电子科技大学格拉斯哥学院
Glasgow College UESTC

标 准 实 验 报 告
Lab Report

（实验）课程名称：信号与系统
(LAB) Course Name：Signals and Systems

电子科技大学教务处制表

Lab Report 4

Student Name：Changgang Zheng 郑长刚
UESTC ID：2016200302027
UoG ID：2289258Z
Instructor：Huijuan Wu 吴慧娟
Location：UESTC, ChengDu, SiChuan Province, China
Date： April 27, 2018

1. **Laboratory name：** Signals and Systems
2. **Project name：** Represent signals using MATLAB
3. **Lab hours：** 2
4. **Theoretical background：**

1. The basic concepts of signals and systems arise in a variety of contexts, from engineering design to financial analysis. In this lab1, you will learn how to represent, manipulate, and analyze basic signals and systems in MATLAB.

2. Some basic MATLAB commands for representing signals include: zeros, ones, cos, sin, exp, real, imag, abs, angle, linspace, plot, stem, subplot, xlabel, ylabel, title.

3. Some useful commands in Symbolic Math Toolbox are as: zeros, ones, cos, sin, exp, real, imag, abs, anglelinspace, plot, stem, subplot, xlabel, ylabel, title. Symbolic Math Toolbox: sym, subs, ezplot, Conv, filter, lsim.

5. **Objective：**

Compute the output of causal LTI system characterized by linear constant-coefficient differential/ difference equations.

6. **Description：**

The following exercises are from the book “John R.Buck, Michael M. Daniel, Andrew C. Singer. Computer Exploration in Signals and Systems —— Using MATLAB.”

1. Compute the output of causal LTI system characterized by linear constant-coefficient difference equations. 2.2(a)(b)(c)(d)(e)(f)(g)

2. Compute the output of causal LTI system characterized by linear constant-coefficient difference equations. 2.3(a)(b)

7. **Required instruments：**

Computer, MATLAB.

8. **Procedures, Analysis of Lab data & result and Conclusion：**

MATLAB codes and results for the exercises:

■ 2.2 Tutorial: `filter`

The `filter` command computes the output of a causal, LTI system for a given input when the system is specified by a linear constant-coefficient difference equation. Specifically, consider an LTI system satisfying the difference equation

$$\sum_{k=0}^{K} a_k y[n-k] = \sum_{m=0}^{M} b_m x[n-m]\,, \tag{2.7}$$

where $x[n]$ is the system input and $y[n]$ is the system output. If `x` is a MATLAB vector containing the input $x[n]$ on the interval $n_x \le n \le n_x + N_x - 1$ and the vectors `a` and `b` contain the coefficients a_k and b_k, then `y=filter(b,a,x)` returns the output of the causal LTI system satisfying

$$\sum_{k=0}^{K} \mathtt{a}(k+1)\mathtt{y}(n-k) = \sum_{m=0}^{M} \mathtt{b}(m+1)\mathtt{x}(n-m)\,. \tag{2.8}$$

Note that $\mathtt{a}(k+1) = a_k$ and $\mathtt{b}(m+1) = b_m$, since MATLAB requires that all vector indices begin at one. For example, to specify the system described by the difference equation $y[n] + 2y[n-1] = x[n] - 3x[n-1]$, you would define these vectors as `a=[1 2]` and `b=[1 -3]`.

The output vector `y` returned by `filter` contains samples of $y[n]$ on the same interval as the samples in `x`, i.e., $n_x \le n \le n_x + N_x - 1$, so that both `x` and `y` contain N_x samples. Note, however, that `filter` needs $x[n]$ for $n_x - M \le n \le n_x - 1$ and $y[n]$ for $n_x - K \le n \le n_x - 1$ in order to compute the first output value $y[n_x]$. The function `filter` assumes that these samples are equal to zero.

(a). Define coefficient vectors `a1` and `b1` to describe the causal LTI system specified by $y[n] = 0.5x[n] + x[n-1] + 2x[n-2]$.

(b). Define coefficient vectors `a2` and `b2` to describe the causal LTI system specified by $y[n] = 0.8y[n-1] + 2x[n]$.

(c). Define coefficient vectors `a3` and `b3` to describe the causal LTI system specified by $y[n] - 0.8y[n-1] = 2x[n-1]$.

(d). For each of these three systems, use `filter` to compute the response $y[n]$ on the interval $1 \le n \le 4$ to the input signal $x[n] = n\,u[n]$. You should begin by defining the vector `x=[1 2 3 4]`, which contains $x[n]$ on the interval $1 \le n \le 4$. The result of using `filter` for each system is shown below.

```
>> x = [1 2 3 4];
>> y1 = filter(b1,a1,x)
y1 =
    0.5000    2.0000    5.5000    9.0000
>> y2 = filter(b2,a2,x)
y2 =
    2.0000    5.6000   10.4800   16.3840
>> y3 = filter(b3,a3,x)
y3 =
         0    2.0000    5.6000   10.4800
```

From `y1(1)=0.5`, you can see that `filter` has set $x[0]$ and $x[-1]$ equal to zero, since both of these samples are needed to determine $y_1[1]$.

The function `filter` can also be used to perform discrete-time convolution. Consider the class of systems satisfying Eq. (2.7) when $a_k = \delta[k]$. In this case, Eq. (2.7) becomes

$$y[n] = \sum_{m=0}^{M} b_m x[n-m]\,. \tag{2.9}$$

If we define the following finite-length signal

$$b[m] = \begin{cases} b_m\,, & 0 \le m \le M\,, \\ 0\,, & \text{otherwise}\,, \end{cases}$$

then Eq. (2.9) can be rewritten as

$$y[n] = \sum_{m=0}^{M} b[m]x[n-m] = \sum_{m=-\infty}^{\infty} b[m]x[n-m]\,. \tag{2.10}$$

Note the similarity between Eq. (2.10) and Eq. (2.3) — the filter given by Eq. (2.9) is a convolution. The signal $b[m]$ is the impulse response of the LTI system which satisfies Eq. (2.9). Because $b[m]$ has finite length, such systems are called finite-length impulse response (FIR) filters.

To illustrate how to use `filter` to implement a discrete-time convolution, consider the convolution of $x[n]$ in Eq. (2.5) with $h[n]$ in Eq. (2.6).

(e). Store $x[n]$ and $h[n]$ on the interval $0 \le n \le 5$ in the vectors `x` and `h`.

