

MATLAB


MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

- Math and computation
- Algorithm development
- Data acquisition
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar noninteractive language such as C or FORTRAN.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects. Today, MATLAB engines incorporate the LAPACK and BLAS libraries, embedding the state of the art in software for matrix computation.

Starting MATLAB

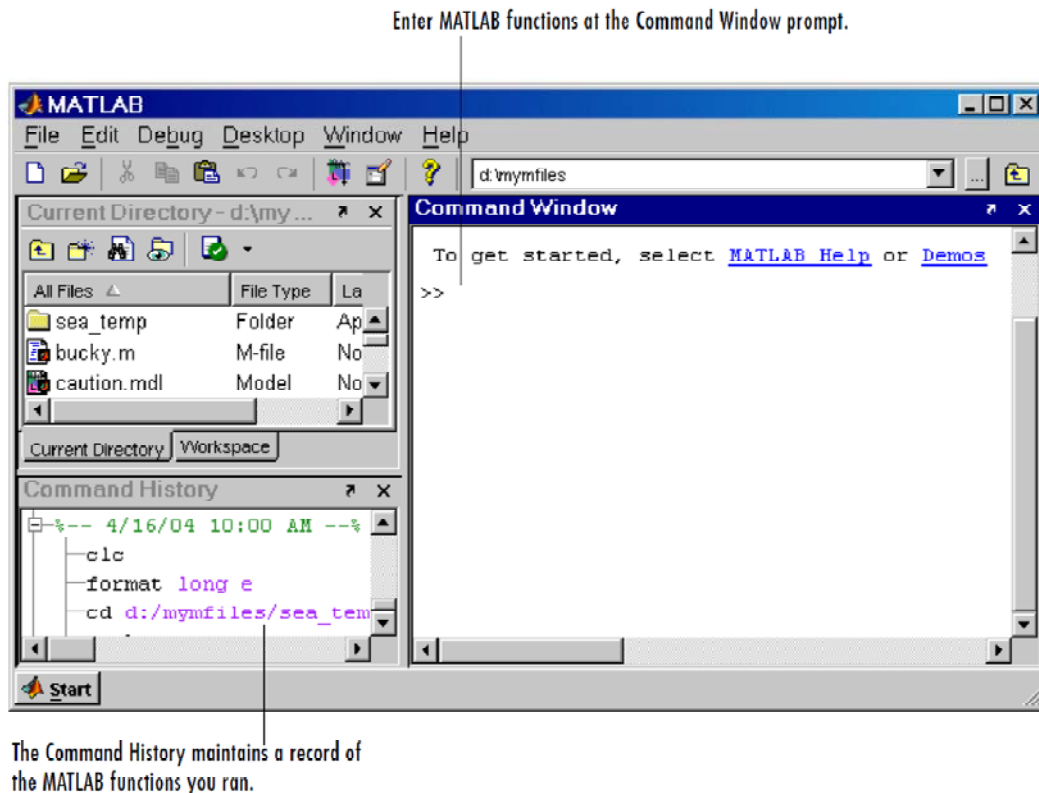
On Windows platforms, start MATLAB by double-clicking the MATLAB  shortcut icon on your Windows desktop.

Quitting MATLAB

To end your MATLAB session, select File -> Exit MATLAB in the desktop, or type quit in the Command Window.

MATLAB Desktop

When you start MATLAB, the MATLAB desktop appears, containing tools (graphical user interfaces) for managing files, variables, and applications associated with MATLAB.



Command Window	Use the Command Window to enter variables and run functions and M-files
Command History	Statements you enter in the Command Window are logged in the Command History. In the Command History, you can view previously run statements, and copy and execute selected statements
Current Directory Browser	MATLAB file operations use the current directory reference point. Any file you want to run must be in the current directory or on the search path
Workspace	The MATLAB workspace consists of the set of variables (named arrays) built up during a MATLAB session and stored in memory

Matrices and Arrays

In MATLAB, a matrix is a rectangular array of numbers. Special meaning is sometimes attached to 1-by-1 matrices, which are scalars, and to matrices with only one row or column, which are vectors. MATLAB has other ways of storing both numeric and nonnumeric data, but in the beginning, it is usually best to think of everything as a matrix. The operations in MATLAB are designed to be as natural as possible.

Entering Matrices

You can enter matrices into MATLAB in several different ways:

- Enter an explicit list of elements.
- Load matrices from external data files.
- Generate matrices using built-in functions.
- Create matrices with your own functions in M-files.

Start by entering Dürer's matrix = $\begin{bmatrix} 16 & 3 & 2 & 13 \\ 5 & 10 & 11 & 8 \\ 9 & 6 & 7 & 12 \\ 4 & 15 & 14 & 1 \end{bmatrix}$ as a list of its elements. You only have to follow a few basic conventions:

- Separate the elements of a row with blanks or commas.
- Use a semicolon (;) to indicate the end of each row.
- Surround the entire list of elements with square brackets, [].

To enter Dürer's matrix, simply type in the Command Window

```
A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
```

MATLAB displays the matrix you just entered.

```
A =  
16 3 2 13  
5 10 11 8  
9 6 7 12  
4 15 14 1
```

Once you have entered the matrix, it is automatically remembered in the MATLAB workspace. You can refer to it simply as A.

Subscripts

The element in row i and column j of A is denoted by A(i,j). For example, A(4,2) is the number in the fourth row and second column. For our magic square, A(4,2) is 15. So to compute the sum of the elements in the fourth column of A, type

```
A(1,4) + A(2,4) + A(3,4) + A(4,4)
```

This produces

```
ans =  
34
```

It is also possible to refer to the elements of a matrix with a single subscript, A(k). This is the usual way of referencing row and column vectors. But it can also apply to a fully two-dimensional matrix, in which case the array is regarded as one long

column vector formed from the columns of the original matrix. So, for our magic square, A(8) is another way of referring to the value 15 stored in A(4,2).

The Colon Operator

The colon (:) is one of the most important MATLAB operators. It occurs in several different forms.

