

Function inputs may be functions. If *fplot* was not part of Matlab, we could easily create something very similar.

```
function [ ] = DIYfplot( f, range )
%DIYfplot Do it yourself version of fplot. Plots a function over a range.
% Inputs: f = function to be plotted
%         range = two element vector containing start and end of range
% No Outputs (like a C++ void function)

n = 100; % number of points

x = linspace (range(1), range(2), n); % generate x values

% generate y values
y = zeros ([1 n]); % preallocate for efficiency
for k = 1 : n
    y(k) = f(x(k));
end

plot (x, y);

end
```

The function provided may be a function m-file (see DIYplot.m) (@ required before name);

```
DIYfplot( @PR, [0 7])
```

```
>> grid on
```

```
>> xlabel ('Mach number');
```

```
>> ylabel ('Pressure Ratio');
```

Built in functions are also acceptable (@ required before name):

```
>> DIYfplot( @sin, [0 4*pi])
```

And so are anonymous functions (no @ required – the @ is in the definition).

```
>> f = @(x) x^3 + 2* x^2 + 7;
```

```
>> DIYfplot( f, [-5 5])
```

Note: We could also create our own version of *fzero* and will soon be doing exactly this.

Another potentially useful function is shown below. It produces a table of function values (see ftable.m)

```
function [ ] = ftable( f, vector )
%ftable Produces table of function values.
% Inputs: f = function to be tabulated
%         vector = vector of values for the independent variable
% No Outputs (like a C++ void function)

fprintf (' Independent      Dependent\n');

for k = 1 : length(vector)
    x = vector(k);
    y = f(x);
    fprintf ('%12f%16f\n', x, y);
end

end
```

Sample usage:

```
>> ftable (@PR, 1:0.5:6)
```

Independent	Dependent
1.000000	1.892929
1.500000	3.413275
2.000000	5.640441
2.500000	8.526136
3.000000	12.060965
3.500000	16.242001
4.000000	21.068081
4.500000	26.538665
5.000000	32.653474
5.500000	39.412352
6.000000	46.815206

Improved Loop:

The loop in the function can be reduced to:

```
for x = vector  
    fprintf ('%12f%16f\n', x, f(x));  
end
```

The table below shows the situations in which a “dot” must be used to avoid having Matlab assume array arithmetic (e.g. array multiplication) . The “dot” tells Matlab to perform an element by element operation instead. **Note:** there is no “.+” or “.-”

op	Array op Scalar	Scalar op Array	Array op Array
+, -	OK [1 2 3] + 4 = [5 6 7]	OK 9 – [1 2 3] = [8 7 6]	Array operation assumed Sizes must match
*	OK [1 2 3] * 2 = [2 4 6]	OK 2 * [1 2 3] = [2 4 6]	Array operation assumed Sizes must be compatible Dot required Sizes must match. [2 3 4].*[1 2 3] = [2 6 12]
/	OK [2 4 6] / 2 = [1 2 3]	Array operation assumed ERROR (as sizes are incompatible) Dot required 8 ./ [1 2 4] = [8 4 2]	Array operation assumed Sizes must be compatible. Dot required Sizes must match. [2 6 12] ./ [1 2 3] = [2 3 4]
^	Array operation assumed Square array required Dot required [1 2 3] .^ 2 = [1 4 9]	Array operation assumed Square array required Dot required 2 .^ [1 2 3] = [2 4 8]	ERROR Dot required Sizes must match. [2 3 4].^[1 2 3] = [2 9 64]

To evaluate $B(t) = \frac{250}{1 + 56.75e^{-0.17t}}$ for a vector of t values:

```
>> t = [ 0 10 20 ];
```

```
>> -0.017 * t
```

```
ans =
```

```
0 -0.1700 -0.3400
```

```
>> exp(ans)
```

```
ans =
```

```
1.0000 0.8437 0.7118
```

```
>> 56.75*ans
```

```
ans =
```

```
56.7500 47.8780 40.3930
```

```
>> 1+ans
```

```
ans =
```

```
57.7500 48.8780 41.3930
```

```
>> 250/ans % a scalar divided by a vector is a no-no (see table)
```

```
??? Error using ==> mrdivide
```

```
Matrix dimensions must agree.
```

```
>> 250./ans
```

```
ans =
```

```
4.3290 5.1148 6.0397
```

```
>> B = @(t) 250./(1 + 56.75*exp(-0.17*t));
```

Problem (text 5.14): You buy a \$25,000 piece of equipment for nothing down at \$5,500 per year for 6 years. What interest rate are you paying?

