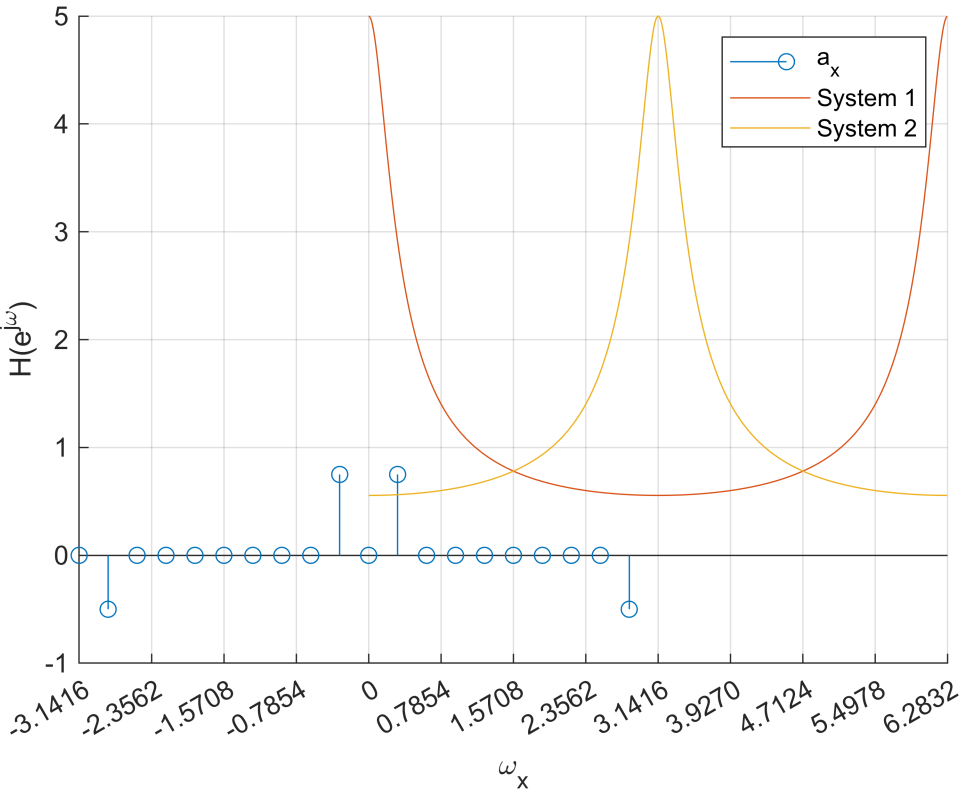
```
a_x = [0 -1/2 zeros(1,7) 3/4 0 3/4 zeros(1,7) -1/2]
```

```
a_x = 1×20
         0   -0.5000         0         0         0         0         0         0···
```

```
k = -pi:2*pi/20:18*pi/20;
```

Generate the plot of the DTFS coefficients and the frequency responses.

```
figure;hold on;grid on;
stem(k,a_x);
plot(omega1,abs(H1));
plot(omega2,abs(H2));
xlabel('\omega_x');
ylabel('H(e^{j\omega})');
axis([-pi 2*pi ylim]);
xticks(-2*pi:pi/4:2*pi);
legend('a_x','System 1','System 2');
```



Shown as the plot, we can tell the conclusion.

For System 1, the frequency components of $x[n]$ will be amplified and when k = ±1 and will be attenuated when k = ±9.

For System 2, the frequency components of $x[n]$ will be amplified and when k = ±9 and will be attenuated when k = ±1.

(d)

(d). Define **x_20** to be one period of $x[n]$ for $0 \le n \le 19$ using **ifft** with **a_x** as specified in Tutorial 3.1. Use **x_20** to create **x**, consisting of 6 periods of $x[n]$ for $-20 \le n \le 99$. Also define **n** to be this range of sample indices, and generate a plot of $x[n]$ on this interval using **stem**.

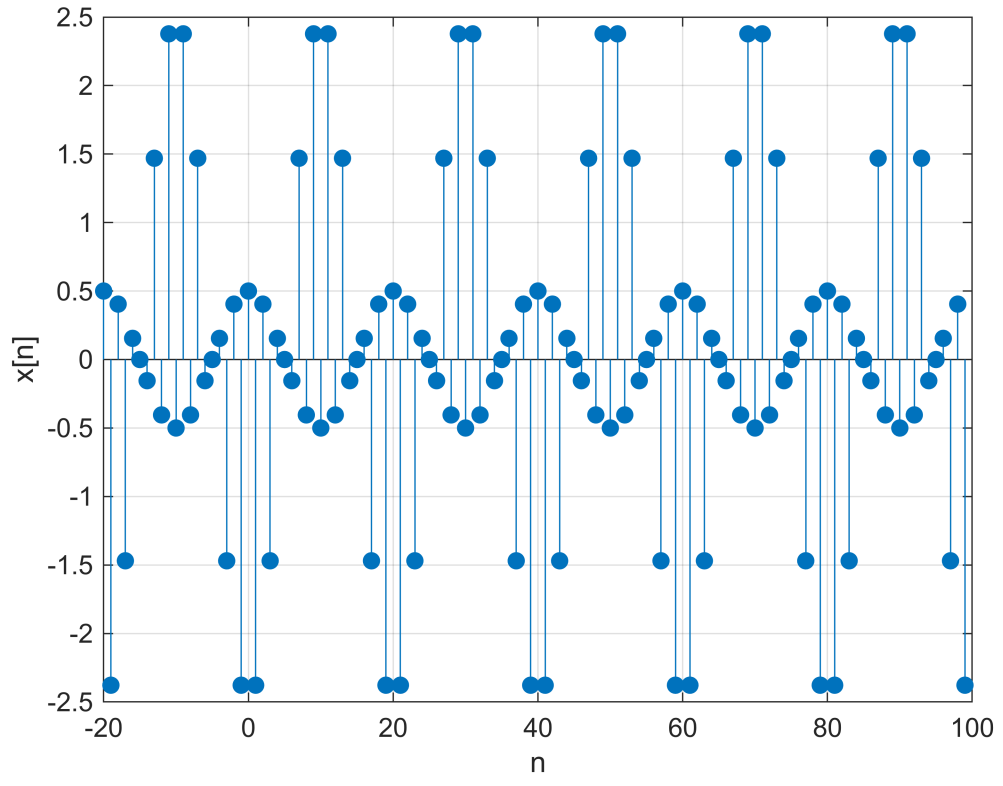
Define x_20, x and n.

```
x_20 = 20*ifft(a_x,20);
x = [x_20 x_20 x_20 x_20 x_20 x_20];
n = -20:99;
```

Generate the plot of $x[n]$ for $-20 \le n \le 99$.

```
figure;
```

```
stem(n,real(x),'filled');
grid on;
xlabel('n');ylabel('x[n]');
```



(e)

(e). Use **filter** to compute **y1** and **y2**, the outputs of Systems 1 and 2 when $x[n]$ is the input. Plot both outputs for $0 \leq n \leq 99$ using **stem**. Based on these plots, state which output contains more high frequency energy and which has more low frequency energy. Do the plots confirm your answers in Part (c)?

