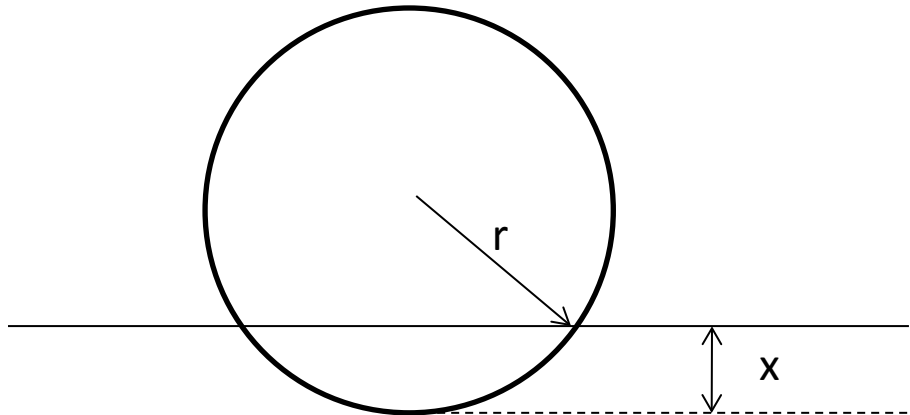


**Problem:** A 50mm diameter cork ball (specific gravity =  $\rho = 0.25$ ) is floating in water. How deep will the ball float?



The total volume of the sphere is:  $V = \frac{4}{3}\pi r^3$

The submerged volume is given by:  $V_s = \frac{1}{3}(\pi x^2(3r - x))$

Ball weight = weight of displaced water:  $V\rho = V_s(1)$

Final result (in root finding form):  $x^3 - 3rx^2 + 4r^3\rho = 0$

Note: problem adapted from Fausett p 48

General formulae for a cubic equation (  $ax^3 + bx^2 + cx + d = 0$ ):

$$\begin{aligned}
 x_1 &= -\frac{b}{3a} \\
 &\quad -\frac{1}{3a} \sqrt[3]{\frac{2b^3 - 9abc + 27a^2d + \sqrt{(2b^3 - 9abc + 27a^2d)^2 - 4(b^2 - 3ac)^3}}{2}} \\
 &\quad -\frac{1}{3a} \sqrt[3]{\frac{2b^3 - 9abc + 27a^2d - \sqrt{(2b^3 - 9abc + 27a^2d)^2 - 4(b^2 - 3ac)^3}}{2}} \\
 x_2 &= -\frac{b}{3a} \\
 &\quad + \frac{1 + i\sqrt{3}}{6a} \sqrt[3]{\frac{2b^3 - 9abc + 27a^2d + \sqrt{(2b^3 - 9abc + 27a^2d)^2 - 4(b^2 - 3ac)^3}}{2}} \\
 &\quad + \frac{1 - i\sqrt{3}}{6a} \sqrt[3]{\frac{2b^3 - 9abc + 27a^2d - \sqrt{(2b^3 - 9abc + 27a^2d)^2 - 4(b^2 - 3ac)^3}}{2}} \\
 x_3 &= -\frac{b}{3a} \\
 &\quad + \frac{1 - i\sqrt{3}}{6a} \sqrt[3]{\frac{2b^3 - 9abc + 27a^2d + \sqrt{(2b^3 - 9abc + 27a^2d)^2 - 4(b^2 - 3ac)^3}}{2}} \\
 &\quad + \frac{1 + i\sqrt{3}}{6a} \sqrt[3]{\frac{2b^3 - 9abc + 27a^2d - \sqrt{(2b^3 - 9abc + 27a^2d)^2 - 4(b^2 - 3ac)^3}}{2}}
 \end{aligned}$$

For cubics using a formula to find the roots is not a practical proposition.

Casio calculator still a possibility (enter 3 for “Degree?” and keep in mind that there may be one real and two complex roots).

In Matlab cubics are a piece of cake:

```
>> D = 50;  
>> r = D/2;  
>> rho = 0.25;  
  
>> p = [ 1 (-3 * r) 0 (4 * r ^3 * rho)];  
  
>> x = roots(p)  
  
x =  
    71.9846  
    16.3176  
   -13.3022
```

As  $x$  must be between 0 and  $D$  the answer is 16.3176 mm.

*fzero* is just as convenient and has the advantage of giving us just the root we want:

```
>> f = @(x) x.^3 - 3 * r * x.^2 + 4 * r^3 * rho;
```

```
>> depth = fzero(f, [0 D])
```

Two possible ways of plotting the function:

```
>> figure (1)
```

```
>> fplot (f, [-D (2 * D)]);
```

```
>> figure (2)
```

```
>> x = linspace (-D, 2 * D, 50);
```

```
>> y = polyval (p, x);      % "p" (polynomial coefficients) from previous slide
```

```
>> plot (x, y);
```

Function *polyval* accepts the coefficients of a polynomial and an *x* value. It produces the corresponding *y* value. The *x* value may be a vector (as it is here). In this case the result is also a vector.