The formula below relates the payment amount (A) to the present worth of the item (P), the number of years (n), and the interest rate (i , expressed as a fraction, $1\% = 0.01$)

$$A = P \frac{i(1+i)^n}{(1+i)^n - 1}$$

If we knew P , n , and i we could easily calculate A , but this isn't the problem. We know A , P , and n and want to calculate i .

Ideally we'd manipulate the equation to obtain an expression for i in terms of A , P , and n (remember that analytic solutions are best when we can obtain them).

$$i = \text{somefunction}(A, P, n)$$

Since we can't do this easily we will convert the problem into root finding form

$$P \frac{i(1+i)^n}{(1+i)^n - 1} - A = 0$$

and use numerical methods (e.g. *fzero*) to find the value of i that satisfies the equation.

Step 1: Define the basic function.

```
>> P = 25000; n = 6;  
>> calcA = @(i) P * (i .* (i + 1).^n) ./ ((i + 1).^n - 1);
```

Step 2: Plot the function to get some idea of its behaviour and the approximate location of the solution.

```
>> fplot (calcA, [0.01 0.20]) % plot for interest rates from 1% to 20%
```

Step 3: Define the function for root finding (the f in $f(x) = 0$).

```
>> A = 5500;  
>> f = @(i) calcA(i) - A;
```

Step 4: Use *fzero* to locate the desired root. In this case the plot tells us that the answer is somewhere between 6% and 10%.

```
>> i = fzero(f, [0.06 0.10])  
i = 0.0856
```

```
>> calcA(i) % check that answer is correct (always a good idea)  
ans = 5.5000e+003
```


The basic idea can be used to produce a useful function m-file that calculate the interest rate for any given P , A , and n .

```
function [ i ] = P5_14( P, A, n )
%P5_14 Calculate interest rate.
% NOTE: Assumes that interest rate is >= 0.005.
% Inputs: P = present value of item
%         A = annual payment
%         n - number of years
% Outputs: i = interest rate (as a fraction)

f = @(i) ( P * ( i .* ( i + 1 ).^n ) ./ (( i + 1 ).^n - 1) ) - A;

maxi = A/P; % if i is very large, eq'n becomes A=Pi, hence i=A/P

i = fzero(f, [ 0.005 maxi ] );

end
```

The search for the root cannot be made to start at zero because f is undefined at $i = 0$ (as the formula for calculating A does not work for $i = 0$).

This difficulty can be overcome by producing a *calcA* function that properly deals with the special case of $i = 0$. This function could be implemented in a separate m-file but it is also possible to place it in the same m-file as the basic function as shown below. This makes it a *subfunction* (interested students should read text s.3.1.3/App. A).

```
function [ i ] = P5_14v2( P, A, n )
%P5_14v2 Calculates interest rate.
% Inputs and outputs as before...

f = @(i) calcA (P, n, i) - A;
maxi = A/P;
i = fzero(f, [ 0 maxi ] );

end

function [A] = calcA (P, n, i)
if i == 0
    A = P / n;
else
    A = P * (i .* (i + 1).^n) ./ ((i + 1).^n - 1);
end

end
```

Problem: Find all of the points that satisfy both $y = x^2 - 17x + 60$ and $y = 50\sin(x/2)$ (i.e. find all of the intersections of the curves defined by these equations).

Step 1: Plot the two curves.

```
>> f1 = @(x) 50 * sin(0.5 * x);  
>> f2 = @(x) x.^2 - 17 * x + 60;  
  
>> x = linspace (0, 20, 100);  
>> y1 = f1(x); y2 = f2(x);  
  
>> plot (x, y1, 'r', x, y2, 'b')  
>> grid on
```

