Classic root-finding algorithms

We have looked at how fixed-point iterations (iterated maps) can be used to locate a solution of the scalar equation $f(x) = 0$. Here are two other methods.

Bisection method

This method is based on the Intermediate Value Theorem of calculus:

> Suppose that the function f is continuous on the interval $[a, b]$. If $f(a)$ and $f(b)$ are of opposite sign, then there exists a number x^* in (a, b) with $f(x^*) = 0$.

To find the root x^* of f, we can look for successively smaller intervals that contain x^*. The idea is to:

- first choose an interval $[l_1, r_1]$ such that $f(l_1)$ and $f(r_1)$ are of opposite signs
- chop the interval in half at the midpoint $m_1 = \frac{1}{2}(l_1 + r_1)$ to make two intervals $[l_1, m_1]$ and $[m_1, r_1]$
- check the sign of $f(m_1)$:
  - if $f(m_1) = 0$, then $x^* = m_1$
  - if $f(l_1)$ and $f(m_1)$ have the same sign, then x^* must lie between m_1 and r_1, so set $l_2 = m_1$ (i.e. change the left boundary) and $r_2 = r_1$
  - if $f(l_1)$ and $f(m_1)$ have opposite signs, then x^* must lie between l_1 and m_1, so set $l_2 = l_1$ and $r_2 = m_1$ (i.e. change the right boundary)

  This gives a new interval $[l_2, r_2]$ which is half the length of $[l_1, r_1]$.
- repeat the process for the interval $[l_2, r_2]$ to get another interval $[l_3, r_3]$ that is half the length of $[l_2, r_2]$
- iterate this procedure to generate a sequence of intervals $[l_k, r_k]$, each of which contains the root x^* and is half the length of the previous one.

The length of the interval $[l_k, r_k]$ will be $\dfrac{1}{2^{k-1}}$ times the length of $[l_1, r_1]$.
If we consider the sequence $\{m_k\}$ of midpoint values as our approximations to x^*, then

$$\text{absolute error} = |x^* - m_k| \le \frac{1}{2}\,|r_k - l_k| = \frac{1}{2} \cdot \frac{1}{2^{k-1}}\,|r_1 - l_1| = \frac{1}{2^k}\,|r_1 - l_1|$$

If we consider the sequence $\{l_k\}$ of left endpoints as our approximations to x^*, then

$$\text{absolute error} = |x^* - l_k| \le |r_k - l_k| = \frac{1}{2^{k-1}}\,|r_1 - l_1|$$

and similarly for the right endpoints $\{r_k\}$.

Now let us write Matlab code to implement the bisection method.
We will use it to approximate:

(i) $\sqrt{10}$

(ii) the solution of $x^3 + 4x^2 - 10 = 0$ in $[1, 2]$

(iii) the solution of $e^x = \tan x$ in $[-4, -2]$

with absolute error not exceeding 10^{-14}.

Newton's (or Newton–Raphson) method

This method is based on tangent-line approximations to the graph of $f(x)$.
Suppose that x_k is an approximation to the solution x^* of $f(x) = 0$.
The tangent line to the graph of f at the point $\big(x_k, f(x_k)\big)$ has equation

$$y - f(x_k) = f'(x_k)(x - x_k)$$

This line intersects the x-axis, i.e. $y = 0$, when $-f(x_k) = f'(x_k)(x - x_k)$, which gives

$$x = x_k - \frac{f(x_k)}{f'(x_k)}$$

We define this x-value to be the new approximation x_{k+1}. So we get the recursive formula

$$\boxed{x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}}$$

Starting from an initial guess x_1, this formula generates a sequence $x_2, x_3, x_4, \ldots$, which hopefully converges to the root x^*.

Note that Newton's method is in fact an iterated map, where the map ϕ is given by

$$\phi(x) = x - \frac{f(x)}{f'(x)}$$

Therefore Newton's method can fail to converge if the map ϕ is not sufficiently "well-behaved".
You can also see that Newton's method will fail if $f'(x_k) = 0$ for some term of the sequence.

Now let us write Matlab code to implement Newton's method.
We will use it to approximate:

(i) $\sqrt{10}$

(ii) the solution of $x^3 + 4x^2 - 10 = 0$ in $[1, 2]$

(ii) the solution of $e^x = \tan x$ in $[-4, -2]$

For the stopping criterion, we will choose $|f(x_k)| \leq 10^{-14}$

(but one of the other criteria $|x_n - x_{n-1}| \leq$ tolerance, $\left|1 - \frac{x_n}{x_{n-1}}\right| \leq$ tolerance or $\left|1 - \frac{x_{n-1}}{x_n}\right| \leq$ tolerance could be used just as well).