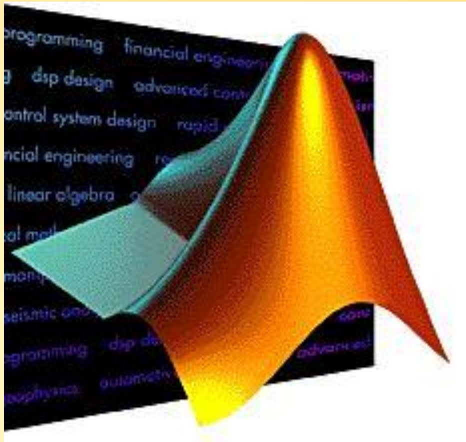
Maetode Numeris & Pemrograman Komputer

01 MATLAB Fundamentals



THE MATLAB ENVIRONMENT

- MATLAB is a computer program that provides the user with a convenient environment for performing many types of calculations.
- In particular, it provides a very nice tool to implement numerical methods.
- The most common way to operate MATLAB is by entering commands one at a time in the command window.

THE MATLAB ENVIRONMENT

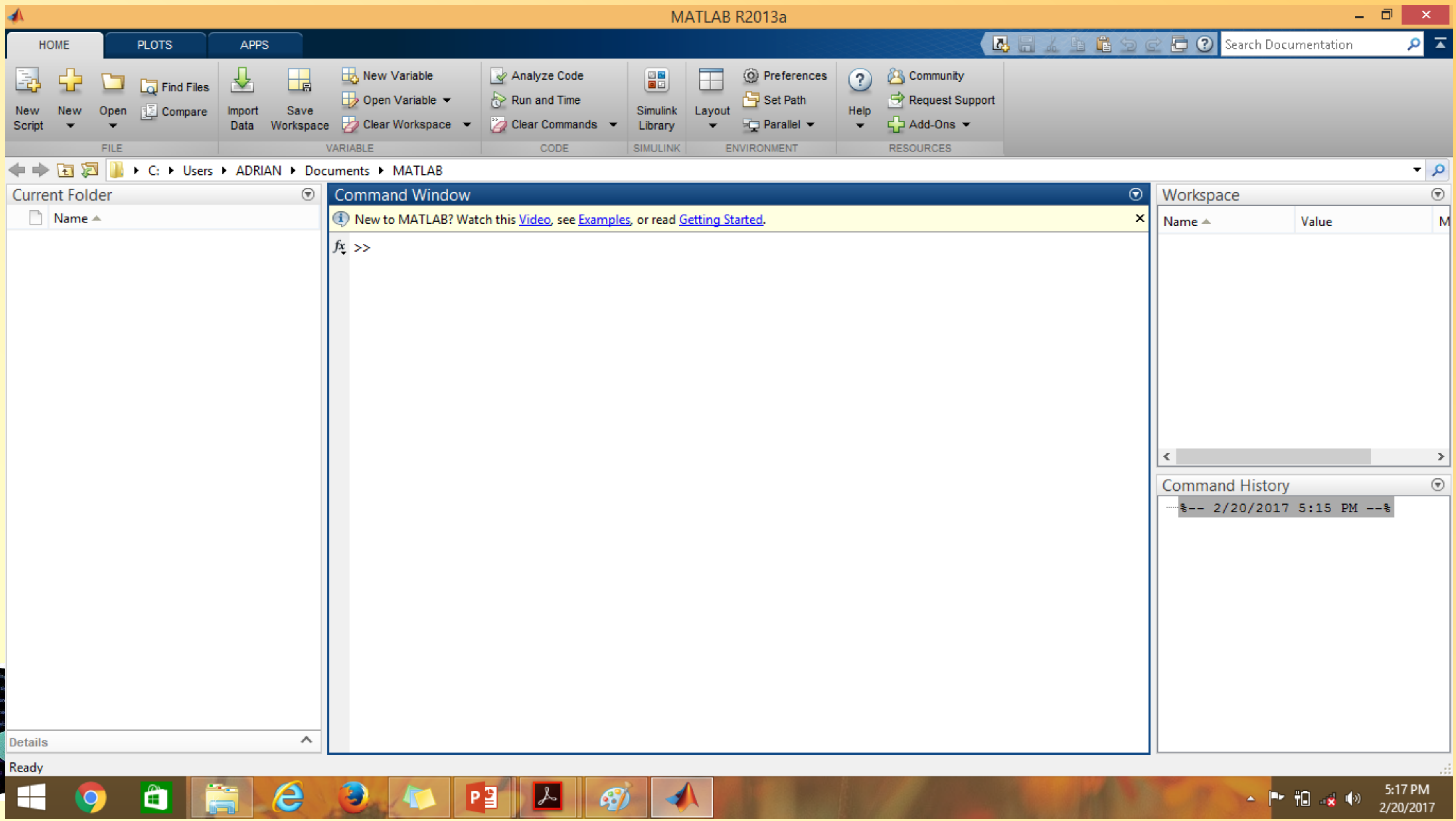
- In this first, we use this interactive or *calculator mode* to introduce you to common operations such as performing calculations and creating plots.
- In next, we show how such commands can be used to create MATLAB programs.

