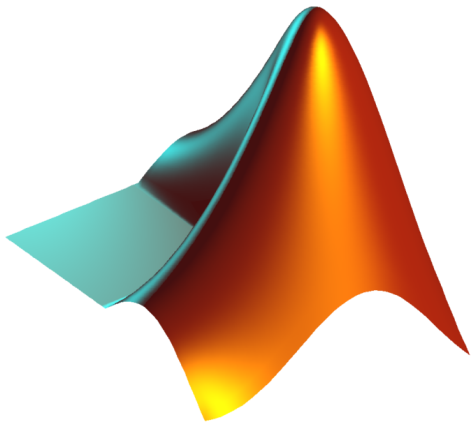
New York City College of Technology
Computer Engineering Technology



MATLAB Basics

Lab 001

CET 4864 Feedback Control Systems
Professor Chen Xu
Spring 2021
Report Written by Galib F. Rahman

Table of Contents

Objective

The objective of this laboratory exercise is to establish a fundamental understanding of the MATLAB software. We will explore the features and capabilities of MATLAB and familiarize ourselves with the environment. In addition we will learn to execute various commands and functions to manage variables of various data types, import and export data, perform calculations, and generate customized plots.

Procedures & Results

Matrix Operations & Plotting

In this portion of the laboratory exercise we will do the following:

Exercise 1

a) **Extract the fourth row of the matrix generated by magic(6)**

First we assigned the variable **n** the output of the **magic()** function which returns a *nxn* matrix constructed from integers from 1 to n^2 with equal rows and column sums. To extract a particular row of a particular matrix,**n**, we will execute the statement n(4,:). To extract a particular column of a particular matrix,**n**, one may execute the statement n(:,x), where x is an integer of the desired column.

Note: The magic function will return a valid magic square if the parameter is greater than or equal to 3.

```
>> n=magic(6);n(4,:)

ans =

     8    28    33    17    10    15
```

b) **Show the results of 'x' multiply by 'y' and 'y' divided by 'x'.**
Given x=[0:0.1:1.1] and y =[10:21]

We will perform these operations in the Command Terminal of MATLAB as demonstrated in the screenshot below:

```
>> x=[0:0.1:1.1];y=[10:21];
>> x.*y

ans =

         0    1.1000    2.4000    3.9000    5.6000    7.5000    9.6000   11.9000   14.4000   17.1000   20.0000   23.1000

>> y.\x

ans =

         0    0.0091    0.0167    0.0231    0.0286    0.0333    0.0375    0.0412    0.0444    0.0474    0.0500    0.0524
```

Note: *A.*B is the element-by-element product of A and B. & A.\B is the matrix with elements B(i,j)/A(i,j).*

c) **Generate a random matrix 'r' of size 4 by 5 with numbers varying between -8 and 9**

To generate a random matrix we will use the **randi()** function which will return uniformly distributed pseudorandom integers. To fulfill the task at hand we will use the constructor that takes in the parameters *imax* and *sz* as follows: randi(imax,sz). The parameter *imax* is the largest integer from the specified sample interval and *sz* is the size vector. This matrix may generated as shown in the following screenshot:

```
>> r = randi([-8, 9], [4,5])


r =

     6     3     9     9    -1
     8    -7     9     0     8
    -6    -3    -6     6     6
     8     1     9    -6     9
```