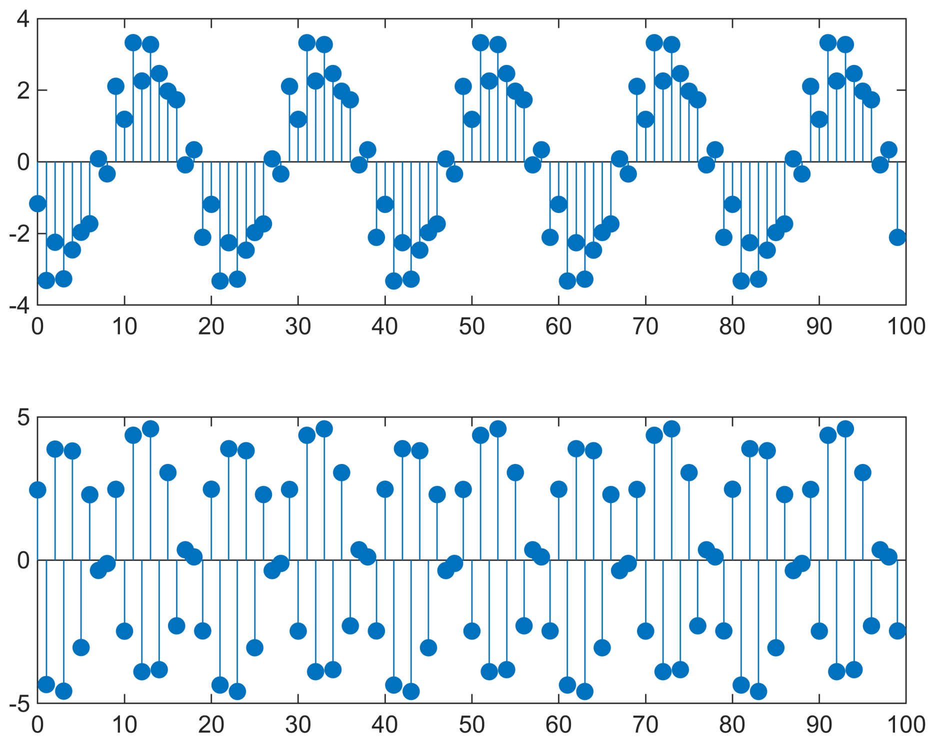
Compute y1 and y2.

```
y1 = filter(b1,a1,x);
y2 = filter(b2,a2,x);
```

Get the outputs for $0 \leq n \leq 99$, and generate the plot of y1 and y2.

```
ny = 0:99;
y1 = y1(end-99:end);
y2 = y2(end-99:end);
figure;
grid on;
subplot(2,1,1);
stem(ny,real(y1),'filled');
```

```
subplot(2,1,2);
stem(ny,real(y2),'filled');
```



The plots confirm the answers in Part (c).

(f)

(f). Define **y1_20** and **y2_20** to be the segments of **y1** and **y2** corresponding to $y_1[n]$ and $y_2[n]$ for $0 \leq n \leq 19$. Use these vectors and **fft** to compute **a_y1** and **a_y2**, the DTFS coefficients of **y1** and **y2**. Use **stem** and **abs** to generate plots of the magnitudes of the DTFS coefficients for both sequences. Do these plots agree with your answers in Part (e)?

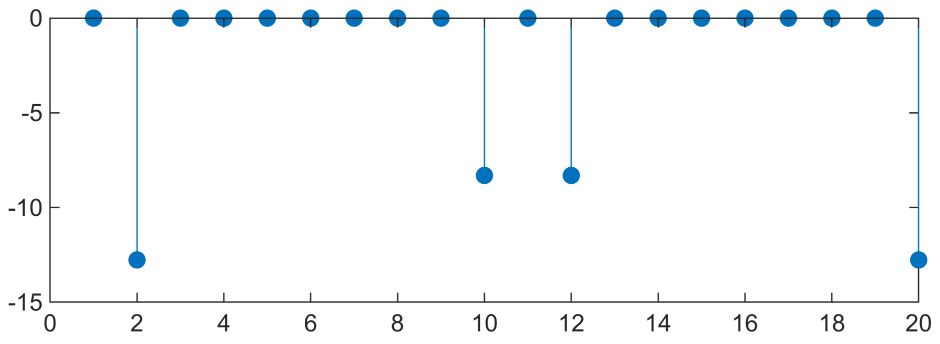
Define $y1\_20$ and $y2\_20$.

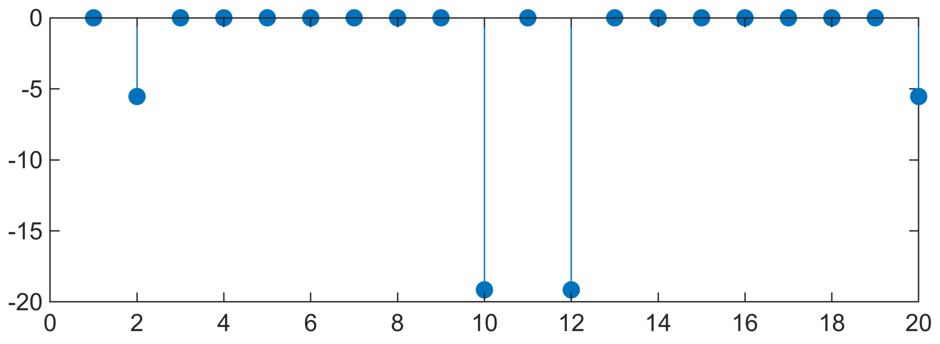
```
y1_20 = y1(20:39);
y2_20 = y2(20:39);
```

Compute a_y1 and a_y2, and generate the plots of the DTFS coefficients for both sequences.

```
a_y1 = fft(y1_20);
a_y2 = fft(y2_20);
k_ay = -10:9;
figure;
tiledlayout(2,1);
```

```
nexttile;
stem(real(a_y1),'filled');
nexttile;
stem(real(a_y2),'filled');
```





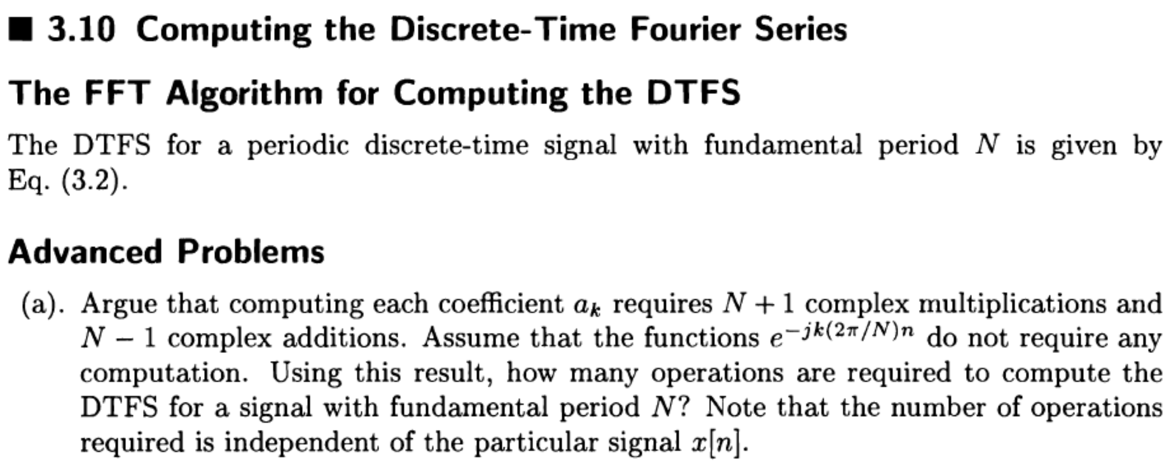
Shown as the plots, the coefficients of outputs confirm the answers in Part (e).