THE MATLAB ENVIRONMENT

MATLAB uses three primary windows:

- Command window. Used to enter commands and data.
- Graphics window. Used to display plots and graphs.
- Edit window. Used to create and edit M-files.

- After starting MATLAB, the command window will open with the command prompt being displayed

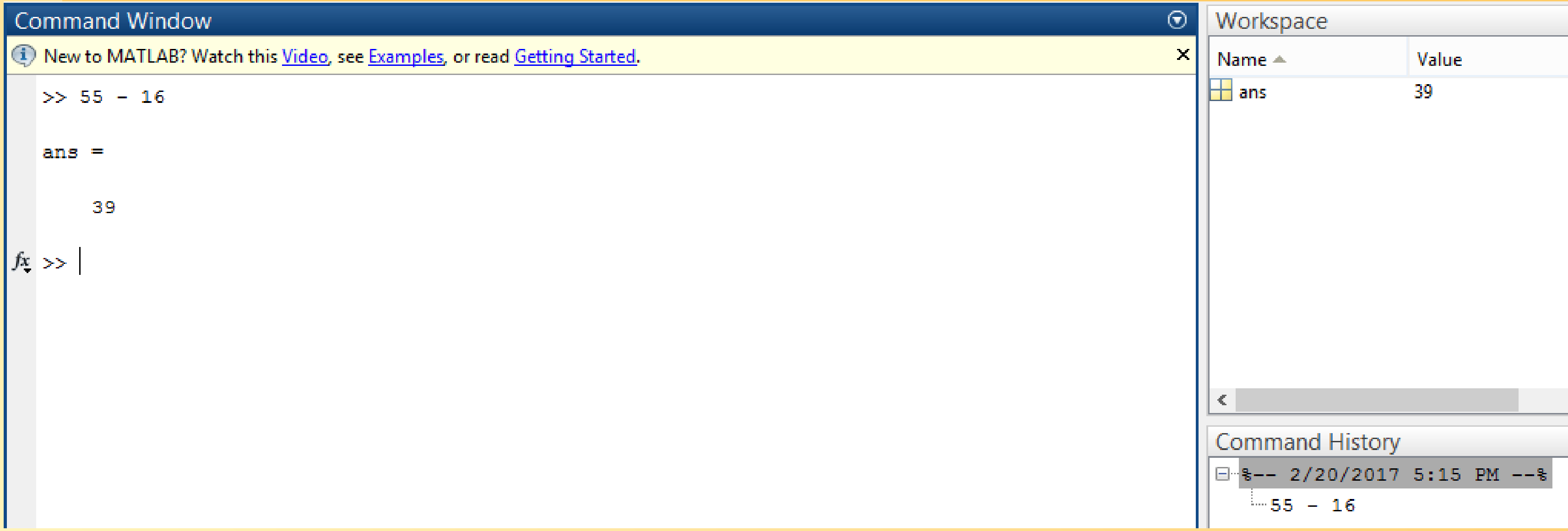


The calculator mode

- The calculator mode of MATLAB operates in a sequential fashion as you type in commands line by line.
- For each command, you get a result. Thus, you can think of it as operating like a very fancy calculator.
- For example, if you type in

```
>> 55 - 16
```