(f). To use `filter`, the impulse response $h[n]$ must be mapped to the coefficients of the difference equation in Eq. (2.9), i.e., $b_m = h[m]$ for $0 \le n \le 5$. In other words, the difference equation coefficients are given by `b=h` and `a=1`. Use `y=filter(h,1,x)` to compute the output of this difference equation on the interval $0 \le n \le 5$, and set `ny=[0:5]`. Remember that `filter` returns a vector `y` with the same number of samples as `x`. Plot your results using `stem(ny,y)`. Your plot should agree with Figure 2.4.

If `filter` is to return the same result as `conv(h,x)`, then the input to `filter` must contain 11 samples of $x[n]$. (Remember that `conv` returns a vector of length $N_x + N_h - 1$, where N_x is the length of `x` and N_h is the length of `h`.)

(g). Define a vector `x2` to contain $x[n]$ on the interval $0 \le n \le 10$, and use

```
>> y2=filter(h,1,x2);
```

to compute the convolution on this interval. Plot your results using `stem([0:10],y2)`, and verify that your plot agrees with Figure 2.2.

Like `conv`, `filter` can also be used to implement an LTI system which has a noncausal impulse response. Again, it is important to keep track of the indices of the input, impulse

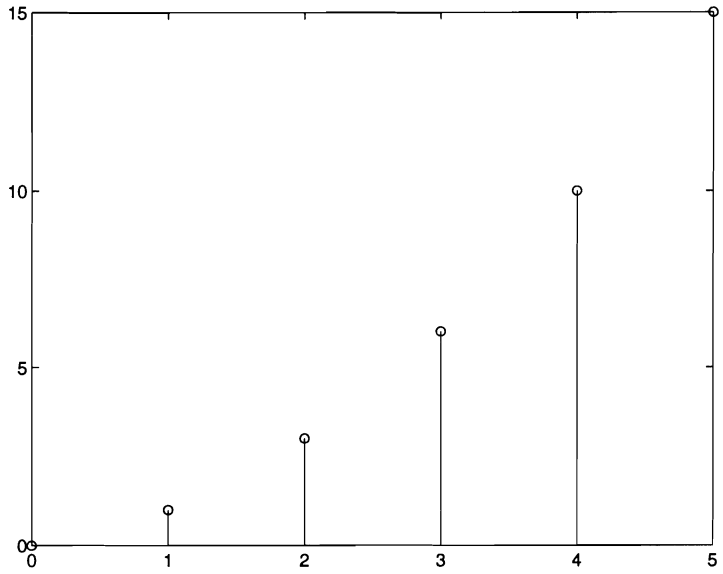


Figure 2.4. Plot of the signal $y[n] = x[n] * h[n]$ over the interval $0 \leq n \leq 5$.

response, and output. To illustrate how, assume that $h[n]$ in Eq. (2.3) is replaced by $h[n+L]$ for some integer L. The convolution sum becomes

$$y_2[n] = \sum_{m=-\infty}^{\infty} h[m+L]x[n-m]\,,$$

which when substituting $m' = m + L$ gives

$$\begin{aligned} y_2[n] &= \sum_{m'=-\infty}^{\infty} h[m']x[n-(m'-L)] \\ &= \sum_{m'=-\infty}^{\infty} h[m']x[(n+L)-m'] \\ &= y[n+L]\,. \end{aligned}$$

In other words, an advance (or delay for $L < 0$) in the impulse response by L samples merely advances the output by L samples. Therefore, if `x` contains $x[n]$ on the interval $0 \leq n \leq N_x - 1$ and `h` contains $h[n]$ on the interval $-L \leq n \leq N_h - 1 - L$, then `y=filter(h,1,x)` will return $y[n]$ on the interval $-L \leq n \leq N_x - 1 - L$. Note that `y` still has the same number of samples as `x`, only the samples represented by `y` have advanced by L samples.

The code:

```
%Name: Matlab/CUDA: Signals and Systems Lab 4th
%Auther: Changgang Zheng
%Student Number UESTC:2016200302027
%Student Number UoG:2289258z
%Institution: Glasgow College UESCT
%Question: Perform convolution. 2.2(a)(b)(c)(d)(e)(f)(g)


function  problem_1st

    %% problem a
    a1=[1];
    b1=[0.5 1 2];
    %%

    %% problem b
    a2=[1 -0.8];
    b2=[2];
    %%

    %% problem c
    a3=[1 -0.8];
    b3=[0 2];
    %%

    %% problem d
    n=[1:4];
    x=[1 2 3 4];
    y1=filter(b1,a1,x);
    y2=filter(b2,a2,x);
    y3=filter(b3,a3,x);


    figure;                         % create a new window for plotting
    stem(n,y1);                     % plot the picture
    title('The graph of y1[n]');    % name the title of the figure as 'The
graph of y1[n]'
    xlabel('n');                    % name the label of x-axis as 't'
    ylabel('y1[n]');                % name the label of y-axis as 'y1[t]'

    figure;                         % create a new window for plotting
    stem(n,y2);                     % plot the picture
    title('The graph of y2[n]');    % name the title of the figure as 'The
graph of y2[n]'
    xlabel('n');                    % name the label of x-axis as 't'
    ylabel('y2[n]');                % name the label of y-axis as 'y2[t]'

    figure;                         % create a new window for plotting
    stem(n,y3);                     % plot the picture
    title('The graph of y3[n]');    % name the title of the figure as 'The
graph of y3[n]'
    xlabel('n');                    % name the label of x-axis as 't'
    ylabel('y3[n]');                % name the label of y-axis as 'y3[t]'
    %%

    %% problem e
```

```
    x_e=ones(1,6);
    n_x=[0:5];
    h=[0:5];
    %%

    %% problem f

    y_f=filter(h,1,x_e);

    figure;                          % create a new window for plotting
    stem(n_x,y_f);                   % plot the picture
    title('The graph of y_f[n]');    % name the title of the figure as 'The
graph of y_f[n]'
    xlabel('n');                     % name the label of x-axis as 't'
    ylabel('y_f[n]');                % name the label of y-axis as 'y_f[t]'
    %%

    %% problem g
    x_g=[ones(1,6) zeros(1,5)];
    y_g=filter(h,1,x_g);

    figure;                          % create a new window for plotting
    stem([0:10],y_g);                % plot the picture
    title('The graph of y_g[n]');    % name the title of the figure as 'The
graph of y_g[n]'
    xlabel('n');                     % name the label of x-axis as 't'
    ylabel('y_g[n]');                % name the label of y-axis as 'y_g[t]'

    %%
```