For System 1, the frequency components will be amplified and when k = ± 1 and will be attenuated when k = ± 9.

For System 2, the frequency components will be amplified and when k = ± 9 and will be attenuated when k = ± 1.

3.10

(a)



■ 3.10 Computing the Discrete-Time Fourier Series

The FFT Algorithm for Computing the DTFS

The DTFS for a periodic discrete-time signal with fundamental period N is given by Eq. (3.2).

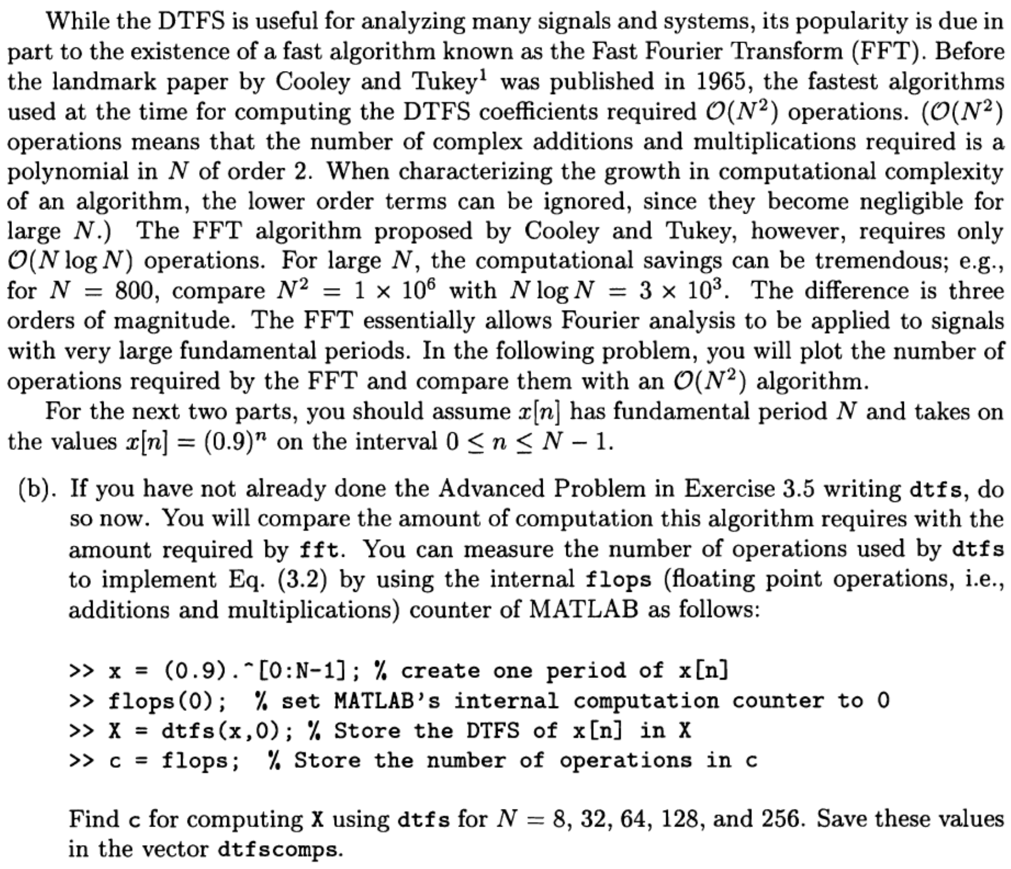
Advanced Problems

(a). Argue that computing each coefficient a_k requires $N+1$ complex multiplications and $N-1$ complex additions. Assume that the functions $e^{-jk(2\pi/N)n}$ do not require any computation. Using this result, how many operations are required to compute the DTFS for a signal with fundamental period N? Note that the number of operations required is independent of the particular signal $x[n]$.

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

Need $N(N+1)$ complex multiplications and $N(N-1)$ complex additions.

(b)



While the DTFS is useful for analyzing many signals and systems, its popularity is due in part to the existence of a fast algorithm known as the Fast Fourier Transform (FFT). Before the landmark paper by Cooley and Tukey[1] was published in 1965, the fastest algorithms used at the time for computing the DTFS coefficients required $\mathcal{O}(N^2)$ operations. ($\mathcal{O}(N^2)$ operations means that the number of complex additions and multiplications required is a polynomial in N of order 2. When characterizing the growth in computational complexity of an algorithm, the lower order terms can be ignored, since they become negligible for large N.) The FFT algorithm proposed by Cooley and Tukey, however, requires only $\mathcal{O}(N \log N)$ operations. For large N, the computational savings can be tremendous; e.g., for $N = 800$, compare $N^2 = 1 \times 10^6$ with $N \log N = 3 \times 10^3$. The difference is three orders of magnitude. The FFT essentially allows Fourier analysis to be applied to signals with very large fundamental periods. In the following problem, you will plot the number of operations required by the FFT and compare them with an $\mathcal{O}(N^2)$ algorithm.

For the next two parts, you should assume $x[n]$ has fundamental period N and takes on the values $x[n] = (0.9)^n$ on the interval $0 \le n \le N-1$.

(b). If you have not already done the Advanced Problem in Exercise 3.5 writing **dtfs**, do so now. You will compare the amount of computation this algorithm requires with the amount required by **fft**. You can measure the number of operations used by **dtfs** to implement Eq. (3.2) by using the internal **flops** (floating point operations, i.e., additions and multiplications) counter of MATLAB as follows:

```
>> x = (0.9).^[0:N-1]; % create one period of x[n]
>> flops(0);  % set MATLAB's internal computation counter to 0
>> X = dtfs(x,0); % Store the DTFS of x[n] in X
>> c = flops;  % Store the number of operations in c
```

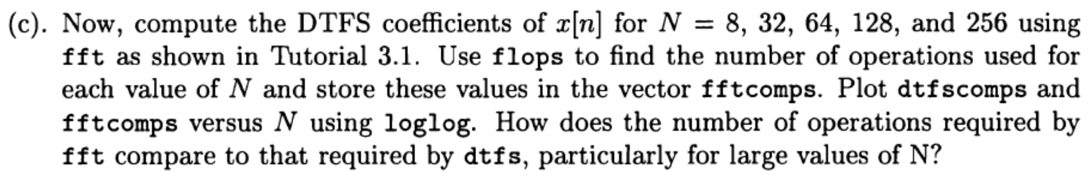
Find c for computing X using dtfs for $N = 8, 32, 64, 128$, and 256. Save these values in the vector dtfscomps.