Command	Result									
1:10	1	2	3	4	5	6	7	8	9	10
1:2:10	1	3	5	7	9					
100:-7:50	100	93	86	79	72	65	58	51		
A(2,:)	5	11	10	8						
A(1:2,1:2)	16	2								
	5	11								

Matrix and Array Functions

$$A = \begin{bmatrix} 16 & 2 & 3 & 13 \\ 5 & 11 & 10 & 8 \\ 9 & 7 & 6 & 12 \\ 4 & 14 & 15 & 1 \end{bmatrix}, \quad D = \begin{bmatrix} 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \end{bmatrix}, \quad B = [5 \ 6 \ 8 \ 9 \ 1 \ 3 \ 2 \ 0]$$

Command	Description
<pre>>> det(A) ans = -1.4495e-012 >> det(D) ans = 0</pre>	Matrix determinant
<pre>>> diag(A) ans = 16 11 6 1</pre>	Diagonal matrices and diagonals of matrix
<pre>>> inv(A)</pre>	Inverse of a square matrix
<pre>>> length(A) ans = 4 >> length(B) ans = 8</pre>	Return the length of a matrix
<pre>>> magic(3) ans = 8 1 6 3 5 7 4 9 2</pre>	Magic square

Command	Description
<pre>>> max(A) ans = 16 14 15 13 >> max(B) ans = 9</pre>	Maximum elements of a matrix
<pre>>> min(A) ans = 4 2 3 1 >> min(B) ans = 0</pre>	Minimum elements of a matrix
<pre>>> numel(A) ans = 16 >> numel(B) ans = 8</pre>	Number of elements in array or subscripted array
<pre>>> ones(2) ans = 1 1 1 1</pre>	Create a matrix of all 1s
<pre>>> size(A) ans = 4 4 >> size(B) ans = 1 8</pre>	Return the size of a matrix
<pre>>> sort(A) ans = 4 2 3 1 5 7 6 8 9 11 10 12 16 14 15 13 >> sort(B) ans = 0 1 2 3 5 6 8 9</pre>	Sort array elements in ascending or descending order
<pre>>> sum(A) ans = 34 34 34 34 >> sum(B) ans = 34</pre>	Sum of matrix elements
<pre>>> tril(A) ans = 16 0 0 0 5 11 0 0 9 7 6 0 4 14 15 1</pre>	Extract lower triangular part
<pre>>> triu(A) ans = 16 2 3 13 0 11 10 8 0 0 6 12 0 0 0 1</pre>	Extract upper triangular part

Command	Description
<pre>>> zeros(3) ans = 0 0 0 0 0 0 0 0 0</pre>	Create a matrix of all zeros
<pre>>> rand(2) ans = 0.8147 0.1270 0.9058 0.9134 >> rand(2,3) ans = 0.6324 0.2785 0.9575 0.0975 0.5469 0.9649 >> rand ans = 0.1576</pre>	Create a matrix of uniformly distributed random elements
<pre>>> randn(2,3) ans = 3.0349 -0.0631 -0.2050 0.7254 0.7147 -0.1241</pre>	Create a matrix of Normally distributed random elements
<pre>>> A' ans = 16 5 9 4 2 11 7 14 3 10 6 15 13 8 12 1 >> B' ans = 5 6 8 9 1 3 2 0</pre>	Matrix transpose
<pre>>> mean(B) ans = 4.2500</pre>	The average of the elements for matrix
<pre>>> A+D ans = 18 4 5 15 7 13 12 10 11 9 8 14 6 16 17 3</pre>	Adding two matrices
<pre>>> A*D ans = 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68 68</pre>	Multiplying two matrices
<pre>>> A+5 ans = 21 7 8 18 10 16 15 13 14 12 11 17 9 19 20 6</pre>	Adding matrix to number

Command	Description
<pre>>> A*3 ans = 48 6 9 39 15 33 30 24 27 21 18 36 12 42 45 3</pre>	Multiplying matrix by number
<pre>>> A^2 ans = 345 257 281 273 257 313 305 281 281 305 313 257 273 281 257 345</pre>	Power matrix by number
<pre>>> A.*D ans = 32 4 6 26 10 22 20 16 18 14 12 24 8 28 30 2</pre>	Element-by-element multiplication
<pre>>> A./D ans = 8.0000 1.0000 1.5000 6.5000 2.5000 5.5000 5.0000 4.0000 4.5000 3.5000 3.0000 6.0000 2.0000 7.0000 7.5000 0.5000</pre>	Element-by-element division
<pre>>> A.^D ans = 256 4 9 169 25 121 100 64 81 49 36 144 16 196 225 1 >> A.^2 ans = 256 4 9 169 25 121 100 64 81 49 36 144 16 196 225 1</pre>	Element-by-element power
<pre>>> sqrt(A) ans = 4.0000 1.4142 1.7321 3.6056 2.2361 3.3166 3.1623 2.8284 3.0000 2.6458 2.4495 3.4641 2.0000 3.7417 3.8730 1.0000</pre>	Element square root

Statistical Functions

Function	Description
corrcoef	Correlation coefficients
mean	Average or mean value of array
median	Median value of array
mode	Most frequent values in array
std	Standard deviation
var	Variance

Variables

MATLAB does not require any type declarations or dimension statements. When MATLAB encounters a new variable name, it automatically creates the variable and allocates the appropriate amount of storage. If the variable already exists, MATLAB changes its contents and, if necessary, allocates new storage. For example,

```
num_students = 25
```

creates a 1-by-1 matrix named `num_students` and stores the value 25 in its single element. Variable names consist of a letter, followed by any number of letters, digits, or underscores. MATLAB uses only the first 31 characters of a variable name. MATLAB is case sensitive; it distinguishes between uppercase and lowercase letters. A and a are *not* the same variable.

Numbers

MATLAB uses conventional decimal notation, with an optional decimal point and leading plus or minus sign, for numbers. *Scientific notation* uses the letter e to specify a power-of-ten scale factor. *Imaginary numbers* use either i or j as a suffix. Some examples of legal numbers are

3	-99	0.0001
9.6397238	1.60210e-20	6.02252e23
1i	-3.14159j	3e5i

All numbers are stored internally using the *long* format specified by the IEEE floating-point standard. Floating-point numbers have a finite *precision* of roughly 16 significant decimal digits and a finite *range* of roughly 10^{-308} to 10^{+308} .

Operators

Expressions use familiar arithmetic operators and precedence rules.