- MATLAB will display the result



- Notice that MATLAB has automatically assigned the answer to a variable, ans. Thus, you could now use ans in a subsequent calculation:

Command Window

New to MATLAB? Watch this Video, see Examples, or read Getting Started.

```
>> 55 - 16

ans =

    39

>> ans + 11

ans =

    50

>>
```

ASSIGNMENT

- The assignment of values to scalar variables is similar to other computer languages.
- Try typing

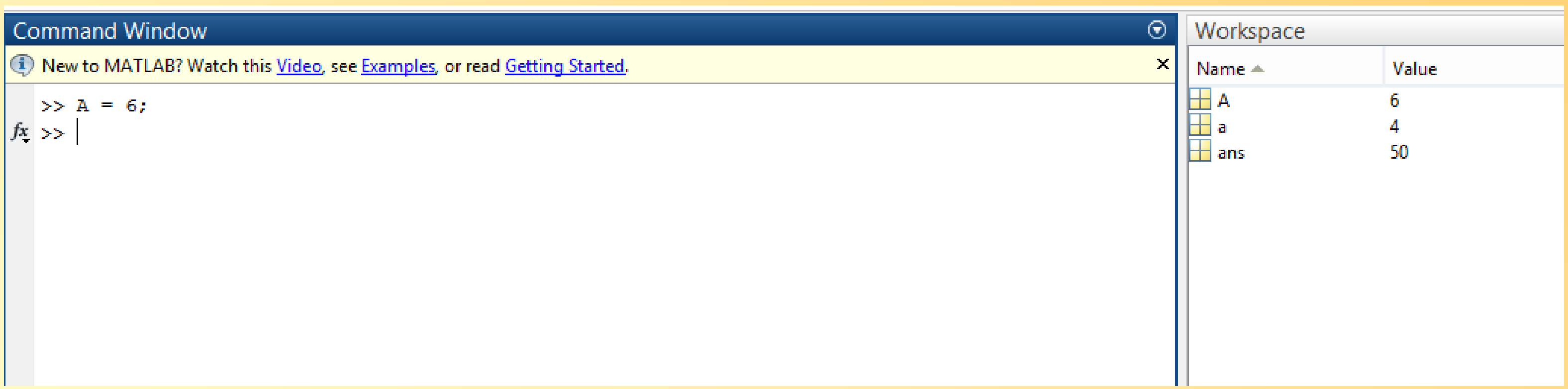
```
>> a = 4

a =

     4

>>
```

- It can be suppressed by terminating the command line with the semicolon (;) character. Try typing



You can type several commands on the same line by separating them with commas or semicolons. If you separate them with commas, they will be displayed, and if you use the semicolon, they will not. For example,

```
>> a = 4,A = 6;x = 1;

a =
      4
```

MATLAB treats names in a case-sensitive manner—that is, the variable `a` is not the same as `A`. To illustrate this, enter

```
>> a
```

and then enter

```
>> A
```

See how their values are distinct. They are distinct names.

We can assign complex values to variables, since MATLAB handles complex arithmetic automatically. The unit imaginary number $\sqrt{-1}$ is preassigned to the variable `i`. Consequently, a complex value can be assigned simply as in

```
>> x = 2+i*4

x =
   2.0000 + 4.0000i
```

It should be noted that MATLAB allows the symbol `j` to be used to represent the unit imaginary number for input. However, it always uses an `i` for display. For example,

```
>> x = 2+j*4

x =
   2.0000 + 4.0000i
```

There are several predefined variables, for example, `pi`.

```
>> pi

ans =
    3.1416
```

Notice how MATLAB displays four decimal places. If you desire additional precision, enter the following:

```
>> format long
```

Now when `pi` is entered the result is displayed to 15 significant figures:

```
>> pi

ans =
   3.14159265358979
```