The result:

Answer for problem (a):

Program core:

```
    %% problem a

    a1=[1];
    b1=[0.5 1 2];

    %%
```

Answer for problem (b):

Program core:

```
    %% problem b

    a2=[1 -0.8];
```

```
b2=[2];

%%
```

Answer for problem (c):

Program core:

```
%% problem c

a3=[1 -0.8];
b3=[0 2];

%%
```

Answer for problem (d):

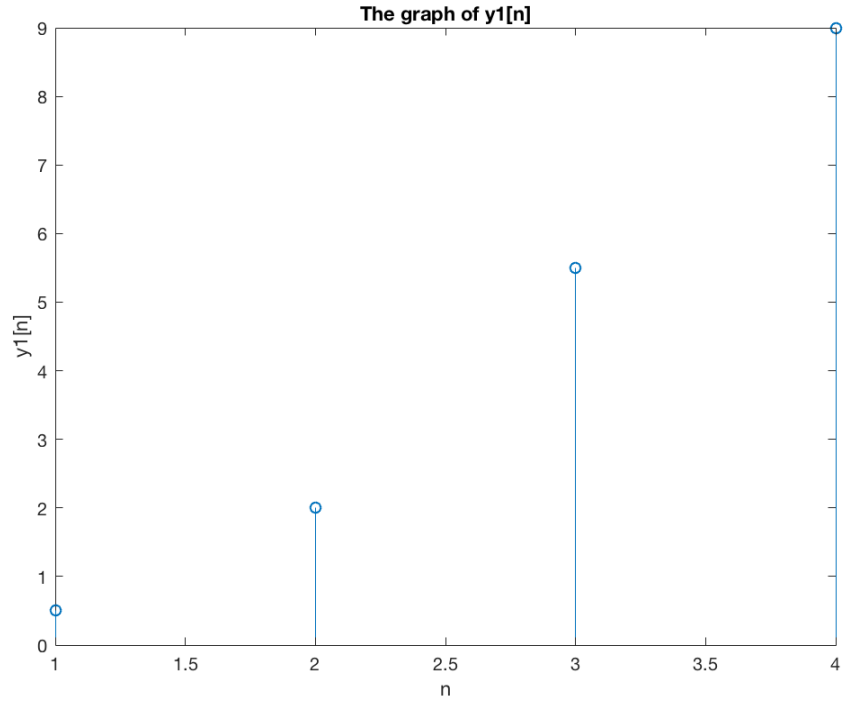
Program core:

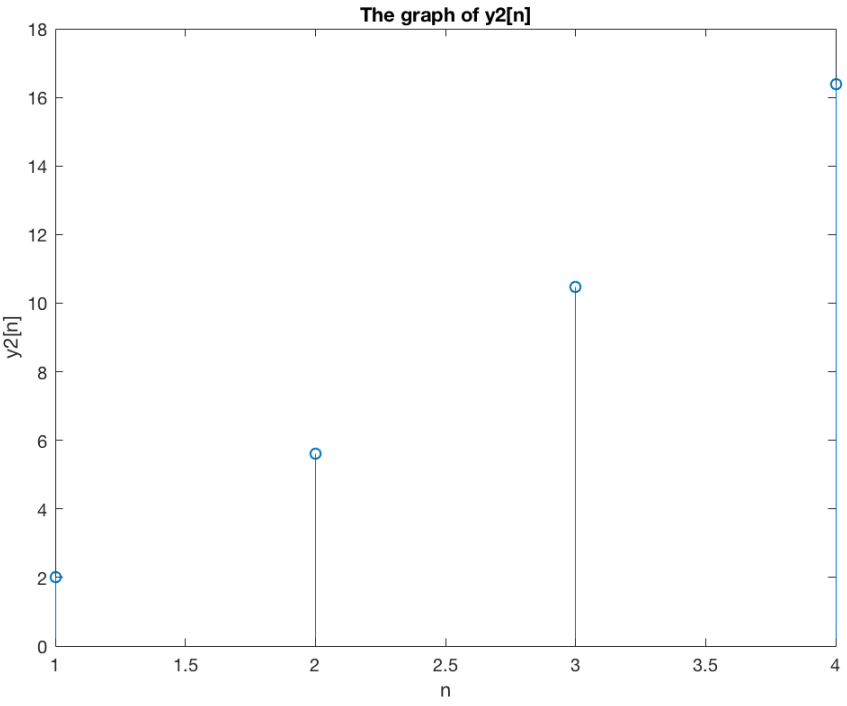
```
%% problem d

n=[1:4];
x=[1 2 3 4];
y1=filter(b1,a1,x);
y2=filter(b2,a2,x);
y3=filter(b3,a3,x);

%%
```

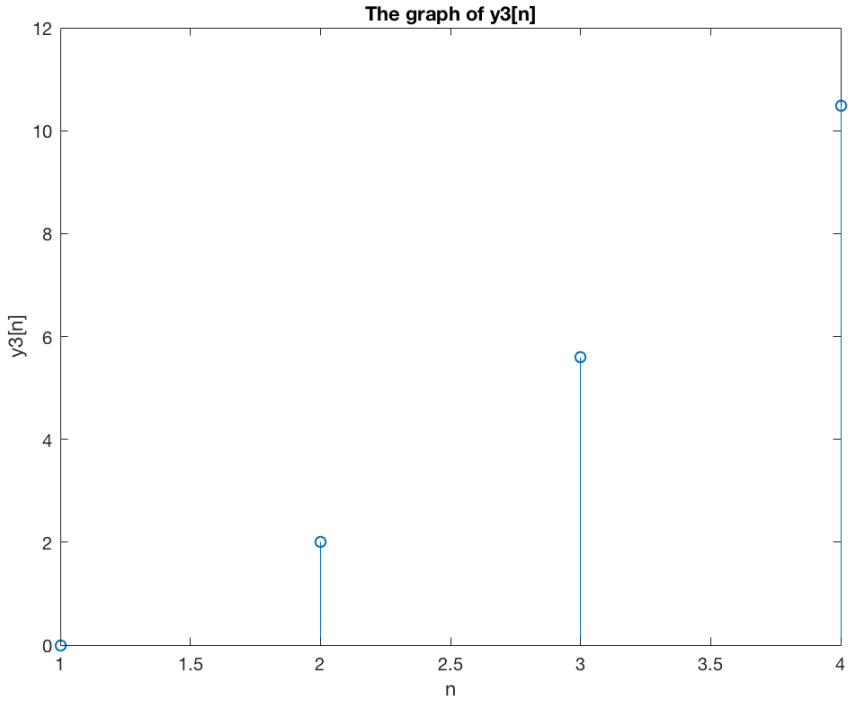
The output result:

```
y1 =

    0.5000    2.0000    5.5000    9.0000

K>> y2

y2 =

    2.0000    5.6000   10.4800   16.3840

K>> y3

y3 =

         0    2.0000    5.6000   10.4800

K>>
```