Note: In this case  $y = f(x)$  would have exactly the same effect.

## Script m-files

- “Canned” collections of Matlab commands
- Created and modified in editor window. Stored as “*script.m*”, where *script* is a user selected name.
- To execute the commands, enter “run *script*” (or just *script*) in the command window.
- Running a script file is logically equivalent to typing in all of the commands contained in it.
- If you are unable to execute a script check that your path includes the folder in which it is stored.
- If the results seem out of date, make sure that the script was saved before being executed (Matlab runs the last SAVED version of the script).
- Scripts may also be executed from the editor window (F5 or via the “Debug” menu). Results appear in the command window. Doing things this way automatically saves the script.

The script file below “cans” the code for the floating ball problem.

```
D = 50;
r = D/2;
rho = 0.25;

p = [ 1 (-3 * r) 0 (4 * r ^3 * rho)];
x = roots(p) % display all roots (semi-colon omitted)

f = @(x) x.^3 - 3 * r * x.^2 + 4 * r^3 * rho;
depth = fzero(f, [0 D]) % display root of interest

% plot function using fplot
figure (1)
fplot (f, [-D (2 * D)]);
grid on;

% plot function using plot
figure (2)
x = linspace (-D, 2 * D, 50);
y = polyval (p, x);
plot (x, y);
grid on;
```

% ball.m

Script m-files can be made more generally useful by having them read in values. The script m-file below reads in the ball diameter and specific gravity, works out how deep the ball will float, and outputs the result in a nicely formatted message.

```
% interactive version of ball script file

rho = input ('Enter specific gravity: ');
D = input ('Enter diameter of ball in mm: ');
r = D / 2;

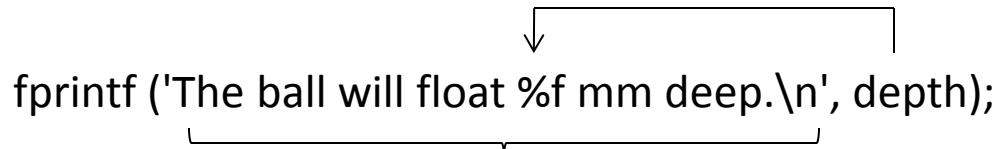
f = @(x) x.^3 - 3 * r * x.^2 + 4 * r^3 * rho;

depth = fzero(f, [0 D]);

fprintf ('The ball will float %f mm deep.\n', depth);           % ball2.m
```

Function *input* displays the specified prompt message, waits for the user to enter a value, and then returns this value.

Aside: What will happen if the specific gravity entered is not between 0 and 1?

  
fprintf ('The ball will float %f mm deep.\n', depth);  
Format string

Format codes:

%d	integer format
%e,%E	scientific format(upper or lower case 'e')
%f	fixed point decimal format
%g	the more compact of %e, %f

Control Codes:

\n	new line
\t	tab

Format codes may include a field width and the number of decimal places required:

%5d	display value as an integer in a field 5 characters wide
%10.6f	display value with 6 decimal places in a field 10 characters wide

See text p 48



**Problem:** Aircraft are typically equipped with a static port and a Pitot tube. The static port measures the static pressure ( $P$ ) of the airflow past the aircraft and the Pitot tube measures the stagnation pressure ( $P_o$ ) of this flow. If an aircraft's speed (expressed as a Mach number) is known, the ratio of the two pressures ( $P_o/P$ ) can be computed:

$$\text{for } M \leq 1: \quad PR(M) = (1 + 0.2M^2)^{3.5}$$

$$\text{for } M \geq 1: \quad PR(M) = (1.2M^2)^{3.5} \left( \frac{7}{6}M^2 - \frac{1}{6} \right)^{-2.5}$$

where  $M$  is the aircraft's Mach number

Typically the situation is the other way around. The pressure ratio is known (from the static port and Pitot tube measurements) and the Mach number must be determined.

This is another root finding problem. We need to find  $M$  such that

$$PR(M) - \text{observed pressure ratio} = 0$$

Function *PR* requires a “function m-file”.

```
function [ ratio ] = PR( M )
%PR  Given a Mach number, returns the corresponding pressure ratio
% Inputs: M = Mach number
% Outputs: ratio = pressure ratio (stagnation pressure / static pressure)

if M <= 1
    ratio = (1 + 0.2 * M^2)^3.5;
else
    ratio = (1.2 * M^2)^3.5 * ((7/6)* M^2 - 1/6)^-2.5;
end

end
```

Such functions are created in the editor window and each function is stored in a file called *function.m*, where *function* is the name of the function (e.g. for this function, the file is *PR.m*).

