# Matlab Examples

(v.01, Fall 2011, Ex. 1-50 prepared by HP Huang; Ex 51-55 prepared by Noel Baker)

*These examples illustrate the uses of the basic commands that will be discussed in the Matlab tutorials. This collection does not reflect the full scope of the tutorials/lectures, which will be more extensive.*

*This collection will be expanded as our Matlab tutorials progress through the semester.*

# Lecture 1. Write your first Matlab programs

## Ex. 1 Write your first Matlab program

```
a = 3;
b = 5;
c = a+b
```

*Output:*

*8*

Remarks: (1) The semicolon at the end of a statement acts to suppress output (to keep the program running in a "quiet mode"). (2) The third statement, *c = a+b*, is not followed by a semicolon so the content of the variable *c* is "dumped" as the output.

## Ex. 2 The meaning of "a = b"

**In Matlab and in any programming language, the statement "a = b" does not mean "a equals b". Instead, it prompts the action of *replacing the content of a by the content of b*.**

```
a = 3;
b = a;
b
```

*Output:*

*3*

Remark: Think of the two "variables" **a** and **b** as two buckets labeled "a" and "b". The first statement puts the number 3 into bucket **a**. The second statement puts **the content of bucket a** into bucket **b**, such that we now have "3" in bucket **b**. (The content of bucket **a** remains unchanged after this action.) The third statement dumps the content of bucket **b** so the final output is "3".

**Ex. 3  Basic math operations**

```
a = 3;
b = 9;
c = 2*a+b^2-a*b+b/a-10
```

*Output:*

*53*

Remark: The right hand side of the third statement includes all 4 of the basic arithmetic operators, + (addition), - (subtraction), * (multiplication), and / (division), in their usual meaning.  It also includes the symbol, ^, which means "to the power of", so "b^2" means (the content of b) to the power of 2, i.e., $9^2 = 81$.  The right hand side of that statement is first evaluated: r.h.s. = $2 \times 3 + 9^2 - 3 \times 9 + 9/3 - 10 = 53$.  The content of r.h.s., now 53, is then assigned to the variable c in the left hand side. Since this statement is not followed by a semicolon, the content of c is dumped as the final output.

**Ex. 4 The meaning of "a = b", continued**

```
a = 3;
a = a+1;
a
```

*Output:*

*4*

Remark: The value of variable **a** is 3 after the first statement. In the second statement, the right hand side is first evaluated to be 3+1 = 4.  This value is then used to replace the old content of **a**, so **a** becomes 4 instead of 3.

**Ex. 5  Formatted output**

```
fprintf('Hello')
```

*Output*:

Hello

**Ex. 6 Formatted output**

```
a = 3;
b = a*a;
c = a*a*a;
d = sqrt(a);
fprintf('%4u square equals %4u \r', a, b)
fprintf('%4u cube equals %4u \r', a, c)
fprintf('The square root of %2u is %6.4f \r', a, d)
```

*Output*:

```
   3 square equals    9
   3 cube equals   27
 The square root of  3 is 1.7321
```

Remarks: The command "fprintf" is for formatted output, using the format specified in the first string '...' in the parentheses. The "%4u" (4-digit integer) and "%6.4f" (real number that preserves 4 digits to the right of the floating point) are the format for the variable(s) for output. The "sqrt" in the 4th statement is the intrinsic function for square root.

**Ex. 7 Arrays**

```
a = [3  6  7];
b = [1  9  4];
c = a + b
```

Output:

```
4  15  11
```

Remarks: (1) Both *a* and *b* are given as a three-element array. In the third line, the operation of "*a*+*b*" results in element-by-element addition

**Ex. 8  Extracting an individual element of an array**

```
a = [3  6  7];
b = [1  9  4  5];
c = a(2) + b(4)
```

Output:

*c = 11*

Remark: If **b** is a one-dimensional array, **b**(*n*) is the *n*-th element of that array. Since a(2) is 6 and b(4) is 5, the 3rd statement leads to c = 6+5 = 11.

