MA3227 Numerical Analysis II

Lecture 2: Nonlinear Equations

Simon Etter

National University of Singapore

Semester II, AY 2020/2021

Nonlinear Equations

Problem statement

$$\text{Given a continuous function } f : \mathbb{R}^n \to \mathbb{R}^n, \\ \text{find } x \in \mathbb{R}^n \text{ such that } f(x) = 0.$$

Examples

- $$ax^2 + bx + c = 0 \quad \Longleftrightarrow \quad x = \tfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$
- $$\begin{Bmatrix} x^2 - y^2 = 0 \\ 1 + xy = 0 \end{Bmatrix} \quad \Longleftrightarrow \quad \begin{Bmatrix} x = \pm 1, \\ y = \mp 1. \end{Bmatrix}$$

Terminology

- A point x such that $f(x) = 0$ is called a *zero* or *root* of f.
- Solving $f(x) = 0$ is also called *root-finding*.

Remark

Setting the right-hand side equal 0 does not reduce generality since any nonlinear equation $g(x) = h(x)$ can be rewritten as $g(x) - h(x) = 0$.

Nonlinear Equations

Applications

Real-world applications:

- Determine the aircon setting so you feel neither hot nor cold.
- Determine launch parameters (time, angle, acceleration, etc.) so your rocket reaches the moon rather than disappears into space.

Mathematical applications:

- Function inversion:

$$x = f^{-1}(y) \quad \iff \quad \text{find } x \quad \text{s.t.} \quad f(x) - y = 0.$$

- Optimisation:

$$x = \arg\min f(x) \quad \iff \quad \text{find } x \quad \text{s.t.} \quad \nabla f(x) = 0.$$

Nonlinear Equations

One equation vs. many equations

The mathematical properties and the algorithms for solving $f(x) = 0$ are quite different depending on whether $f : \mathbb{R}^n \to \mathbb{R}^n$ is a scalar function ($n = 1$) or a multi-dimensional function ($n > 1$).

Correspondingly, we will discuss these two cases separately, starting with the scalar case $f : \mathbb{R} \to \mathbb{R}$.

Problem statement in one dimension

$$\text{Given a continuous function } f : \mathbb{R} \to \mathbb{R},$$
$$\text{find } x \in \mathbb{R} \text{ such that } f(x) = 0.$$

Nonlinear Equations

Existence and uniqueness of solutions

It is clear that an arbitrary function $f : \mathbb{R} \to \mathbb{R}$ may have zero, one or many roots. At this point, we thus cannot interpret root-finding as a mapping

$$\text{root} : \big(\mathbb{R} \to \mathbb{R}\big) \;\to\; \mathbb{R}$$

but rather we need to think of it as a mapping

$$\text{roots} : \big(\mathbb{R} \to \mathbb{R}\big) \;\to\; \bigcup_{k=0}^{\infty} \mathbb{R}^k.$$

For the purpose of this lecture, I will assume that finding any root of $f(x)$ is good enough. This reduces root-finding to a mapping

$$\text{root} : \big(\mathbb{R} \to \mathbb{R}\big) \;\to\; \mathbb{R}^0 \cup \mathbb{R}^1$$

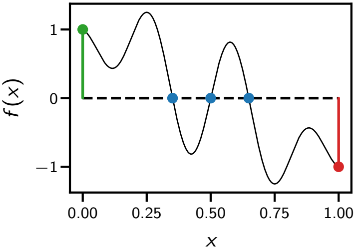
where $\text{root}(f)$ may assume one of several values if $f(x)$ has several roots. Handling the possibility that $\text{root}(f)$ may find no root is often a nuisance in practice. Fortunately, we can usually avoid it using the bracketing theorem presented next.

Nonlinear Equations

Bracketing theorem (also known as Bolzano's theorem)

If $f : [a, b] \to \mathbb{R}$ is continuous and $\text{sign}(f(a)) \neq \text{sign}(f(b))$, then $f(x)$ has at least one root in $[a, b]$.