Operators	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
\	Left division (described in “Matrices and Linear Algebra” in the MATLAB documentation)
^	Power
'	Complex conjugate transpose
()	Specify evaluation order

Relational and logic operators

Operators	Description
==	Equal
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal
~=	Not equal
&	And
 	Or
~	Not

Functions

MATLAB provides a large number of standard elementary mathematical functions.

cos(x)	sin(x)	tan(x)	sec(x)	csc(x)	cot(x)
acos(x)	asin(x)	atan(x)	atan2(y,x)		
exp(x)	log(x)	[log(x) is ln(x)]	log10(x)	log2(x)	sqrt(x)
cosh(x)	sinh(x)	tanh(x)	sech(x)	csch(x)	coth(x)
acosh(x)	asinh(x)	atanh(x)			

Function	Description
abs(x)	the absolute value of a number (real or complex)
clc	clears the command window; useful for beautifying printed output
ceil(x)	the nearest integer to x looking toward +1
Clear	clears all assigned variables
close all	closes all figure windows
close 3	closes figure window 3
fix(x)	the nearest integer to x looking toward zero
mod(x,y)	the integer remainder of x/y
rem(x,y)	the integer remainder of x/y
round(x)	the nearest integer to x
sign(x)	the sign of x and returns 0 if x=0

For a list of the elementary mathematical functions, type

help elfun

For a list of more advanced mathematical and matrix functions, type

help specfun

help elmat

Several special functions provide values of useful constants.

Pi	3.14159265...
I	Imaginary unit, $\sqrt{-1}$
J	Same as i
Eps	Floating-point relative precision, $\varepsilon = 2^{-52}$
realmin	Smallest floating-point number, 2^{-1022}
realmax	Largest floating-point number, $(2 - \varepsilon)^{1023}$
Inf	Infinity
NaN	Not-a-number

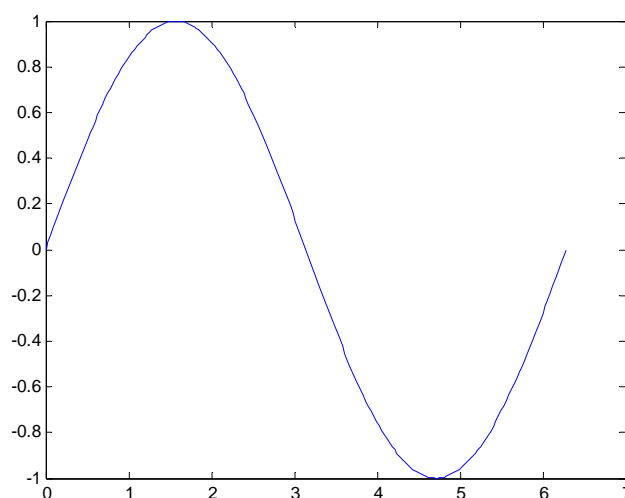
Basic Plotting Functions

Creating a Plot

The plot function has different forms, depending on the input arguments. If y is a vector, `plot(y)` produces a piecewise linear graph of the elements of y versus the index of the elements of y . If you specify two vectors as arguments, `plot(x,y)` produces a graph of y versus x .

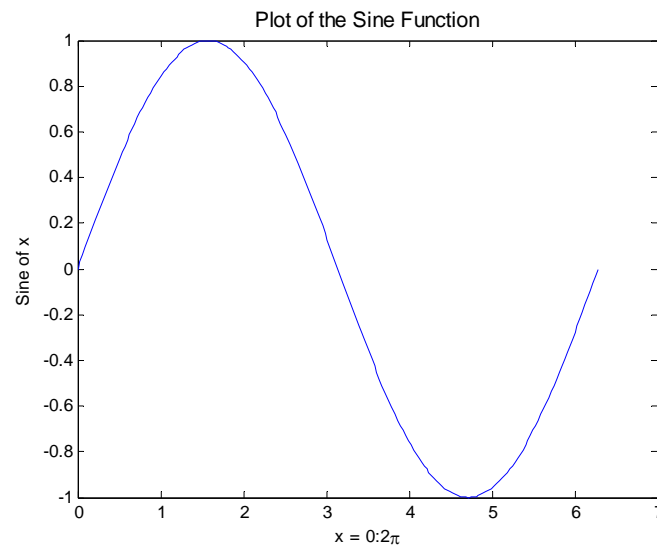
For example, these statements use the colon operator to create a vector of x values ranging from 0 to 2π , compute the sine of these values, and plot the result.

```
x = 0:pi/100:2*pi;  
y = sin(x);  
plot(x,y)
```



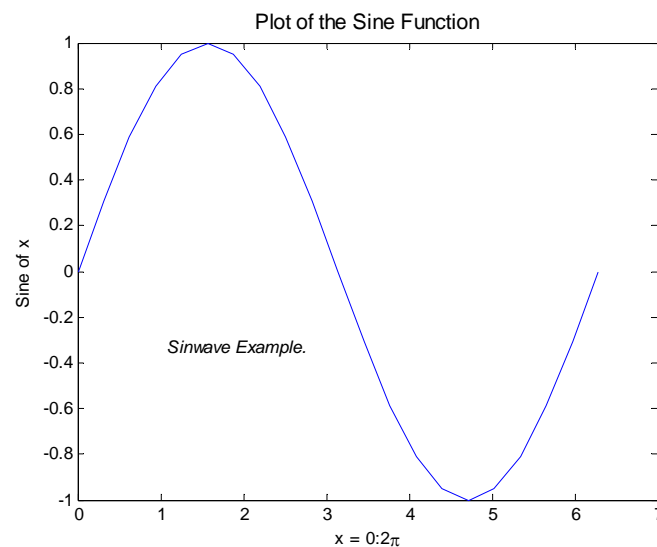
Now label the axes and add a title. The characters `\pi` create the symbol π .

```
xlabel('x = 0:2\pi')  
ylabel('Sine of x')  
title('Plot of the Sine Function','FontSize',12)
```



It can produce mathematical symbols using LaTeX notation in the text string, as the following example illustrates.

```
text(1,-1/3,'\it Sinwave Example.')
```