**Ex. 9  Comment**

```
%
% This program demonstrates how to "comment out"
% a segment of code
%
A = 3;
B = A*A;
%
% B = 2*B   <--- This statement is not executed
%
C = A+B
```

Output:

*c = 12*

**Ex. 10 Continuation to next line**

```
summation1 = 1 + 3 + 5 + 7 ...
 + 9 + 11
```

Note: The three periods (...) allow continuation to the next line of commands. The two lines in the above example are essentially one line of "summation1 = 1+3+5+7+9+11".

**Ex. 11 "Clear" a variable**

```
c1 = 3;
c2 = c1+5;
clear c1
c1
```

Output:

*??? Undefined function or variable 'c1'.*

Remarks: We see an error message because the variable "c1" no longer exists. It is purged from the computer memory by the "clear" command. Note that the command does not just act to delete the content of a variable, but it kills the variable outright. The 3rd statement can be useful if c1 is a big array that occupies a lot of memory but is no longer needed for the rest of the program. The 3rd statement only kills c1, while c2 (= 8) still exists. A "clear" command not followed by any variable will kill all variables.

**Ex. 11 Intrinsic math functions and constants**

```
x = pi;
y = sin(pi/2)
z = exp(-sin(pi/2))
```

Output:

*y = 1*
*z = 0.3679*

Remarks: "pi" (= 3.14159...) is a reserved intrinsic constant. A function within a function is allowed. The innermost function will be evaluated first, so the 3rd statement leads to z = exp(-1) = 0.3679.

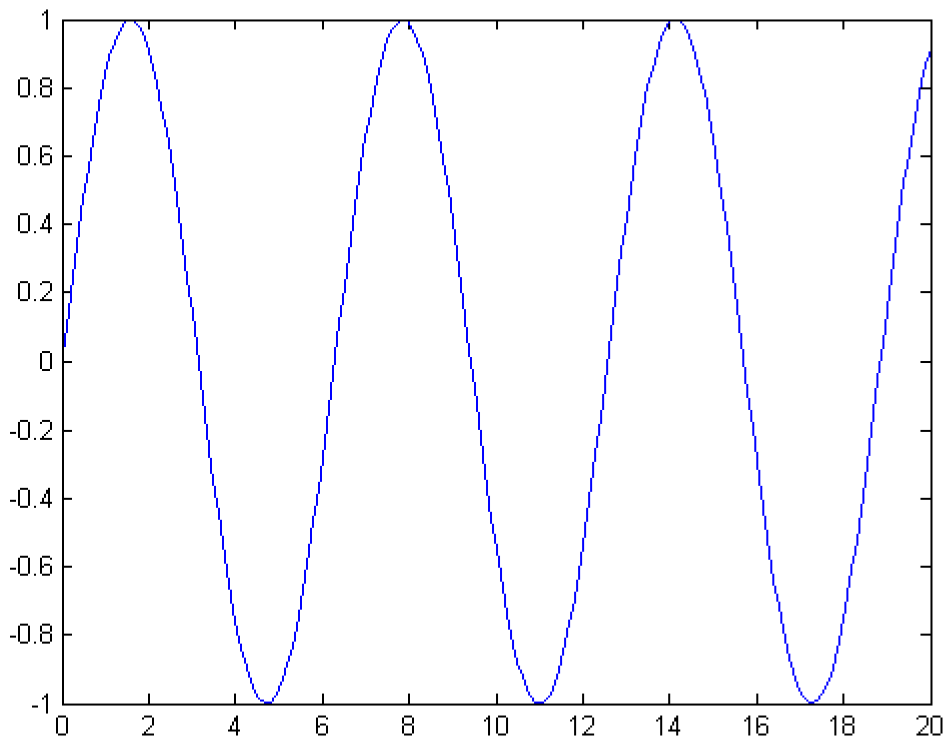
**Ex. 12 Naming a variable**

(i) Matlab variables are case sensitive. For example, "ASU" and "asu" are two different variables. (ii) An underscore (_) or a number (0-9) can also be part of the name of a variable. For example, "MAE_384" is a legitimate variable name. (iii) Some names are reserved for special constants. For example (see Ex. 11), "pi" is an intrinsic constant with a fixed value of 3.14159...