Exercise 2

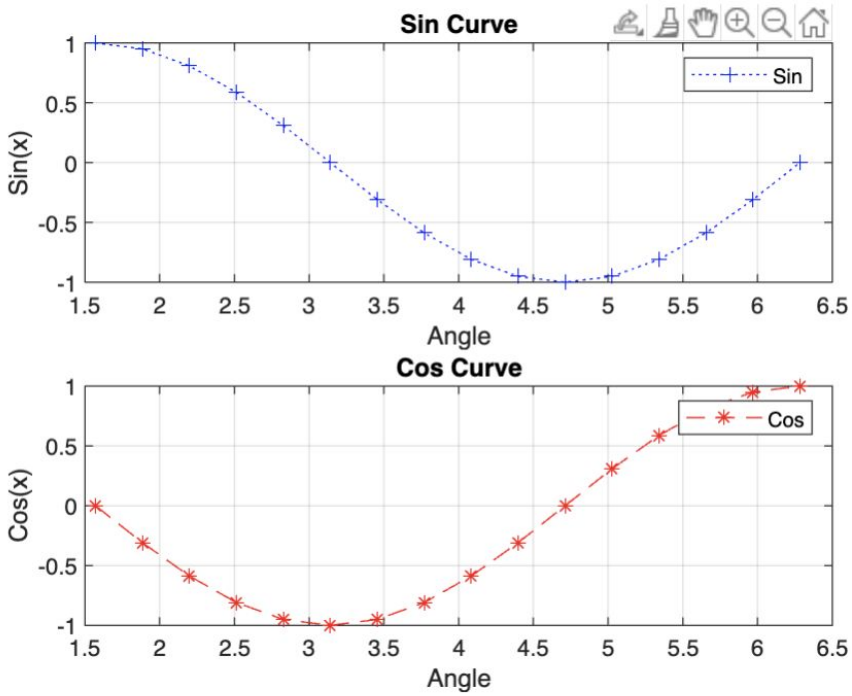
a) **Use MATLAB commands to obtain a plot of the sine and cosine curves**

To generate a plot we must first define our data points. The x-axis will be a matrix ranging from $\pi/2$ to 2π with an incrementation of $\pi/10$. The y axes will be a function of sin(x) and a function of cos(x). We will plot both curves onto subplots such that they may be displayed as one figure.. The **subplot()** function may take in three parameters as follows: subplot(m,n,p) which divides the figure into a *m*-by-*n* grid and creates axes specified by the position, *p*.

Next we used the **plot()** function to create a 2-D line plot by specifying the **x** & **y** vectors (which must be of equal length). To customize the line style,color, marker symbol and color we may add another parameter, referred to as *LineSpec*. For this figure we will plot the sine curve with dotted lines as line style, plus signs as markers, with a blue color - thus the parameter for *LineSpec* would be ':+b'. The cosine curve will be plotted with dashed lines as line style, asterisks as markers, with a red color - therefore the parameter for *LineSpec* would be '--*r'. For more information on customizing plots with *LineSpec* click here.

Plots may have titles and are applied using the **title()** function with the desired title text passed as a parameter. To apply a legend we will use the **legend()** function and pass the label as a parameter (e.g. Sin and Cos). The axes may be labeled using the **xlabel()** & **ylabel()** functions for the x & y axes. Grids may also be applied to plots using the **grid on** command and removed using **grid off**. In the figure shown below, we have also customized the x-axis by applying ticks on particular values using the **xticks()** function and passing the desired set values as a parameter.

Finally we set the limits of the x-axis using **xlim()** function and pass the *limits* parameter which is a two element vector of the form [xmin,xmax].



Polynomial Representation & Operations

In this portion of the laboratory exercise we will learn to represent polynomials, determine the roots of polynomials, and generate polynomials of known roots & obtain the corresponding partial fractions.

Polynomials are represented, in MATLAB, as row vectors which contain the coefficients ordered from the highest to lowest power (e.g. The polynomial $p(x)= x^3 - 2x -5$ is represented as p=[1 0 -2 -5]). To calculate the roots of a polynomial, one may use the function **roots()** and pass the corresponding row vector representing the polynomial as a parameter. The roots are then stored as column vectors. To determine the polynomial coefficients from the roots, one may use the **poly()** function and pass the column vector as a parameter - which returns a row vector representing the corresponding polynomial.

Evaluation of polynomials may be executed using the **polyval()** function as follows: polyval(p,x). The parameter *p* is a polynomial represented as a vector and *x* is a vector specifying the desired query points in which the polynomial is to be evaluated. The convolution and deconvolution of these polynomials may be performed using the **conv()** and **deconv()** functions respectively where the two parameters that are passed, are two vectors representing the coefficients of the polynomials. Derivatives of polynomials may also be computed using MATLAB, with the **polyder()** function - which returns a vector representing the coefficients of the derivative.

Partial fraction expansion, or partial fraction decomposition, is an operation that *decomposes* a rational function into the sum of simpler rational terms. This operation is often used in the integration of rational functions and deducing the inverse z and laplace transforms of complex functions. In MATLAB, the **residue()** function returns the residues, poles, and direct terms of a partial fraction expansion of the ratio of the given two polynomials, passed as parameters.