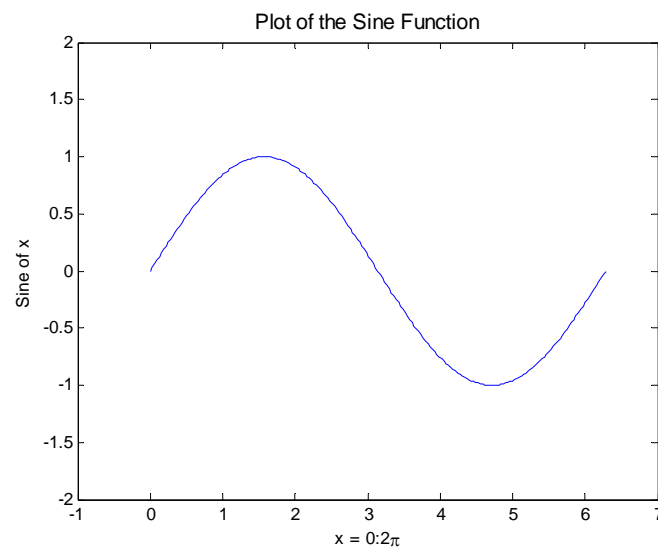
Setting Axis Limits

By default, MATLAB finds the maxima and minima of the data and chooses the axis limits to span this range. The `axis` command enables you to specify your own limits:

```
axis([xmin xmax ymin ymax])
```

for example :

```
axis([-1 7 -2 2])
```

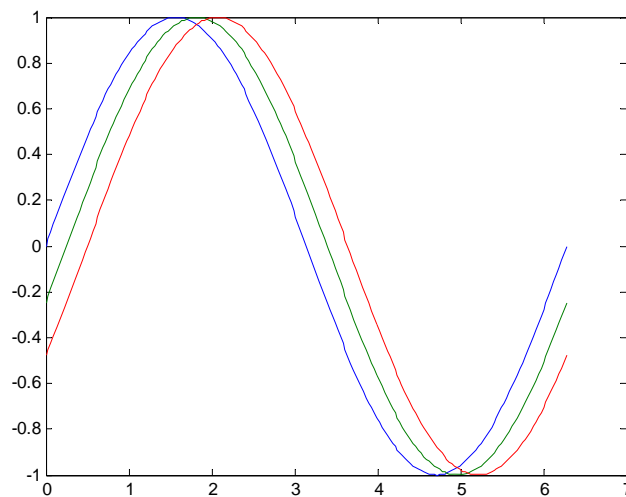


Multiple Data Sets in One Graph

Multiple x-y pair arguments create multiple graphs with a single call to plot. MATLAB automatically cycles through a predefined (but user settable) list of colors to allow discrimination among sets of data.

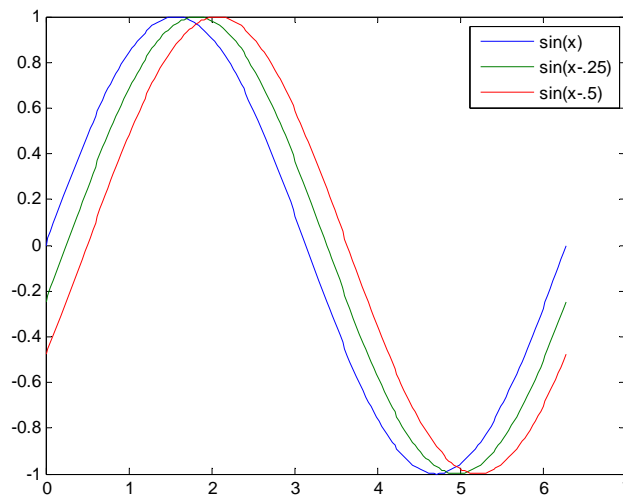
For example, these statements plot three related functions of x, with each curve in a separate distinguishing color.

```
x = 0:pi/100:2*pi;  
y = sin(x);  
y2 = sin(x-.25);  
y3 = sin(x-.5);  
plot(x,y,x,y2,x,y3)
```



The legend command provides an easy way to identify the individual plots.

```
legend('sin(x)', 'sin(x-.25)', 'sin(x-.5)')
```



Specifying Line Styles and Colors

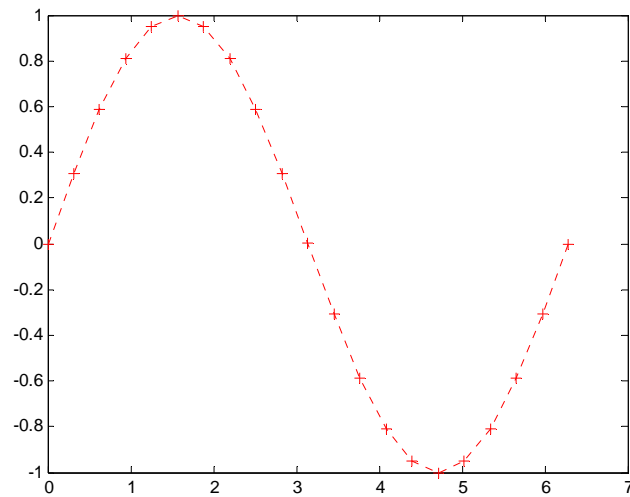
It is possible to specify color, line styles, and markers (such as plus signs or circles) when you plot your data using the plot command.

```
plot(x,y,'color_style_marker')
```

color_style_marker is a string containing from one to four characters (enclosed in single quotation marks) constructed from a color, a line style, and a marker type:

- Color strings are 'c', 'm', 'y', 'r', 'g', 'b', 'w', and 'k'. These correspond to cyan, magenta, yellow, red, green, blue, white, and black.
- Line style strings are '-' for solid, '--' for dashed, ':' for dotted, '-.' for dash-dot. Omit the line style for no line.
- The marker types are '+', 'o', '*', and 'x', and the filled marker types are 's' for square, 'd' for diamond, '^' for up triangle, 'v' for down triangle, '>' for right triangle, '<' for left triangle, 'p' for pentagram, 'h' for hexagram, and none for no marker.

```
plot(x,y,'r:+')
```



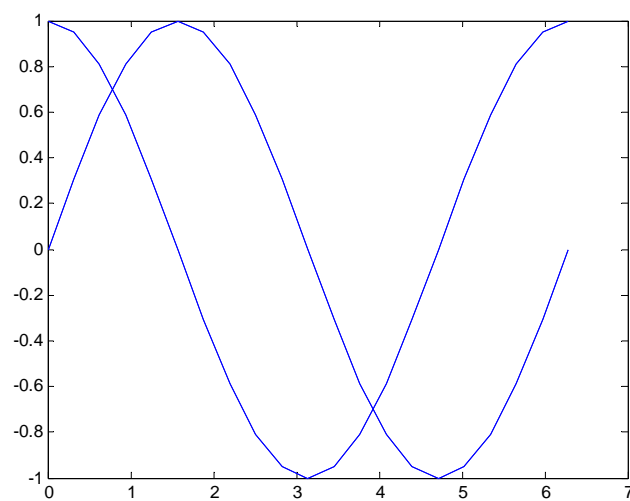
Adding Plots to an Existing Graph

The hold command enables you to add plots to an existing graph. When you type

`hold on`

For example

```
x = 0:pi/10:2*pi;
y1 = sin(x);
y2 = cos(x);
plot(x,y1)
hold on
plot(x,y2)
```



