roots of equations aka zeros of functions

Finding a **root** r of the equation $f(?) = 0$ or, equivalently, finding a **zero** r of the function f involves three aspects:
(i) *where* to look;
(ii) *what* to look *for*;
(iii) *how* to look.

where to look: typically, one is given some interval $[a\,..\,b]$ in which to find a zero. Different choices of $[a\,..\,b]$ may well lead to different roots even if one interval is contained in the other. If the specified interval is *too large*, one may never find a zero.

what to look for: This seems to be a stupid question; one is, after all, looking for a zero, i.e., some r for which $f(r) = 0$. But, consider: The equation $\sin(?) = 0$ does have exactly one root in the interval $[\pi/2\,..\,3\pi/2]$, namely $r = \pi$. However, none of the typical floatingpoint numbers in a machine are exactly equal to π. Correspondingly, there may be *no* x in our machine for which $\sin(x) = 0$ (exactly). An even simpler example is provided by the equation $(?)^2 = 2$ whose sole positive root is the number $\sqrt{2} = 1.4142\ldots$ which is not one of the floatingpoint numbers available in one's machine.

In addition, there is the question of noise. Typically, if a function **vanishes** (i.e., has a zero) at r, then the formula for that function is likely to involve a sum of terms which, at r, add up to zero. This can only happen through loss of significance. It means that, very close to r, the computed function values are essentially just noise, as the figure on page 45 of the book which shows values near its only zero, $x = 1$, of the polynomial

$$p(x) := 1 - 6x + 15x^2 - 20x^3 + 15x^4 - 6x^5 + x^6$$

evaluated in single-precision (i.e., carrying about 7 decimal digits), using the nested form to keep down the noise (which doesn't help much in this case).

From this plot, any number z in the interval $[(.9)\,..\,(1.1)]$ could be judged a 'zero' for the function $f = p$, by either one of the two **standard criteria** for z to be a 'zero':

- $|f(z)|$ is no bigger than the noise in the computed function values.
- There are points $a \le z \le b$ close to z for which $f(a)f(b) \le 0$, hence, assuming that f is continuous, f must have a zero, r, in this interval, hence our z is close to that zero in the sense that the resulting bound $|r - z| \le |a - b|/2$ is small.

how to look for a zero: There are two basic ideas for this.

first idea: bracketing the zero If we know that f is continuous on the interval $[a\,..\,b]$ and that $f(a)f(b) \le 0$, then, by the *Intermediate-Value Theorem*, we know that $f(r) = 0$ for some $r \in [a\,..\,b]$. In particular, we know that the *midpoint*, $c := (a + b)/2$ is within $|a - b|/2$ of any such zero. This leads to the bisection method and its more sophisticated variants, with the c just computed the initial guess, x_0, for the zero.

The basic method along these lines is **bisection**.

second idea: approximating the function We have already some good approximation, x_n say, and know a straight line which is a 'good' approximation to f near x_n. Then we compute exactly the zero of that straight line, and that becomes our next approximation, x_{n+1}. Newton's method is the best-known method in this class.

termination criteria: All root-finding methods are iterative; one repeats the same steps "until satisfied". This can mean several things. With c our most recent guess at the root, r, one might stop if (i) $|f(c)| <$ `Tolf` or if $|c - r| <$ `Tolx` (this second criterion requires some way of knowing how close one is to r; this is something one knows with root bracketing, but may not know with Newton's method or the Secant method). In addition, one should also terminate if the number of iteration steps exceeds a certain bound, either built-in or explicitly specified.

comparison of methods

Bisection is reliable (one always brackets a root (unless noise becomes overwhelming)), but slow (one gains one binary digit of accuracy with every step, i.e., per one function evaluation).

Newton is much faster, but not at all reliable, unless one starts *sufficiently close* to a zero. Difficulties: (i) **shot in the dark**, meaning that the straight line (tangent) is relatively flat, so its zero is far away from the current guess. (ii) **cycling** example: for the function $f(x) = x^3 - x$, the guess $a = 1/\sqrt{5}$ for its zero 0 gives the next guess $b = -a$, hence the next guess after that is a again.

The methods actually used in good software are **hybrid**, combining the safety of bisection with the speed of secant. One such method is the **Modified Regula Falsi** aka **Illinois method**: At every step, one has in hand an interval with endpoints a and b, of which b is the most recently obtained approximation to the root, and f changes sign on that interval, i.e., $f(a)f(b) \le 0$. One computes c as the unique point at which the straight line through the points (a, fa) and (b, fb) crosses the x-axis and, correspondingly, $fc := f(c)$.

I also discussed `matlab`'s general-pupose root-finding command `fzero`.

comment concerning functions in `matlab`: Rootfinding commands, like `fzero` or your very own `bisect` and `secant`, must somehow be told about the function f whose zero(s) they are supposed to find. This is done by using the *name* of that function as an input argument.

If, e.g., the function is $f(x) = \sin(x)$, then its appropriate name here is the *string* `'sin'`.

If you have made up a function m-file called `myfun.m` so that the `matlab` command `v = myfun(x);` provides the value of f at the point `x`, then the appropriate name to use is the string `'myfun'`. E.g., for the function $f(x) = x^2 - 3x + 1$, you might write the function m-file having just the two lines

```
function v = myfun(x)
v = (x - 3)*x + 1;
```

If the formula for $f(x)$ is pretty simple, you might want to make up an **inline function**. E.g., if $f(x) = x^2 - 3x + 1$, the command

```
f = inline('(x-3)*x+1');
```

provides, in `f`, a name that you can pass on to your favorite rootfinding command. (I believe that inline functions were not available prior to version 5 of `Matlab`.)

How is this name actually used inside the rootfinding m-file? If `f` is the name, then

```
v = feval(f,x);
```

provides the value of that function at the point `x`.

As you make up functions, be sure to enable the function to deal with a whole vector or even matrix `x`, as do the built-in elementary functions, like `sin`, in order to avoid using loops. E.g., in the above example, the multiplication should be *pointwise*, meaning that the formula should be `(x-3).*x + 1` .