Some Matlab

Here is a quick introduction to Matlab and a couple of its symbolic and control functions.

Matlab is an interpreted language. When you enter a command in the Command window, the line is executed immediately. Matlab was designed to handle arrays or matrices of numbers easily. You do not need to declare a variable before you use it. Variable names are case dependent, i.e., A is a different variable than a. Normal variables that hold numbers hold double-precision numbers by default. Comments start with %.

The >> below is the prompt in the Command window.

Below, a is a scalar that holds one value. If we have no semicolon at the end of a line, the result of the line is displayed. When you do not want to see intermediate results, put a semicolon at the end of the line.

```
>> a = 5

a =

    5

>> a = 5;
>>
```

rherz@ucsd.edu

In addition to entering commands into the Command window, you can also create a disk file to hold a program or "script."

Then you type your program or list of commands into the disk file, save the disk file with a name without spaces and the extension .m, which is the default extension when saving from Matlab.

These files are called m-files. You can execute an m-file by entering the name of the file in the Command window without the .m extension. You can also execute the m-file by clicking the > button in the Editor window.

In the Command window, if you want to repeat an earlier command, you can scroll to an earlier command using the up arrow on the keyboard.

When you create a new m-file, it will be saved by default into the default folder. The path to this folder will be shown near the top of the Command window. If you want to change the default folder to another location, for example your Desktop if you have a file on the Desktop, then click the ... button to the left of the file path field in the Command window.

Matlab can execute files in the default folder and also folders in its search path. In order to add folders permanently to the search path, select Set Path under the File menu.

Below, b is a row vector. The elements in a row can be separated by spaces or commas.

```
>> b = [1 4 7]

b =

    1    4    7

>> b = [1,4,7]

b =

    1    4    7
```

The transpose operator is an apostrophe.

```
>> bt = b'

bt =

    1
    4
    7
```

rherz@ucsd.edu

Some Matlab

Below, c is a column vector. The rows can be separated by semicolons or new lines.

```
>> c = [2; 3; 9]

c =

    2
    3
    9

>> c = [2
3
9]

c =

    2
    3
    9
```

rherz@ucsd.edu

Below, d is a 3x3 matrix. Here we have separated elements in a row by spaces and separated rows by semicolons as an example.

```
>> d = [3 6 9; 4 8 0; 3 2 1]

d =

    3    6    9
    4    8    0
    3    2    1
```

Below, t and y are row vectors that we construct. The colon here is used in specifying a range.

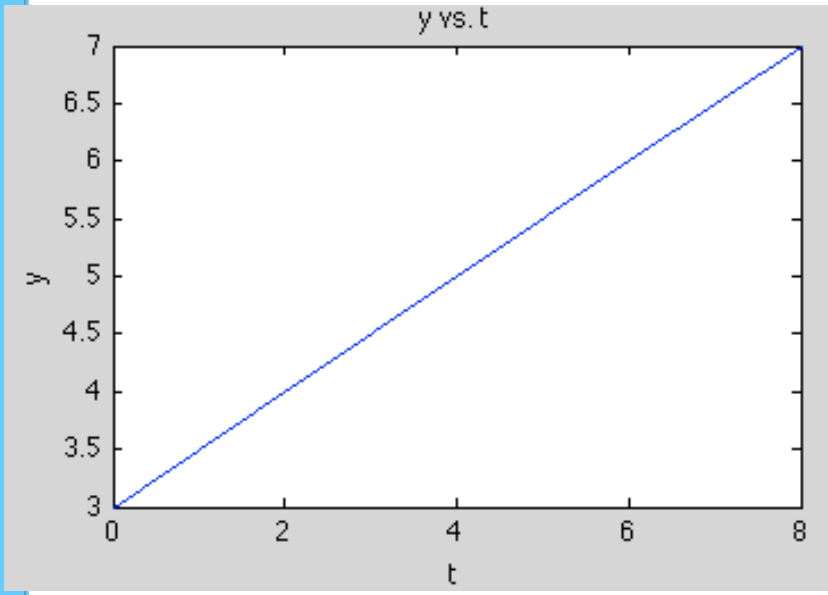
```
>> t = 0:2:8   % = start : increment : end

t =

    0    2    4    6    8

>> y = 3:7     % = start : end (default increment is 1)

y =

    3    4    5    6    7
```