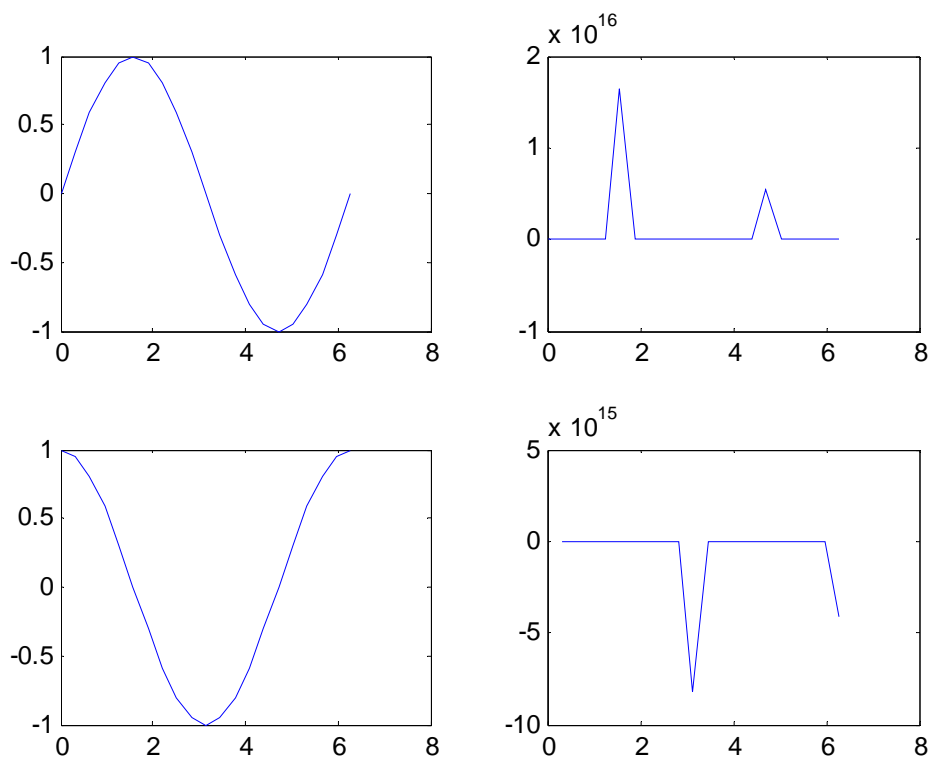
Multiple Plots in One Figure

The subplot command enables you to display multiple plots in the same window or print them on the same piece of paper. Typing

subplot(m,n,p)

partitions the figure window into an m-by-n matrix of small subplots and selects the pth subplot for the current plot. The plots are numbered along first the top row of the figure window, then the second row, and so on. For example, these statements

```
x = 0:pi/10:2*pi;  
y1 = sin(x);y3=tan(x)  
y2 = cos(x);y4=cot(x)  
subplot(2,2,1); plot(x,y1)  
subplot(2,2,2); plot(x,y3)  
subplot(2,2,3); plot(x,y2)  
subplot(2,2,4); plot(x,y4)
```



Complex Numbers

Matlab works as easily with complex numbers as with real ones. The variable i is the usual imaginary number $i = \sqrt{-1}$,

$$A = 3 + 4i$$

$$B = -1 - 3i$$

Command	Description
<pre>>> A=3+4i A = 3.0000 + 4.0000i >> A = complex(3,4) A = 3.0000 + 4.0000i >> B = complex(-1,-3) B = -1.0000 - 3.0000i</pre>	Construct complex data from real and imaginary components
<pre>>> abs(A) ans = 5 >> abs(B) ans = 3.1623</pre>	Absolute value and complex magnitude
<pre>>> angle(A) ans = 0.9273 >> angle(A)*180/pi ans = 53.1301 >> angle(B)*180/pi ans = -108.4349</pre>	Phase angle
<pre>>> conj(A) ans = 3.0000 - 4.0000i >> conj(B) ans = -1.0000 + 3.0000i</pre>	Complex conjugate
<pre>>> real(A) ans = 3 >> real(B) ans = -1</pre>	Real part of complex number
<pre>>> imag(A) ans = 4 >> imag(B) ans = -3</pre>	Imaginary part of complex number

Command	Description
<pre>>> A+B ans = 2.0000 + 1.0000i >> A-B ans = 4.0000 + 7.0000i >> A*B ans = 9.0000 -13.0000i >> A/B ans = -1.5000 + 0.5000i >> A^2 ans = -7.0000 +24.0000i >> sin(A) ans = 3.8537 -27.0168i</pre>	Complex Arithmetic

M-Files

You can create your own program using *M-files*, which are text files containing MATLAB code. Use the MATLAB Editor or another text editor to create a file containing the same statements you would type at the MATLAB command line. Save the file under a name that ends in .m.

Flow Control

If-statement

The **if** statement evaluates a logical expression and executes a group of statements when the expression is *true*. The optional **elseif** and **else** keywords provide for the execution of alternate groups of statements. An **end** keyword, which matches the if, terminates the last group of statements.

For example the following program check the value of A if it even or odd:

Program	Run results
<pre>A=input('input the value of A ='); if mod(A,2)==0 disp('A is even') else disp('A is odd') end</pre>	<pre>input the value of A = 4 A is even input the value of A = 5 A is odd</pre>
<p>Note: the function (input) is used to assign a value to the variable A the function (disp) is used display the text.</p>	

Switch and case

The **switch** statement executes groups of statements based on the value of a variable or expression. The keywords **case** and **otherwise** delineate the groups. Only the first matching case is executed. There must always be an end to match the switch.

For example the following program calculates the value of Y according to the following conditions:

$$y = \begin{cases} 20 & \text{if } A = 1 \text{ or } 2 \text{ or } 3 \\ 50 & \text{if } A = 4 \\ 70 & \text{if } A = 5 \end{cases}$$

Program	Run results
<pre>A=input('input the value of A ='); switch A case {1,2,3} y=20 case 4 y=50 case 5 y=70 end</pre>	<pre>input the value of A =2 y = 20 input the value of A =5 y = 70</pre>

For

The **for** loop repeats a group of statements a fixed, predetermined number of times. A matching end delineates the statements.