Since flops has been removed so use the execution time to be the substitude.

```
N_list = [8 32 64 128 256];
dtfscomps = zeros(1,length(N_list));
for i = 1:5
```

```
    N = N_list(i);
    x = (0.9).^(0:N-1);
    tic;
    dtfs(x,0);
    dtfscomps(i) = toc;
end
```

(c)

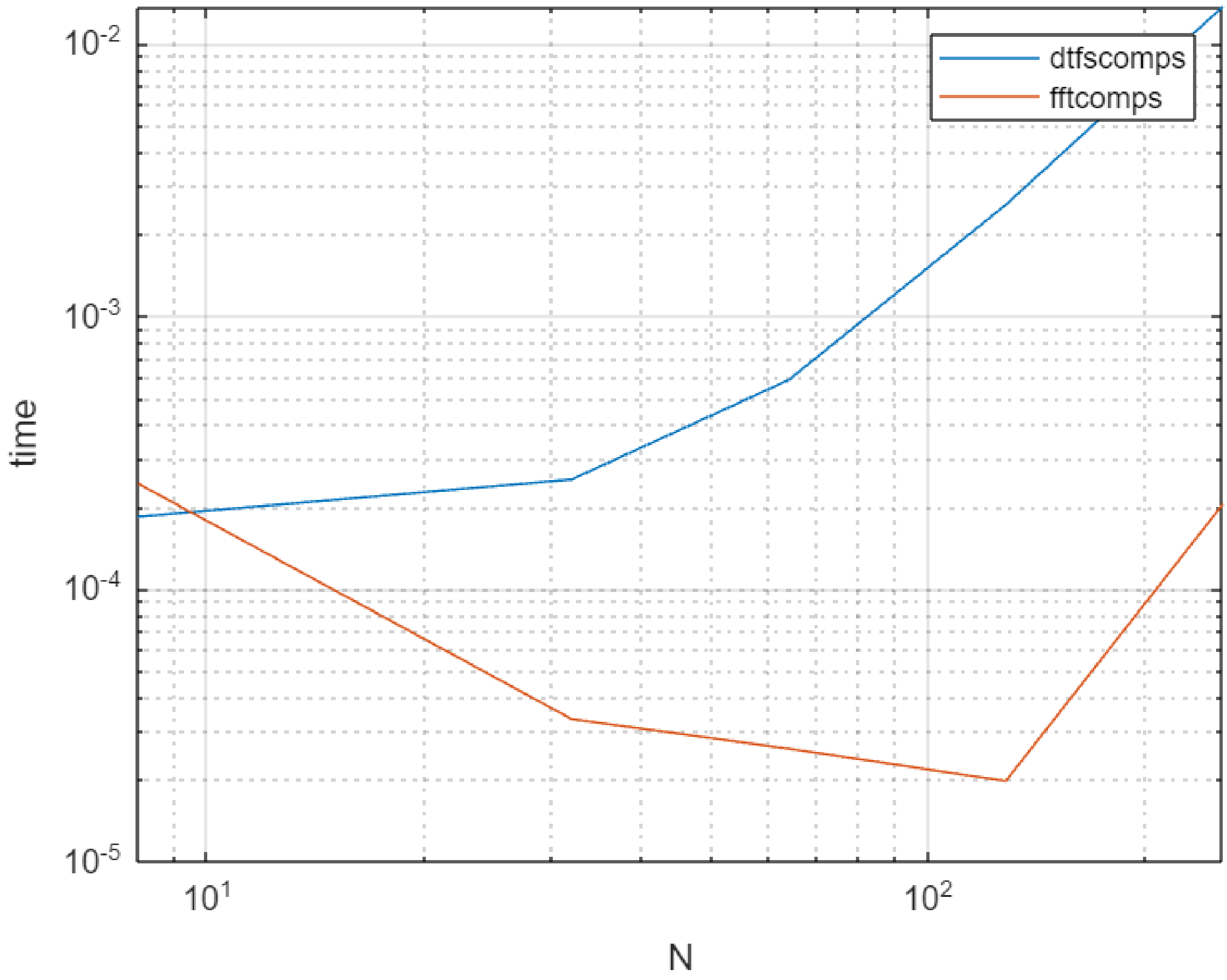

(c). Now, compute the DTFS coefficients of $x[n]$ for $N = 8$, 32, 64, 128, and 256 using **fft** as shown in Tutorial 3.1. Use **flops** to find the number of operations used for each value of N and store these values in the vector **fftcomps**. Plot **dtfscomps** and **fftcomps** versus N using **loglog**. How does the number of operations required by **fft** compare to that required by **dtfs**, particularly for large values of N?

Since flops has been removed so use the execution time to be the substitude.

```
fftcomps = zeros(1,length(N_list));
for i = 1:5
    N = N_list(i);
    x = (0.9).^(0:N-1);
    tic;
    fft(x,0);
    fftcomps(i) = toc;
end
```

Plot $dtfscomps$ and $ffcomps$.

```
figure;
loglog(N_list,dtfscomps);
hold on;
loglog(N_list,fftcomps);
grid on;
xlabel('N');
ylabel('time');
legend('dtfscomps','fftcomps');
```



According to the figure, the cost time of $dtfs$ increases obviously while ff increases too slowly to be observed.

(d)

Periodic Convolution with the FFT

In many applications, LTI systems are implemented using periodic convolution. In the following problems, you will implement the periodic convolution of two discrete-time signals $x[n]$ and $h[n]$, both with fundamental period N, is

$$y[n] = \sum_{r=0}^{N-1} x[r]h[n-r]\,. \tag{3.11}$$

(d). What is the fundamental period, N_y, of $y[n]$? Argue that directly implementing the periodic convolution according to Eq. (3.11) requires $\mathcal{O}(N^2)$ operations (additions and multiplications). Remember that computing one period of $y[n]$ is sufficient to characterize the entire signal.

Because $h[n]$ has a fundamental period N, $h[n+N-r] = h[n-r]$.

So $y[n] = \sum_{r=0}^{N-1} x[r]h[n-r] = \sum_{r=0}^{N-1} x[r]h[n+N-r] = y[n+N]$, $N_y = N$.

(e)

(e). Assume both $x[n]$ and $h[n]$ have fundamental period $N = 40$, and are given by $x[n] = (0.9)^n$ and $h[n] = (0.5)^n$ over the interval $0 \le n \le N-1$. Compute the periodic convolution of $x[n]$ with $h[n]$ and plot $y[n]$ for $0 \le n \le N_y - 1$. Store the number of operations, given by **flops**, required to implement the convolution in **f40c**. Remember to set **flops(0)** after creating **x** and **h**, the vectors representing $x[n]$ and $h[n]$. Hint: To implement the periodic convolution, first store $x[n]$ and $h[n]$ over the interval $0 \le n \le N-1$ in the row vectors **x** and **h**, respectively. Set **flops(0)** and then use **conv([x x],h)**. The periodic convolution can be extracted from a portion of this signal.

Initialize $N, x[n], h[n]$.