Plot signals y1[n] y2[n] y3[n], outcomes are as follows:


The graph of y1[n]
y1[n]
n
0
1
2
3
4
5
6
7
8
9
1
1.5
2
2.5
3
3.5
4


The graph of y2[n]
y2[n]
n
0
2
4
6
8
10
12
14
16
18
1
1.5
2
2.5
3
3.5
4



The similarity between these two formula:

Formula 2.3

$$y[n] = \sum_{m=-\infty}^{\infty} h[m]x[n-m]$$

Formula 2.10

$$y[n] = \sum_{m=0}^{M} b[m]x[n-m] = \sum_{m=-\infty}^{\infty} b[m]x[n-m]$$

From the two formulas, we can see that formula 2.3 is the convolution, which aims at calculating the result of a LTI system, for formula 2.10, it is a LCCDE where $a_k = \delta(k)$ which means it has only the first order term. In this way, the form of the two formula are similar. Besides, we can see that the aim of the two lines are the same, which is getting the result of a LTI system. Moreover, if we set $a_k = \delta(k)$, the method to calculate the LCCDE is the same as doing calculation, which means that b[m]=h[m] if $a_k = \delta(k)$.

Answer for problem (e):

$$\boldsymbol{x[n] = 1 \quad when \;\; 0 \le n \le 5}$$
$$\boldsymbol{x[n] = 0 \quad otherwise}$$

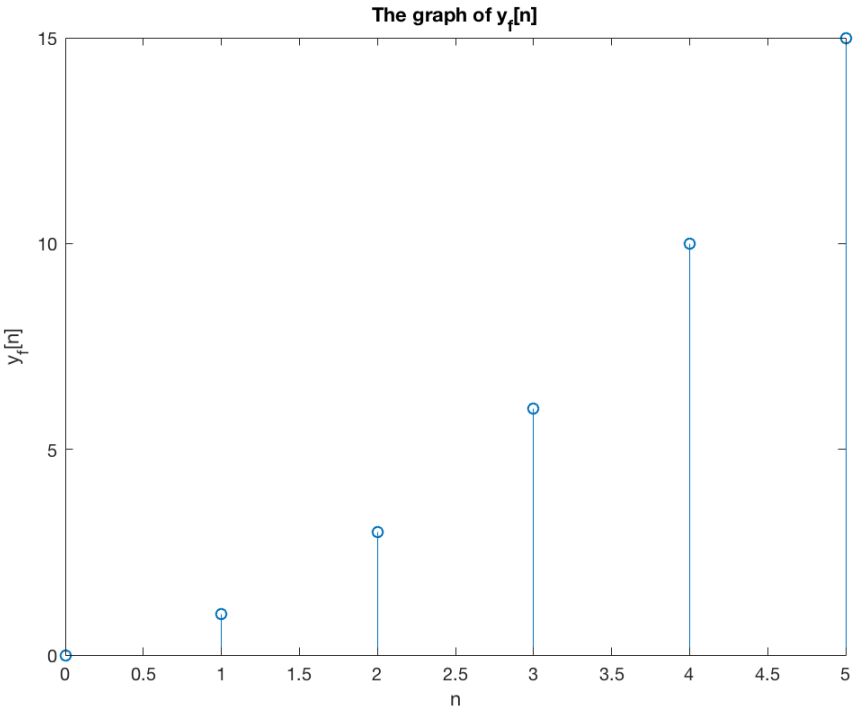
$$\boldsymbol{h[n] = n \quad when \;\; 0 \le n \le 5}$$
$$\boldsymbol{h[n] = 0 \quad otherwise}$$

The output of data which was saved by me:
Program core:

```
>> x=ones(1,6)

x =

     1     1     1     1     1     1

>> h=[0:5]

h =

     0     1     2     3     4     5
```

Answer for problem (f):

The result of the plot: (which is the same as the figure 2.4)

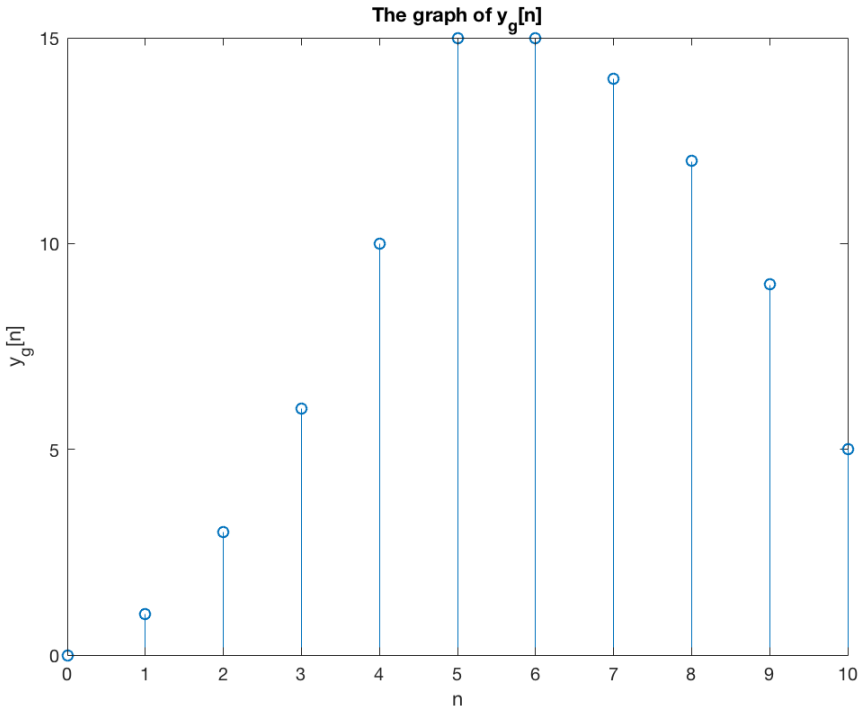


Answer for problem (g):

Program core:

```
%% problem g

x_g=[ones(1,6) zeros(1,5)];
y_g=filter(h,1,x_g);

%%
```

The result of the plot: (which is the same as the figure 2.2)



Summary and comments：

I use the MATLAB to solve some problems such as creating and processing signals as well as plotting them.