Some Matlab

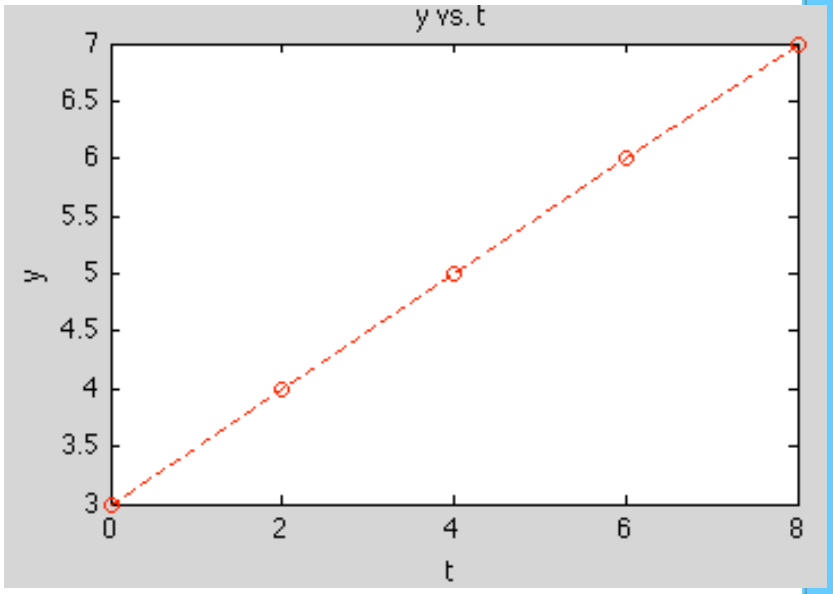
Of of the great features of Matlab is that it has large libraries of functions.

You can plot two vectors of equal length against each other using the standard function "plot"

```
>> plot(t,y)
>> title('y vs. t')
>> ylabel('y')
>> xlabel('t')
```



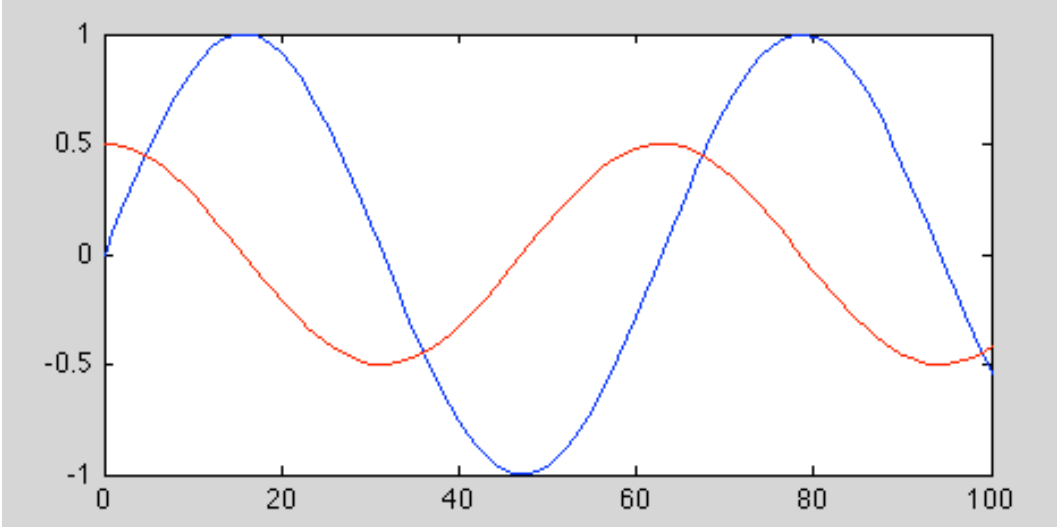

```
>> plot(t,y,'ro--') % note symbol/line specification
>> title('y vs. t')
>> ylabel('y')
>> xlabel('t')
```



rherz@ucsd.edu

You can plot more than one curve on a single plot.

```
>> t = 0:100;
>> y1 = sin(0.1*t);
>> plot(t,y1)
>> y2 = 0.5*cos(0.1*t);
>> plot(t,y1,'b',t,y2,'r')
```