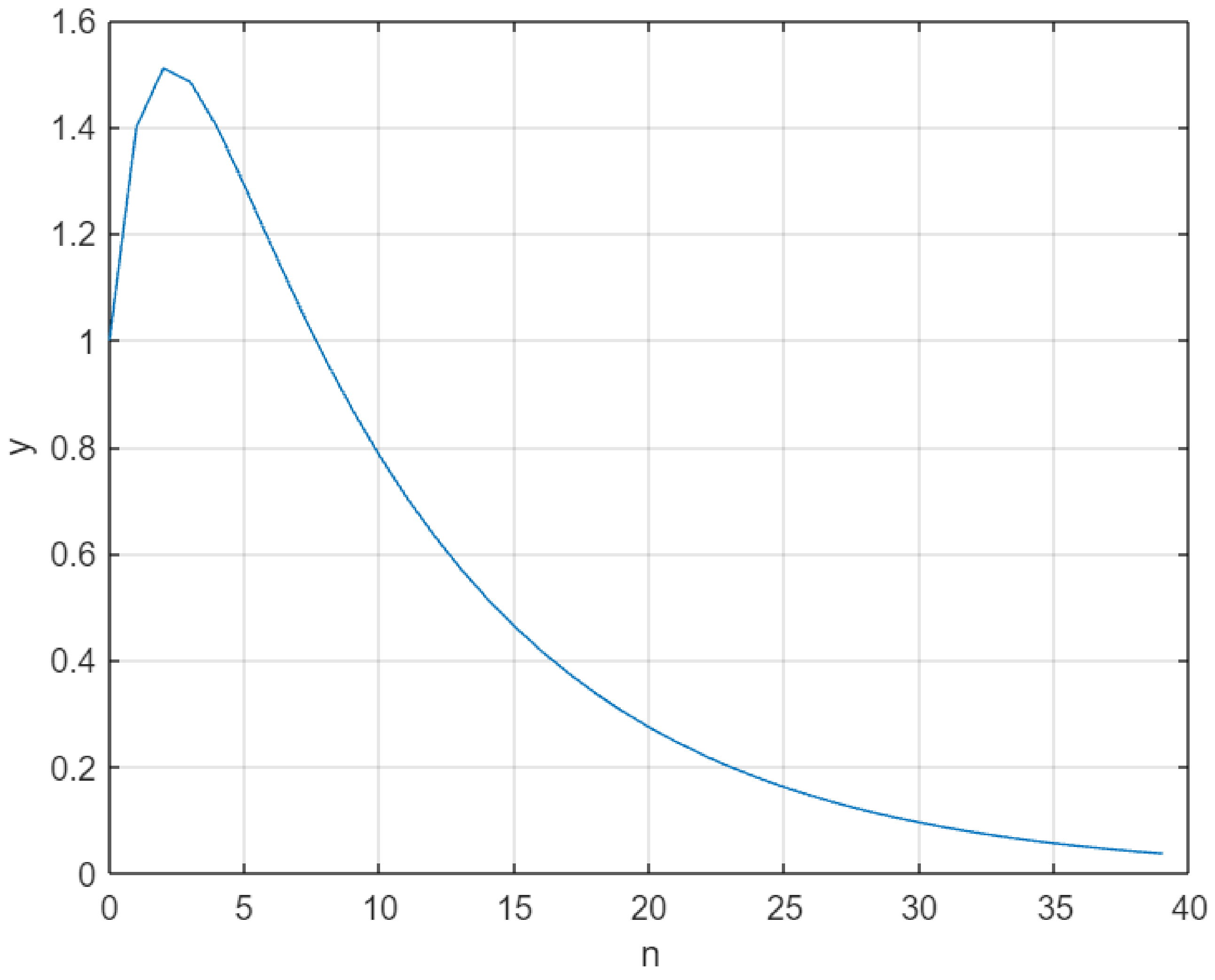
```
N = 40;
x = 0.9.^(0:(N-1));
h = 0.5.^(0:(N-1));
```

Calculate $N_y, y[n], f40c$.

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

Plot $y[n]$.

```
figure;
plot(0:(Ny-1),y);
grid on;
xlabel('n');
ylabel('y');
```



(f)

(f). Repeat Part (e) for $N = 80$, again plotting a period of $y[n]$ and storing the number of operations in `f80c`.

Initialize $N, x[n], h[n]$.

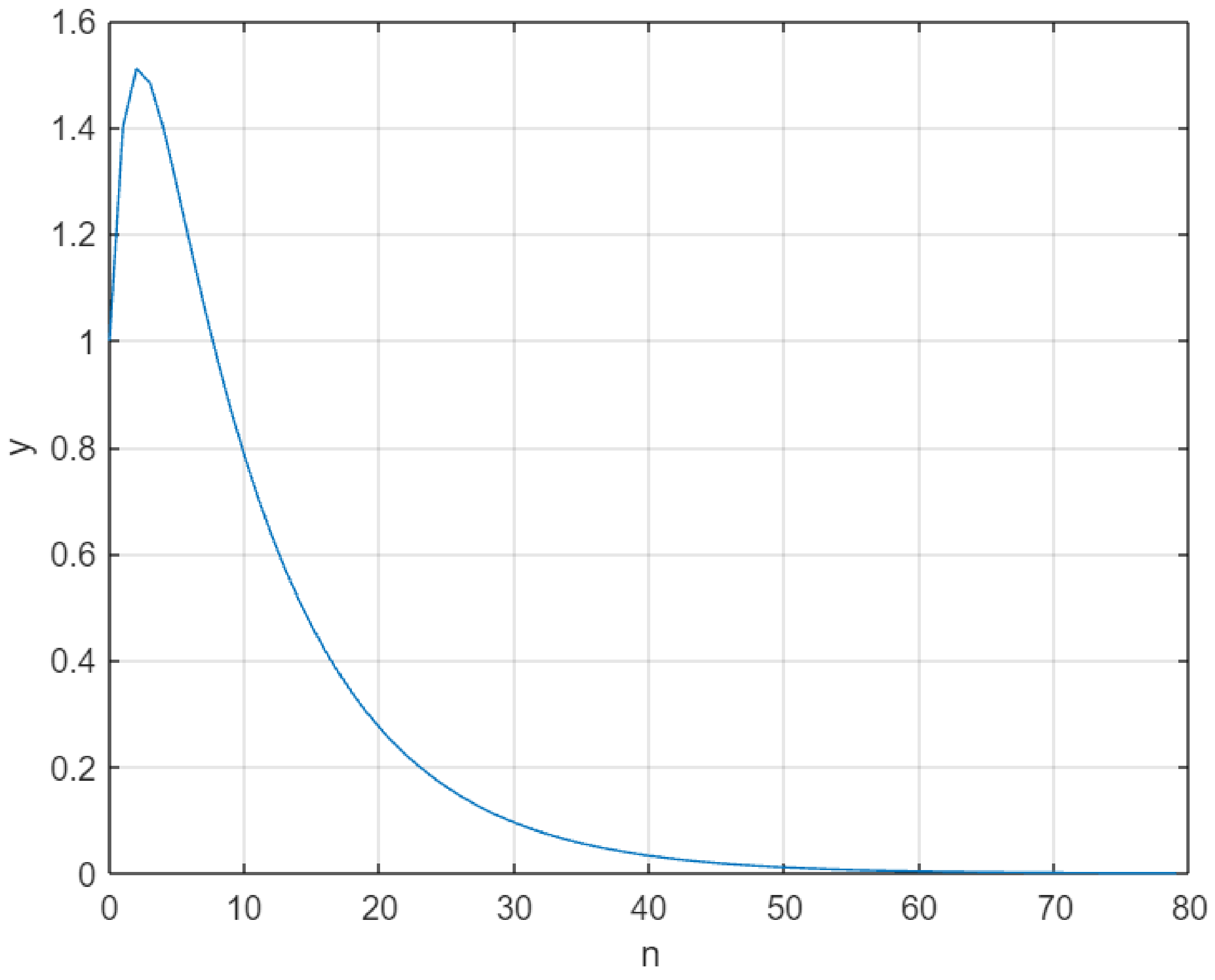
```
N = 80;
x = 0.9.^(0:(N-1));
h = 0.5.^(0:(N-1));
```