I gained more impression of the concept of MATLAB and how it could be used in processing of the signals and systems, especially understand more about the function 'filter', which could help us to calculate the output of the LCCDE system. It is interesting to know that if the $a_k = \delta(k)$, which means it has only the first order term, b_k would become the UIR of the system. This could be derived from precise mathematical calculation, for instance formula 2.3 and 2.10.

Also I am more familiar with the LTI systems which are described by the LCCDE and UIR, especially on how the system work and what the result would be like.

■ 2.3 Tutorial: `lsim` with Differential Equations

The function `lsim` can be used to simulate the output of continuous-time, causal LTI systems described by linear constant-coefficient differential equations of the form

$$\sum_{k=0}^{N} a_k \frac{d^k y(t)}{dt^k} = \sum_{m=0}^{M} b_m \frac{d^m x(t)}{dt^m} . \tag{2.11}$$

To use `lsim`, the coefficients a_k and b_m must be stored in MATLAB vectors `a` and `b`, respectively, in descending[1] order of the indices k and m. Rewriting Eq. (2.11) in terms of the vectors `a` and `b` gives

$$\sum_{k=0}^{N} \mathtt{a}(N+1-k) \frac{d^k y(t)}{dt^k} = \sum_{m=0}^{M} \mathtt{b}(M+1-m) \frac{d^m x(t)}{dt^m} . \tag{2.12}$$

Note that `a` must contain $N+1$ elements, which might require appending zeros to `a` to account for coefficients a_k that equal zero. Similarly, the vector `b` must contain $M+1$ elements. With `a` and `b` defined as in Eq. (2.12), executing

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

simulates the response of Eq. (2.11) to the input signal specified by the vectors `x` and `t`. The vector `t` contains the time samples for the input and output, `x` contains the values of the input $x(t)$ at each time in `t`, and `y` contains the simulated values of the output $y(t)$ at each time in `t`. The accuracy of the simulated values depends upon how well `x` and `t` represent the true function $x(t)$.

While this tutorial does not describe the numerical methods used by `lsim` to compute `y`, it is important to know how `lsim` interprets the inputs `x` and `t`. Basically, `lsim` interpolates the pair `t,x` in much the same way as does `plot`. For instance, consider the plot produced by the following

```
>> t = [0 1 2 5 8 9 10];
>> x = [0 0 0 3 0 0 0];
>> plot(t,x)
```

which is given in Figure 2.5. The function `lsim(b,a,x,t)` will consider $x(t)$ to be equal to

$$x(t) = \begin{cases} 3 - |t-5|, & 2 \leq t \leq 8, \\ 0, & \text{otherwise}. \end{cases}$$

[1] As noted in Tutorial 2.2, the vectors `a` and `b` for the function `filter` contain a_k and b_m in ascending order of the indices k and m, rather than descending order as used by `lsim`.

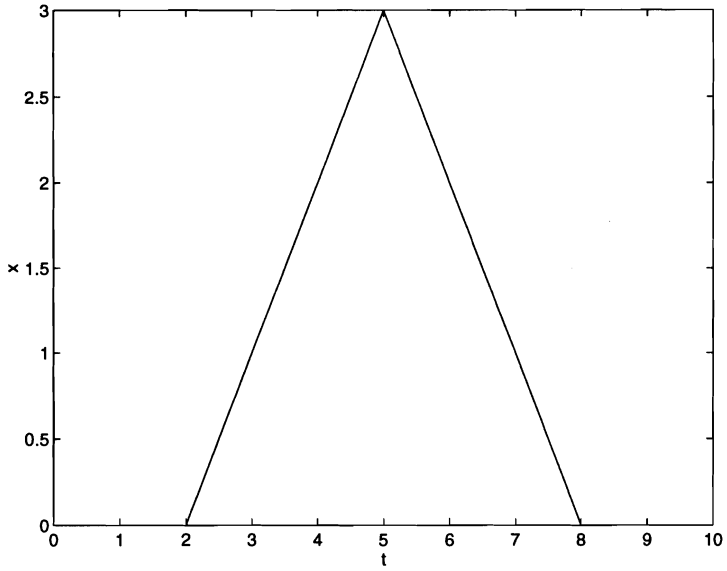


Figure 2.5. The linear interpolation used by `plot(t,x)` is similar to that used by `lsim` to create its input function.

on the interval $0 \leq t \leq 10$. Thus the linear interpolation of the points specified by the pair `[t(n),x(n)]` is the continuous-time function $x(t)$ which `lsim` uses as the input to Eq. (2.12).

Consider the causal LTI system described by the first-order differential equation

$$\frac{dy(t)}{dt} = -\frac{1}{2}y(t) + x(t)\,. \tag{2.13}$$

The step response of this system can computed by first defining the input step function as

```
>> t = [0:10];
>> x = ones(1,length(t));
```

The simulated step response can then be computed and plotted by executing

```
>> b = 1;
>> a = [1 0.5];
>> s = lsim(b,a,x,t);
>> plot(t,s,'y--')
```

The plot is shown in Figure 2.6, where the solid line represents the actual step response

$$s(t) = 2\left(1 - e^{-t/2}\right)u(t)\,. \tag{2.14}$$

Note that at each value of `t`, the step response computed by `lsim` is essentially identical to the true step response. The only difference is in the interpolation produced by `plot`. The