When the **residue()** function is provided three parameters, the residues,poles, and a polynomial - the original ratio of the two polynomials is returned in the form of vector coefficients.

To demonstrate our understanding of polynomial representation and operations in MATLAB we will execute the following tasks:

Exercise 1

Consider the two polynomials $p(s) = s^2 + 2s + 1$ and $q(s) = s + 1$.
Using MATLAB compute the following:

a) **$p(s) \times q(s)$**

MATLAB represents the polynomials as row vectors with coefficients ordered by descending powers. Therefore $p(s)$ = s^2 + 2s + 1 is represented as p=[1 2 1] and $q(s)$ = s + 1 as q=[1 1].

The multiplication of polynomials is referred to convolution and we will use the function **conv()** to implement said operation as follows:

```
>> p=[1 2 1];q=[1 1];c=conv(p,q)


c =

     1     3     3     1
```

b) **Roots of $p(s)$ and $q(s)$**

We will use the roots function to calculate the roots of the polynomials p(s) & q(s) as shown below:

```
>> p=[1 2 1];q=[1 1];
rootp=roots(p)
rootq=roots(q)

rootp =

    -1
    -1


rootq =
```

c) **p(-1) and q(6)**

To evaluate a polynomial at a particular value we will use the polyval() function as shown below:

```
>> p=[1 2 1];q=[1 1];
polyval(p,-1)
polyval(q,6)

ans =

     0


ans =

     7
```

Exercise 2

Use MATLAB to find the partial fractions of the following equations:

a) $\frac{B(s)}{A(s)} = \frac{2s^3+5s^2+3s+6}{s^3+6s^2+11s+6}$

Given,

$B(s) = 2s^3 + 5s^2 + 3s + 6$

$A(s) = s^3 + 6s^2 + 11s + 6$

```
>> b=[2 5 3 6];
a=[1 6 11 6];
[r,p,k] = residue(b,a)

r =

   -6.0000
   -4.0000
    3.0000


p =

   -3.0000
   -2.0000
   -1.0000


k =

     2
```

We will use the **residue()** function to find the partial fraction expansion of the ratio of two defined polynomials, as shown below:

b) $\frac{B(s)}{A(s)} = \frac{s^2+2s+3}{(s+1)^3}$

Given,

$B(s) = s^2 + 2s + 3$

$A(s) = (s+1)^3$

We will use the **residue()** function to find the partial fraction expansion of the ratio of two defined polynomials, as shown below:

```
>> b=[0 1 2 3];a=[1 3 3 1];
[r,p,k] = residue(b,a)

r =

    1.0000
    0.0000
    2.0000


p =

   -1.0000
   -1.0000
   -1.0000


k =

     []
```

Execution of Scripts,Function & Flow Controls

MATLAB is a high level programming language & an interactive development environment where users can create scripts and functions using the native language and store them as M-files. A script is the simplest type of MATLAB program that is a file which contains several sequential lines of MATLAB commands and function calls. To create a script one may use the **edit** command in the terminal followed by the name of the M-file. To execute the written script, one may simply type the name of the program in the terminal. The MATLAB IDE also supports the development of *live scripts* which enable users to interact with both the code and the output (e.g. images,equations, and formatted text). These live scripts, unlike MATLAB scripts and functions, are stored as .mlx files.