Proof. Straightforward application of the intermediate value theorem.



The sign function $\text{sign}(x)$ is given by $\text{sign}(x) = \begin{cases} +1 & \text{if } x > 0, \\ 0 & \text{if } x = 0, \\ -1 & \text{if } x < 0. \end{cases}$

Nonlinear Equations

Discussion

The bracketing theorem motivates the following definition.

Def: Bracketing interval

An interval $[a, b]$ such that $\text{sign}(f(a)) \neq \text{sign}(f(b))$.

It further suggests that if we add a bracketing interval $[a, b]$ as an argument to the root-finding function, then we obtain a mapping

$$\text{root} : \big(\mathbb{R} \to \mathbb{R}, \text{bracketing interval}\big) \;\to\; \mathbb{R}$$

which is guaranteed to return a single number $\text{root}(f, [a, b]) \in \mathbb{R}$.

Finally, we can obtain an algorithm for evaluating this $\text{root}(f, [a, b])$ function by combining the bracketing theorem with a simple observation presented next.

Nonlinear Equations

The bisection idea

Assume we have a bracketing interval $[a, b]$ and we evaluate $f(x)$ at the midpoint $m = \frac{a+b}{2}$. Then, $\text{sign}\big(f(m)\big)$ must be either equal or not equal to $\text{sign}\big(f(a)\big)$, and we observe:

- If $\text{sign}\big(f(a)\big) \neq \text{sign}\big(f(m)\big)$, then $[a, m]$ is a bracketing interval.
- If $\text{sign}\big(f(a)\big) = \text{sign}\big(f(m)\big)$, then $\text{sign}\big(f(m)\big) \neq \text{sign}\big(f(b)\big)$ and hence $[m, b]$ is a bracketing interval.

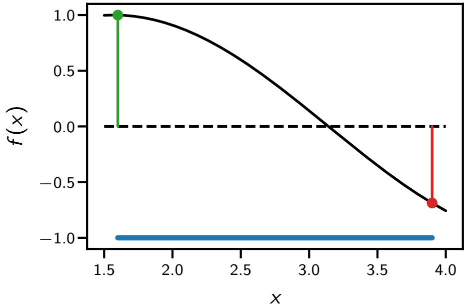
In either case, we can thus determine another bracketing interval whose length is only half of that of $[a, b]$. Applying this idea repeatedly, we can hence gradually shrink the width of the bracketing interval until it becomes negligibly small.

This algorithm is known as the bisection method and demonstrated in detail on the next slide.

Nonlinear Equations

Algorithm Bisection method

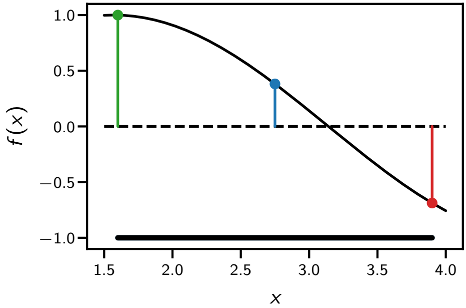
1: Start with any bracketing interval $[a_0, b_0]$.
2: **for** $k = 0, 1, 2, \ldots,$ **do**
3: Compute $m_k = \frac{a_k + b_k}{2}$.
4: Update $[a_{k+1}, b_{k+1}] = \begin{cases} [a_k, m_k] & \text{if } \operatorname{sign}\big(f(a_k)\big) \neq \operatorname{sign}\big(f(m_k)\big), \\ [m_k, b_k] & \text{otherwise.} \end{cases}$
5: **end for**