Two kinds of m-files:

- script files : contain “canned” instructions
- function files: contain a function (file name = function name)

Two kinds of functions (at least for now):

- “anonymous” functions: defined by a single command
- file functions: generally more complex, defined within a file

File functions are comparable to functions in C++ (and other languages)

- self-contained units
- do NOT share variables with the command window
- accept inputs and produce outputs

The initial comments document the function:

```
>> lookfor mach
```

TargetsComms\_Machine - class to hold information about a machine

PR - Given a Mach number, returns the corresponding pressure ratio

PRv - Given a Mach number, returns the corresponding pressure ratio

sfoopen - Opens a new machine.

sfsave - Saves a machine.

wmachdep - Machine dependent values.

```
>> help PR
```

PR Given a Mach number, returns the corresponding pressure ratio

Inputs: M = mach number

Outputs: ratio = pressure ratio (stagnation pressure / static pressure)

The *lookfor* command looks for functions having the specified keyword in their initial comment line.

The *help* command outputs all of a function's initial comments.

The Matlab “if” is logically equivalent to a C++ “if”. Only the details differ.

```
if M <= 1      % round brackets around expression allowed but not required
    ratio = (1 + 0.2 * M^2)^3.5;
else
    ratio = (1.2 * M^2)^3.5 * ((7/6)* M^2 - 1/6)^-2.5;
end
```

### **Relational and logical operators:**

==	is equal to
~=	is not equal to ** different from C++ **
<	is less than
>	is greater than
<=	is less than or equal to
>=	is greater than or equal to
~	NOT ** different from C++ **
&&	AND note: & is array version
	OR note:   is array version

The script file below plots the pressure ratio for Mach numbers from 0 to 7 (world record: the X-15 rocket plane achieved Mach 6.72), reads in a pressure ratio, and outputs the corresponding Mach number.

```
figure (1);  
fplot (@PR, [0, 7]);  
xlabel ('Mach Number');  
ylabel ('Pressure Ratio');  
grid on;  
  
ratio = input ('Enter pressure ratio: ');  
  
% define function for root finding  
f = @(M) PR(M) - ratio;  
  
M = fzero (f, [0 7]);  
  
fprintf ('The mach number is %f\n', M);
```

Note: When a file function is used as an input to another function, the name of the function must be preceded with a '@'.

Function PR is not vector friendly. Unlike most Matlab functions, it will not work properly if it is given a vector of inputs values.

In order to convert a vector of inputs into a vector of outputs it is necessary to use a loop:

```
M = 0 : 0.1 : 7; % from 0 to 7 in steps of 0.1
```

```
for k = 1 : length(M) % from 1 to length(M) in steps of 1  
    ratio(k) = PR(M(k));  
end
```

```
figure (1)  
plot (M, ratio);  
xlabel ('Mach Number');  
ylabel ('Pressure Ratio');  
grid on;
```

Vectors grow automatically as new elements are assigned values.

The function can be made to handle vectors:

```
function [ ratio ] = PRv( M )
%PRv  Given a Mach number, returns the corresponding pressure ratio
%    Vector friendly version of PR
% Inputs: M = Mach number (may be a vector of Mach numbers)
% Outputs: ratio = pressure ratio (stagnation pressure / static pressure)

ratio = ones(size(M)); % preallocate for efficiency
for k = 1:length(M)
    if M(k) <= 1
        ratio(k) = (1 + 0.2 * M(k)^2)^3.5;
    else
        ratio(k) = (1.2 * M(k)^2)^3.5 * ((7/6)* M(k)^2 - 1/6)^-2.5;
    end
end

end
```

Initializing a vector to its final size (preallocation) improves efficiency.



## Ways of creating vectors:

```
>> v1 = linspace (2, 6, 5)    % 5 values between 2 and 6
```

```
v1 =
```

```
    2    3    4    5    6
```

```
>> v2 = 2 : 6                % steps of size 1 are the default
```

```
v2 =
```

```
    2    3    4    5    6
```

```
>> v3 = 2 : 1 : 6
```

```
v3 =
```

```
    2    3    4    5    6
```

```
>> v4 = [2 3 4 5 6]          % values may be separated with commas
```

```
v4 =
```

```
    2    3    4    5    6
```

```
>> v5 = ones([1 5])
```

```
v5 =
```

```
    1    1    1    1    1
```

```
>> v6 = zeros([1 5])
```

```
v6 =
```

```
    0    0    0    0    0
```

```
>> v7 = ones(size(v4))      % size returns a two element vector (# rows, # cols)
```

```
v7 =
```

```
    1    1    1    1    1
```

Matlab also has while loops (just like C++).

The script below uses an infinite loop to process pressure ratios until a ratio that is less than 1 is entered.

```
while true % true is 1, false is 0

    ratio = input ('Enter pressure ratio: ');

    if ratio < 1; break; end % could be return instead of break

    % define function for root finding
    f = @(M) PR(M) - ratio;

    M = fzero (f, [0 7]);

    fprintf ('The mach number is %f\n', M);

end
```