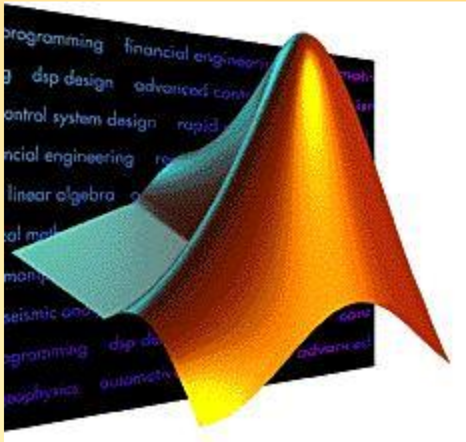
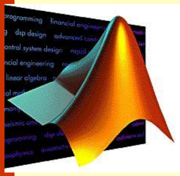
Programming with Matlab





Review minggu lalu

Problem Statement. A bungee jumper with a mass of 68.1 kg leaps from a stationary hot air balloon. Use Eq. (1.9) to compute velocity for the first 12 s of free fall. Also determine the terminal velocity that will be attained for an infinitely long cord (or alternatively, the jumpmaster is having a particularly bad day!). Use a drag coefficient of 0.25 kg/m.

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)
```

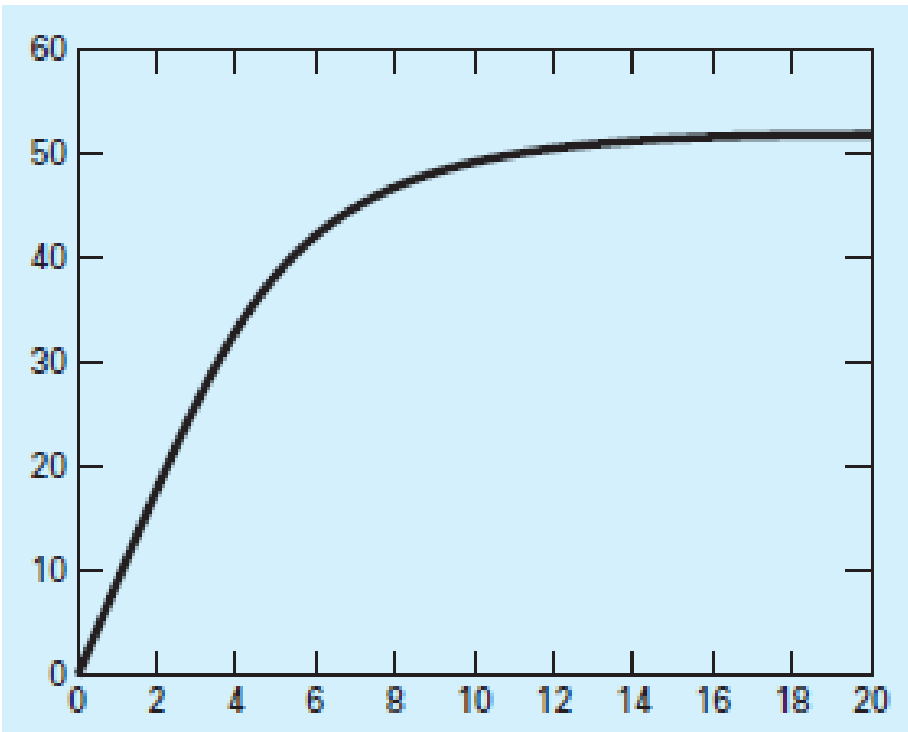
```
v =
          0
    18.7292
    33.1118
    42.0762
    46.9575
    49.4214
    50.6175
    51.1871
    51.4560
    51.5823
    51.6416
```

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)
```