To return to the four decimal version, type

```
>> format short
```

type	Result	Example
`short`	Scaled fixed-point format with 5 digits	`3.1416`
`long`	Scaled fixed-point format with 15 digits for double and 7 digits for single	`3.14159265358979`
`short e`	Floating-point format with 5 digits	`3.1416e+000`
`long e`	Floating-point format with 15 digits for double and 7 digits for single	`3.141592653589793e+000`
`short g`	Best of fixed- or floating-point format with 5 digits	`3.1416`
`long g`	Best of fixed- or floating-point format with 15 digits for double and 7 digits for single	`3.14159265358979`
`short eng`	Engineering format with at least 5 digits and a power that is a multiple of 3	`3.1416e+000`
`long eng`	Engineering format with exactly 16 significant digits and a power that is a multiple of 3	`3.14159265358979e+000`
`bank`	Fixed dollars and cents	`3.14`

Arrays, Vectors and Matrices

- An array is a collection of values that are represented by a single variable name.
- One dimensional arrays are called *vectors* and two-dimensional arrays are called *matrices*.

```
>> a = [1 2 3 4 5]

a =
      1      2      3      4      5
```

Note that this assignment overrides the previous assignment of `a = 4`.

In practice, row vectors are rarely used to solve mathematical problems. When we speak of vectors, we usually refer to column vectors, which are more commonly used. A column vector can be entered in several ways. Try them.

```
>> b = [2;4;6;8;10]
```

or

```
>> b = [2
4
6
8
10]
```

or, by transposing a row vector with the ' operator,

```
>> b = [2 4 6 8 10]'
```

The result in all three cases will be

```
b =
      2
      4
      6
      8
     10
```

A matrix of values can be assigned as follows:

```
>> A = [1 2 3; 4 5 6; 7 8 9]

A =
      1     2     3
      4     5     6
      7     8     9
```

In addition, the Enter key (carriage return) can be used to separate the rows. For example, in the following case, the Enter key would be struck after the 3, the 6 and the] to assign the matrix:

```
>> A = [1 2 3
        4 5 6
        7 8 9]
```

Finally, we could construct the same matrix by *concatenating* (i.e., joining) the vectors representing each column:

```
>> A = [[1 4 7]' [2 5 8]' [3 6 9]']
```

At any point in a session, a list of all current variables can be obtained by entering the `who` command:

```
>> who

Your variables are:
A     a     ans  b    x
```

or, with more detail, enter the `whos` command:

```
>> whos

  Name        Size                    Bytes  Class

  A           3x3                        72  double array
  a           1x5                        40  double array
  ans         1x1                         8  double array
  b           5x1                        40  double array
  x           1x1                        16  double array (complex)

  Grand total is 21 elements using 176 bytes
```

Note that subscript notation can be used to access an individual element of an array. For example, the fourth element of the column vector `b` can be displayed as

```
>> b(4)

ans =
     8
```

For an array, `A(m,n)` selects the element in `m`th row and the `n`th column. For example,

```
>> A(2,3)

ans =
     6
```

There are several built-in functions that can be used to create matrices. For example, the `ones` and `zeros` functions create vectors or matrices filled with ones and zeros, respectively. Both have two arguments, the first for the number of rows and the second for the number of columns. For example, to create a 2 × 3 matrix of zeros:

```
>> E = zeros(2,3)

E =
     0     0     0
     0     0     0
```

Similarly, the `ones` function can be used to create a row vector of ones:

```
>> u = ones(1,3)

u =
     1     1     1
```

The Colon Operator

- The colon operator is a powerful tool for creating and manipulating arrays.
- If a colon is used to separate two numbers, MATLAB generates the numbers between them using an increment of one:

```
>> t = 1:5

t =
     1     2     3     4     5
```

If colons are used to separate three numbers, MATLAB generates the numbers between the first and third numbers using an increment equal to the second number:

```
>> t = 1:0.5:3

t =
     1.0000     1.5000     2.0000     2.5000     3.0000
```

Note that negative increments can also be used

```
>> t = 10:-1:5

t =
     10      9      8      7      6      5
```

Aside from creating series of numbers, the colon can also be used as a wildcard to select the individual rows and columns of a matrix. When a colon is used in place of a specific subscript, the colon represents the entire row or column. For example, the second row of the matrix `A` can be selected as in

```
>> A(2,:)

ans =
      4      5      6
```

We can also use the colon notation to selectively extract a series of elements from within an array. For example, based on the previous definition of the vector `t`:

```
>> t(2:4)

ans =
      9      8      7
```

Thus, the second through the fourth elements are returned.