**Ex. 13 Making a quick plot**

```
x = [0:0.1:20];
y = sin(x);
plot(x,y)
```

The outcome will be the following plot:




Remarks: This only serves as a very quick example of what Matlab can do in making plots. will have more detailed discussions on the use of arrays and Matlab's graphic tools in later lectures. The first line is equivalent to x = [0 0.1 0.2 0.3 ... 19.8 19.9 20]. It assigns the content of x which is an array of 201 elements. The "0:0.1:20" means the 201 numbers are evenly spaced. They start from 0 and end at 20 with an increment of 0.1. The second line gives the content of the new array, y, as

y = [sin(x(1)) sin(x(2)) sin(x(3)) ... sin(x(200)) sin(x(201))] ,

or

y = [sin(0) sin(0.1) sin(0.2) ... sin(19.9) sin(20)] .

The 3rd line makes a plot of y vs. x.

# Lecture 2. Basic looping

## 1. The *for* loop

**Ex. 14** Loop: Using **for** command

```
b = 3;
for k = 1:5
 b
end
```

*Output:*

*3*
*3*
*3*
*3*
*3*

Remark: The blue-colored segment in lines 2-4 forms a "for-loop". The statement sandwiched between "for k = 1:5" and "end" is repeated 5 times, with the "k" index going from 1 to 5 step 1.

**Ex. 15** *For* loop: Utility of the dummy index

```
b = 3;
for k = 1:5
 b^k
end
```

*Output:*

*3*
*9*
*27*
*81*
*243*

Remark: The outputs are 3^1, 3^2, 3^3, 3^4, and 3^5. the value of "k" keeps changing as we go through the loop

**Ex. 16** *For* loop: More on the dummy index

```
sum1 = 0;
for k = 1:9
  sum1 = sum1+k;
end
sum1
```

Output:

*45*

Remark: this program performs the summation of 1+2+3+4+5+6+7+8+9 (= 45).

**Ex. 17** *For* loop: More on the dummy index

```
sum1 = 0;
for k = 1:2:9
  sum1 = sum1+k;
end
sum1
```

Output:

*25*

Remark: this program performs the summation of 1+3+5+7+9 (= 25). The command "for k = 1:2:9" means we go through the loop only 5 times. First time with k = 1, second time with k = 1+2 (=3), third time with k = 1+2+2 (=5), and so on. The looping stops once k reaches 9.

**Ex. 18** Treatment of array within a loop

```
b = [3  8  9  4  7  5];
sum1 = 0;
for k = 1:4
  sum1 = sum1+b(k);
end
sum1
```

*Output:*

*24*

Remark: This program performs the summation of sum1 = b(1)+b(2)+b(3)+b(4) = 3+8+9+4 = 24

**Ex. 19** Treatment of array within a loop

```
b = [3  8  9  4  7  5];
sum1 = 0;
for k = 1:2:5
  sum1 = sum1+b(k);
end
sum1
```

*Output:*

*19*

Remark: This program performs the summation of sum1 = b(1)+b(3)+b(5) = 3+9+7 = 19

**Ex. 20** Double loop

```
sum1 = 0;
for n = 1:2
   for m = 1:3
      sum1 = sum1+n*m;
   end
end
sum1
```

*Output:*

*16*

Remark: this program performs the summation of
Sum1 = 1*1+1*2+1*3 +2*1+2*2+2*3 = 16