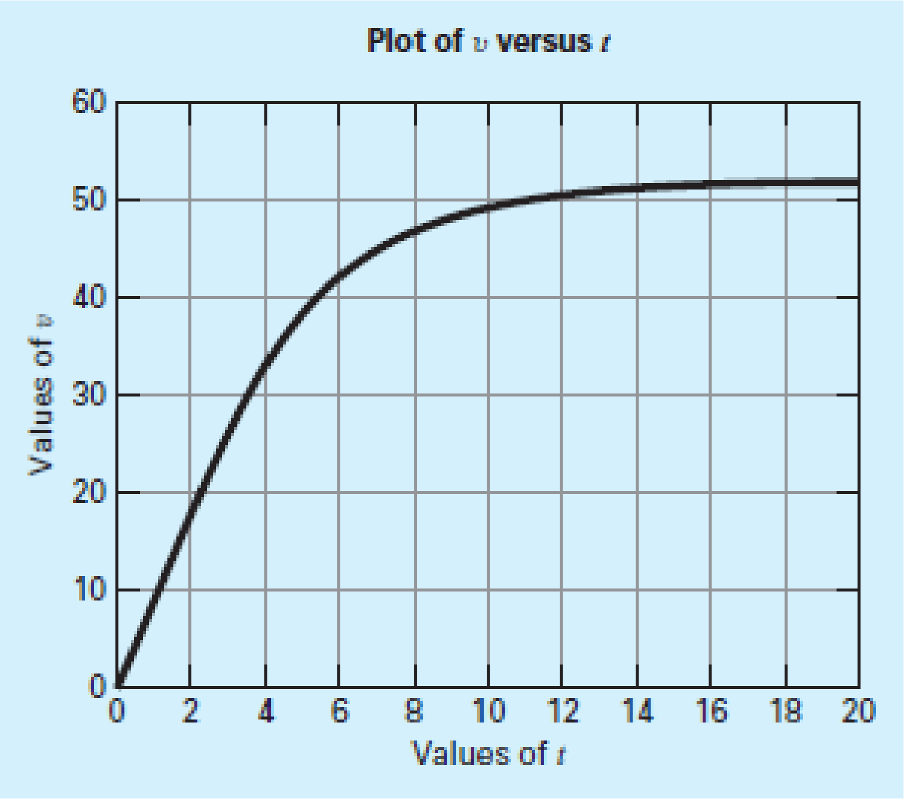
The graph appears in the graphics window and can be printed or transferred via the clipboard to other programs.



02 Programming with Matlab

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)')
```

M-FILES

- The most common way to operate MATLAB is by entering commands one at a time in the command window.
- M-files provide an alternative way of performing operations that greatly expand MATLAB.s problem-solving capabilities.

- An *M-file* consists of a series of statements that can be run all at once.
- Note that the nomenclature m-file. comes from the fact that such files are stored with a .m extension.
- M-files come in two flavors: script files and function files.

Script Files

- *A script file* is merely a series of MATLAB commands that are saved on a file.
- They are useful for retaining a series of commands that you want to execute on more than one occasion.
- The script can be executed by typing the file name in the command window or by invoking the menu selections in the edit window: **Debug, Run.**

Back this problem

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

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

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).

Open the editor with the menu selection: **File, New, M-file.** Type in the following statements to compute the velocity of the free-falling bungee jumper at a specific time [recall Eq. (1.9)]:

```
g = 9.81; m = 68.1; t = 12; cd = 0.25;
v = sqrt(g * m / cd) * tanh(sqrt(g * cd / m) * t)
```

Save the file as `scriptdemo.m`. Return to the command window and type

```
>>scriptdemo
```

The result will be displayed as

```
v =
    50.6175
```

Function Files

- *Function files* are M-files that start with the word function.
- In contrast to script files, they can accept input arguments and return outputs.

- The syntax for the function file can be represented generally as

  function *outvar* = *funcname*(*arglist*)

  % *helpcomments*

  statements

  outvar = *value*;

- where *outvar* = the name of the output variable,
- *funcname* = the function's name,
- *arglist* = the function's argument list (i.e., comma-delimited values that are passed into the function),
- *helpcomments* = text that provides the user with information regarding the function (these can be invoked by typing Help *funcname* in the command window), and
- *statements* = MATLAB statements that compute the *value* that is assigned to *outvar*.

mming with Matlab

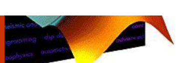
```
function v = freefall(t, m, cd)
% freefall: bungee velocity with second-order drag
% v=freefall(t,m,cd) computes the free-fall velocity
%                    of an object with second-order drag
% input:
%   t = time (s)
%   m = mass (kg)
%   cd = second-order drag coefficient (kg/m)
% output:
%   v = downward velocity (m/s)