For example the following program calculates the summation of the even number between A and B:

Program	Run results
<pre>A=input('input the value of A ='); B=input('input the value of B ='); sum=0; for i=A:B if mod(i,2)==0 sum=sum+i; end end sum</pre>	<pre>input the value of A =5 input the value of B =20 sum = 104</pre>

while

The **while** loop repeats a group of statements an indefinite number of times under control of a logical condition. A matching end delineates the statements.

For example the following program calculates the summation of the number between A and B:

Program	Run results
<pre>A=input('input the value of A ='); B=input('input the value of B ='); sum=0; while A<=B sum=sum+A; A=A+1; end sum</pre>	<pre>input the value of A =5 input the value of B =20 sum = 200</pre>

Continue

The **continue** statement passes control to the next iteration of the for loop or while loop in which it appears, skipping any remaining statements in the body of the loop. In nested loops, continue passes control to the next iteration of the for loop or while loop enclosing it.

For example the following program calculates the summation of the even number between A and B:

Program	Run results
<pre>A=input('input the value of A ='); B=input('input the value of B ='); sum=0; for i=A:B if mod(i,2)==1 continue end sum=sum+i; end sum</pre>	<pre>input the value of A =5 input the value of B =20 sum = 104</pre>

Break

The **break** statement lets you exit early from a for loop or while loop. In nested loops, **break** exits from the innermost loop only.

For example the following program calculates the summation of the number between A and B:

Program	Run results
<pre>A=input('input the value of A ='); B=input('input the value of B ='); sum=0; for i=A:10000 sum=sum+i; if i==B break end end sum</pre>	<pre>input the value of A =5 input the value of B =20 sum = 200</pre>

Scripts

Scripts M-file does not accept input arguments or return output arguments. They operate on data in the workspace.

When you invoke a *script*, MATLAB simply executes the commands found in the file. Scripts can operate on existing data in the workspace, or they can create new data on which to operate. Although scripts do not return output arguments, any variables that they create remain in the workspace, to be used in subsequent computations. In addition, scripts can produce graphical output using functions like plot.

For example the following program calculates the summation of the number between 1 and 10, the program saved as (sum1to10.m):

Program (sum1to10.m)	Run results
<pre>sum=0; for i=1:10 sum=sum+i; end sum</pre>	<pre>>> sum1to10 sum = 55</pre>

Functions

Functions are M-files that can accept input arguments and return output arguments. The names of the M-file and of the function should be the same. Functions operate on variables within their own workspace, separate from the workspace you access at the MATLAB command prompt.

For example the following program calculates the summation of the number between A and B, the program saved as (sumAtoB.m):

Program (sumAtoB.m)	Run results
<pre>function [sum]=sumAtoB(A,B) sum=0; for i=A:B sum=sum+i; end</pre>	<pre>>> sumAtoB(5,20) ans = 200</pre>

Strings

A string is a set of characters, like this
s='This is a string'

The string is actually a vector whose components are the numeric codes for the characters (the first 127 codes are ASCII). The actual character displayed depends on the character set encoding for a given font. The length of S is the number of characters. A quote within the string is indicated by two quotes.

Command	Description
>> ['this','is','string'] ans = thisisstring	concatenates character arrays in to new character array
>> strcat('this','is','string') ans = thisisstring	Concatenates character arrays or cell arrays of strings in to new character array
>> char(65) ans = A >> char([65 66 67 68]) ans = ABCD	convert an array that contains positive integers representing character codes into a MATLAB character array
>> double('A') ans = 65 >> double('ABCD') ans = 65 66 67 68	converts the character array to a numeric matrix containing floating-point representations of the ASCII codes for each character
>> char('this','is','string') ans = this is string	vertically concatenates character arrays
>> lower('ABCdef') ans = abcdef	Make all letters lowercase.
>> upper('ABCdef') ans = ABCDEF	Make all letters uppercase.
>> sort('cdbafe') ans = abcdef >> sort('cdbafe','descend') ans = fedcba	Sort elements in ascending or descending order.
>> '5+2' ans = 5+2 >> eval('5+2') ans = 7	Execute a string with MATLAB expression.

Data Type Conversion

The following table gives some function to convert the data from form to another:

Function	Description
<pre>>> dec2base(10, 2) ans = 1010 >> dec2base(10, 2,6) ans = 001010</pre>	Convert decimal to base N number in string
<pre>>> base2dec('1010', 2) ans = 10 >> base2dec('1010', 8) ans = 520</pre>	Convert base N number string to decimal number
<pre>>> dec2bin(9) ans = 1001 >> dec2bin(9,8) ans = 00001001</pre>	Convert decimal to binary number in string
<pre>>> bin2dec('010111') ans = 23</pre>	Convert binary number string to decimal number
<pre>>> dec2hex(10) ans = A >> dec2hex(105) ans = 69 >> dec2hex(105,4) ans = 0069</pre>	Convert decimal to hexadecimal number in string
<pre>>> hex2dec('3ff') ans = 1023 >> hex2dec('A') ans = 10</pre>	Convert hexadecimal number string to decimal number
<pre>>> dec2binvec(10) ans = 0 1 0 1 >> dec2binvec(10,6) ans = 0 1 0 1 0 0</pre>	Convert digital input and output decimal value to binary vector
<pre>>> binvec2dec([1 0 1 0]) ans = 5</pre>	Convert digital input and output binary vector to decimal value
<pre>>> str2num('500') ans = 500 >> str2num('500e-3') ans = 0.5000</pre>	Convert string to number

Function	Description
<pre>>> num2str(pi) ans = 3.1416 >> num2str(0.5000) ans = 0.5</pre>	Convert number to string

Bitwise Operations

The Bitwise Operator block performs the bitwise operation that you specify on one or more operands. Unlike logic operations of the Logical Operator block, bitwise operations treat the operands as a vector of bits rather than a single value.

$$A=[1\ 0\ 1\ 0] \quad B=[1\ 0\ 0\ 0]$$

Command	Description
<pre>>> and(A,B) ans = 1 0 0 0</pre>	AND operation
<pre>>> or(A,B) ans = 1 0 1 1</pre>	OR operation
<pre>>> xor(A,B) ans = 0 0 1 1</pre>	XOR operation
<pre>>> not(A) ans = 0 1 0 0</pre>	NOT operation
<pre>>> not(and(A,B)) ans = 0 1 1 1</pre>	NAND operation
<pre>>> not(or(A,B)) ans = 0 1 0 0</pre>	NOR operation
<pre>>> bitand(A,B) ans = 1 0 0 0</pre>	Bit-wise AND
<pre>>> bitor(A,B) ans = 1 0 1 0</pre>	Bit-wise OR
<pre>>> bitxor(A,B) ans = 0 0 1 0</pre>	Bit-wise XOR
A&B	AND operation
A B	OR operation
A~=B	XOR operation
~A	NOT operation
A==B	XNOR operation

Solving Simultaneous Linear Equations

Matrix division is especially useful in solving simultaneous linear equations. A system of linear equations is a set of linear equations that you usually want to solve at the same time; that is, simultaneously. A basic principle for exact answers in solving simultaneous linear equations requires that there be as many equations as there are unknowns.

