Problem: A room is 4m longer than it is wide. The area of the room is $20m^2$. What is the width of the room?

Let x be the width of the room.

Then the area of the room is $x(x + 4) = x^2 + 4x$

Equating this to the given area gives $x^2 + 4x = 20$

Rearranging gives $x^2 + 4x – 20 = 0$

The problem is a "root finding" problem.

Root Finding Problems:

- General form: find x such that $f(x) = 0$
- The values of x for which $f(x) = 0$ are the *roots* of $f(x)$

For our problem $f(x)$ happens to be a *quadratic*.

The roots can be found using the quadratic formula.

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

$$roots = x_1, x_2 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

In general there are two roots.

One is obtained by using + in the formula and the other by using -.

If the quantity under the square root is zero the roots are equal.

If this quantity is negative the roots are complex numbers.

Casio Calculator Note

Casio calculators can solve quadratics.

Hit *Mode* until menu include “EQN”. Then select this option.

Use the right arrow to move from “Unknowns?” to “Degree?”

Enter 2 (a quadratic is a second degree polynomial)

Enter values for *a*, *b*, and *c* (hit “=” after each value)

Use the up and down arrows to move between the two solutions.

If the roots are complex numbers *shift* plus “=” toggles between the real and imaginary parts of each solution. If the roots are real this key combination has no effect. The display does not indicate the roots are real or complex (use *shift* plus “=” to find out).

At its simplest Matlab is just a high priced calculator

Typing an expression after the command prompt causes Matlab to output the value of the expression.

```
>> 1+1
ans =
    2
>> 2^3
ans =
    8
>> 2*3+4
ans =
   10
>> 2*(3+4)
ans =
   14
```

Points to remember: ^ is exponentiation
The usual rules of precedence apply

Assigning to a name creates a memory (variable) and gives it a value.

```
>> x = 67
x =
    67
```

Output can be suppressed by ending commands with a semi-colon.

```
>> y = 50;
```

In general spaces don’t matter, but there is one exception...

```
>> y/2
ans =
    25
>> y / 2
ans =
    25
>> y /2
??? Error: "y" was previously used as a variable,
 conflicting with its use here as the name of a function or
command.
```

Point to remember: The space issue can’t arise in assignments.

Our Problem In Matlab

```
>> a = 1;
>> b = 4;
>> c = -20;

>> disc = b^2 - 4 * a * c;

>> x1 = (-b + sqrt(disc)) / (2 * a)  % no semi-colon to see result
x1 =
    2.8990

>> x2 = (-b - sqrt(disc)) / (2 * a)
x2 =
   -6.8990
```

For our problem the root that matters (the one with physical meaning) is clearly 2.8990.

Point to remember: ‘%’ starts a comment (except within strings)

Some Matlab Basics:

clc – clear command window

who – list all variables

whos – list all variables with sizes and other information

clear – delete all variables

clear var1 var2 … – delete selected variables

help command – get help with command (also works with operators and functions)

format compact – single space output
format loose – double space output

up and down arrows – allow recall of previous commands

Point to remember: in Matlab scalars are treated as 1x1 arrays

Matlab Function "roots": The roots of polynomials can be quickly found using function *roots*. It expects a array containing the coefficients of the polynomial (highest order coefficient first) and the result is also an array.

```
>> x = roots([a b c])    % values in square brackets creates an array
x =
   -6.8990
    2.8990

>> p = [a b c];
>> x = roots(p);
>> x(1)                  % x(1) is the first element of the result
ans =
   -6.8990
>> x(2)
ans =
    2.8990
```

Points to remember:

- [v1 v2 v3 …. vn] is a array
- Array indices start at 1 (not at zero as in C++)
- A(n) selects the nth element of array A

Matlab Function "fzero"

Function *fzero* is a general root finding function.

In order to use it the function of interest must be defined:

```
>> f = @(x) a*x^2 + b*x + c;
```

fzero must be given either a range of interest or a ballpark value

```
>> x = fzero(f, [0 4])   % find root between 0 and 4
x =
   2.8990

>> x = fzero(f, 4)       % find root in vicinity of 4
x =
   2.8990

>> x = fzero(f, -4)      % find root in vicinity of -4
x =
  -6.8990
```

Function definition command

```
>> f = @(x) a*x^2 + b*x + c;
```

Function definition.

Function arguments (inputs).
Does not have to be *x*.
Multiple inputs are possible.

Name by which the
function will be known.
This does not have to be *f*.

The argument names serve only to identify arguments within the definition. They have no other significance (variables are not created and any existing variables having the same names are not affected).

When variables (e.g. *a*, *b*, *c*) are used in a function definition the function is based on the current values of these variables. If the variables are subsequently changed the function does NOT change.

See p66 of Chapra (2nd ed), p74 of Chapra (3rd ed), or p470 of Gilat

Creating a Plot

Suppose that we would like a plot showing the area of the room vs. its width.

