Problem: A room is 4m longer than it is wide. The area of the room is $20m^2$. What is the area 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 $x^2 + 4x - 20 = 0$

This 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.

In Matlab

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

>> disc = b^2 - 4 * a * c;
>> x1 = (-b + sqrt(disc)) / (2 * a)

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.

Matlab Function "roots"

The roots of polynomials can be quickly found using function *roots*.

```
>> x = roots([a b c])
x =
   -6.8990
    2.8990

>> p = [a b c];
>> x = roots(p);
>> x(1)
ans =
   -6.8990
>> x(2)
ans =
    2.8990
```

The function expects a vector containing the coefficients of the polynomial (highest order coefficient first).

The result is also a vector.

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 variable. If the variables are subsequently changed the function does NOT change.

Creating a Plot

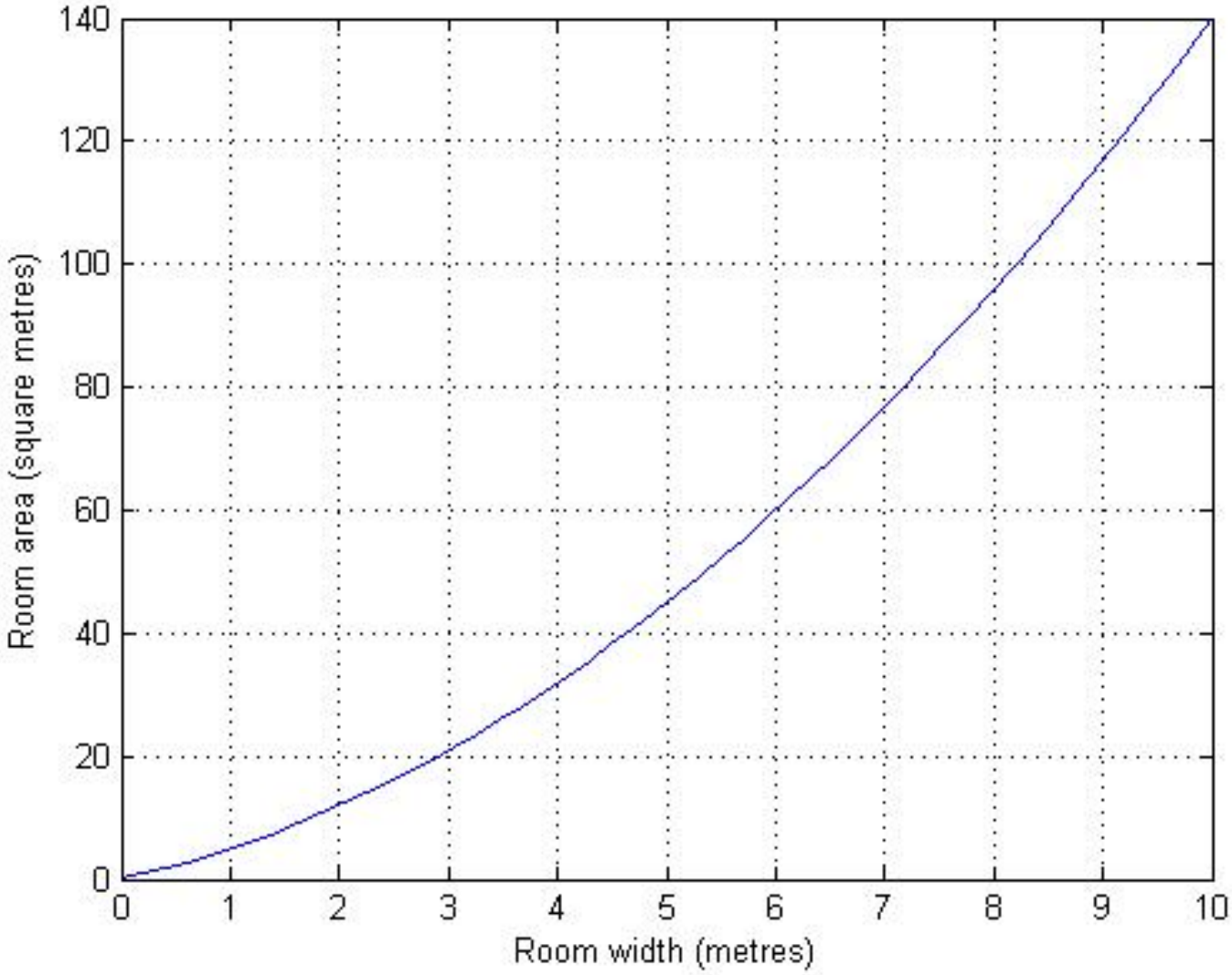
Suppose that we would like a plot showing the area of the room vs its width.

```
>> 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*. “element by element” multiplication (note the dot) is essential.

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.


Room area (square metres)
Room width (metres)
0
20
40
60
80
100
120
140
0
1
2
3
4
5
6
7
8
9
10

More Plotting

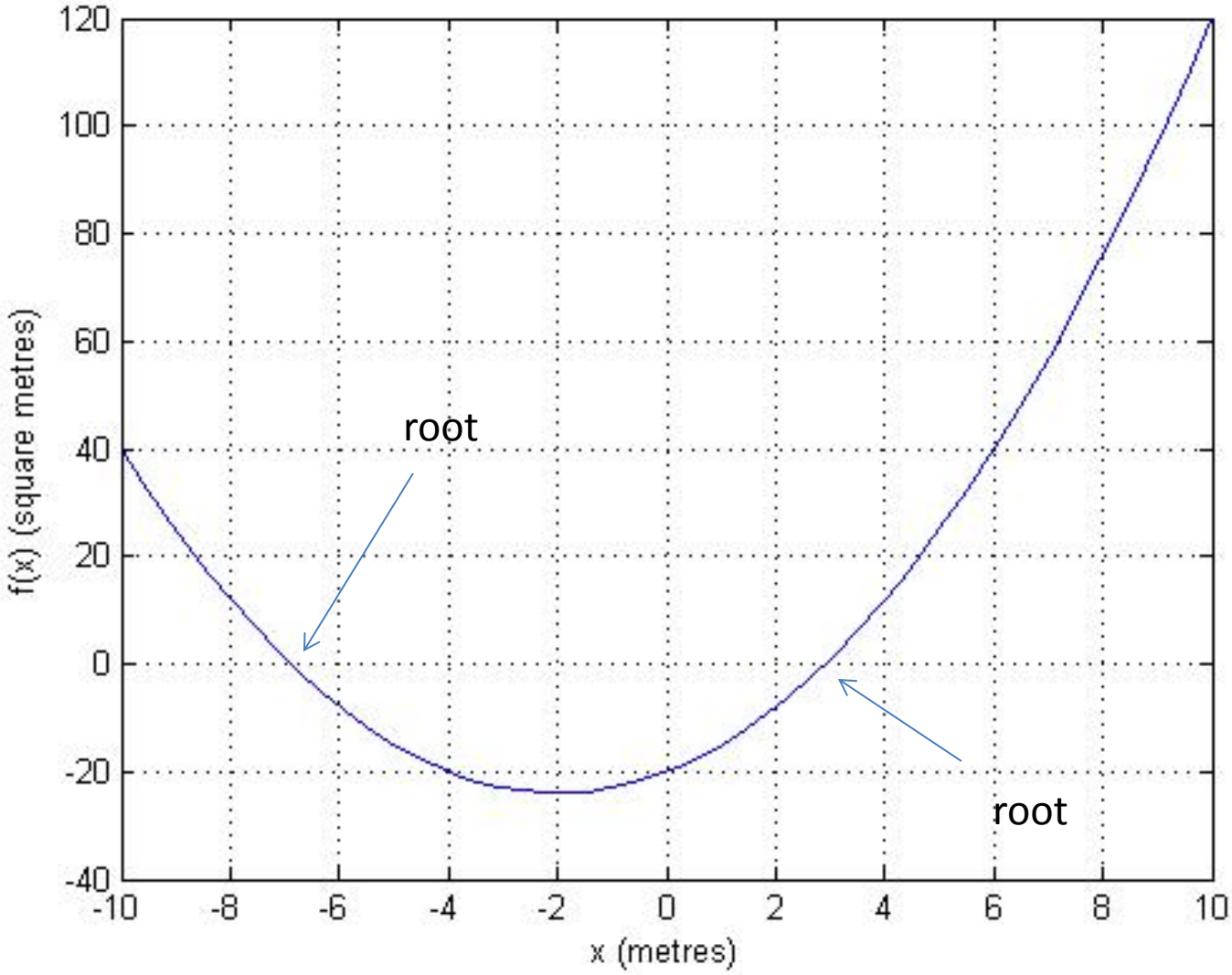
We might also like to plot f(x) (perhaps to get some idea where the roots are).

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;
```


120
100
80
60
40
20
0
-20
-40
-10
-8
-6
-4
-2
0
2
4
6
8
10
f(x) (square metres)
x (metres)
root
root

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.