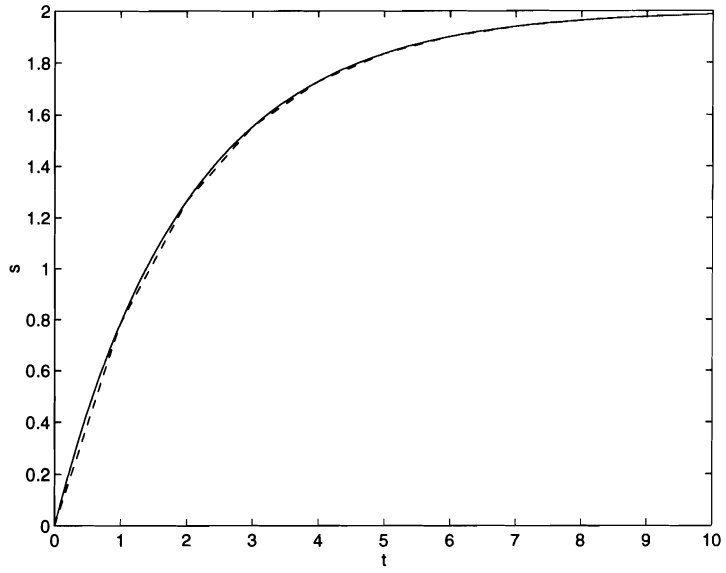


Figure 2.6. For the causal LTI system described by Eq. (2.13), the dotted line is a plot of the step response computed by `lsim`. The solid line is the true step response.

function `lsim` will return more samples of $s(t)$ if the samples in `t` are chosen more closely spaced, e.g., `t=[0:0.1:10]`.

(a). On your own, use `lsim` to compute the response of the causal LTI system described by

$$\frac{dy(t)}{dt} = -2\,y(t) + x(t)\,. \tag{2.15}$$

to the input $x(t) = u(t-2)$. Your response should look like the plot in Figure 2.7, which is computed using `t=[0:0.5:10]`.

While `lsim` can simulate the response of Eq. (2.11) to any input which can be approximated by linear interpolation, the functions `impulse` and `step` can be used to compute the impulse and step responses of such systems. With the vectors `t`, `b`, and `a` defined by

```
>> t = [0:1:10];
>> b = 1;
>> a = [1 0.5];
```

then typing

```
>> s = step(b,a,t);
>> h = impulse(b,a,t);
```

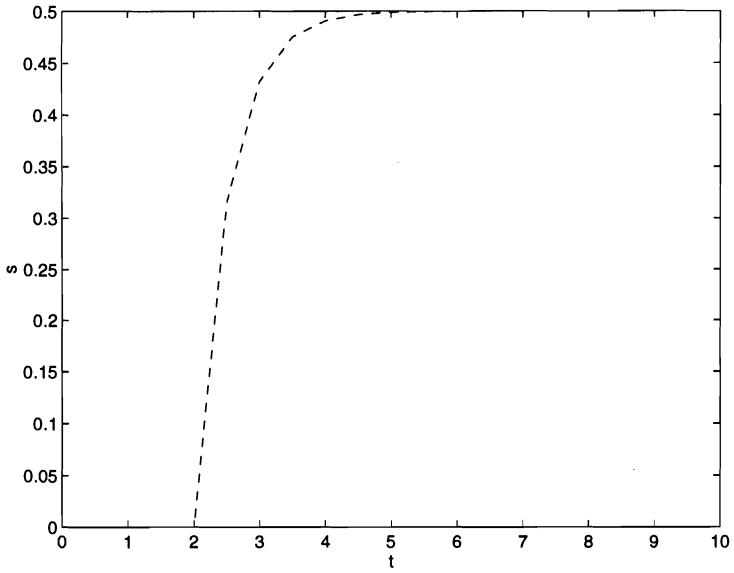


Figure 2.7. The response of Eq. (2.15) to $u(t-2)$, as computed by `lsim` when the time vector is `t=[0:0.5:10]`.

will return the step and impulse responses in the vectors `s` and `h`, respectively. Note that the inputs `a` and `b` to `step` and `impulse` have the same form required by `lsim`.

(b). Use `step` and `impulse` to compute the step and impulse responses of the causal LTI system characterized by Eq. (2.13). Compare the step response computed by `step` with that shown in Figure 2.6. Compare the signal returned by `impulse` with the exact impulse response, given by the derivative of $s(t)$ in Eq. (2.14).

The code:

```
%Name: Matlab/CUDA: Signals and Systems Lab 4th
%Auther: Changgang Zheng
%Student Number UESTC:2016200302027
%Student Number UoG:2289258z
%Institution: Glasgow College UESCT
%Question: Perform convolution. 2.3(a)(b)

function  problem_2nd

    %% problem 2.3(a)
    t_a=[0:0.5:10];
    num_0=2/0.5;
    x_a=[zeros(1,num_0) ones(1,length(t_a)-num_0)];
    b_a=[1];
    a_a=[1 2];
    y_a=lsim(b_a,a_a,x_a,t_a);      % calculate the result after the
(continuous LCCDE) system
```