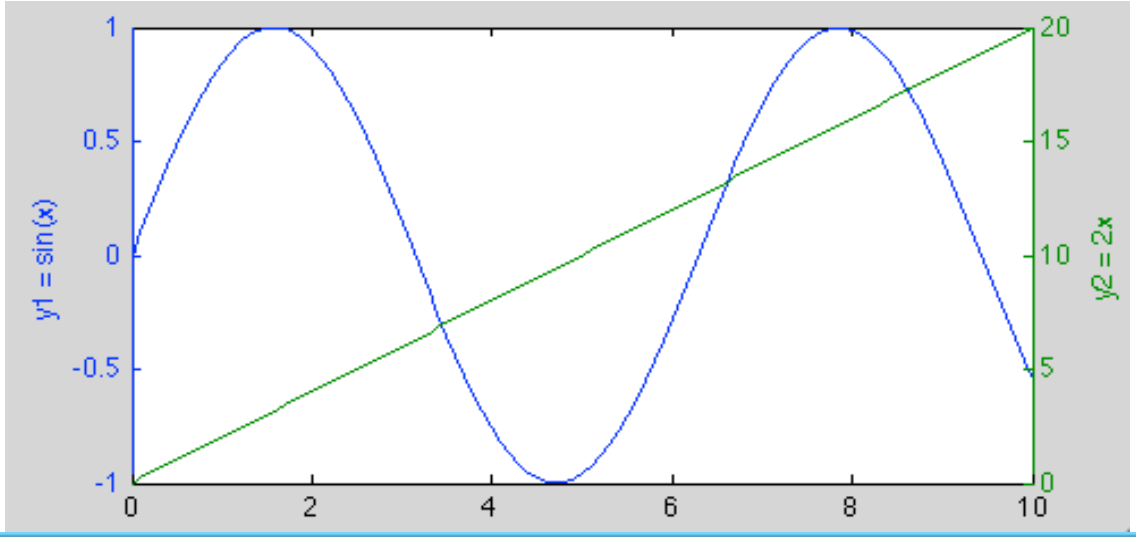


You can also plot the first curve, then issue the hold command, then plot the second curve which adds it to the same plot, and then issue the hold command again to toggle to hold off.

rherz@ucsd.edu

Sometimes you want to plot two variables on a single plot when the magnitudes of the two variables differ greatly. You really want to have different scales for each of the curves. You can do this with a double-y plot using the Matlab function plotyy.

```
>> x = 0:0.1:10;
>> y1 = sin(x);
>> y2 = 2*x;
>> [ax, h1, h2] = plotyy(x,y1,x,y2);   % the function plotyy returns values in [ax, h1, h2]
>>                                     % these are 'handles' or references to the plot objects
>> ylabel(ax(1),'y1 = sin(x)')         % label axis 1 with handle ax(1)
>> ylabel(ax(2),'y2 = 2x')             % label axis 2 with handle ax(2)
```



rherz@ucsd.edu

If you know the name of a Matlab function, you can get instructions on how to use it by entering help and the name of the function.

Enter "help help" to get info on help. Also enter "help lookfor" to see how to find key words if you do not know the exact function name.

Here is some of the info on the function plot.

Note that PLOT is capitalized in help but you must use it as lowercase plot in Matlab!

```
>> help plot
 PLOT   Linear plot.
    PLOT(X,Y) plots vector Y versus vector X. If X or Y is a matrix,
    then the vector is plotted versus the rows or columns of the matrix,
    whichever line up.  If X is a scalar and Y is a vector, disconnected
    line objects are created and plotted as discrete points vertically at
    X.

    PLOT(Y) plots the columns of Y versus their index.
    If Y is complex, PLOT(Y) is equivalent to PLOT(real(Y),imag(Y)).
    In all other uses of PLOT, the imaginary part is ignored.
```