Nonlinear Equations

Algorithm Bisection method

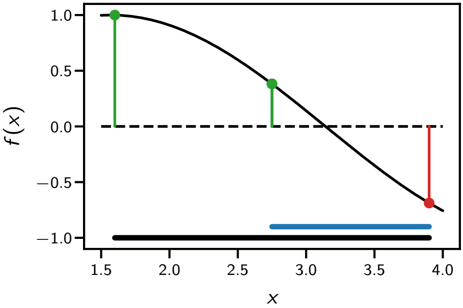
1: Start with any bracketing interval $[a_0, b_0]$.
2: **for** $k = 0, 1, 2, \ldots,$ **do**
3: Compute $m_k = \frac{a_k + b_k}{2}$.
4: Update $[a_{k+1}, b_{k+1}] = \begin{cases} [a_k, m_k] & \text{if } \operatorname{sign}\big(f(a_k)\big) \neq \operatorname{sign}\big(f(m_k)\big), \\ [m_k, b_k] & \text{otherwise.} \end{cases}$
5: **end for**



Nonlinear Equations

Algorithm Bisection method

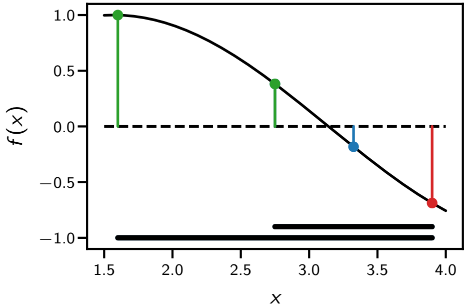
Start with any bracketing interval $[a_0, b_0]$.
for $k = 0, 1, 2, \ldots,$ **do**
    Compute $m_k = \frac{a_k + b_k}{2}$.
    Update $[a_{k+1}, b_{k+1}] = \begin{cases} [a_k, m_k] & \text{if } \operatorname{sign}(f(a_k)) \neq \operatorname{sign}(f(m_k)), \\ [m_k, b_k] & \text{otherwise.} \end{cases}$
end for



Nonlinear Equations

Algorithm Bisection method

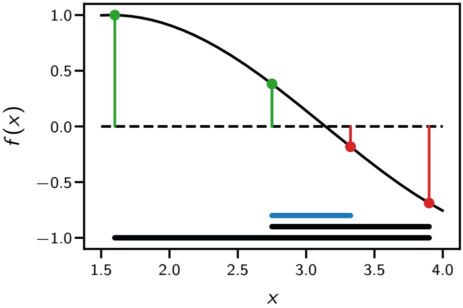
Start with any bracketing interval $[a_0, b_0]$.
for $k = 0, 1, 2, \ldots,$ **do**
  Compute $m_k = \frac{a_k + b_k}{2}$.
  Update $[a_{k+1}, b_{k+1}] = \begin{cases} [a_k, m_k] & \text{if } \operatorname{sign}\big(f(a_k)\big) \neq \operatorname{sign}\big(f(m_k)\big), \\ [m_k, b_k] & \text{otherwise.} \end{cases}$
end for



Nonlinear Equations

Algorithm Bisection method

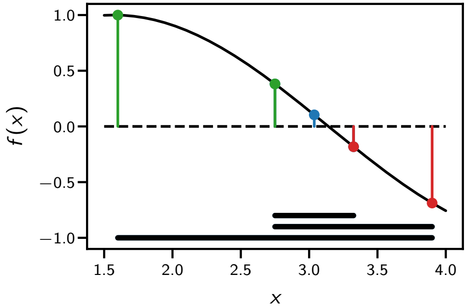
1: Start with any bracketing interval $[a_0, b_0]$.
2: **for** $k = 0, 1, 2, \ldots,$ **do**
3: Compute $m_k = \frac{a_k + b_k}{2}$.
4: Update $[a_{k+1}, b_{k+1}] = \begin{cases} [a_k, m_k] & \text{if } \operatorname{sign}\big(f(a_k)\big) \neq \operatorname{sign}\big(f(m_k)\big), \\ [m_k, b_k] & \text{otherwise.} \end{cases}$
5: **end for**