**Ex. 21** Double loop

```
for n = 1:2
   for m = 1:3
      fprintf('n = %3u  m = %3u \r', n, m)
   end
end
```

*Output:*

```
  n = 1    m = 1
  n = 1    m = 2
  n = 1    m = 3
  n = 2    m = 1
  n = 2    m = 2
  n = 2    m = 3
```

**Ex. 22** More complicated use of loop and index

```
b = [2  5  7  4  9  8  3];
c = [2  3  5  7];
sum1 = 0;
for k = 1:4
   sum1 = sum1+b(c(k));
end
sum1
```

Output:

24

Remark: This program performs the summation of

```
sum1 = b(c(1))+b(c(2))+b(c(3))+b(c(4))
        = b(2)+b(3)+b(5)+b(7)
        = 5+7+9+3
        = 24
```

# Lecture 3. Basic branching

**Ex. 23** The *if* command

```
num1 = 7;
if (num1 > 5)
   fprintf('%4u is greater than 5 \r', num1)
else
   fprintf('%4u is less than or equal to 5 \r', num1)
end
```

*Output:*

*7 is greater than 5*

Same program, but change first line to "num1 = 3;"

*Output:*

*3 is less than or equal to 5*

Remark: In this program, if (num1 > 5) (num1 is greater than 5) is true, the statement "fprintf('%4u is greater than 5 \r', num1)" is executed. Otherwise, the statement "fprintf('%4u is less than or equal to 5 \r', num1)" is executed.

**Ex 24** *if* - *elseif* - *else* (This example is self-explanatory. Try to change the given value of num1 and observe the outcome.)

```
num1 = 4;
if (num1 >= 5)
   fprintf('%4u is greater than or equal to 5 \r', num1)
elseif (num1 > 1)
   fprintf('%4u is less than 5 but greater than 1 \r', num1)
elseif (num1 == 1)
   fprintf('%4u equals 1 \r', num1)
elseif (num1 > -3)
   fprintf('%4u is less than 1 but greater than -3 \r', num1)
else
   fprintf('%4u is less than or equal to -3 \r', num1)
end
```

**Ex 25** An application - determine whether a given year is a leap year (try to change the given value of nyear and observe the outcome)

```
nyear = 1975;
if (mod(nyear, 400) == 0)
  fprintf('%6u is a leap year', nyear)
elseif (mod(nyear,4) == 0) & (mod(nyear,100) ~= 0)
  fprintf('%6u is a leap year', nyear)
else
  fprintf('%6u is not a leap year', nyear)
end
```

*Output:*

*1975 is not a leap year*

Remarks:

(1) In the *elseif* command (4th line), "&" means "AND". Both statements "(mod(nyaer,4) == 0)" and "(mod(nyear,100) ~= 0)" have to be true for Matlab to execute the command, "fprintf('%6u is a leap year', nyear)". Also commonly used in an *if* statement is "|" (a vertical line), which means "OR".

(2) The symbols "~=" in line 4 means "NOT EQUAL TO". There are 6 commonly used expressions to compare two numbers in an *if* command:

| | |
|---|---|
| A > B | A is greater than B |
| A < B | A is less than B |
| A >= B | A is greater than or equal to B |
| A <= B | A is less than or equal to B |
| A == B | A equals B |
| A ~= B | A does not equal B |

(3) The "mod(A,B)" function returns the remainder of A divided by B. For example, mod(7,2) = 1, mod(10,4) = 2, and mod(25,5) = 0. If A is divisible by B, mod(A,B) = 0. This is a very useful function in many applications related to numerical methods.

**Ex 26** Combine looping and branching

```
sum1 = 0;
sum2 = 0;
N = 9
for k = 1:N
   sum1 = sum1+k;
   if (mod(k,3) == 0)
      sum2 = sum2+k;
   end
end
sum1
sum2
```

*Output:*

*sum1 = 45*
*sum2 = 18*

Remark: Sum1 = 1+2+3+4+5+6+7+8+9, while sum2 = 3+6+9.