g = 9.81;     % acceleration of gravity
v = sqrt(g * m / cd)*tanh(sqrt(g * cd / m) * t);
```

Save the file as `freefall.m`. To invoke the function, return to the command window and type in

```
>> freefall(12,68.1,0.25)
```

The result will be displayed as

```
ans =
   50.6175
```



Save the file as `freefall.m`. To invoke the function, return to the command window and type in

```
>> freefall(12,68.1,0.25)
```

The result will be displayed as

```
ans =
   50.6175
```

One advantage of a function M-file is that it can be invoked repeatedly for different argument values. Suppose that you wanted to compute the velocity of a 100-kg jumper after 8 s:

```
>> freefall(8,100,0.25)

ans =
   53.1878
```

To invoke the help comments type

```
>> help freefall
```

which results in the comments being displayed

```
freefall: bungee velocity with second-order drag
  v=freefall(t,m,cd) computes the free-fall velocity
                     of an object with second-order drag
input:
  t = time (s)
  m = mass (kg)
  cd = second-order drag coefficient (kg/m)
output:
  v = downward velocity (m/s)
```

INPUT-OUTPUT

- ***The input Function***. This function allows you to prompt the user for values directly from the command window. Its syntax is

  n = input('*promptstring*')

- The function displays the *promptstring*, waits for keyboard input, and then returns the value from the keyboard.

- For example,

  m = input('Mass (kg): ')

- When this line is executed, the user is prompted with the message

  Mass (kg):

- If the user enters a value, it would then be assigned to the variable m.

- The input function can also return user input as a string.
- To do this, an 's' is appended to the function.s argument list.
- For example,

```
name = input('Enter your name: ','s')
```

- ***The disp Function***. This function provides a handy way to display a value. Its syntax is

  disp(*value*)

- where *value* = the value you would like to display. It can be a numeric constant or variable, or a string message enclosed in hyphens.
- Its application is illustrated in the following example.

```
function freefalli
% freefalli: interactive bungee velocity
%   freefalli interactive computation of the
%             free-fall velocity of an object
%             with second-order drag.
g = 9.81;     % acceleration of gravity
m = input('Mass (kg): ');
cd = input('Drag coefficient (kg/m): ');
t = input('Time (s): ');
disp(' ')
disp('Velocity (m/s):')
disp(sqrt(g * m / cd)*tanh(sqrt(g * cd / m) * t))
```

Save the file as `freefalli.m`. To invoke the function, return to the command window and type

```
>> freefalli

Mass (kg): 68.1
Drag coefficient (kg/m): 0.25
Time (s): 12

Velocity (m/s):
   50.6175
```

Save the file as `freefalli.m`. To invoke the function, return to the command window and type

```
>> freefalli

Mass (kg): 68.1
Drag coefficient (kg/m): 0.25
Time (s): 12

Velocity (m/s):
   50.6175
```

- ***The fprintf Function***. This function provides additional control over the display of information.
- A simple representation of its syntax is

  fprintf('format', x, ...)

- where *format* is a string specifying how you want the value of the variable *x* to be displayed.
- The operation of this function is best illustrated by examples.

A simple example would be to display a value along with a message. For instance, suppose that the variable `velocity` has a value of 50.6175. To display the value using eight digits with four digits to the right of the decimal point along with a message, the statement along with the resulting output would be

```
>> fprintf('The velocity is %8.4f m/s\n', velocity)

The velocity is 50.6175 m/s
```

Format Code	Description
%d	Integer format
%e	Scientific format with lowercase e
%E	Scientific format with uppercase E
%f	Decimal format
%g	The more compact of %e or %f

Control Code	Description
\n	Start new line
\t	Tab

The `fprintf` function can also be used to display several values per line with different formats. For example,