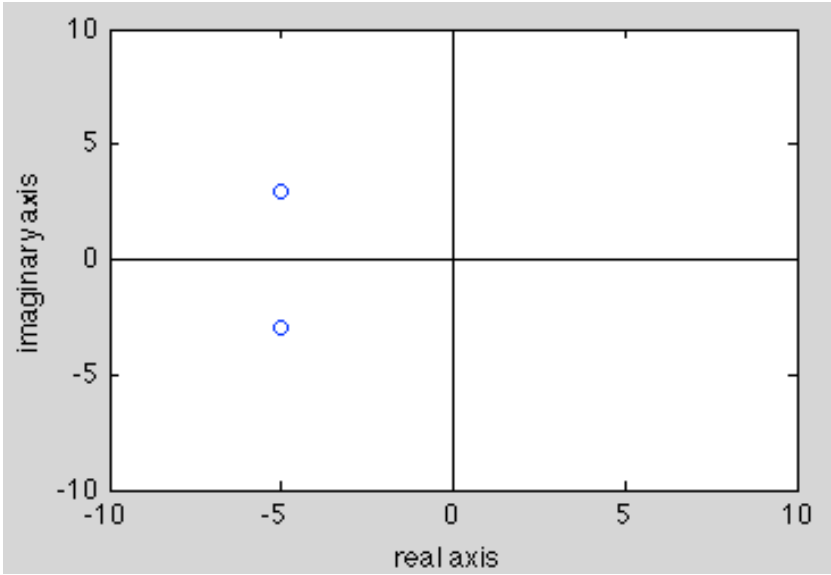
... etc.

Note the last couple lines above in help about plot.

In a control course, we may want to plot zeros (o) and poles (x) on a complex plane.

```
>> z = [-5+3i, -5-3i]

z =

  -5.0000 + 3.0000i  -5.0000 - 3.0000i
```

```
>> plot(z,'bo')
>> axis([-10 10 -10 10])
>> hold
Current plot held
>> xa = [-10 10];
>> ya = xa;
>> plot(xa,[0 0],'k')
>> plot([0 0],ya,'k')
>> ylabel('imaginary axis')
>> xlabel('real axis')
>> hold  % toggle hold off
Current plot released
```



rherz@ucsd.edu

Sometimes you will want to extract one element or a submatrix out of a matrix.

```
>> d = [3 6 9; 4 8 0; 3 2 1]

d =

   3   6   9
   4   8   0
   3   2   1

>> e = d(:,2)  % ( all rows, column 2 )

e =

   6
   8
   2

>> f = d(1:2,2:3)  % ( rows 1 to 2, columns 2 to 3 )

f =

   6   9
   8   0
```

Other times you may want to combine two arrays into a larger array.

```
>> a = [5 6];
>> b = [7 8];
>> c = [a; b]

c =

    5    6
    7    8
```

Math operators in Matlab include:

Matrix operations: * (matrix multiplication), ' (matrix transpose), / (matrix right division)

Array or element-by-element operators: .* (array multiplication), ./ (array division)

When these operations are performed on two arrays, the arrays must be of the proper dimensions.

For matrix multiplication, for example, a * b, the number of columns of a must equal the number of rows of b.

For array multiplication, for example, a .* b, the dimension of a must equal the dimension of b

Here is an example where we want to get the result of b = t*exp(-5*t). We first create the t array. Then we create the a array using the t array. Finally we use array (element-by-element) multiplication to get the final result

```
>> t = 0:5;
>> a = exp(-5*t);
>> b = t  .* a

b =

      0    0.0067    0.0001    0.0000    0.0000    0.0000

>> format short e   % change the display format so we can see more digits, see help format
>> b

b =

 Columns 1 through 5

       0   6.7379e-03   9.0800e-05   9.1771e-07   8.2446e-09

 Column 6

  6.9440e-11
```

What would happen in the above example if we forget the dot before the * such that we specified matrix multiplication instead of array multiplication?

```
>> b = t * a
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

Matrix operations are used when we do linear algebra, of course. We encountered linear algebra problems when doing material balances in CENG 100.

Array or element-by-element operations are used frequently when we process experimental data. In experiments, we often get columns of values where the columns are of the same length. Each row of the data may represent an experiment or an experimental measurement time in which we collect several values.

Let's say vector Cin holds concentrations of a reactant component at the inlet of a reactor and vector Cout holds concentrations at the outlet. We can do the following array operations to compute a new vector that holds the fractional conversion X.

```
>> X = (Cin - Cout) ./ Cin   % note the dot before the /
```

Some Matlab

Now let's look at some functions that are useful in process control.

The roots function gives us the roots of a polynomial.

For example, for the polynomial $5s^2 + 3s + 1$

```
>> % argument is array of coefficients of polynomial in order higher to lower
>> roots([5 3 1])

ans =

  -0.3000 + 0.3317i
  -0.3000 - 0.3317i
```

The function poly returns the coefficients given the input of a polynomials roots. Here we input the roots just obtained. Note that the coefficients returned are normalized so the coefficient of the highest order term is one.