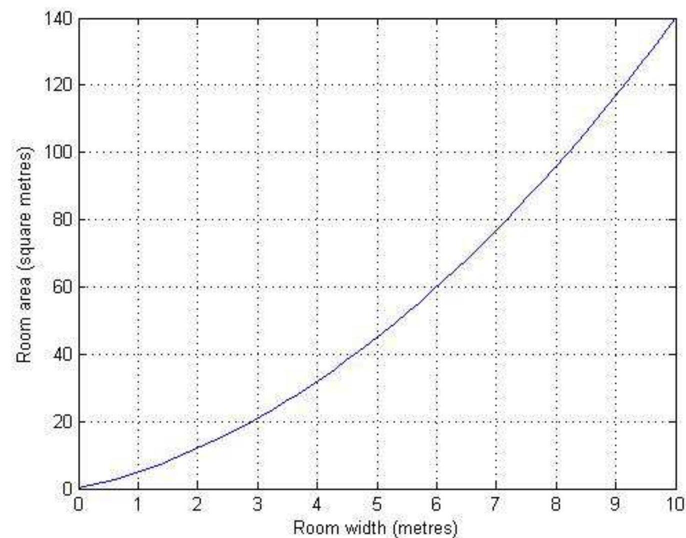
```
>> figure(1);  % the plot will be "Fig 1"
>> width = linspace(0, 10, 50);
>> area = width .* (width + 4);
>> plot (width, area);
>> xlabel ('Room width (metres)');
>> ylabel ('Room area (square metres)');
>> grid on;
```

width is a vector containing 50 equally spaced values between 0 and 10

area is also a 50 element vector. It is created by performing operations upon vector *width*. The dot indicates "element by element" multiplication.

The *plot* command creates an *x-y* plot using the values in *width* as the *x* values and the values in *area* as the *y* values.

Point to remember: linspace(start, end, n) creates an array containing *n* evenly spaced values starting at *start* and ending at *end*.



The example below illustrates the difference between regular matrix multiplication (assumed) and element by element multiplication (specified by using the “dot” form of the operator).

```
>> a = [ 1 2 ; 3 4]
a =
    1    2
    3    4
>> b = [ 5 6 ; 1 2]
b =
    5    6
    1    2

>> a*b                      % Matrix multiplication
ans =
    7   10
   19   26

>> a.*b                     % Element by element multiplication
ans =
    5   12
    3    8
```

op	Array **op** Scalar	Scalar **op** Array	Array **op** Array
+, -	**Dot not permitted** [1 2 3] + 4 = [5 6 7]	**Dot not permitted** 9 – [1 2 3] = [8 7 6]	**Dot not permitted** [1 2 3] + [4 5 6] = [5 7 9]
*	**Dot makes no difference** [1 2 3] * 2 = [2 4 6] [1 2 3] .* 2 = [2 4 6]	**Dot makes no difference** 2 * [1 2 3] = [2 4 6] 2 .* [1 2 3] = [2 4 6]	**No dot: array operation** **Dot: element by element** [2 3 4].*[1 2 3] = [2 6 12]
/	**Dot makes no difference** [2 4 6] / 2 = [1 2 3] [2 4 6] ./ 2 = [1 2 3]	**No dot: illegal (1)** **Dot: see below** 8 ./ [1 2 4] = [8 4 2]	**No dot: array operation** **Dot: element by element** [2 6 12] ./ [1 2 3] = [2 3 4]
^	**No dot: array operation (2)** **Dot: see below** [1 2 3] .^ 2 = [1 4 9]	**No dot: array operation (3)** **Dot: see below** 2 .^ [1 2 3] = [2 4 8]	**No dot: illegal** **Dot: element by element** [2 3 4].^[1 2 3] = [2 9 64]

(1) Array operation and dimensions guaranteed to be invalid.
(2) The array (which must be square) is raised to the specified power.
(3) S^A is (e^A)^S, where e^A is the matrix exponential of A. Don’t worry about this.

Points to remember:

- Never use a dot with the + and - operators
- Unless you actually want array multiplication, division, or exponentiation, you can always use a dot with the *, / , and ^ operators

More Plotting

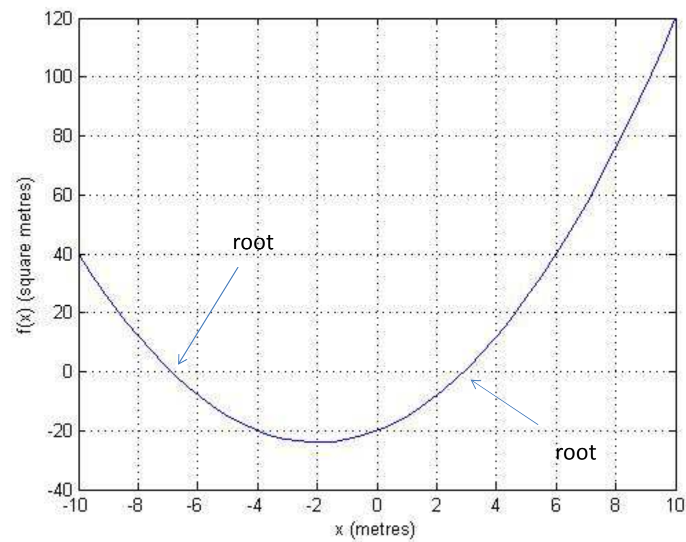
We might also like to plot f(x) (perhaps to get some idea where the roots are).

A good first step is to redefine our function so that it works properly with vector inputs (so that vector inputs produce vector outputs).

```
>> f = @(x) a*x.^2 + b*x + c; % redefine using dot version of exponentiation
```

Once this is done the function can be used to produce *y* values.

```
>> figure(2);
>> x = linspace(-10, 10, 50);
>> y = f(x);
>> plot (x, y);
>> xlabel ('x (metres)');
>> ylabel ('f(x) (square metres)');
>> grid on;
```



Yet More Plotting

Function *fplot* makes plotting a function even easier.

```
>> figure(3);
>> fplot (f, [-10 10]);  % plot function “f” for “x” from -10 to 10
>> xlabel ('x (metres)');
>> ylabel ('f(x) (square metres)');
>> grid on;
```

fplot looks after selecting the *x* values used in the plotting process. It uses closely spaced values where the function is changing rapidly and places values further apart where the function is changing relatively slowly.