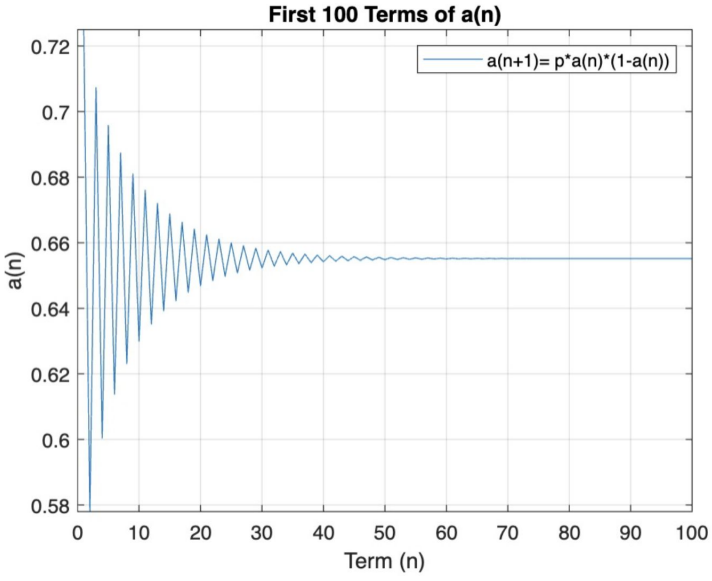
Like most programming languages, MATLAB supports the implementation of loops and conditionals. Loop control statements execute a particular section of code of which is enclosed within it repeatedly (e.g. for loops, parallel for loops & while loops) and conditional statements control which section of code to execute during the runtime of the program or script (e.g. if,else,if,else,switch,case,otherwise).

In this portion of the laboratory exercise we will perform the following tasks to demonstrate our understanding of MATLAB script and function generation and implementation of various flow control operations :

Exercise 1

Use MATLAB commands to generate the first 100 terms in the sequence a(n) defined recursively by $a(n+1)= p \times a(n) \times (1-a(n))$ with p=2.9 and a(1)=0.5. Upon generation of the sequence plot the resulting terms.

To generate the terms of the sequence, we first created the symbolic variables *a*,*p*, and *n*. Then we defined the variables accordingly to the provided initial terms. Next, we utilized the for loop to iterate 100 times and created an array to store the calculated terms dynamically. Finally, we used the acquired array and created a plot appropriately, using our knowledge of customizing figures and plots as shown below:



Exercise 2

Consider the following equation:

$$y(t) = \frac{y(0)}{\sqrt{1-\zeta^2}} e^{-\zeta\omega t} \sin(\omega\sqrt{1-\zeta^2}t + \theta)$$

a) **Write a MATLAB M-file function to obtain numerical values of y(t). Your function must take y(0), ζ, ω_n, t, θ as function inputs and y(t) as output argument.**

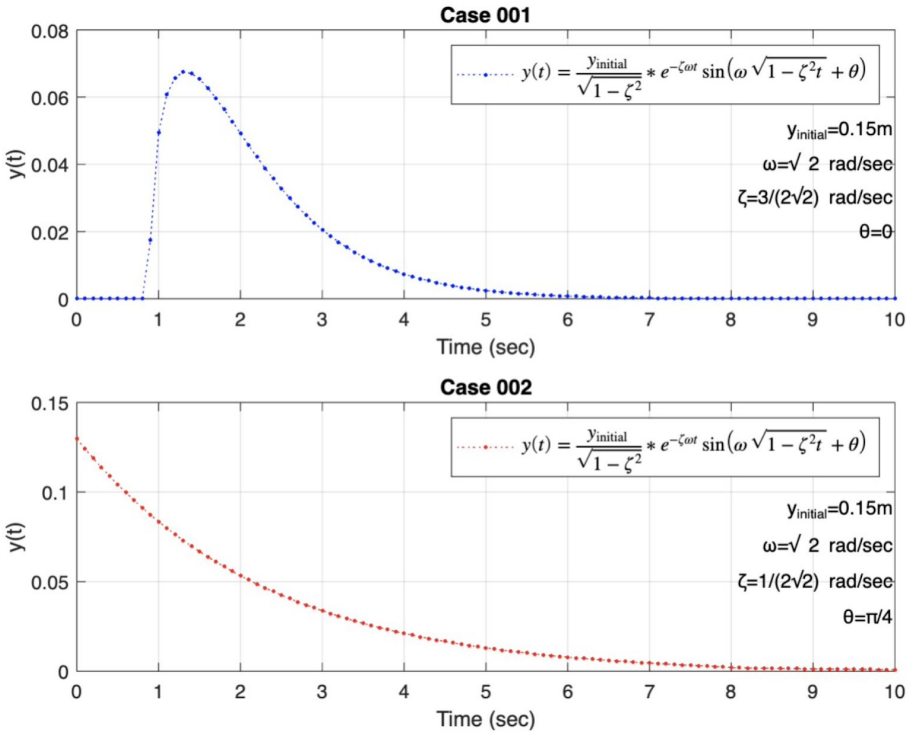
```
function [y] = f(yinital,zeta,omega,t,theta)
   y= yinital / sqrt(1- zeta^2) * exp(-zeta*omega*t) * sin(omega*(sqrt(1-(zeta^2)*t)) + theta);

end
```

b) **Write a second script m-file to obtain the plot for y(t) for 0<t<10 with an increment of 0.1, by considering the following two cases:**

