Lab 2 System Analysis in Time Domain

Objective

- Learn to create functions and scripts.
- Understand the zero-input and zero-state response of a system.
- Find out impulse and step response of the system.
- Learn the convolution operation with MATLAB.

Content

Scripts and Functions

All of the pre-built commands that you use in MATLAB are script files or functions (plot, exp, mean …). MATLAB allows the user to create his/her own customized m-files for specific applications or problems.

A script file is simply a collection of executable MATLAB commands saved together in a file. It behaves exactly the same as entering each command sequentially in the command window. Every single variable defined appears in the workspace.

To create a new script file, follow the steps below:

1. Click on the New Script icon on the left side of the Home Tab as shown in Figure 1.
2. Type a set of executable commands in the editor window.
3. Save the file in an appropriate folder. The naming rules for a script are the same as the rules for variables as mentioned in Lab1.
4. Set the current directory in MATLAB to the same place where you saved the script file.
5. To run the script file, hit the green run arrow in the editor window or simply type the name of the file (without the .m extension) at the command prompt in the MATLAB command window.

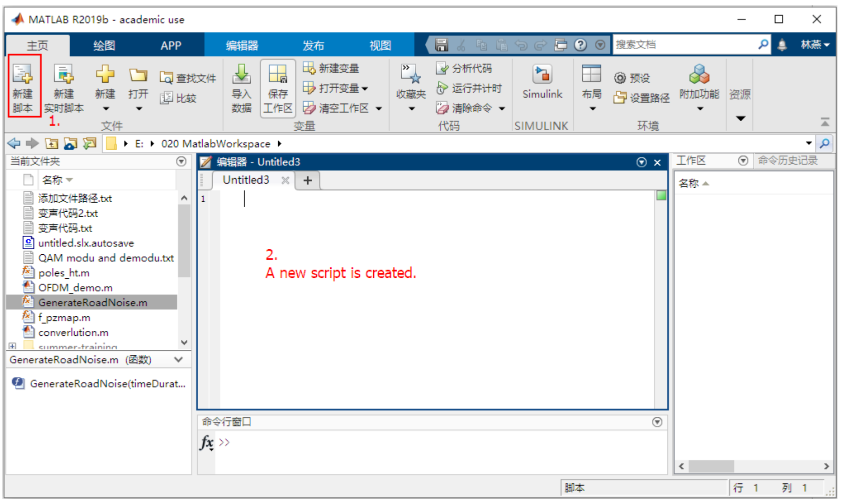
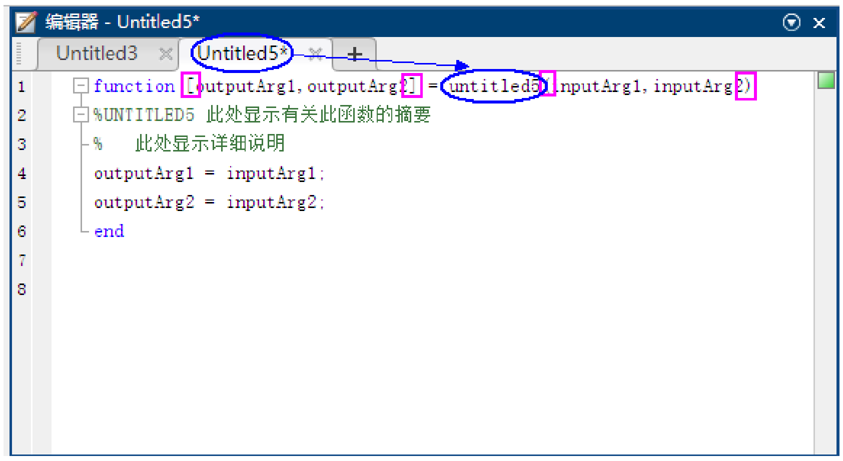


Figure 1 Create a New Script

A function operates on local variables. Click the New icon (on the Home Tab) and then Function to create a function. A function usually consists of three parts, function name, comment, and body. The structure of a function file is shown in Figure 2.



```
function [outputArg1,outputArg2] = untitled5(inputArg1,inputArg2)
%UNTITLED5 此处显示有关此函数的摘要
%   此处显示详细说明
outputArg1 = inputArg1;
outputArg2 = inputArg2;
end
```

Figure 2 the Structure of Function File

The first line in a function file must begin with a *function definition line* that has a list of inputs and outputs. This line distinguishes a function m-file from a script m-file. Note that the output variables are enclosed in square brackets, while the input variables must be enclosed with parentheses. The function name (here Untitled5) should be the same as the file name in which it is saved (with the .m extension).