**Ex 27** The *while* loop

```
x = 3;
while (x < 100)
   x = x*3;
end
x
```

*Output*:

x = 243

Remark: One can think of a *while* loop as a combination of a *for* loop and an *if* statement. Here, the looping will keep going indefinitely as long as the condition, (x < 100), is satisfied. Therefore, the value of x progresses from 3, 9, 27, 81, to 243 when the loop is terminated.

# Lecture 4A. Array and Matrix

## 1. Assign the content of an array/matrix; Basic operations

**Ex. 28** Assign the content of a (one-dimensional) array; Addition of two arrays

```
a = [2 12 25];
b = [3 7 4];
c = a+b
```

*Output*:

```
c = 5  19  29
```

**Ex. 29** Assign the content of a matrix; Addition of two matrices

```
a = [3 4; 1 6];
b = [5 2; 11 7];
c = a+b
```

*Output*:

```
c =    8    6
      12   13
```

This program performs the following acts:

$$\boldsymbol{a} = \begin{bmatrix} 3 & 4 \\ 1 & 6 \end{bmatrix}$$

$$\boldsymbol{b} = \begin{bmatrix} 5 & 2 \\ 11 & 7 \end{bmatrix}$$

$$\boldsymbol{c} = \boldsymbol{a} + \boldsymbol{b} = \begin{bmatrix} 8 & 6 \\ 12 & 13 \end{bmatrix}$$

**Ex. 30** Multiplication involving a scalar and an array (or a matrix)

```
a = [3 5; 1 4];
b = 2*a
```

Output:

```
b =   6   10
      2    8
```

**Ex. 31** Element-by-element multiplication involving two 1-D arrays or two matrices of the same dimension

```
a = [2 3 5];
b = [2 4 9];
c = a.*b
```

*Output*:

```
c = 4   12   45
```

Remark: The period preceding the mathematical operation, "*", indicates that the operation will be performed element-by-element. In this case, the content of c is

```
c = [a(1)*b(1)  a(2)*b(2)  a(3)*b(3)]
```

Also, c is automatically assigned as a 1-D array with 3 elements

**Ex. 32** Element-by-element multiplication of two matrices

```
a = [2 3; 1 4];
b = [5 1; 7 2];
c = a.*b
```

*Output*:

```
c = 10   3
     7   8
```

**Ex. 33** Direct (not element-by-element) multiplication of two matrices

```
a = [2 3; 1 4];
b = [5 1; 7 2];
c = a*b
```

*Output*:

```
c = 31   8
    33   9
```

Remark: Observe how the outcome of this example differs from Ex. 32.

**Ex. 34** Elementary functions with a vectorial variable

```
a = [2 3 5];
b = sin(a)
```

Output:

```
b = 0.9092   0.1411   -0.9589
```

Remark: The content of b is [sin(2)  sin(3)  sin(5)].

**Ex. 35** Another example of elementary functions with a vectorial variable

```
a = [2 3 5];
b = 2*a.^2+3*a+4
```

Output:

```
b = 18  31  69
```

Remark: The content of b is

b = [2*(a(1))^2+3*a(1)+4    2*(a(2))^2+3*a(2)+4    2*(a(3))^2+3*a(3)+4].

**Ex. 36** An efficient way to assign the content of an array

```
a = [0:0.5:4];
a
```

*Output*:

```
a = 0  0.5  1  1.5  2  2.5  3  3.5  4
```

**Ex 37. Extracting the individual element(s) of a matrix**

```
A = [3  5; 2  4];
c = A(2,2)+A(1,2)
```

Output:

```
c = 9
```

Remark: With the given A matrix, we have A(1,1) = 3, A(1,2) = 5, A(2,1) = 2, and A(2,2) = 4.

**Ex. 38** Another example for the usage of index for a matrix

```
A = [3  5; 2  4];
norm1 = 0;
for m = 1:2
for n = 1:2
   norm1 = norm1+A(i,j)^2;
end
end
norm1 = sqrt(norm1)
```

*Output*:

```
norm1 = 7.348
```

Remark: This program calculates the Euclidean norm of the A matrix.