Nonlinear Equations

Algorithm Bisection method

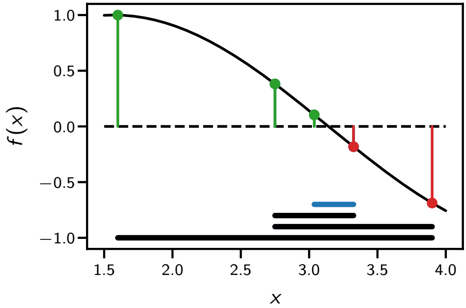
1: Start with any bracketing interval $[a_0, b_0]$.
2: **for** $k = 0, 1, 2, \ldots,$ **do**
3: Compute $m_k = \frac{a_k + b_k}{2}$.
4: Update $[a_{k+1}, b_{k+1}] = \begin{cases} [a_k, m_k] & \text{if } \operatorname{sign}\big(f(a_k)\big) \neq \operatorname{sign}\big(f(m_k)\big), \\ [m_k, b_k] & \text{otherwise.} \end{cases}$
5: **end for**



Nonlinear Equations

Algorithm Bisection method

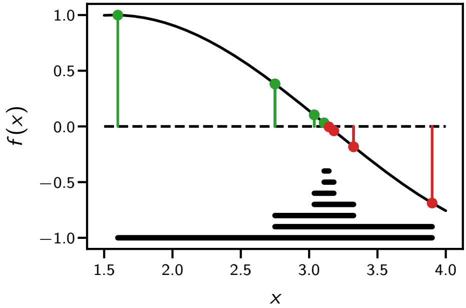
1: Start with any bracketing interval $[a_0, b_0]$.
2: **for** $k = 0, 1, 2, \ldots,$ **do**
3: Compute $m_k = \frac{a_k + b_k}{2}$.
4: Update $[a_{k+1}, b_{k+1}] = \begin{cases} [a_k, m_k] & \text{if } \operatorname{sign}(f(a_k)) \neq \operatorname{sign}(f(m_k)), \\ [m_k, b_k] & \text{otherwise.} \end{cases}$
5: **end for**



Nonlinear Equations

Algorithm Bisection method

1: Start with any bracketing interval $[a_0, b_0]$.
2: **for** $k = 0, 1, 2, \ldots,$ **do**
3: Compute $m_k = \frac{a_k + b_k}{2}$.
4: Update $[a_{k+1}, b_{k+1}] = \begin{cases} [a_k, m_k] & \text{if } \operatorname{sign}\big(f(a_k)\big) \neq \operatorname{sign}\big(f(m_k)\big), \\ [m_k, b_k] & \text{otherwise.} \end{cases}$
5: **end for**



Performance of the bisection method

Now that we have a concrete algorithm for finding increasingly tighter bracketing intervals, the natural next question is: how fast is this algorithm?

Answering this question is complicated by the fact that the bisection method only produces an exact root (i.e. a bracketing interval of length 0) after an infinite number of bisection steps. In practice, we must therefore artificially terminate the bisection procedure after a finite number of iterations and accept that a finite amount of computation buys us only finite accuracy.

The best we can do under these circumstances is to estimate separately how the runtime and accuracy change as we increase the number of bisection steps. This is achieved by the results presented next.

Nonlinear Equations

Thm: Runtime of the bisection method

n iterations of the bisection method require n evaluations of $f(x)$ and $O(n)$ other operations.

This estimate assumes that we do not need to verify that the initial interval $[a_0, b_0]$ is indeed bracketing. The runtime increases to $n+2$ function evaluations if we do need to check this condition.

Proof. Obvious.

Thm: Convergence of bisection method

Denote by $[a_k, b_k]$ the search interval after k bisection steps. We have

$$|b_k - a_k| = 2^{-k} \, |b_0 - a_0|,$$

i.e. the bisection method is exponentially convergent with rate 2.

Proof. Obvious.