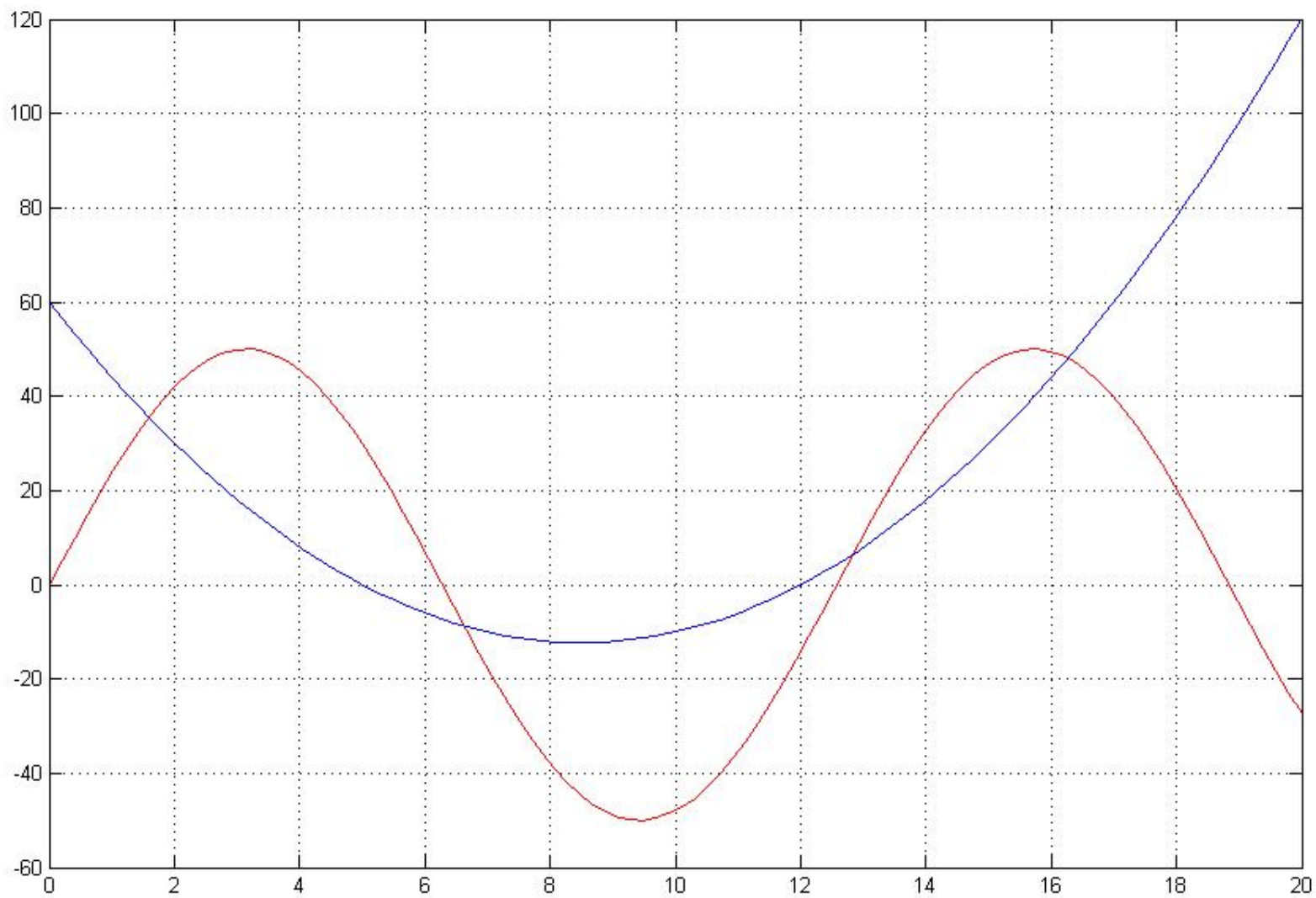
The plot function will accept more than one pair of x and y vectors. This allows multiple plots to be placed on the same graph. Each pair of vectors may be followed by a string containing plotting options. Some of the permitted characters are:

colour: 'r' = red, 'b' = blue, 'k' = black, 'g' = green

line style: '-' = solid, ':' = dotted, '--' = dashed, '-.' = dash dot

data point markers: 'x' = crosses, 'o' = circles

Options may be combined (e.g. 'r:x' gives a dotted red plot with a crosses)



Note: The hold command is another way of placing multiple plots on the same graph.

```
>> plot (x, y1, 'r')  
>> hold on  
>> plot (x, y2, 'b')  
>> hold off          % be careful - hold for a figure stays in effect until it is turned off  
>> grid on
```

Step 2: Define the function for root finding (and perhaps plot it)

```
>> f = @(x) f1(x) - f2(x);  
  
>> figure (2)  
>> fplot (f, [0 20])  
>> grid on;
```

Step 3: Find the roots:

```
>> x1 = fzero(f, [0 2]);  
>> x2 = fzero(f, [6 8]);  
>> x3 = fzero(f, [12 14]);  
>> x4 = fzero(f, [16 18]);
```