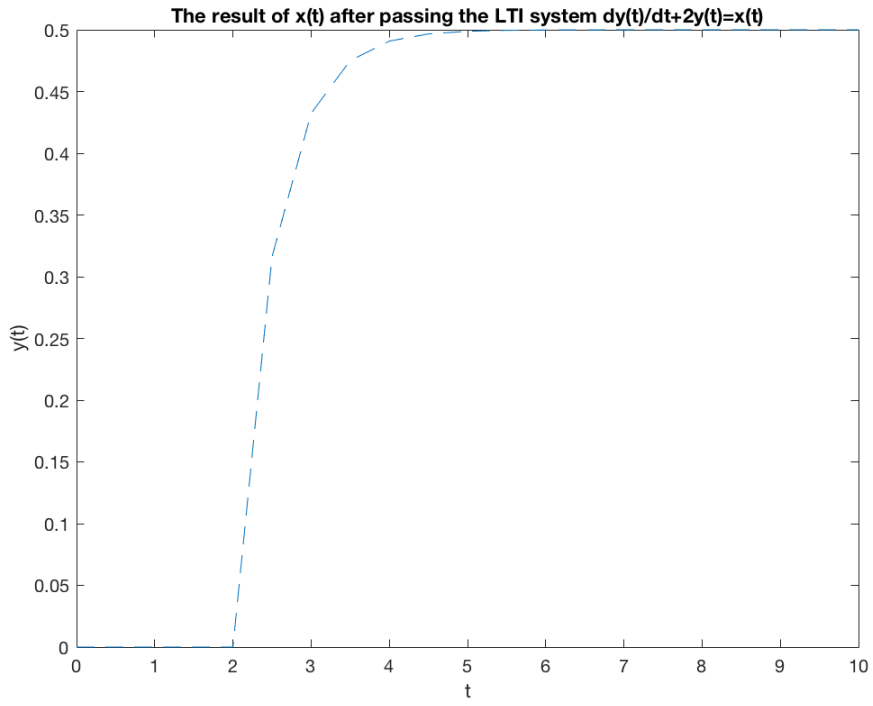
```
    figure;                             % create a new window for plotting
    plot(t_a,y_a,'--');                 % plot the picture
    title('The result of x(t) after passing the LTI system
dy(t)/dt+2y(t)=x(t)');
                                        % name the title of the figure as 'The
result of x(t) after passing the LTI system dy(t)/dt+2y(t)=x(t)'
    xlabel('t');                        % name the label of x-axis as 't'
    ylabel('y(t)');                     % name the label of y-axis as 'y(t)'
    %%

    %% problem 2.3(b)

    intrerval=1;
    t_b=[0:intrerval:10];
    b_b=1;
    a_b=[1 0.5];

    s=step(b_b,a_b,t_b);                % calculate the result after the LTI
system (unit step response)

    h=impulse(b_b,a_b,t_b);             % calculate the result after the LTI
system (unit impluse response)

    syms t;
    s_real=heaviside(t)*2*(1-(exp(-t/2)));
    h2=diff(s_real);                    % calculate the differencial

    t=[0:1:10];
    x=ones(1,length(t));
    b=1;
    a=[1 0.5];
    s2=lsim(b,a,x,t);                   % calculate the result after the LTI
system(continuous LCCDE)


    figure;                             % create a new window for plotting
    fplot(s_real,[0,10],'g');           % plot the picture
    title('The out put from "step" function compare with picture 2.6');
                                        % name the title of the figure as
    xlabel('t');                        % name the label of x-axis as 't'
    ylabel('s(t)');                     % name the label of y-axis as 's(t)'

    hold on;                            % plot two lines in one picture
    plot(t,s2,'b');                     % plot the picture

    hold on;                            % plot two lines in one picture
    plot(t_b,s,'--');                   % plot the picture
    legend('real response','unit impulse response by "lsim" function','unit
impulse response by "step" function');
                                        % set the label for each line


    figure;                             % create a new window for plotting
    plot(t_b,h,'g');                    % plot the picture
    title('The result get by "impulse" function compare with using the real
response');
                                        % name the title of the figure as 'The
result get by impulse compare with using the real response'
```

```
    xlabel('t');                    % name the label of x-axis as 't'
    ylabel('h(t) and ds(t)/dt');    % name the label of y-axis as 'h(t) and
ds(t)/dt'

    hold on;
    fplot(h2,[0,10],'--');          % plot the picture
    legend('The result get by "impulse" function','The result of the real
response');
                                    % set the label for each line
    %%
```

The result:

Answer for problem (a):

The result of the plot:



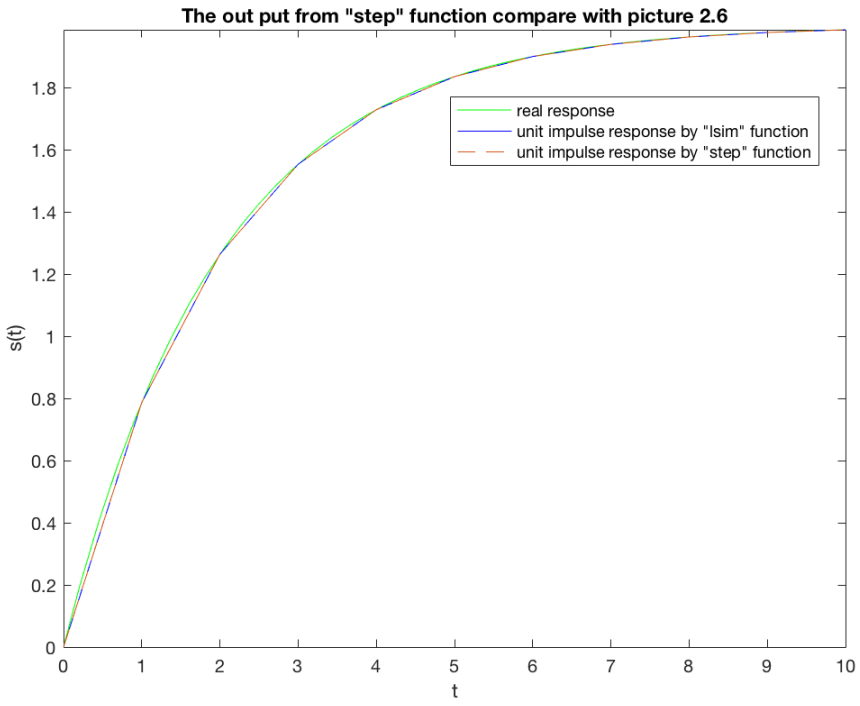
The result is got from t=[0:0.5:10]

Answer for problem (b):

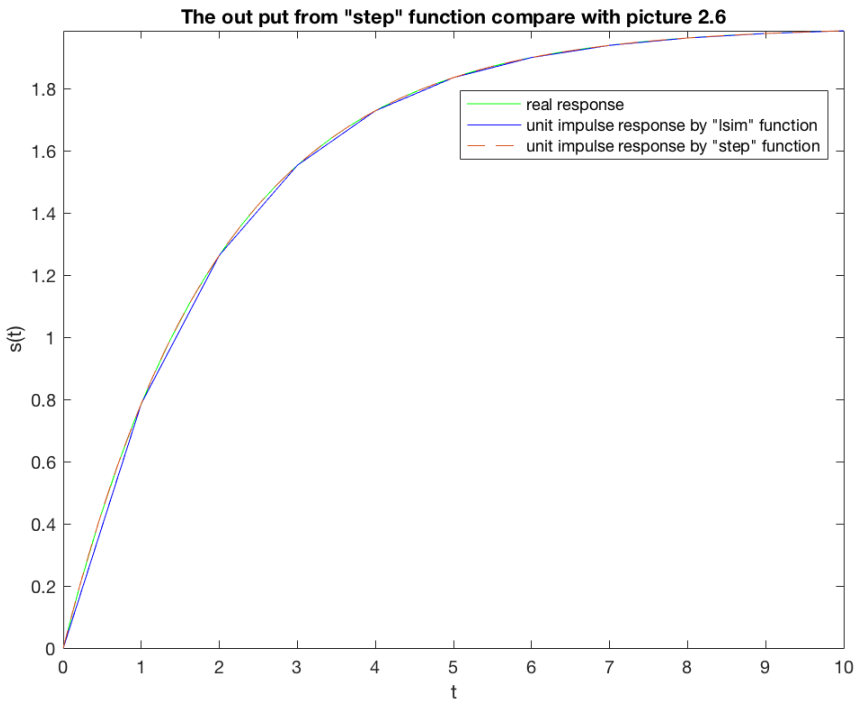
- **When we compare the output getting from the 'step' function with t=[0:1:10], we can easily see that the result (showed by the orange dash line) is extremely close to the result got from 'lsim' function (t=[0:1:10] as well) in picture 2.6.**
- **When we compare the output getting from the 'step' function with t=[0:0.1:10], we can easily see that the result (showed by the orange dash line) is extremely close to the real result in picture 2.6.**