Function evaluations as a performance metric

The runtime estimate on the previous slide presented a precise count for the number of evaluations of $f(x)$ and a big O estimate for the number of other operations.

The reason for doing so is that evaluating $f(x)$ is usually by far the most time-consuming part of the bisection method. It is then justified to assume that a method which requires X times more function evaluations than another method will also be X times slower, i.e. some of the details which we usually prefer to hide using the big O notation actually do matter under these circumstances.

Outlook

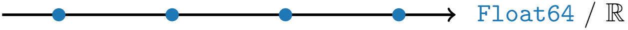
The following slides will discuss two further features of the bisection method before we move on to Newton's method as an alternative root-finding algorithm.

Nonlinear Equations

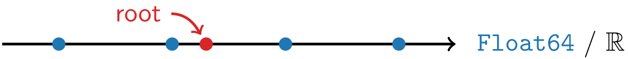
Bisection over `Float64`

I mentioned earlier that the bisection method is in principle an infinite algorithm: the more bisection steps we take, the tighter a bracketing interval we obtain.

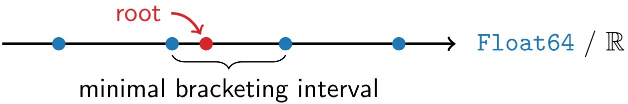
This statement is true if the bisection method is implemented over the real numbers $\mathbb{R}$, but it is no longer true once we replace $\mathbb{R}$ by the set of machine-representable numbers `Float64`.



The exact root is generally not in `Float64`,



and if the bisection method is restricted to choose interval endpoints $a, b \in$ `Float64`, then clearly there is a smallest bracketing interval $[a, b]$ beyond which no further improvement is possible.



Bisection over `Float64` (continued)

The upshot of the previous slide is that while bisection over the real numbers can achieve arbitrary accuracy given a sufficient amount of computing time, bisection over `Float64` will after a finite number of steps reach a bracketing interval $[a_k, b_k]$ beyond which no further improvement is possible.

Estimating the number of bisection steps required to reach a minimal bracketing interval is quite straightforward.

- Since `Float64` numbers occupy 64 bits of memory, there can be at most 2^{64} such numbers.
- A clever implementation of the bisection method can choose bisection points m_k such that the number of `Float64` in the bracketing interval is cut in half in each step.

Thus, even when starting from a largest possible bracketing interval, the bisection method will require at most 63 steps to reach a minimal bracketing interval.

This is demonstrated in `bisection()` and `bisection_demo()`.

Bisection over `Float64` (conclusion)

Our insights from the last two steps can be summarised as follows.

The bisection method determines the best possible root approximation $x \in$ `Float64` using at most 63 function evaluations.

This is already a very powerful statement, and it gets even better.

Thm: Optimality of the bisection method

No algorithm can reduce an initial bracketing interval $[a_0, b_0]$ to another bracketing interval $[a_k, b_k]$ with $|b_k - a_k| \le 2^{-k} \, |b_0 - a_0|$ using less than k function evaluations for every function $f : \mathbb{R} \to \mathbb{R}$.

Proof. See next slide.

Nonlinear Equations

Proof of optimality of bisection (not examinable, continued).

Observations:

1. Any algorithm reducing an initial bracketing interval $[a_0, b_0]$ to another bracketing interval $[a_k, b_k]$ with $|b_k - a_k| \leq 2^{-k} |b_0 - a_0|$ must be able to return at least 2^k different intervals $[a_k, b_k]$: if not, there are points $x \in [a_0, b_0]$ which are not contained in any of the output intervals and hence the algorithm must be wrong for functions with roots at these points.

   Example. Four intervals of length $\frac{b_0 - a_0}{4}$ can cover all of $[a_0, b_0]$:

   Three intervals of length $\frac{b_0 - a_0}{4}$ cannot cover all of $[a_0, b_0]$:

   An algorithm which only outputs these three intervals must be wrong for functions whose only root is at the indicated point x.