Examples:

Linear Equations	Program	Run results
$2x+y=13$ $x-3y=-18$	<pre>A=[2 1 1 -3]; B=[13 -18]; xy=A\B</pre>	<pre>xy = 3 7</pre>
$6x-2y+z=20$ $-2x+6y=0$ $-x-12y+3z=12$	<pre>A=[6 -2 1 -2 6 0 -1 -12 3]; B=[20 0 12]; xyz=A\B</pre>	<pre>xyz = 2.2857 0.7619 7.8095</pre>

Polynomial roots

The `root(x)` returns a column vector whose elements are the roots of the polynomial `x`.

Examples:

Polynomial	Program	Run results
$2x-4=0$	<pre>x=[2 -4]; p=roots(x)</pre>	<pre>p = 2</pre>
$x^2-2x+3=0$	<pre>x=[1 -2 3]; p=roots(x)</pre>	<pre>p = 1.0000 + 1.4142i 1.0000 - 1.4142i</pre>
$x^2-4=0$	<pre>x=[1 0 -4]; p=roots(x)</pre>	<pre>p = 2.0000 -2.0000</pre>
$x^3-6x^2-72x-27=0$	<pre>x=[1 -6 -72 -27]; p=roots(x)</pre>	<pre>p = 12.1229 -5.7345 -0.3884</pre>

Applications Examples

Transistor Analysis

- 1) Write a MATLAB function M-file to calculate the I_C , I_B , I_E , V_{CE} , and the operation region of the circuit shown in Fig(1):

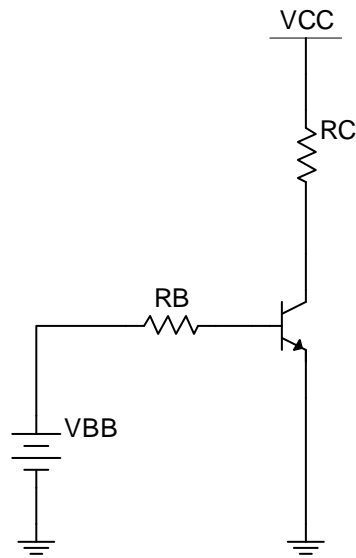


Fig (1)

Solution:

$$I_B = \frac{V_{BB} - V_{BE}}{R_B} \quad I_{Csat} = \frac{V_{CC} - V_{CEsat}}{R_C}$$

if $I_B \leq 0$ then $I_B = 0$, $I_C = 0$, $I_E = 0$, $V_{CE} = V_{CC}$ cutoff region

*if $\beta I_B < I_{Csat}$ then $I_B = I_B$, $I_C = \beta I_B$, $I_E = (\beta + 1)I_B$,
 $V_{CE} = V_{CC} - I_C R_C$ Active region*

*if $\beta I_B \geq I_{Csat}$ then $I_B = I_B$, $I_C = I_{Csat}$, $I_E = I_{Csat}$,
 $V_{CE} = V_{CEsat}$ Saturation region*

Program (trans.m)

```
function [IB,IC,IE,VCE]=trans(Vcc,VBB,RC,RB,VBE,B,VCEsat)
IB=(VBB-VBE)/RB;
ICsat=(Vcc-VCEsat)/RC;
if IB <= 0
    IB=0
    IC=0
    IE=0
    VCE=Vcc
    disp('cut off region')
elseif (B*IB)>= ICsat
    IB
    IC=ICsat
    IE=ICsat
    VCE=VCEsat
    disp('saturation region')
else
    IB
    IC=B*IB
    IE=(B+1)*IB
    VCE=Vcc-(IC*RC)
    disp('Active region')
end
```

Run results

```
>> trans(20,5,1e3,10e3,0.7,100,0)
IB =
    4.3000e-004
IC =
    0.0200
IE =
    0.0200
VCE =
     0
saturation region
```

```
>> trans(20,2,1e3,10e3,0.7,100,0)
IB =
    1.3000e-004
IC =
    0.0130
IE =
    0.0131
VCE =
    7.0000
Active region
```

```
>> trans(20,0.5,1e3,10e3,0.7,100,0)
IB =
     0
IC =
     0
IE =
     0
VCE =
    20
cut off region
```

Output Characteristics

- 2) Write a MATLAB commands to plot the Output Characteristics of the circuit shown in Fig (1).

Solution:

The trans.m function written in previous example will be implemented in this program to plot the Output Characteristics when

$$V_{BE} = 0.7\text{V}, \quad R_C = 1\text{K}\Omega, \quad R_B = 500\text{K}\Omega, \quad V_{CEsat} = 0.2\text{V}, \quad \beta = 100$$