The linspace and logspace Functions

The `linspace` and `logspace` functions provide other handy tools to generate vectors of spaced points. The `linspace` function generates a row vector of equally spaced points. It has the form

```
linspace(x1, x2, n)
```

which generates *n* points between *x1* and *x2*. For example

```
>> linspace(0,1,6)

ans =
         0    0.2000    0.4000    0.6000    0.8000    1.0000
```

The linspace and logspace Functions

The `logspace` function generates a row vector that is logarithmically equally spaced. It has the form

```
logspace(x1, x2, n)
```

which generates n logarithmically equally spaced points between decades 10^{x1} and 10^{x2}. For example,

```
>> logspace(-1,2,4)

ans =
     0.1000    1.0000   10.0000  100.0000
```

Character Strings

Aside from numbers, *alphanumeric* information or *character strings* can be represented by enclosing the strings within single quotation marks. For example,

```
>> f = 'Miles ';
>> s = 'Davis';
```

Each character in a string is one element in an array. Thus, we can *concatenate* (i.e., paste together) strings as in

```
>> x = [f s]

x =
Miles Davis
```

Note that very long lines can be continued by placing an *ellipsis* (three consecutive periods) at the end of the line to be continued. For example, a row vector could be entered as

```
>> a = [1 2 3 4 5 ...
6 7 8]

a =
     1     2     3     4     5     6     7     8
```

However, you cannot use an ellipsis within single quotes to continue a string. To enter a string that extends beyond a single line, piece together shorter strings as in

```
>> quote = ['Any fool can make a rule,' ...
' and any fool will mind it']

quote =
Any fool can make a rule, and any fool will mind it
```

MATHEMATICAL OPERATIONS

Operations with scalar quantities are handled in a straightforward manner, similar to other computer languages. The common operators, in order of priority, are

^	Exponentiation
–	Negation
* /	Multiplication and division
\	Left division[2]
+ –	Addition and subtraction

These operators will work in calculator fashion. Try

```
>> 2*pi

ans =
    6.2832
```

Also, scalar real variables can be included:

```
>> y = pi/4;
>> y ^ 2.45

ans =
    0.5533
```

Results of calculations can be assigned to a variable, as in the next-to-last example, or simply displayed, as in the last example.

Results of calculations can be assigned to a variable, as in the next-to-last example, or simply displayed, as in the last example.

As with other computer calculation, the priority order can be overridden with parentheses. For example, because exponentiation has higher priority then negation, the following result would be obtained:

```
>> y = -4 ^ 2

y =
    -16
```

Thus, 4 is first squared and then negated. Parentheses can be used to override the priorities as in

```
>> y = (-4) ^ 2

y =
     16
```

Calculations can also involve complex quantities. Here are some examples that use the values of x ($2 + 4i$) and y (16) defined previously:

```
>> 3 * x

ans =
   6.0000 + 12.0000i

>> 1 / x

ans =
   0.1000 - 0.2000i

>> x ^ 2

ans =
 -12.0000 + 16.0000i

>> x + y

ans =
  18.0000 + 4.0000i
```

```
>> a = [1 2 3];
```

and

```
>> b = [4 5 6]';
```

```
>> A = [1 2 3; 4 5 6; 7 8 9]
```

Now, try

```
>> a * A

ans =
     30     36     42
```

```
>> A * b

ans =
    32
    77
   122
```

Matrices cannot be multiplied if the inner dimensions are unequal. Here is what happens when the dimensions are not those required by the operations. Try

```
>> A * a
```

MATLAB automatically displays the error message:

```
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

MATLAB automatically displays the error message:

```
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

Matrix-matrix multiplication is carried out in likewise fashion:

```
>> A * A

ans =
    30    36    42
    66    81    96
   102   126   150
```

Mixed operations with scalars are also possible:

```
>> A/pi

ans =
    0.3183    0.6366    0.9549
    1.2732    1.5915    1.9099
    2.2282    2.5465    2.8648
```

We must always remember that MATLAB will apply the simple arithmetic operators in vector-matrix fashion if possible. At times, you will want to carry out calculations item by item in a matrix or vector. MATLAB provides for that too. For example,

```
>> A^2

ans =
    30    36    42
    66    81    96
   102   126   150
```

results in matrix multiplication of A with itself.

What if you want to square each element of A? That can be done with

```
>> A.^2

ans =
     1     4     9
    16    25    36
    49    64    81
```

The . preceding the ^ operator signifies that the operation is to be carried out element by element. The MATLAB manual calls these *array operations*. They are also often referred to as *element-by-element operations.*

MATLAB contains a helpful shortcut for performing calculations that you've already done. Press the up-arrow key. You should get back the last line you typed in.

```
>> A.^2
```

Pressing Enter will perform the calculation again. But you can also edit this line. For example, change it to the line below and then press Enter.

```
>> A.^3

ans =
     1     8    27
    64   125   216
   343   512   729
```

USE OF BUILT-IN FUNCTIONS

MATLAB and its Toolboxes have a rich collection of built-in functions. You can use online help to find out more about them. For example, if you want to learn about the `log` function, type in

```
>> help log

 LOG    Natural logarithm.
    LOG(X) is the natural logarithm of the elements of X.
    Complex results are produced if X is not positive.

    See also LOG2, LOG10, EXP, LOGM.
```

For a list of all the elementary functions, type

```
>> help elfun
```

One of their important properties of MATLAB's built-in functions is that they will operate directly on vector and matrix quantities. For example, try

```
>> log(A)

ans =
         0    0.6931    1.0986
    1.3863    1.6094    1.7918
    1.9459    2.0794    2.1972
```

```
>> sqrtm(A)

ans =
    0.4498 + 0.7623i   0.5526 + 0.2068i   0.6555 - 0.3487i
    1.0185 + 0.0842i   1.2515 + 0.0228i   1.4844 - 0.0385i
    1.5873 - 0.5940i   1.9503 - 0.1611i   2.3134 + 0.2717i
```

There are several functions for rounding. For example, suppose that we enter a vector:

```
>> E = [-1.6 -1.5 -1.4 1.4 1.5 1.6];
```

The `round` function rounds the elements of `E` to the nearest integers:

```
>> round(E)

ans =
    -2    -2    -1     1     2     2
```

The `ceil` (short for ceiling) function rounds to the nearest integers toward infinity:

```
>> ceil(E)

ans =
    -1    -1    -1     2     2     2
```

The `floor` function rounds down to the nearest integers toward minus infinity:

```
>> floor(E)

ans =
    -2    -2    -2     1     1     1
```

There are also functions that perform special actions on the elements of matrices and arrays. For example, the `sum` function returns the sum of the elements:

```
>> F = [3 5 4 6 1];
>> sum(F)

ans =
     19
```

In a similar way, it should be pretty obvious what's happening with the following commands:

```
>> min(F),max(F),mean(F),prod(F),sort(F)

ans =
     1

ans =
     6

ans =
     3.8000

ans =
     360

ans =
     1     3     4     5     6
```

A common use of functions is to evaluate a formula for a series of arguments. Recall that the velocity of a free-falling bungee jumper can be computed with [Eq. (1.9)]:

$$v = \sqrt{\frac{gm}{c_d}} \tanh\left(\sqrt{\frac{gc_d}{m}}\, t\right)$$

where v is velocity (m/s), g is the acceleration due to gravity (9.81 m/s^2), m is mass (kg), c_d is the drag coefficient (kg/m), and t is time (s).

```
>> t = [0:2:20]'

t =
     0
     2
     4
     6
     8
    10
    12
    14
    16
    18
    20
```