**Ex. 39** Solving a system of linear equation

```
A = [4  1  2; 0  3  1; 0  1  2];
b = [17 ; 19 ; 13];
x = inv(A)*b
```

Output:

```
x  =  1
      5
      4
```

Remark: This program solves [**A**] **x** = **b**. The solution is obtained by **x** = [**A**]$^{-1}$ **b**.

**Ex. 40** An alternative to Ex. 39

```
A = [4  1  2; 0  3  1; 0  1  2];
b = [17 ; 19 ; 13];
x = b\A
```

The output is the same as Ex. 39. Here, b\A is essentially inv(A)*b. (The "\" is called "back divide" in Matlab documentations.)

## Lecture 4B. Basic graphic applications

(This section will be updated soon)

**Ex. 41** A quick example of plot command: Draw a curve

```
a = [0:0.5:5];
b = 2a.^2 + 3*a -5;
plot(a,b)
```

Remarks:
(1) In "plot(a,b)", the array "a" should contain the data of the coordinate or "grid point) on the x-axis and "b" should be the corresponding values on the y-axis.
(2) After a plot is made, it can be further modified by using the interactive tool for graphics. For example, the labels of the x and y axes can be manually added to the plot.
(3) The plot can be saved in various formats (jpg, tif, eps, etc.).

**Ex. 42** Refine the plot: Line pattern, color, and thickness

```
a = [0:0.5:5];
b = 2a.^2 + 3*a -5;
plot(a,b,'-or','MarkerFaceColor','g','LineWidth',2)
xlabel('X'); ylabel('Y'); legend('Test')
```

**Ex. 43** Draw multiple curves

```
a = [0:0.5:5];
b = 2a.^2 + 3*a -5;
c =
plot(a,b)
```

**Ex. 44** Draw symbols

**Ex. 45** Control the range of the plot

**Ex. 46** Control the aspect ratio of the plot

**Ex. 47** Plot with multiple panels

**Ex. 48** Plotting a 2-D color map

**Ex. 49** Plotting a 2-D contour map

**Ex. 50** Variations of color/contour maps

## Lecture 5 & 6. More on graphic applications and input/output

(Examples 51-55 are prepared by Noel Baker.)

### Ex. 51

```
x = 0:0.5:10;
y1 = -sin(x);
y2 = (x-5).^2-1;
y3 = x.^2;
y4 = 2*x.^2;
y5 = 4*x.^2;

figure
plot(x,y1,'*r','LineWidth',2)
hold on
plot(x,y2,'Marker','p','Color',[0 .3 .2],'LineWidth',2)
axis([0 10 -2 2])
legend('y = -sin(x)','y = (x-5)^2 - 1')

figure
subplot(2,1,1)
plot(x,y1,'*r','LineWidth',2)
title('y = -sin(x)')

subplot(2,1,2)
plot(x,y2,'Color',[0 .3 .2],'Marker','s','MarkerEdgeColor','m', ...
    'MarkerSize',5,'LineWidth',2)
title('y = (x-5)^2 - 1')

figure
plot(x,y3,':k',x,y4,'--k',x,y5,'-.k')
legend('y = x^2','y = 2x^2','y = 4x^2',2)
axis([0 10 0 100])

Z = peaks;
figure
subplot(2,2,1)
contour(Z)
title('Contour plot (2D)')

subplot(2,2,2)
contourf(Z)
title('Filled contour plot (2D)')

subplot(2,2,3)
mesh(Z)
title('Mesh plot (3D)')

subplot(2,2,4)
surf(Z)
title('Surface plot (3D)')
```