```
>> fprintf('%5d %10.3f %8.5e\n',100,2*pi,pi);

  100      6.283 3.14159e+000
```

It can also be used to display vectors and matrices. Here is an M-file that enters two sets of values as vectors. These vectors are then combined into a matrix, which is then displayed as a table with headings:

```
function fprintfdemo
x = [1 2 3 4 5];
y = [20.4 12.6 17.8 88.7 120.4];
z = [x;y];
fprintf('      x         y\n');
fprintf('%5d %10.3f\n',z);
```

The result of running this M-file is

```
>> fprintfdemo

      x         y
      1    20.400
      2    12.600
      3    17.800
      4    88.700
      5   120.400
```

The result of running this M-file is

```
>> fprintfdemo
```

x	y
1	20.400
2	12.600
3	17.800
4	88.700
5	120.400

Next meeting

STRUCTURED PROGRAMMING

- The simplest of all M-files perform instructions sequentially.
- That is, the program statements are executed line by line starting at the top of the function and moving down to the end.
- Because a strict sequence is highly limiting, all computer languages include statements allowing programs to take nonsequential paths.

These can be classified as

- *Decisions* (or Selection). The branching of flow based on a decision.

- *Loops* (or Repetition). The looping of flow to allow statements to be repeated.

Decisions

The ***if*** Structure.

- This structure allows you to execute a set of statements if a logical condition is true.
- Its general syntax is

```
if condition
  statements
end
```

```
function grader(grade)
% grader(grade):
%   determines whether grade is passing
% input:
%   grade = numerical value of grade (0-100)
% output:
%   displayed message
if grade >= 60
  disp('passing grade')
end
```

The following illustrates the result

```
>> grader(95.6)
passing grade
```

Example	Operator	Relationship
`x == 0`	`==`	Equal
`unit ~= 'm'`	`~=`	Not equal
`a < 0`	`<`	Less than
`s > t`	`>`	Greater than
`3.9 <= a/3`	`<=`	Less than or equal to
`r >= 0`	`>=`	Greater than or equal to

The ***if...else*** Structure.

- This structure allows you to execute a set of statements if a logical condition is true and to execute a second set if the condition is false.
- Its general syntax is

```
if condition
   statements1
else
   statements2
end
```

The ***if...elseif*** Structure.

- It often happens that the false option of an if...else structure is another decision. This type of structure often occurs when we have more than two options for a particular problem setting. For such cases, a special form of decision structure, the if...elseif has been developed.
- It has the general syntax

```
if condition1
  statements1
elseif condition2
  statements2
```

The price of an apel is \$ 25. If someone buy more than 5 he will get a discount of 20 %.

Script m-file

```
n = input('the number of apples = ');
price = n*25;
if n > 5
    price = (1-20/100)*price;
end
t = ['price= ' num2str(price)];
disp(t)
```

The students will be pass if their grade is more than 60

```
N = input(‘ The grade of student = ’);
if N > 60;
   disp ‘passing grade’
else
   disp ‘NOT passing grade’
end
```

```
% The grade of the students
n = input('The grade of student = ');
if n<  40
    disp 'E'
elseif n< 60
    disp 'D'
elseif n< 74
    disp 'C'
elseif n< 87
    disp 'B'
else
    disp 'A'
end
```

Loops

The *for…end* Structure.

- Afor loop repeats statements a specific number of times.
- Its general syntax is

```
for index = start:step:finish
  statements
end
```

```
for i = 1:5
   disp(i)
end
```

```
for j = 10:-1:1
   disp(j)
end
```

- Develop an M-file to compute the factorial.

$$
\begin{aligned}
0! &= 1\\
1! &= 1\\
2! &= 1 \times 2 = 2\\
3! &= 1 \times 2 \times 3 = 6\\
4! &= 1 \times 2 \times 3 \times 4 = 24\\
5! &= 1 \times 2 \times 3 \times 4 \times 5 = 120\\
&\vdots
\end{aligned}
$$