```
>> poly(ans)

ans =

    1.0000    0.6000    0.2000
```

Some Matlab

The function residue does partial-fraction expansion for us. For the transfer function

$$G(s) = \frac{4s + 3}{5s^2 + 3s + 1}$$

```
>> [n p k] = residue([4 3], [5 3 1])    % the results are returned in [n p k], see help residue

n =

  0.4000 - 0.5427i
  0.4000 + 0.5427i


p =

 -0.3000 + 0.3317i
 -0.3000 - 0.3317i

k =

    []
```

With the result that the partial-fraction expansion can be written as:

$$G(s) = \frac{0.40 - 0.54i}{s - (-0.30 + 0.33i)} + \frac{0.40 + 0.54i}{s - (-0.30 - 0.33i)}$$

rherz@ucsd.edu

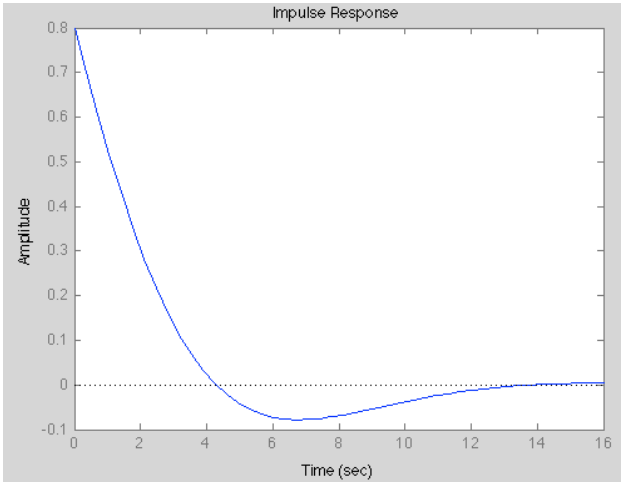
Some Matlab

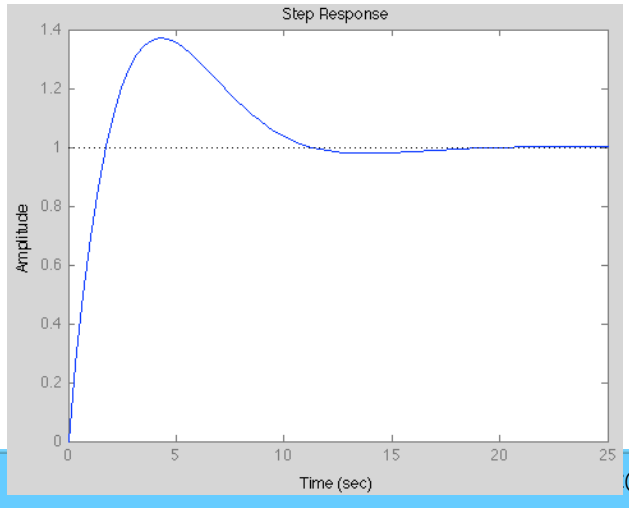
We can enter a transfer function and then plot its impulse and step responses using the functions tf, impulse, and step.

```
>> s = tf('s')     % create the transform variable s

Transfer function:
s

>> G = (4*s + 1)/(5*s^2 + 3*s + 1)

Transfer function:
   4 s + 1
---------------
5 s^2 + 3 s + 1

>> impulse(G)  % plot on right at top
>> step(G)       % plot on right at bottom
```

You can also do algebra with transfer functions, e.g.,

```
>> Y = G*X
>> Y = G1*G2*X
>> Y = (G1/G2)*X
>> G3 = G1 + G2
```





Some Matlab

To plot the response to any arbitrary input, use the function lsim. See help lsim.

Matlab can do the Laplace transform and inverse Laplace transform to symbolic equations using the functions syms, laplace, and ilaplace

```
>> syms t                  % declare t as a symbolic variable
>> y = 5*exp(-3*t);        % enter a symbolic time-domain function
>> Y = laplace(y)          % use the function laplace to apply the Laplace transform

Y =

5/(s + 3)

>> yp = ilaplace(Y)        % apply the inverse Laplace transform to Y to see if we get y(t) back

yp =

5/exp(3*t)                 % we get our original y(t) back, y(t) = t/exp(3*t) = 5*exp(-3*t)
```

rherz@ucsd.edu