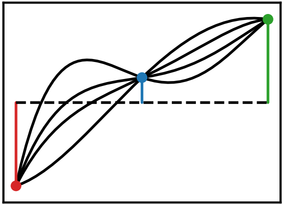
Nonlinear Equations

Proof of optimality of bisection (not examinable, continued).

Observations:

2. Every evaluation of $f(x)$ can tell us only whether the root is to the left or right of the evaluation point

   Example. The below plots shows several functions which assume the same values in the points a, m and b but have different roots x.

   This demonstrates that knowing $f(a)$, $f(m)$ and $f(b)$ only allows us to decide whether the root is in the interval $[a, m]$ or $[m, b]$ but provides no guarantees regarding the location of the root within these intervals.

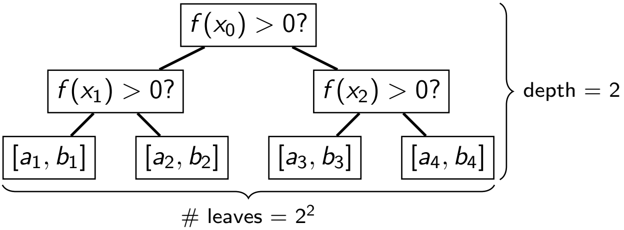
Nonlinear Equations

Proof of optimality of bisection (not examinable, continued).

Consequences:

- Item 2 implies that we can visualise A as a binary tree where each interior node represents one evaluation of $f(x)$ and each leaf node represents an output interval $[a_k, b_k]$.
- Item 1 implies that this tree must have at least 2^k leaves.

One easily verifies that a binary tree with 2^k leaves must have a depth of at least k, i.e. there must be at least one leaf which requires k function evaluations to be reached.



The bisection method is essentially binary search over the real line, and the above proof is essentially the same as the proof regarding the optimality of binary search.

Nonlinear Equations

Discussion

Put differently, the above theorem says that for any root-finding algorithm A, we can find some function $f(x)$ such that A takes at least as long as the bisection method when applied to $f(x)$.

Moreover, the proof indicates that if a root-finding algorithm A is faster than bisection for some functions $f(x)$, then at least one of the following conditions must be true.

- A is slower and/or wrong for other functions $f(x)$ (i.e. some leaves are further from the root, or some leaves are missing).
- A assumes that more information than just point values of $f(x)$ is available (i.e. we can ask questions other than $f(x_k) > 0$ to decide which branch to pursue).

We will return to these observations on slide 32.

From bisection to Newton

We have now reached the end of our discussion of the bisection method. To summarise, we have seen:

- The bisection method is guaranteed to converge to a root.
- The bisection method finds the best possible root approximation $x \in$ `Float64` using at most 63 function evaluations.
- No other method can have better worst-case performance than the bisection method.

These points indicate that other root-finding algorithms can "beat" bisection only if they provide better performance for some specific classes of functions $f(x)$.

Many alternative root-finding algorithms have been proposed over the years, and each of them represents a particular compromise between guaranteed convergence, best-case performance and worst-case performance. The remainder of this lecture will discuss the most well-known of these alternatives, namely Newton's method.

Nonlinear Equations

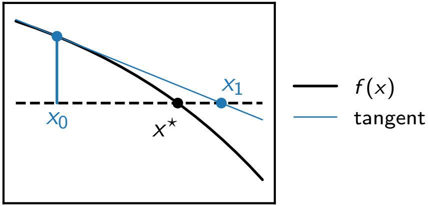
Newton's method

Assume $f(x)$ is a two-times differentiable function with root $x^\star \in \mathbb{R}$, and assume we have some guess $x_0 \in \mathbb{R}$ for where we expect this root to be.