Case 1: y_0=0.15m, $\omega = \sqrt{2}\ \frac{rad}{sec}$, $\zeta = \frac{3}{2\sqrt{2}}$ and θ=0

Case 2: y_0=0.15m, $\omega = \sqrt{2}\ \frac{rad}{sec}$, $\zeta = \frac{1}{2\sqrt{2}}$ and $\theta = \frac{\pi}{4}$



Exercise 3

Use a 'for' or 'while loop to convert degrees Fahrenheit (T_f) to degrees Celsius using the following equation: $T_f = \frac{9}{5} T_C + 32$. Use any starting temperature,increment and ending temperature (e.g. Starting Temperature=0, Increment=10, Ending Temperature=200)

For this exercise we generated a table to display T_C and T_F from 0°C to 100°C and the corresponding value in °F with an increment of 1°C as shown below:

	Degrees Celsius	Degrees Farenheit
1	0	32
2	1	33.8000
3	2	35.6000
4	3	37.4000
5	4	39.2000
6	5	41
7	6	42.8000
8	7	44.6000
9	8	46.4000
10	9	48.2000
11	10	50
12	11	51.8000
13	12	53.6000
14	13	55.4000
15	14	57.2000

Conclusion

In this lab we explored the MATLAB IDE and programming language by performing various exercises. We learned the MATLAB syntax and various predefined functions and how to write scripts & custom functions to perform desired computations and operations. This laboratory exercise explored matrix operations (e.g. extracting a particular row, generating a randomized matrix) and methods to plot data with the addition of customizing the output (e.g. titling plots, labelling axes, applying legends, and customizing lines on a 2D plot). We also explored polynomial representation using appropriate MATLAB syntax and performing convolution and deconvolution of said polynomials. In addition, we learned to use the residue() function to find the partial fraction expansions of the ratio of two polynomials and perform the inverse of this operation using the same function. At the final stage of this lab, we combined our knowledge of the aforementioned topics and created scripts and functions to execute computational operations within control structures such as for loops and while loops. As a result, we have established a fundamental understanding of the MATLAB environment and language to perform future tasks which may require the execution of the explored operations and methods.

Appendix

Matrix Operations & Plotting (Code)

Exercise 1

a)
```
n=magic(6);n(4,:)
```
b)
```
x=[0:0.1:1.1];y=[10:21];
x.*y
y.\x
```
c)
```
r = randi([-8, 9], [4,5])
```

Exercise 2

```
x=pi/2:pi/10:2*pi;
y=sin(x);
z=cos(x);


subplot(2,1,1);
plot(x,y,':+b')
grid on
title('Sin Curve')
legend('Sin')
xlabel('Angle')
ylabel('Sin(x)')
xlim([1.5,6.5])
xticks([1.5 2 2.5 3 3.5 4 4.5 5 5.5 6 6.5])

hold on

subplot(2,1,2);
plot(x,z,'--*r')
title('Cos Curve')
legend('Cos')
xlabel('Angle')
ylabel('Cos(x)')
xlim([1.5,6.5])
xticks([1.5 2 2.5 3 3.5 4 4.5 5 5.5 6 6.5])
grid on
```

Polynomial Representation & Operations (Code)

Exercise 1

a)
```
p=[1 2 1];
q=[1 1];
c=conv(p,q)
```

b)
```
p=[1 2 1];
q=[1 1];
rootp=roots(p)
rootq=roots(q)
```

c)
```
p=[1 2 1];
q=[1 1];
polyval(p,-1)
polyval(q,6)
```

Exercise 2

a)
```
b=[2 5 3 6];
a=[1 6 11 6];
[r,p,k] = residue(b,a)
```

b)
```
b=[0 1 2 3];
a=[1 3 3 1];
[r,p,k] = residue(b,a)
```

Execution of Scripts,Function & Flow Controls (Code)