In the comment part, write some description for the function, which will be displayed in the help command.

Write the function body to realize what you want the function to do. Note the use of a semicolon at the end of the lines in the function body. This prevents the value of the functions being displayed.

To execute the function, type the command

```
[y1, y2]=function_name (x1, x2);        % with explicit out
function_name(x1, x2);                  % without any output
```

Zero-Input and Zero-State Response

Usually, a dynamic system can be represented by a linear differential equation with constant coefficients, which is shown as below:

$$\sum_{i=0}^{N} a_i y^{(i)}(t) = \sum_{j=0}^{M} b_j f^{(j)}(t)$$

$$\text{with initial condition: } y^k(0)(\mathrm{k}=0,\ldots,\mathrm{N}-1), \mathrm{f(t)}=0 \text{ for } \mathrm{t}<0$$

For such a system there are several ways to classify the system response, one of which is:

$$y(\mathrm{t}) = y_{zs}(t) + y_{zi}(t)$$

Zero-Input Response

MATLAB symbolic toolbox provide **dsolve** function to solve differential equation with constant coefficients. When using the symbolic method, $\delta(\mathrm{t})$ is represented by **dirac(t)** and $\mathrm{u(t)}$ is represented by **heaviside(t)**.

The use of **dsolve**:

1. The format of dsolve is $\mathrm{dsolve([eqn1, eqn2, \ldots], [cond1, cond2, \ldots])}$. Both eqn and cond are strings. Eqn refers to the differential equation. Cond refers to the initial condition.
2. When solving zero-state response, the **heaviside** function may be contained in the input signal. When define the initial condition, take t a bit greater than 0 to represent the 0_+ moment. This is because when used as a symbolic function, **heaviside** has no value at t=0, as shown in Lab 1. The zero-input response will not encounter the above problem, sine no **heaviside** function is involved.
3. Use function **simplify** to simplify the uncertainty model.

Eg:$y''(t) + 4y(t) = 0, y(0\_) = 1, y'(0\_) = 1$, find the zero_input response.

1. Represent differentiation by using the **diff** function.
2. Specify a differential equation by using ==.

```
syms y(t)
D2y = diff(y,t,2);
Dy = diff(y,t);
eqn = D2y+4*y==0;
conds = [y(0)==1, Dy(0)==1];
ysol = dsolve(eqn,conds);
yzi = simplify(ysol);
```

Zero-State Response

Find out Zero-State Response by Solving Differential Equation (Symbolic Method)

By solving the differential equation, we can also find the zero-state response of the system. However,

we should take 0+ instead of 0- as the initial condition.
Eg:
$y''(t) - 5y'(t) + 6y(t) = f(t), f(t) = e^{-2t}u(t), \mathrm{y}(0_+) = 0, y'(0_+) = 0$,find the zero_state response.

```
syms y(t)
D2y = diff(y,t,2);
Dy = diff(y,t);
eqn = D2y-5*Dy+6*y==exp(-2*t)*heaviside(t);
conds = [y(0.01)==0, Dy(0.01)==0];
ysol = dsolve(eqn, conds);
yzs = simplify(ysol);
```