Calculate $N_y, y[n], f80c$.

```
Ny = N;
tic;
y = conv([x x],h);
y = y(1:Ny);
f80c = toc;
```

Plot $y[n]$.

```
figure;
plot(0:(Ny-1),y);
grid on;
xlabel('n');
ylabel('y');
```



(g)

(g). Assume $x[n]$ and $h[n]$ are defined as in Part (e). Use the periodic convolution property of the DTFS to implement the periodic convolution[2]. Namely, compute the DTFS coefficients of both $x[n]$ and $h[n]$ using **fft** as described in Tutorial 3.1. Then, use the periodic convolution property to form the DTFS coefficients of $y[n]$. Finally, synthesize $y[n]$ from the DTFS coefficients using **ifft**. The **ifft** algorithm is nearly identical to the FFT, and also requires $\mathcal{O}(N \log N)$ operations for a signal with fundamental period N. To check the validity of your implementation, plot $y[n]$ for $0 \le n \le N_y - 1$ and compare this signal to that computed in Part (e). Remember, as discussed in Tutorial 3.1, the signal $y[n]$ might have a small imaginary component due to numerical round-off errors. Store the total number of operations required to compute $y[n]$ in **f40f**, and remember to set **flops(0)** after creating the representations of $x[n]$ and $h[n]$ in MATLAB.

Initialize $N, x[n], h[n]$.

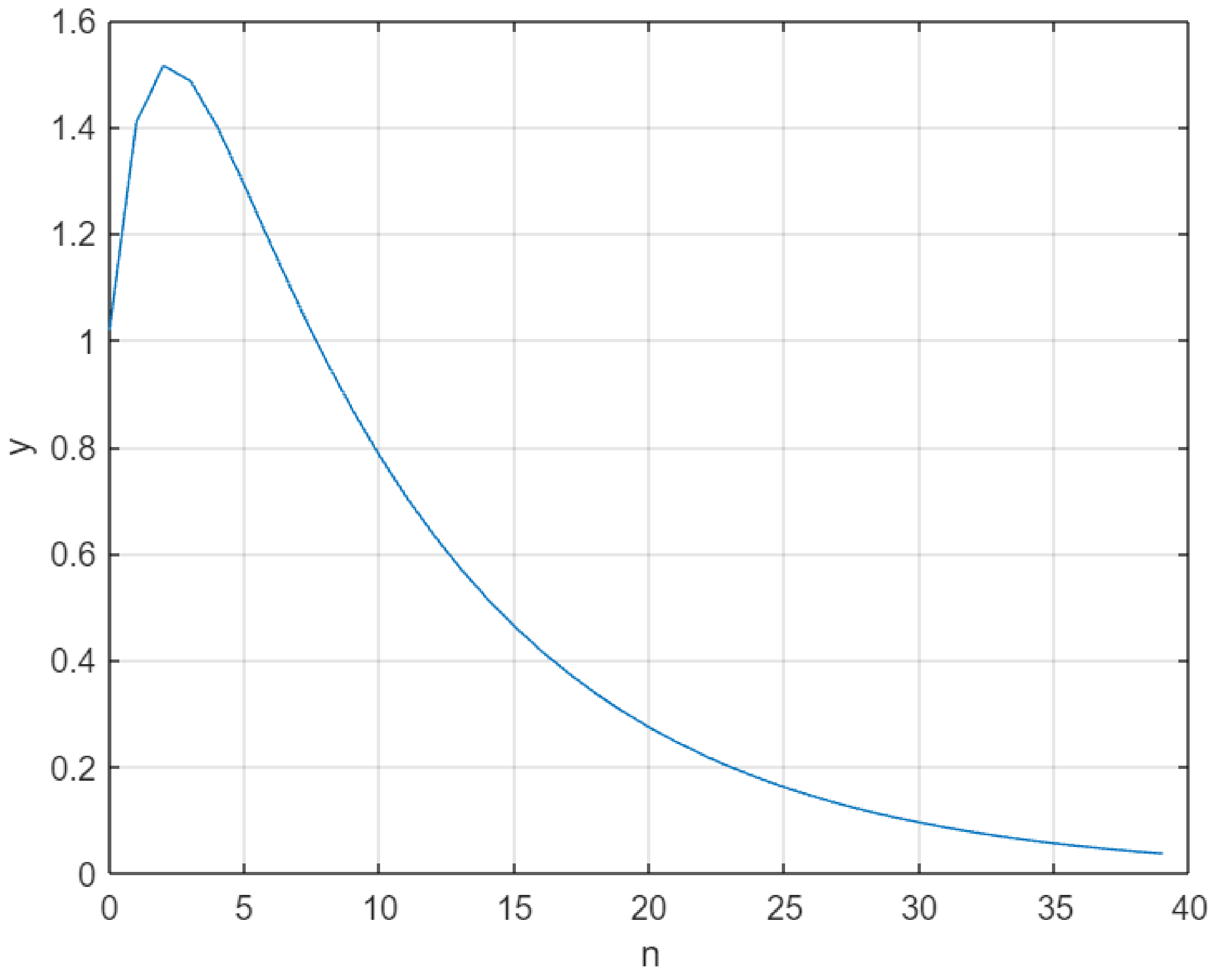
```
N = 40;
x = 0.9.^(0:(N-1));
h = 0.5.^(0:(N-1));
```

Calculate N_y, $y[n]$, $f40f$.

```
Ny = N;
tic;
DTFSx = fft(x);
DTFSh = fft(h);
DTFSy = DTFSx.*DTFSh;
y = ifft(DTFSy);
f40f = toc;
```

Plot $y[n]$.

```
figure;
plot(0:(Ny-1),y);
grid on;
xlabel('n');
ylabel('y');
```



(h)

(h). Repeat Part (g) for $N = 80$, again plotting a period of $y[n]$ and storing the number of operations in **f80f**. Again check the validity of your implementation by comparing $y[n]$ with that computed in Part (f).

Initialize $N, x[n], h[n]$.

```
N = 80;
x = 0.9.^(0:(N-1));
h = 0.5.^(0:(N-1));
```