Exercise 1

```
syms a p n mat tick

a=0.5;
p=2.9;
tick=(0:10:100)

   for i= 1:100
    a= p* a * (1-a);
    mat(i)= vpa(a,3);
   end

plot((1:100),mat)
title('First 100 Terms of a(n)')
legend('a(n+1)= p*a(n)*(1-a(n))')
xlabel('Term (n)')
ylabel('a(n)')
grid on
ylim([.578,.725])
xticks(tick)
```

Exercise 2

```
syms yinital zeta omega t theta yt

%%%%%%%%%%%%%%%%%%%%%%%%%%%%Case 1%%%%%%%%%%%%%%%%%%%%%%%%%%%%

yinital=0.15;       % meters
omega=sqrt(2);      % rad/sec
zeta=3/(2*sqrt(2)); %
theta=0;            %

t=(0:.1:10);

for i= 1:length(t)
   yt(i)=f(yinital,zeta,omega,t(i),theta);
end

subplot(2,1,1);
plot(t,yt,':.b')
grid on
title('Case 001')
```

```
h=legend(['$$y\left(t\right)=\frac{y_{\mathrm{initial}\;} }' ...
   '{\sqrt{1-\zeta^2 }}*e^{-\zeta \omega t} \;\mathrm{sin}\' ...
   'left(\omega \sqrt{1-\zeta^2 t\;}+\theta \right)\;$$'])
set(h,'Interpreter','latex','fontsize',10)

text(10,.05,['y_{initial}=0.15m'],'HorizontalAlignment','right')
text(10,.04,['\omega=\surd 2 rad/sec'],'HorizontalAlignment','right')
text(10,.03,['\zeta=3/(2\surd2) rad/sec'],'HorizontalAlignment','right')
text(10,.02,['\theta=0'],'HorizontalAlignment','right')


xlabel('Time (sec)')
ylabel('y(t)')

%%%%%%%%%%%%%%%%%%%%%%%%%Case 2%%%%%%%%%%%%%%%%%%%%%%%%%

yinital=0.15;       % meters
omega=sqrt(2);      % rad/sec
zeta=1/(2*sqrt(2)); %
theta=pi/4;         %

t=(0:.1:10);

for i= 1:length(t)
   yt(i)=f(yinital,zeta,omega,t(i),theta);
end

subplot(2,1,2);
plot(t,yt,':.r')
grid on
title('Case 002')

h=legend(['$$y\left(t\right)=\frac{y_{\mathrm{initial}\;} }' ...
   '{\sqrt{1-\zeta^2 }}*e^{-\zeta \omega t} \;\mathrm{sin}\' ...
   'left(\omega \sqrt{1-\zeta^2 t\;}+\theta \right)\;$$'])
set(h,'Interpreter','latex','fontsize',10)

text(10,.09,['y_{initial}=0.15m'],'HorizontalAlignment','right')
text(10,.07,['\omega=\surd 2 rad/sec'],'HorizontalAlignment','right')
text(10,.05,['\zeta=1/(2\surd2) rad/sec'],'HorizontalAlignment','right')
text(10,.03,['\theta=\pi/4'],'HorizontalAlignment','right')

xlabel('Time (sec)')
ylabel('y(t)')
```

Exercise 3

```
% MATLAB Flow Control
% Use ‘for’ or while loop to convert degrees Fahrenheit (Tf) to degrees
% Celsius using the following equation Tf= (9/5)*Tc + 32. Use any starting
% temperature, increment and ending temperature.

syms tf tc iterator


iterator=1;

for x= 0:1:100
  tc(iterator)=x;
  tf(iterator)=(9/5)*tc(iterator) + 32;
  iterator=iterator+1;
end

%Transpose the Row Vector into a Column Vector
tc=double(transpose(tc));
tf=double(transpose(tf));

%Create a Figure containing a Table for Visualization
f=figure;
headers={'Degrees Celsius','Degrees Farenheit'}
data=[tc,tf]
h = uitable(f,'Data',data, 'ColumnName',headers);
```