Check the number of items in the `t` array with the `length` function:

```
>> length(t)

ans =
    11
```

Assign values to the parameters:

```
>> g = 9.81; m = 68.1; cd = 0.25;
```

MATLAB allows you to evaluate a formula such as $v = f(t)$, where the formula is computed for each value of the t array, and the result is assigned to a corresponding position in the v array. For our case,

```
>> v = sqrt(g*m/cd)*tanh(sqrt(g*cd/m)*t)
```

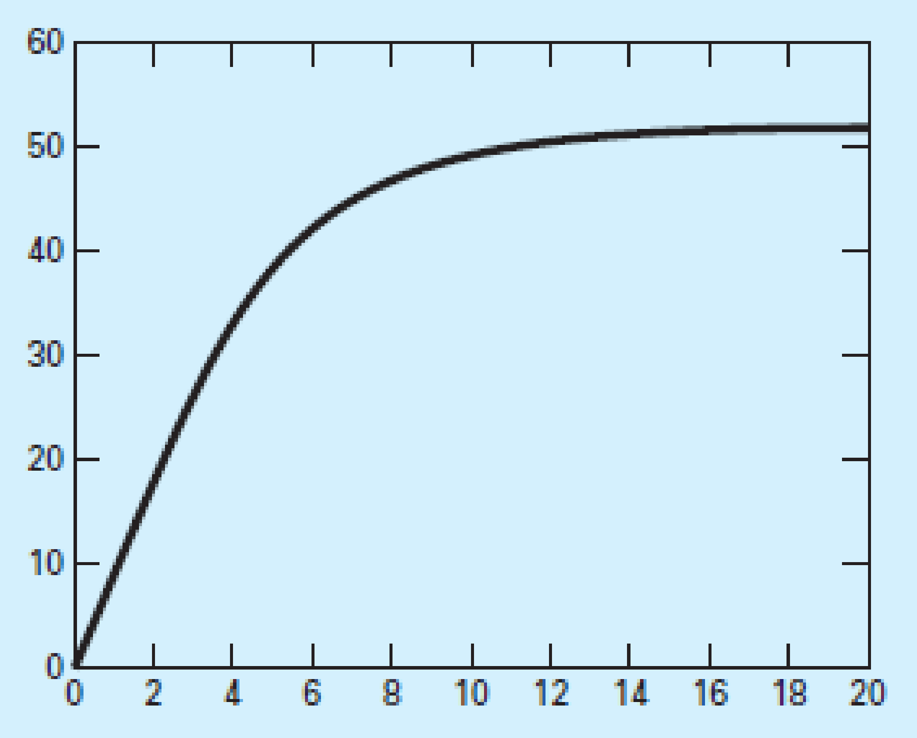
```
v =
         0
   18.7292
   33.1118
   42.0762
   46.9575
   49.4214
   50.6175
   51.1871
   51.4560
   51.5823
   51.6416
```

GRAPHICS

MATLAB allows graphs to be created quickly and conveniently. For example, to create a graph of the t and v arrays from the data above, enter