However, when you calculate the zero_state response manually, the result is:

$$y_{zsr}(\mathrm{t}) = -\frac{1}{20}[-e^{-2t} + 5e^{2t} - 4e^{3t}]u(t)$$

which is somewhat different from yzs. This is because of the bias when y(0.01) and Dy(0.01) are used instead of y(0) and Dy(0) as the initial condition. Take $e^{-\frac{1}{20}}$ and $e^{-\frac{1}{25}}$ as 1, then yzs will be the same as the one calculated manually. Try to change 0.01 to 0.0001 to observe the difference in the result.
Since our goal is to find a zero state response, which means $e^{-2t}u(t)$ is the same as e^{-2t} when $\mathrm{t} \geq 0$, then we can do as following:

```
syms y(t)
D2y = diff(y,t,2);
Dy = diff(y,t);
eqn = D2y-5*Dy+6*y==exp(-2*t);
conds = [y(0)==0, Dy(0)==0];
ysol = dsolve(eqn, conds);
yzs = simplify(ysol);
```

This time we get the exactly the same result as $y_{zsr}(\mathrm{t})$.

Function **dsolve** can have several equations. When the right side of the differential equation also has n-order derivative, set the input signal as the second equation.

Use Function lism to Find out the Zero-State Response (Numerical Method)

In MATLAB, **lsim** is used to solve the numerical solution of differential equations under zero initial conditions. Its form is like:

$$\mathrm{y} = \mathrm{lsim}(\mathrm{sys}, \mathrm{f}, \mathrm{t})$$

t is the sampling point of the system response. **f** is the input signal. **sys** is the model of the LTI system.
Sys can be generated by function **tf**:

$$\mathrm{sys} = \mathrm{tf}(\mathrm{b}, \mathrm{a})$$

b refers to the numerator part of the transfer function and **a** refers to the denominator part of the transfer function.
For the differential equation:

$$a_3 y'''(t) + a_2 y''(t) + a_1 y'(t) + a_0 y(t) = b_3 f'''(t) + b_2 f''(t) + b_1 f'(t) + b_0 f(t)$$

the value of a and b is:

$$\mathrm{a} = [a_3, a_2, a_1, a_0]$$
$$\mathrm{b} = [b_3, b_2, b_1, b_0]$$

and the transfer function is:

$$\mathrm{sys} = \mathrm{tf}(\mathrm{b}, \mathrm{a})$$

Note if the Nth derivative is missing in the differential equation, the corresponding element in the vector should be set to zero.
Example: $y''(t) + 2y'(t) + 3y(t) = f'(t) + 2f(t), f(t) = 5\sin 2\pi t$, try find out the zero-state response.

```
sys = tf([1,2],[1,2,3]);
t = 0:0.01:5;
f = 5*sin(2*pi*t);
y = lsim(sys,f,t);
plot(t,y);xlabel('t');ylabel('y(t)');title('Zero-State
Response'),grid on;
```

Subs

Function **subs** is used to replace an old symbolic variable or string in the symbolic expression with a new symbolic or numeric variable or expression. The format of subs is like **subs**(S, old, new). It is useful when calculating $\mathrm{y} = y_{zi} + y_{zs}$, where y_{zi} is calculated by symbolic method and y_{zs} by numerical method.

Find out Zero-State Response by Convolution

Impulse Response and Step Response

MATLAB provides impulse and step function to achieve the impulse response and step response for the LTI system. The format of the two function is shown below:

$$\mathrm{y} = \mathrm{impluse}(\mathrm{sys}, \mathrm{t}) \ \text{ or } \ [\mathrm{y}, \mathrm{t}] = \mathrm{impluse}(\mathrm{sys})$$
$$\mathrm{y} = \mathrm{step}(\mathrm{sys}, \mathrm{t}) \ \text{ or } \ [\mathrm{y}, \mathrm{t}] = \mathrm{step}(\mathrm{sys})$$

Eg1:$y''(t) + 3y'(t) + 2y(t) = f(t)$, find the impulse response and step response.

```
t = 0:0.001:4;
sys = tf([1],[1,3,2]);
h = impulse(sys,t);
s = step(sys,t);
subplot(2,1,1);plot(t,h);grid on;title('Impulse Response');
subplot(2,1,2);plot(t,s);grid on;title('Step Response');
```

or we do it this way:

```
sys = tf([1],[1,3,2]);
[h,t1] = impulse(sys);
[s,t2] = step(sys);
subplot(2,1,1);plot(t1,h);grid on;title('Impulse Response');
subplot(2,1,2);plot(t2,s);grid on;title('Step Response');
```

Convolution Operation

Convolution is extensively used in signal processing, communication and control engineering. It can be viewed as a description of the input-output relationship for an LTI system. In addition, it can be considered as an operation performed between two arbitrary signals. Convolution can be used as a tool for analysis the properties of signals or the processes that produced those signals.
The unit impulse response of a system is the response of the system to the unit impulse signal and it has particular significance when a system is LTI, because it can completely characterize the system. The impulse response provides all of the information that is required to determine the output of an LTI system with any input signal.

