4.0 Programming with MATLAB

4.1 M-files

The term “M-file” is obtained from the fact that such files are stored with .m extension. M-files are alternative means of performing computations so as to expand MATLAB’s problem-solving capabilities. An *M-file* consists of series of statements that can be run all at once to execute a given problem. M-files come in two forms: script files and function files.

4.1.1 Script files

Script file is a series of MATLAB commands that are saved on a file. They are useful for retaining a series of commands that can be executed in 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.**

Example: Develop a script file to compute the velocity of the free-falling body when the initial velocity is zero.

Solution:

Open the editor with the menu selection: **File, New, M-file.**

Type in the following statements to compute the velocity of the free-falling body at a specific time.

```
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 velfall.m.

Return to the command window and type

```
>>velfall
```

The result should be displayed as

```
v =
        50.6175
```

4.1.2 Function files

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

Examples:

i. Create an M-file called falling.m.

```
function h = falling(t)
```

```
GRAVITY = 32;
h = 1/2*GRAVITY*t.^2;
```

at the command window, type

```
y = falling((0:.1:5)');
```

ii.
```
function w = twosum(x,y)
      w = x+y;
```

type on the command window

```
>> twosum(1,2)

>> x = [1 2]; y = [3 4];
>> twosum(x,y)
>> A = [1 2; 3 4]; B = [5 6; 7 8];
>> twosum(A,B);
```

iii.
```
function s = threesum(x,y,z)% threesum Add three variables
% and return the result
s = x+y+z;

>> a = threesum(1,2,3);

>> b= threesum(7,8,9);
```

iv.
```
function [s,p] = addmult(x,y)
% addmult Compute sum and product
% of two matrices
s = x+y;
p = x*y;

>> [a,b] = addmult(3,2)
>> v = addmult(3,2)
```

4.2 Flow Control

MATLAB flow control consists of the following:

•if statements

•switch statements

•for loops

•while loops

•continue statements

•break statements

if

The **if** statement is fundamental in decision-making for all computing languages. It 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.

e.g:

i.
```
x = 0
if x == 0
disp( 'x equals zero' )
end
```

ii.
```
x = 2;
if x < 0
disp( 'neg' )
else disp( 'non-neg' )
end
```

iii. MATLAP script to compute the root of a quadratic equation

$ax^2 + bx + c = 0.$

```
a = 2;
b  = -10;
c = 12;
d = b^2 - 4*a*c;
if a ~= 0
```

```
if d < 0
disp( 'Complex roots' )
else
x1 = (-b + sqrt( d )) / (2*a)
x2 = (-b - sqrt( d )) / (2*a)
end % first end
end
```

switch

The **switch** statement executes groups of statements based on the value of a variable or expression. There must always be an end to match the switch.

e.g:

```
d = floor(10*rand);
switch d
case {2, 4, 6, 8}
disp( 'Even' );
case {1, 3, 5, 7, 9}
disp( 'Odd' );
otherwise
disp( 'Zero' );
end
```

for

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

i.
```
for n = 3:32
r(n) = rank(magic(n));
end
r
```

ii.
```
for i = 1:m
for j = 1:n
H(i,j) = 1/(i+j);
end
end
```

while

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

A complete program illustrating while, if, else, and end, is written as:

```
a = 0; fa = -Inf;
b = 3; fb = Inf;
while b-a > eps*b
x = (a+b)/2;
fx = x^3-2*x-5;
if sign(fx) == sign(fa)
a = x; fa = fx;
else
b = x; fb = fx;
end
end
x
```

continue

The continue statement passes control to the next iteration of the **for** 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** or **while** loop enclosing it.

```
fid = fopen('magic.m','r');
count = 0;
while ~feof(fid)
    line = fgetl(fid);
    if isempty(line) | strncmp(line,'%',1)
        continue
    end
    count = count + 1;
end
disp(sprintf('%d lines',count));
```

break

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

```
a = 0; fa = -Inf;
b = 3; fb = Inf;
while b-a > eps*b
x = (a+b)/2;
fx = x^3-2*x-5;
if fx == 0
break
elseif sign(fx) == sign(fa)
a = x; fa = fx;
else
b = x; fb = fx;
end
end
```

4.3 Function of Functions

A class of functions, called "function functions," works with nonlinear functions of a scalar variable. That is, one function works on another function. The function functions include:

•Zero finding

•Optimization

•Quadrature

•Ordinary differential equations

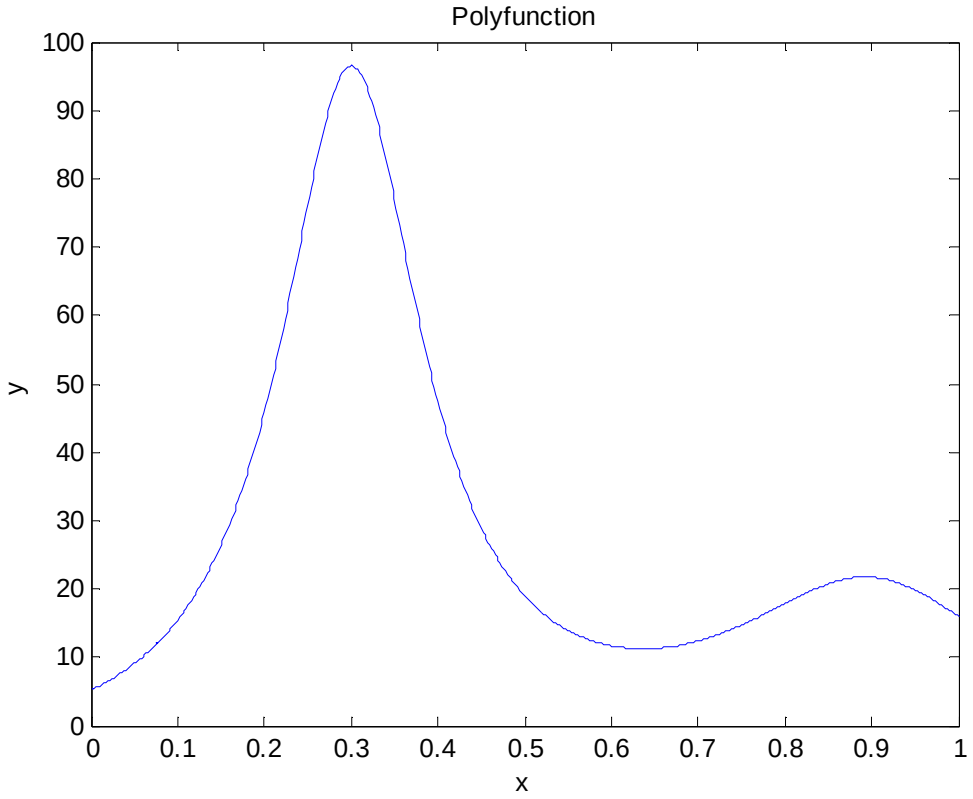
```
function y = humps(x)
y = 1./((x-.3).^2 + .01) + 1./((x-.9).^2 + .04) - 6;
```

Evaluate this function at a set of points in the interval $0 \le x \le 1$ in command window as follow:

```
>>x = 0:.002:1;
>>y = humps(x);
```

Then plot the function with

```
plot(x,y),title('Polyfunction'),xlabel('x'),ylabel('y')
```




```
p = fminsearch(@humps,0.5)
```
% finds the value of *x* where the function is minimum.

p =

0.6370

To evaluate the function at the minimum,

```
humps(p)
ans =

   11.2528
```

```
Q = quadl(@humps,0,1)
```
% computes the area under the curve in the graph

Q =

29.8583