**Ex. 52**

```
plot([1 4],[4 4],'k','LineWidth',2)
hold on
plot([1 4],[1 1],'k','LineWidth',2)
hold on
plot([1 1],[1 4],'k','LineWidth',2)
hold on
plot([4 4],[1 4],'k','LineWidth',2)
hold on
plot(2.5,4,'>k',2.5,1,'<k',1,2.5,'^k',4,2.5,'vk', ...
    'MarkerFaceColor','k','MarkerSize',15)
axis([0 5 0 5])

set(gca,'XTick',[1 4],'YTick',[1 4],'XTickLabel',
[],'YTickLabel',[],'Color',[.9 .9 .9])

xlabel('Entropy','FontSize',14)
ylabel('Temperature','FontSize',14)
title('Carnot Cycle','FontSize',16)

text(1,-.25,'S_A','HorizontalAlignment','center','FontSize',14)
text(4,-.25,'S_B','HorizontalAlignment','center','FontSize',14)
text(-.25,1,'T_C','HorizontalAlignment','center','FontSize',14)
text(-.25,4,'T_H','HorizontalAlignment','center','FontSize',14)
```

**Ex. 53**

```
salaries = [320616 0;242570 0;189217 0;95199 86311;70055 65692;17784
0;1057305 0];

bar(salaries)
set(gca,'YTick',0:200000:600000,'YTickLabel',
{'$000,000','$200,000', ...
    '$400,000','$600,000'})
set(gca,'XTick',[])


positions = [.8 1.8 2.8 0 0 5.8 6.8];

values = [320616 242570 189217 0 0 17784 1057305];
values2 = [95199 86311 70055 65692];
values3 =
{'$320,616','$242,570','$189,217','','','$17,784','$1,057,305'};
values4 = {'$95,199','$86,311','$70,055','$65,692'};

for q = [1:3,6,7]
    text(positions(q),values(q),values3(q),'HorizontalAlignment', ...
        'center','VerticalAlignment','bottom','FontSize',8)
end

text(.9,-80000,
{'University';'President/CEO'},'HorizontalAlignment','center', ...
    'FontSize',8)
text(1.9,-80000,'Provost','HorizontalAlignment','center', ...
    'FontSize',8)
text(2.9,-80000,'Deans','HorizontalAlignment','center', ...
    'FontSize',8)
text(4,-80000,{'Tenured';' '},'HorizontalAlignment','center', ...
    'FontSize',8)
text(5,-80000,{'Un-Tenured';' '},'HorizontalAlignment','center', ...
    'FontSize',8)
text(4.5,-100000,'Professors','HorizontalAlignment','center', ...
    'FontSize',8)
text(5.9,-80000,
{'Grad';'Students'},'HorizontalAlignment','center', ...
    'FontSize',8)
text(6.9,-80000,
{'Football';'Coaches'},'HorizontalAlignment','center', ...
    'FontSize',8)
text(3.8,-20000,'Male','HorizontalAlignment','center', ...
    'FontSize',6)
```

```
text(4.2,-20000,'Female','HorizontalAlignment','center', ...
    'FontSize',6)
text(4.8,-20000,'Male','HorizontalAlignment','center', ...
    'FontSize',6)
text(5.2,-20000,'Female','HorizontalAlignment','center', ...
    'FontSize',6)
text(3.8,values2(1)+30000,values4(1),'HorizontalAlignment', ...
        'center','VerticalAlignment','bottom','FontSize',8)
text(4.2,values2(2),values4(2),'HorizontalAlignment', ...
        'center','VerticalAlignment','bottom','FontSize',8)
text(4.8,values2(3)+30000,values4(3),'HorizontalAlignment', ...
        'center','VerticalAlignment','bottom','FontSize',8)
text(5.2,values2(4),values4(4),'HorizontalAlignment', ...
        'center','VerticalAlignment','bottom','FontSize',8)
text(.5,900000,'"Academic"
Salaries','FontSize',16,'FontWeight','bold')
text(.7,790000,{'Actual average and median salaries at US'; ...
    'Doctoral-granting Universities (2008 figures)'},'FontSize',12)

colormap summer
```