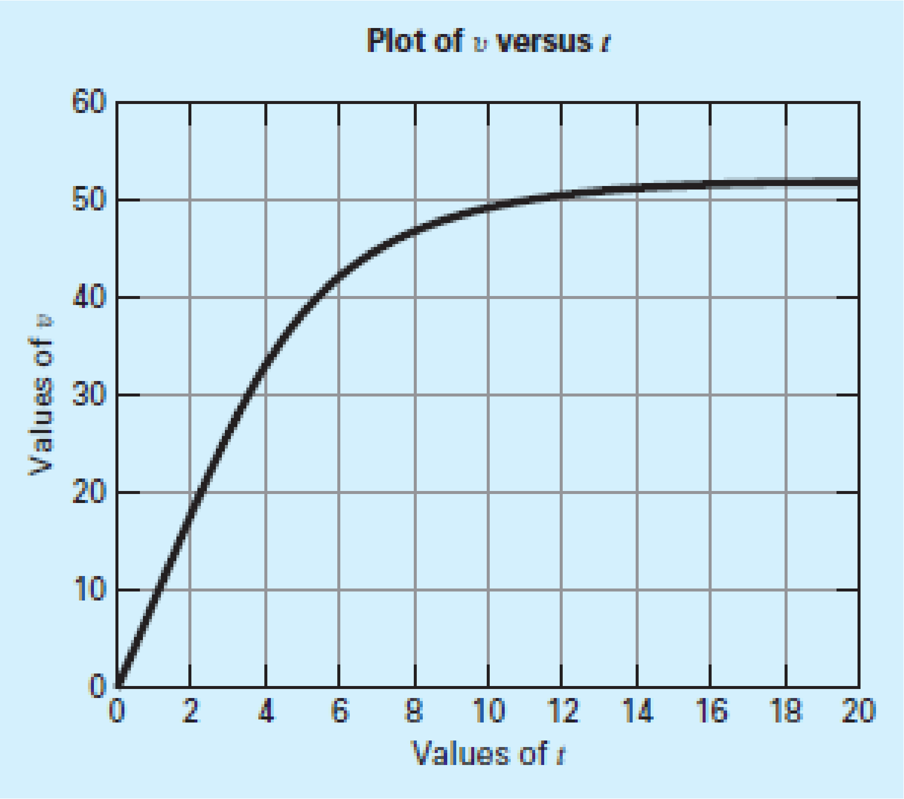
```
>> plot(t, v)
```

The graph appears in the graphics window and can be printed or transferred via the clipboard to other programs.



You can customize the graph a bit with commands such as the following:

```
>> title('Plot of v versus t')
>> xlabel('Values of t')
>> ylabel('Values of v')
>> grid
```



Colors		Symbols		Line Types	
Blue	b	Point	.	Solid	-
Green	g	Circle	o	Dotted	:
Red	r	X-mark	x	Dashdot	-.
Cyan	c	Plus	+	Dashed	--
Magenta	m	Star	*		
Yellow	y	Square	s		
Black	k	Diamond	d		
White	w	Triangle(down)	v		
		Triangle(up)	^		
		Triangle(left)	<		
		Triangle(right)	>		
		Pentagram	p		
		Hexagram	h		

The `plot` command displays a solid thin blue line by default. If you want to plot each point with a symbol, you can include a specifier enclosed in single quotes in the `plot` function. Table 2.2 lists the available specifiers. For example, if you want to use open circles enter

```
>> plot(t, v, 'o')
```

You can also combine several specifiers. For example, if you want to use square green markers connected by green dashed lines, you could enter

```
>> plot(t, v, 's--g')
```

You can also control the line width as well as the marker's size and its edge and face (i.e., interior) colors. For example, the following command uses a heavier (2-point), dashed, cyan line to connect larger (10-point) diamond-shaped markers with black edges and magenta faces:

```
>> plot(x,y,'--dc','LineWidth',2,...
        'MarkerSize',10,...
        'MarkerEdgeColor','k',...
        'MarkerFaceColor','m')
```

```
>> t = 0:pi/50:10*pi;
>> subplot(1,2,1);plot(sin(t),cos(t))
>> axis square
>> title('(a)')
```

```
>> subplot(1,2,2);plot3(sin(t),cos(t),t);
>> title('(b)')
```

Problems

2.20 The temperature dependence of chemical reactions can be computed with the *Arrhenius equation:*

$$k = Ae^{-E/(RT_a)}$$

where k = reaction rate (s^{-1}), A = the preexponential (or frequency) factor, E = activation energy (J/mol), R = gas constant [8.314 J/(mole · K)], and T_a = absolute temperature (K). A compound has $E = 1 \times 10^5$ J/mol and $A = 7 \times 10^{16}$. Use MATLAB to generate values of reaction rates for temperatures ranging from 253 to 325 K. Use `subplot` to generate a side-by-side graph of (a) k versus T_a (green line) and (b) $\log_{10} k$ (red line) versus $1/T_a$. Employ the `semilogy` function to create (b). Include axis labels and titles for both subplots. Interpret your results.