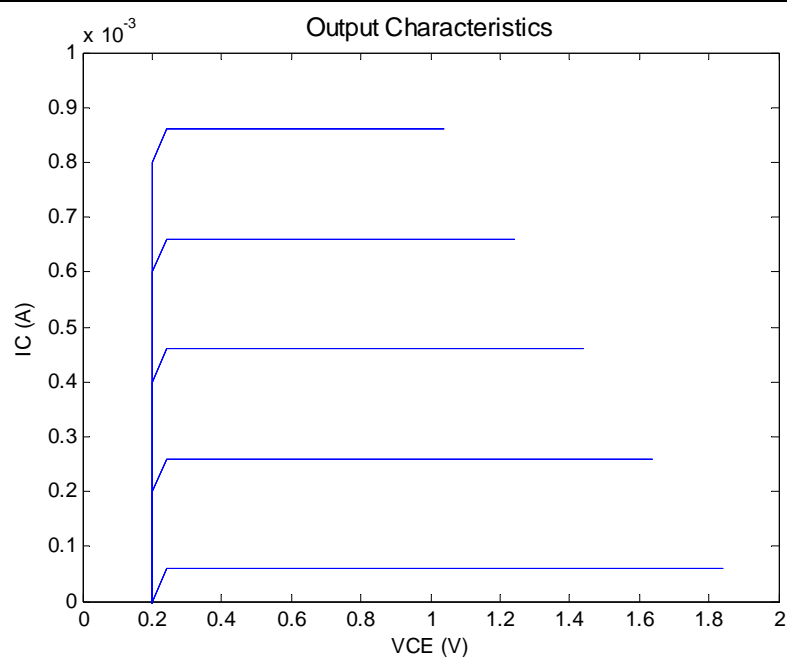
V_{BB} is change from 1 to 5 step 1

V_{CC} is change from 0 to 1.9 step 0.1

Program

```
clc
clear
RB=500e3;VCEsat=0.2;B=100;VBE=0.7;RC=1e3;
for VBB=1:5
    Vcc=0;
    for i=1:20
        [IB,IC(i),IE,VCE(i)]=trans(Vcc,VBB,RC,RB,VBE,B,VCEsat);
        Vcc=Vcc+0.1
    end
    plot(VCE,IC)
    hold on
    xlabel('VCE (V)')
    ylabel('IC (A)')
    title('Output Characteristics','FontSize',12)
    axis([0 2 0 10e-4])
end
```

Run results

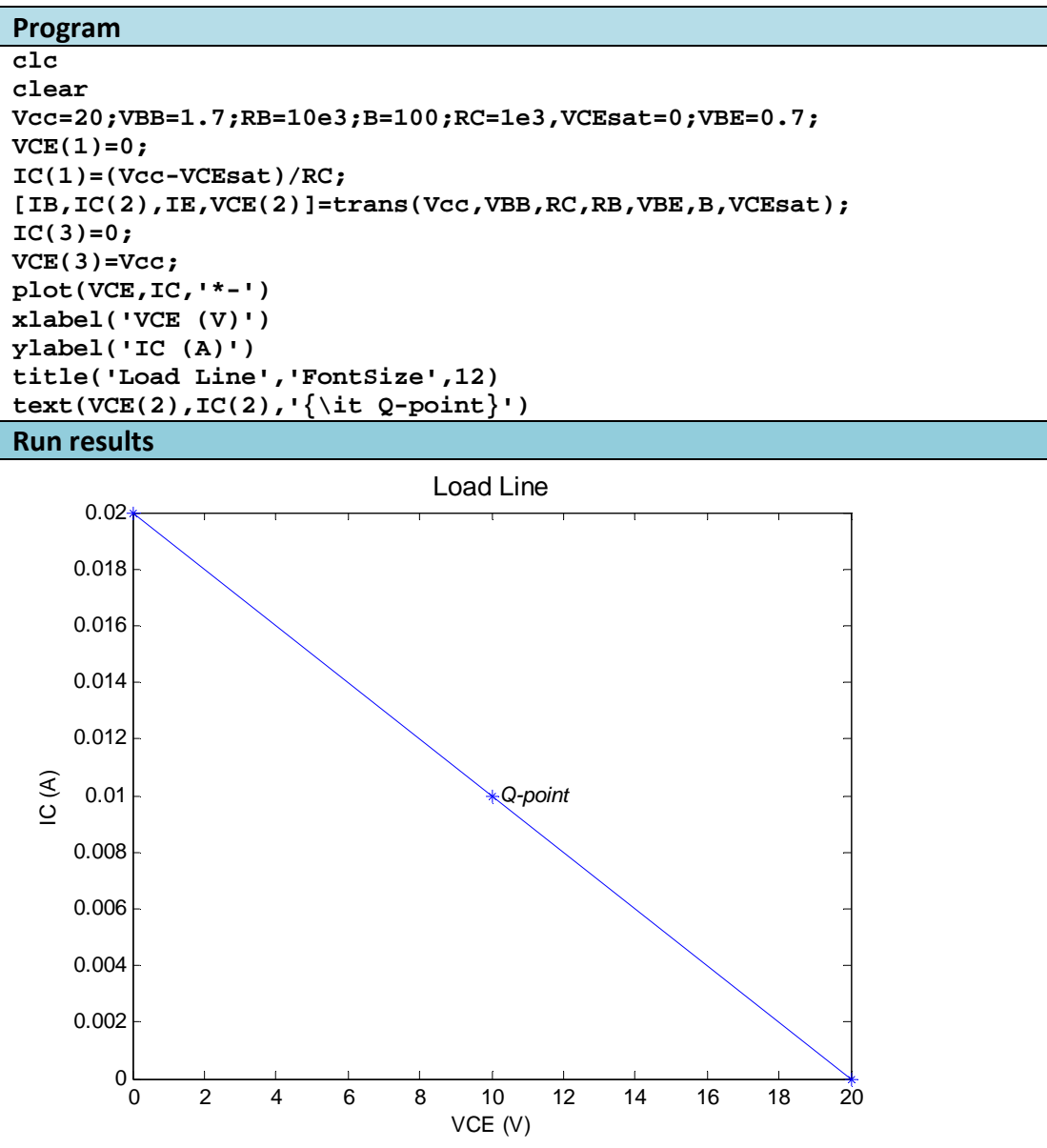


Load line and Q-point

- 3) Write a MATLAB commands to plot the Load line and of the circuit shown in Fig (1) when $V_{CC} = 20\text{V}$, $V_{BB} = 1.7\text{V}$, $V_{BE} = 0.7\text{V}$, $V_{CEsat} = 0\text{V}$, $R_B = 10\text{K}\Omega$, $R_C = 1\text{K}\Omega$, $\beta = 100$.

Solution:

The trans.m function written in previous example will be implemented in this program to plot the load line.



Diode analysis

4) Write a MATLAB commands to plot the Diode Characteristics .

Solution:

$$I_D = I_s \left(e^{\frac{KV_D}{T}} - 1 \right) \quad \text{where } K = \frac{11600}{2} \text{ for Si, } T = 300, I_s = 10 \mu A$$

V_D change from -0.5 to 0.75

Program

```
clc
clear
Is=10e-6;K=11600/2;T=300;V=-0.5;
for i=1:120
    Vd(i)=V;
    Id(i)=Is*(exp(K*Vd(i)/T)-1);
    V=V+0.01;
end
plot(Vd,Id)
xlabel('Vd (V)')
ylabel('Id (A)')
title('Diode Characteristics','FontSize',12)
```

Run results