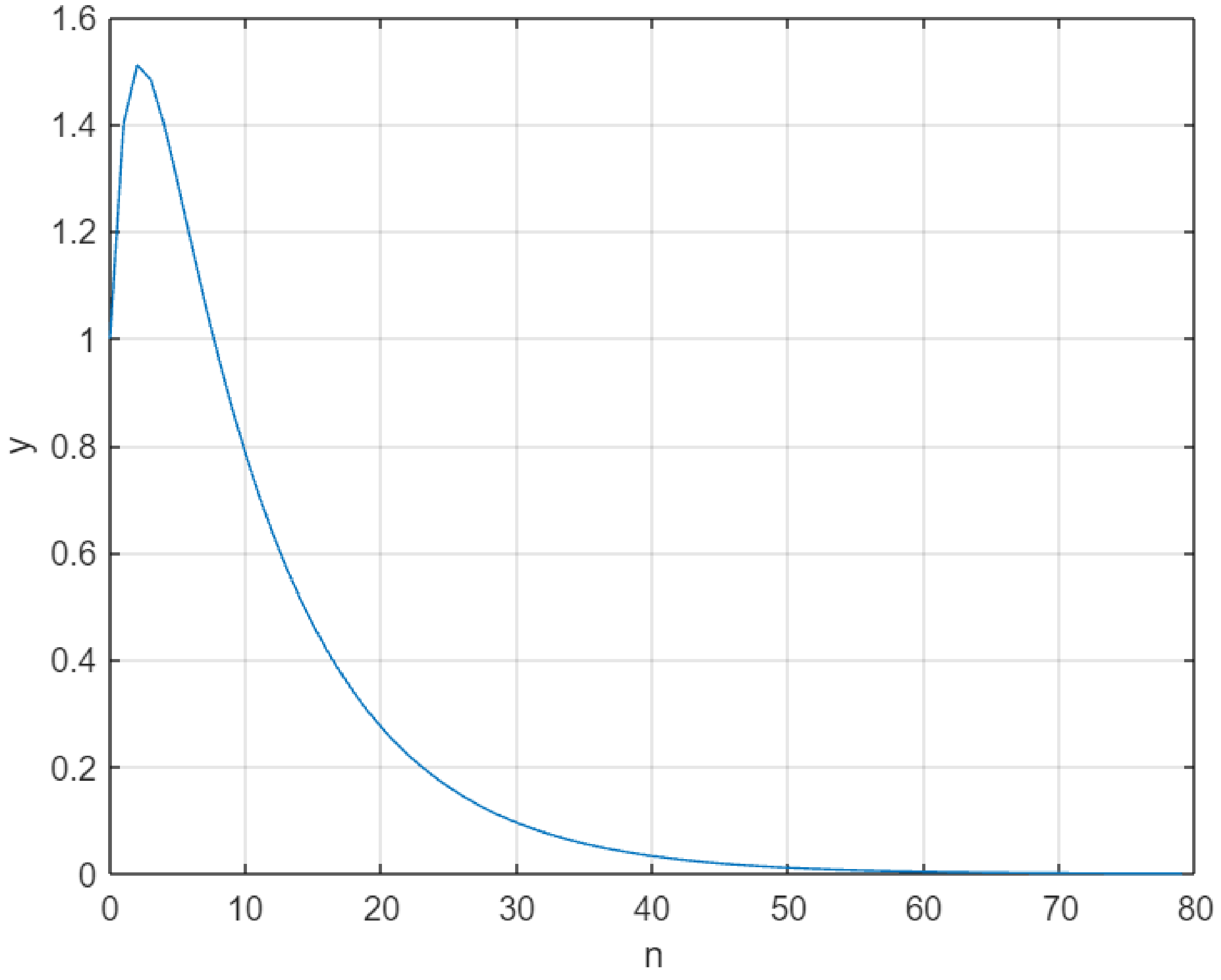
Calculate $N_y, y[n], f80f$.

```
Ny = N;
tic;
DTFSx = fft(x);
DTFSh = fft(h);
```

```
DTFSy = DTFSx.*DTFSh;
y = ifft(DTFSy);
f80f = toc;
```

Plot $y[n]$.

```
figure;
plot(0:(Ny-1),y);
grid on;
xlabel('n');
ylabel('y');
```



(i)

(i). Compute the ratios of **`f40c`** to **`f40f`** and **`f80c`** to **`f80f`**. How do these ratios compare for $N = 40$ and $N = 80$? Which method of convolution is more efficient for each value of N? Which method would you choose for $N > 80$? Justify your answer.

Compute the ratios of $\frac{f40c}{f40f}$ and $\frac{f80c}{f80f}$.

```
ratio40 = f40c/f40f
```

```
ratio40 = 0.7496
```

```
ratio80 = f80c/f80f
```

```
ratio80 = 0.6801
```

Convolution directly are more efficient in both $N = 40$ and $N = 80$. The ratio is also decreasing as N increases. But it's because the data size is so small that the time can't reflect the actual efficiency of the algorithm. Cache, instruction pipeline, matlab's optimization when data size increasing, etc. Many factors influence the time. So we can't figure out which is more efficient until the data size is large enough to make the calculate time takes the main parts of the whole time. But theoretically time complexity of the first method is $O(n^2)$ while the second is $O(nlogn)$. So I would use fft and ifft when $N > 80$.

Test the time when $N = 1e6$.

```
N = 10^6;
x = 0.9.^(0:(N-1));
h = 0.5.^(0:(N-1));
Ny = N;
tic;
y = conv([x x],h);
y = y(1:Ny);
f1e6c = toc;
tic;
DTFSx = fft(x);
DTFSh = fft(h);
DTFSy = DTFSx.*DTFSh;
y = ifft(DTFSy);
f1e6f = toc;
f1e6c, f1e6f
```

```
f1e6c = 1.3915
f1e6f = 0.0290
```

Functions

Unit step function

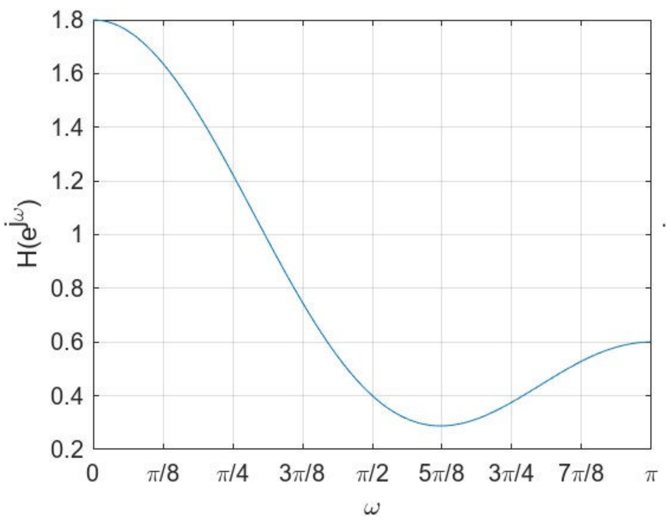
```
function a = dtfs(x,n_init)
    m = zeros(length(x),length(x));
    for i = 1:length(x)
        for j = 1:length(x)
            m(j,i) = exp(-1i*2*pi/length(x)*(i-1)*(j+n_init-1))/length(x);
        end
    end
    a = x*m;
```