In conclusion, when we use the 'step' function, if we set the 'interval' (t=[0:interval:10]) to be an extremely small number, it would be close to the real result (the 'lsim' function is the same) and if we set the 'interval' to be the same number when we use 'lsim' or the 'step' function, the two results would be the same as well.

The result of the plot:



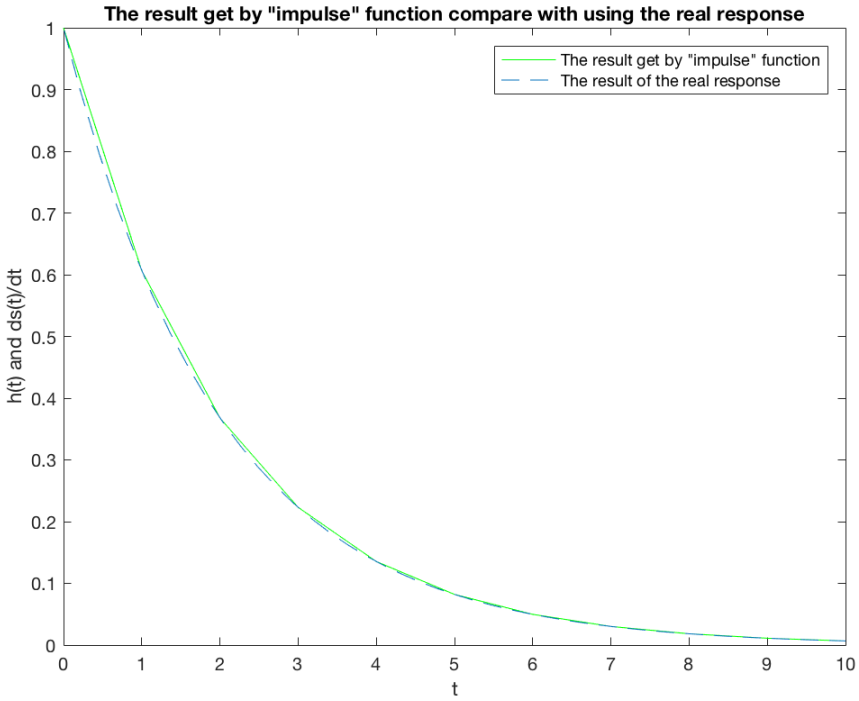
The result is got from t=[0:1:10]



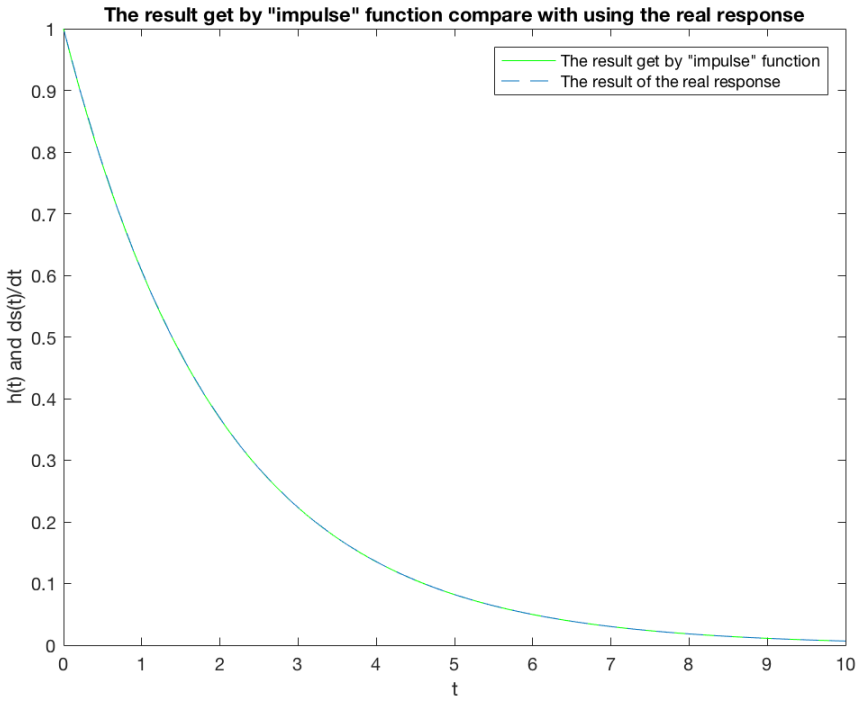
The result is got from t=[0:0.1:10]

The smaller the 'interval', the smaller difference between the outcome and the real result.

The result of the plot:



The result is got from t=[0:1:10]



The result is got from t=[0:0.1:10]

Summary and comments：

I use the MATLAB to solve some problems such as creating and processing signals as well as plotting them.

I gained more impression of the concept of MATLAB and how it could be used in processing of the signals and systems, especially on the part of LTI system (represented by UIR USR and LCCDE). For instance, how I could use the function 'step' and 'impulse' to calculate the USR and UIR respectively. Besides, I know how to use the function 'lsim' to get the output signals from the LCCDE and know more about how these functions work (for instance: b[m]=h[m] if $a_k = \delta(k)$). Meanwhile, I am more familiar with the use of the MATLAB and how I could organize my program (and use comment to make it readable).

Also I am more familiar with the LTI systems which are described by the LCCDE, UIR and USR (The form of convolution), especially on how the system work and what the result would be like.

Suggestion for this lab：

It would be better if we can get the answers of the lab to check if our work is staying on track. Thanks!

Score：

Instructor：