For continuous-time system, the output $y(t)$ of a continuous-time LTI system is related to its input $x(t)$ and the system impulse response $h(t)$ through the convolution integral expressed as:

$$y(t) = \int_{-\infty}^{+\infty} x(\tau)h(t-\tau)d\tau = x(t) * h(t)$$

For a computer program to perform the above continuous-time convolution integral, a numerical approximation of the integral is needed (Computers operate in a discrete and not continuous fashion). One way to approximate the continuous functions in the integral is to use piecewise constant functions by defining

$$\delta_\Delta(t) = \begin{cases} 1 & -\frac{\Delta}{2} \le t \le \frac{\Delta}{2} \\ 0 & otherwise \end{cases}$$

Then a continuous function $x(t)$ can be approximated with a piecewise constant function $x_\Delta(t)$ as a sequence of pulses that every Δ second in time with heights of $x(k\Delta)$, which is :

$$x_\Delta(t) = \sum_{k=-\infty}^{\infty} x(k\Delta)\delta_\Delta(t-k\Delta)$$

In the limit as $\Delta \to 0$, $x_\Delta(t) \to x(t)$. Similarly, $h(t)$ can be approximated by

$$h_\Delta(t) = \sum_{k=-\infty}^{\infty} h(k\Delta)\delta_\Delta(t-k\Delta)$$

The convolution integral can thus be approximated by convolving the two piecewise constant signals as follows:

$$y_\Delta(t) = \int_{-\infty}^{\infty} x_\Delta(\tau) h_\Delta(t-\tau) d\tau$$

Notice that $y_\Delta(t)$ is not necessarily a piecewise constant function. For computer representation purposes, the output needs to be generated in a discrete manner. This is achieved by further approximating the convolution integral as indicated below:

$$y_\Delta(n\Delta) = \Delta \sum_{k=-\infty}^{\infty} x(k\Delta) h((n-k)\Delta)$$

The discrete convolution sum $\sum_{k=-\infty}^{\infty} x(k\Delta)h((n-k)\Delta)$ can be computed with the MATLAB function c**onv**. Then this sum can be multiplied by Δ to get an estimate of the continuous signal $y(t)$ at $t = n\Delta$. Note that as Δ is made smaller, one obtains a closer approximation to $y(t)$.

For a discrete-time system, the output signal $y(n)$ is given by the convolution of the input signal $x(n)$ with the system impulse response $h(n)$.

$$y(n) = \sum_{m=-\infty}^{\infty} x(m) h(n-m) = x(n) * h(n)$$

Function **conv(u, v)** is used to calculate the convolution of vectors u and v. We can also use convolution to find out the zero-state response.
Eg:$f1(t) = u(t) - u(t-2), f2(t) = e^{-3t}u(t)$, find out the convolution of *f1* and *f2*.
(Since *f1* is infinite, while it is impossible for a computer to process infinite signal. So the scale of t should ensure that f2 can be attenuated to be small enough. Here we take t=2.5.)

```
dt = 0.01; t = 0:dt:2.5;
f1 = heaviside(t)-heaviside(t-2);
f2 = exp(-3*t).*heaviside(t);
f = conv(f1,f2)*dt;
n = length(f); tt = (0:n-1)*dt;
subplot(3,1,1); plot(t,f1); grid on; title('f1(t)');
subplot(3,1,2); plot(t,f2); grid on; title('f2(t)');
subplot(3,1,3); plot(tt,f); grid on; title('f(t)=f1(t)*f2(t)');
```