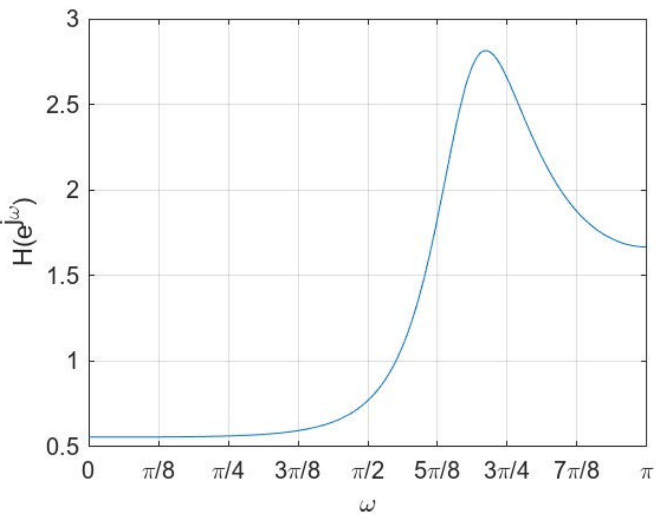
```
  end
```

Expeience

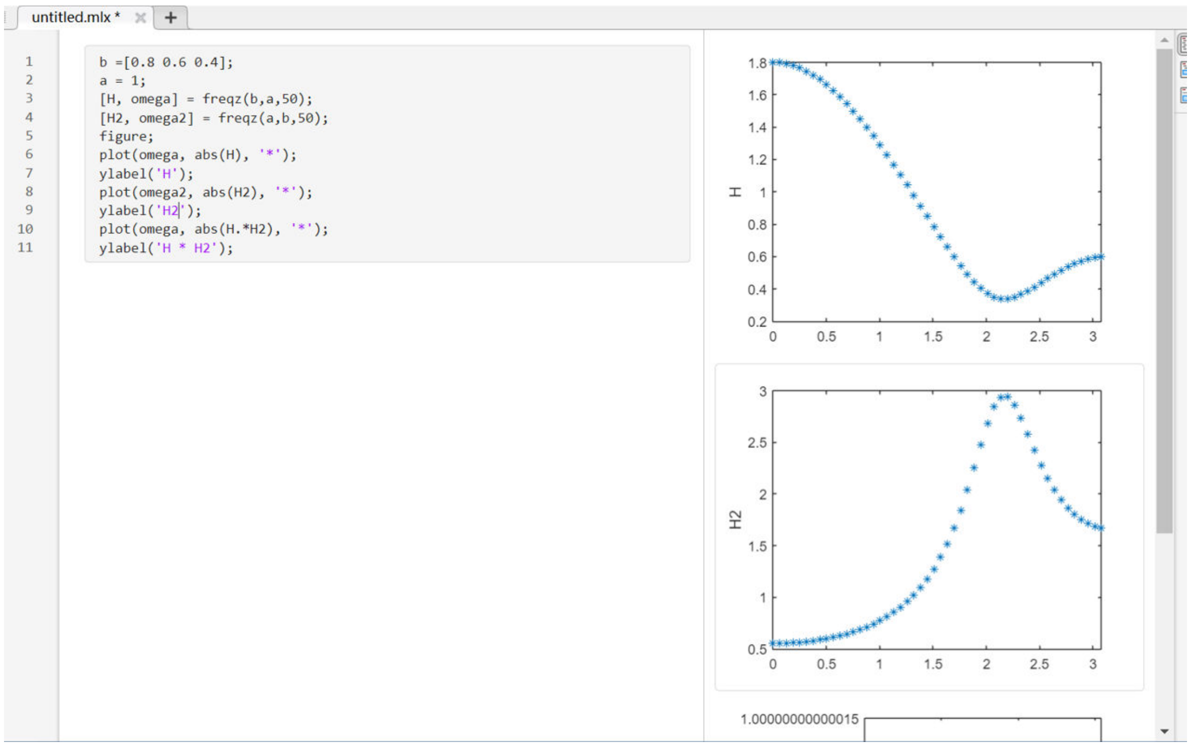
Through Matlab work, I have a preliminary understanding of Matlab simulation tools in the signal and system of the application of the discipline. I'm more famililar with the usage of ff and iff. And I have a better understanding of the dtfs and the differences between dtfs and fft. I also learned the efficiency characteristic of different algorithm. An additional knowledge is that the execution time can't entirely equals to the calculation times.

欧阳安男（12211831）：

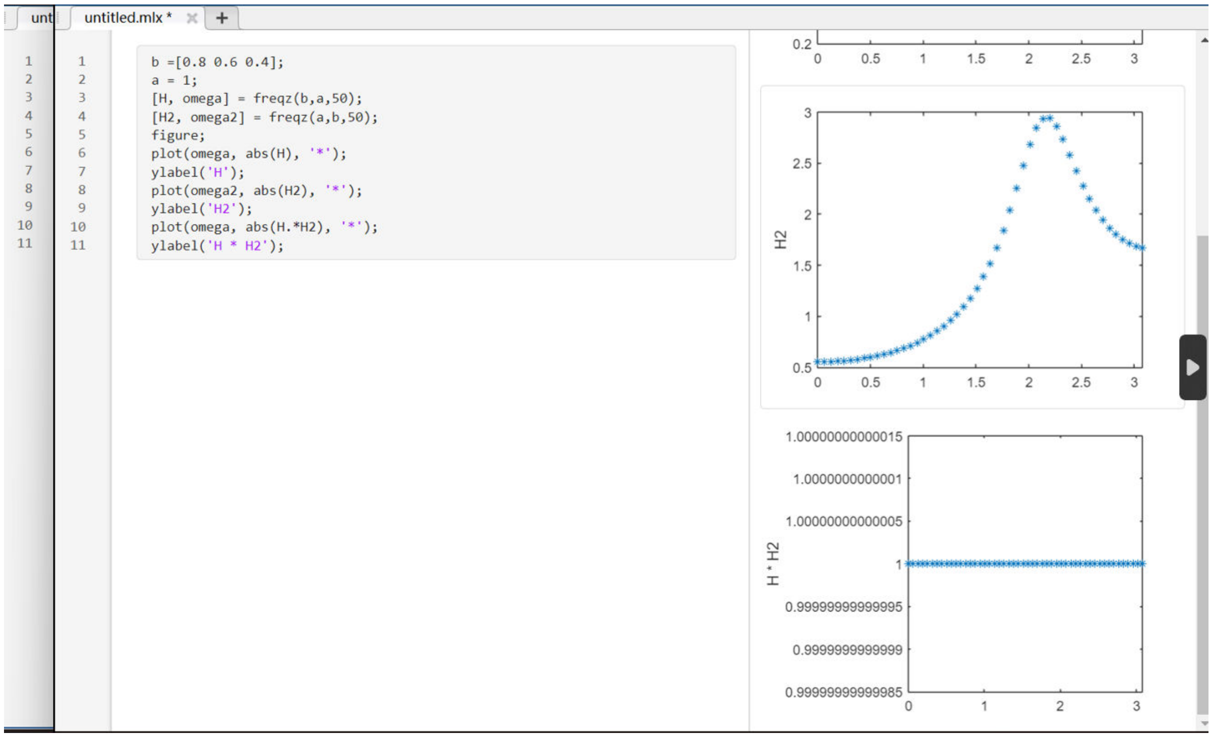




涂峻绫（12213010）：



```
b =[0.8 0.6 0.4];
a = 1;
[H, omega] = freqz(b,a,50);
[H2, omega2] = freqz(a,b,50);
figure;
plot(omega, abs(H), '*');
ylabel('H');
plot(omega2, abs(H2), '*');
ylabel('H2');
plot(omega, abs(H.*H2), '*');
ylabel('H * H2');
```



```
b =[0.8 0.6 0.4];
a = 1;
[H, omega] = freqz(b,a,50);
[H2, omega2] = freqz(a,b,50);
figure;
plot(omega, abs(H), '*');
ylabel('H');
plot(omega2, abs(H2), '*');
ylabel('H2');
plot(omega, abs(H.*H2), '*');
ylabel('H * H2');
```

Score

涂峻绫（100） 欧阳安男（100）