```
function fout = factor(n)
% factor(n):
%   Computes the product of all the integers from 1 to n.
x = 1;
for i = 1:n
  x = x * i;
end
fout = x;
end
```

which can be run as

```
>> factor(5)

ans =
   120
```

The ***while*** Structure.

- A while loop repeats as long as a logical condition is true.
- Its general syntax is

```
while condition
  statements
end
```

```
x = 8
while x > 0
   x = x - 3;
   disp(x)
end
```

```
x =

     8

     5

     2

    -1
```

```
num = 0; n = 10;
while n>1;
        n=n/2
        num = num+1
 end
```

Problem Statement. The roots of a quadratic equation

$$f(x) = ax^2 + bx + c$$

can be determined with the quadratic formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Develop a function to implement this formula given values of the coeffcients.

First check

determinant $D = B^2 - 4*A*C$

If D > 0; roots is real, $x1 = \frac{-B + \sqrt{D}}{2A}$ $x2 = \frac{-B - \sqrt{D}}{2A}$

If D < 0; roots is imajiner $x1 = \frac{-B}{2A} + \frac{\sqrt{-D}}{2A} i$ $x2 = \frac{-B}{2A} - \frac{\sqrt{-D}}{2A} i$

If D = 0; roots is the same $x1 = x2 = \frac{-B}{2A}$

```
clear all
clc

a = input(' a =  ');
b = input(' b =  ');
c = input(' c =  ');

d = b^2-4*a*c;
if d > 0
    x1=((-b+sqrt(d))/2*a);
    x2=((-b-sqrt(d))/2*a);
    disp 'Roots of this equation is '
    t = [' x1 = ' num2str(x1) ' and x2 = ' num2str(x2)];
    disp(t)
elseif d<0
    x1=((-b)/(2*a))+(sqrt(-d)/2*a)*i;
    x2=((-b)/(2*a))-(sqrt(-d)/2*a)*i;
    disp 'Roots of this equation is '
    t = [' x1 = ' num2str(x1) ' and x2 = ' num2str(x2)];
    disp(t)
else
    x =((-b)/2*a);
    disp 'Roots of this equation is '
    t = [' x1 = x2 = ' num2str(x)];
    disp(t)
end
```

```
nilai a = 1
nilai b = -1
nilai c = -6
Akar-akar Persamaannya adalah
 x1 = 3 dan x2 = -2


nilai a = 1
nilai b = -4
nilai c = 4
Akar-akar Persamaannya adalah
 x1 = x2 = 2


nilai a = 1
nilai b = 0
nilai c = 4
Akar-akar Persamaannya imajiner, ya
 x1 = 0+2i
 x1 = 0-2i
```

Koefisien drag partikel bola untuk terminal velocity bervariasi dengan bilangan Reynolds (Re) sebagai berikut :

$$C_D = \frac{24}{Re} \qquad \text{untuk} \quad Re < 0{,}1$$

$$C_D = \frac{24}{Re}\left(1 + 0{,}14\,Re^{0,7}\right) \qquad \text{untuk} \quad 0{,}1 \le Re \le 1000$$

$$C_D = 0{,}44 \qquad \text{untuk} \quad 1000 < Re \le 35000$$

$$C_D = 0{,}19 - 8.10^4/Re \qquad \text{untuk} \quad 35000 \le Re$$

dengan Re adalah bilangan Reynold.

Tentukan koefisien drag untuk partikel batubara yang jatuh dalam air. (Re ditanyakan di dalam program)

```
clear all
clc
Re=input('Besarnya bilangan Reynold (Re) = ');
if Re < 0.1
    C_D = 24/Re;
elseif Re < 1000
    C_D = 24*(1+0.14*Re^0.7)/Re;
elseif Re < 350000;
    C_D = 0.44;
else
    C_D = 0.19-80000/Re;
end
disp 'Koefisien drag partikel adalah '
t=[' CD = ' num2str(C_D)];
   disp(t)
```