According to Taylor's theorem, $f(x)$ is well approximated by the tangent to $f(x)$ through x_0 for x close to x_0, i.e. the function

$$x \;\mapsto\; f(x_0) + f'(x_0)\,(x - x_0).$$

Thus if x_0 is close to $x^\star$, then we expect that the root x_1 of this tangent should be a good approximation to the root of $f(x)$.



Nonlinear Equations

Newton's method (continued)

Straightforward algebra reveals that the root x_1 of the tangent through x_0 is given by

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

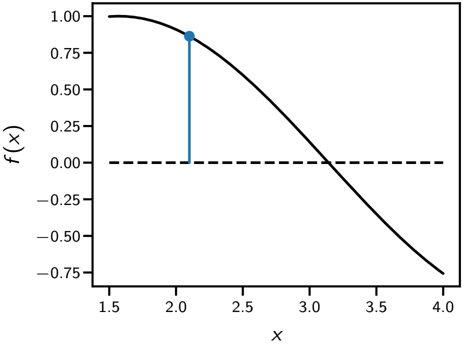
Newton's method consists in repeating this "root of tangent" process indefinitely, i.e. it constructs a sequence $(x_k)_{k=0}^{\infty}$ defined recursively by

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

Nonlinear Equations

Algorithm Newton's method

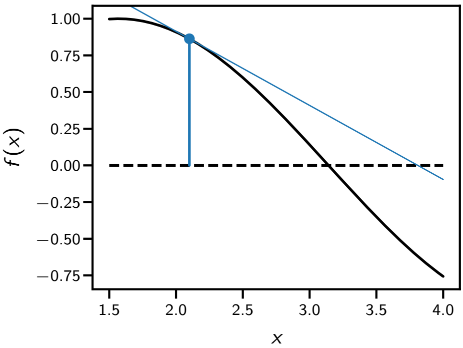
1: Start with any x_0.
2: **for** $k = 0, 1, 2, \ldots,$ **do**
3: Compute tangent of $f(x)$ at x_k.
4: Update $x_{k+1} =$ [root of tangent]
5: **end for**



Nonlinear Equations

Algorithm Newton's method

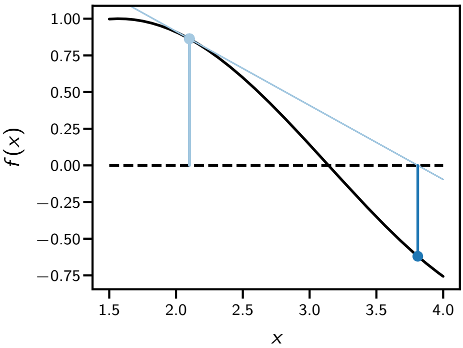
Start with any x_0.
for $k = 0, 1, 2, \ldots,$ **do**
  Compute tangent of $f(x)$ at x_k.
  Update $x_{k+1} =$ [root of tangent]
end for



Nonlinear Equations

Algorithm Newton's method

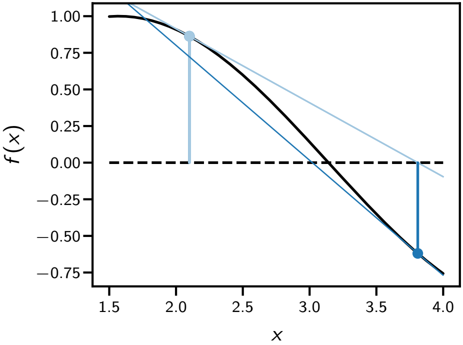
Start with any x_0.
for $k = 0, 1, 2, \ldots,$ **do**
    Compute tangent of $f(x)$ at x_k.
    Update $x_{k+1} =$ [root of tangent]
end for