**Ex. 54**

```
clear all  % Clears variables from the workspace
close all  % Closes all open figures/GUIs
clc        % Clears command window

A = 3.141592;
fprintf('Pi is both %f and delicious. \n',A)
           % Outputs string to text in the command window

           % For full documentation on fprintf, see
           % http://www.mathworks.com/help/techdoc/ref/fprintf.html

B = [4 8 15 16 23 42];

newfile = fopen('MyTextFile.txt','w');
fprintf(newfile,'%f\n',B);
fclose(newfile);
           % Creates a .txt file named 'My_File.txt'; the 'w'
           % gives write permission for the file. The fprintf
           % command writes to the specified file in the format
           % '%f', creates a new line while writing with '\n',
           % and closes the file with fclose.

           % Make sure that the correct working directory is
           % chosen, as this is the folder in which your new file
           % will be created.

save MyMatFile.mat
           % Saves all current variables in the workspace to a
           % .mat file

MyDataVariables = who('-file','MyMatFile.mat')
           % Outputs all variables in the specified file to the
           % command window

load MyMatFile.mat
           % Loads all variables contained in the specified file

d = {'Time','Temperature';12 98;13 99;14 97};

xlswrite('climate.xls',d,'Temperatures','E1');
           % Creates an Excel file and writes the variable 'd' to
           % a worksheet called 'Temperatures'. The upper left
           % corner of the matrix begins on cell 'E1'.
```

**Ex. 55**

```
clear all
close all
clc

%%%
%Define data (y-values) by loading from Excel file

grad_enrollment = xlsread('Example6.xlsx','A2:A23');
unemployment_rate = xlsread('Example6.xlsx','B2:B25');

%%%
%Define x-values

years = 1985:2006;
years2 = 1985:2008;

%%%
% Create plot with two y-axes. The vector [my_axes,line1,line2]
creates
% user-defined labels for the three objects:
% 1) my_axes defines the axes, such that my_axes(1) calls on the
first set
% of axes, and my_axes(2) calls on the second set
% 2) line1 defines the first set of data specified by X1 and Y1, set
here
% as [years,grad_enrollment]
% 3) line2 defines the second set of data specified by X2 and Y2, set
here
% as [years2,unemployment_rate]

[my_axes,line1,line2] = plotyy(years,grad_enrollment, ...
    years2,unemployment_rate);

%%%
%Call up the first set of axes (my_axes(1)) and set axis properties

axes(my_axes(1))
axis([1983 2008 -40000 40000])
set(my_axes(1),'YTickLabel',-40000:20000:40000)
ylabel('Fluctuations in Grad Student Enrollment')

%%%
%Call up the second set of axes (my_axes(2)) and set axis properties
```

```
axes(my_axes(2))
axis([1983 2008 -2 2])
set(my_axes(2),'YColor','r','XTick',[1985 1990 1995 2000 2005], ...
    'YTick',-1.5:.5:1.5)
ylabel('Fluctuations in Unemployment Rate')

title('Unemployment Rate vs Grad Student Enrollment
(Engineering/Science)', ...
    'FontWeight','bold')

%%%
%Call up the specified data (line1 and line2) and set line properties

set(line1,'Marker','s','Color','b','LineWidth',2,'MarkerFaceColor','b
', ...
    'MarkerSize',5)
set(line2,'Marker','^','Color','r','LineWidth',2,'MarkerSize',7)

%%%
%Define a line at x = 0

years3 = 1983:2008;
zeroline = 0*years3;

hold on
plot(years3,zeroline,'LineWidth',2,'Color','k')

%%%
%Add text to the plot

text(1995,1.6,'Correlation coefficient \rho = 0.75583')




%%%%%%%%%%%
% Note (by Noel): at the tutorial on Thursday 09/22, the question was
raised regarding
% whether a figure could be modified using the Matlab figure GUI, then
% the code could be exported. To do this, edit the figure, then in the
% figure window choose File > Generate M-File. The code will be created in
% a new script in the Editor window.
```