Nonlinear Equations

Algorithm Newton's method

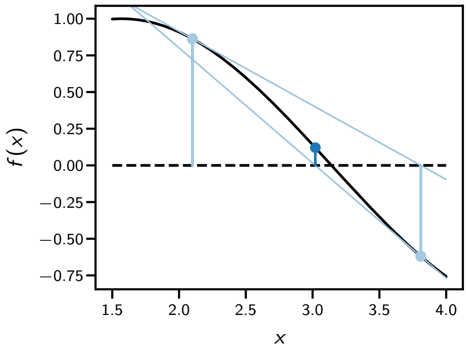
1: Start with any x_0.
2: **for** $k = 0, 1, 2, \ldots,$ **do**
3: Compute tangent of $f(x)$ at x_k.
4: Update $x_{k+1} =$ [root of tangent]
5: **end for**



Nonlinear Equations

Algorithm Newton's method

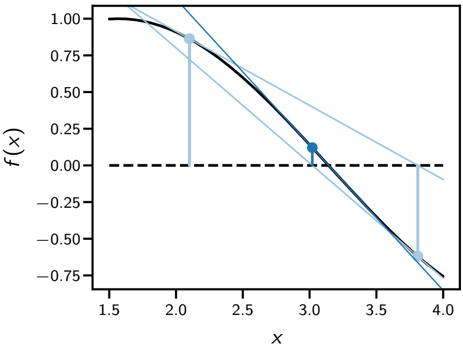
1: Start with any x_0.
2: **for** $k = 0, 1, 2, \ldots,$ **do**
3: Compute tangent of $f(x)$ at x_k.
4: Update $x_{k+1} =$ [root of tangent]
5: **end for**



Nonlinear Equations

Algorithm Newton's method

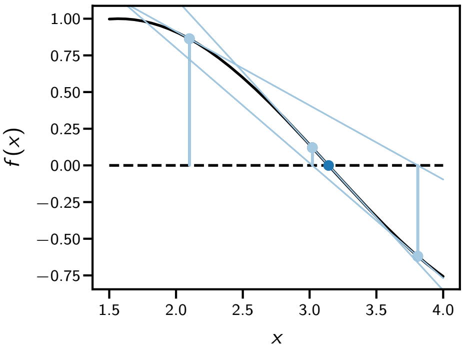
```
Start with any x_0.
for k = 0, 1, 2, ..., do
    Compute tangent of f(x) at x_k.
    Update x_{k+1} = [root of tangent]
end for
```



Nonlinear Equations

Algorithm Newton's method

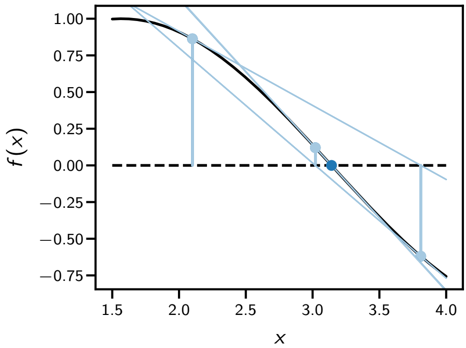
1: Start with any x_0.
2: **for** $k = 0, 1, 2, \ldots,$ **do**
3: Compute tangent of $f(x)$ at x_k.
4: Update $x_{k+1} =$ [root of tangent]
5: **end for**



Nonlinear Equations

Algorithm Newton's method

Start with any x_0.
for $k = 0, 1, 2, \ldots,$ **do**
  Compute tangent of $f(x)$ at x_k.
  Update $x_{k+1} =$ [root of tangent]
end for



Performance of Newton's method

Analogous to what I did for the bisection method, I will next address the question "how fast is Newton's method", and I will do so by analysing separately the runtime and convergence of Newton's method as a function of the number of iterations.

Thm: Runtime of Newton's method

n iterations of Newton's method require n evaluations of $f(x)$ and $f'(x)$, and $O(n)$ other operations.

Proof. Immediate consequence of the iteration formula

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

Nonlinear Equations

Thm: Error recursion for Newton's method

Assume $f(x)$ is two times differentiable and has a root $x^\star \in \mathbb{R}$ such that $f'(x^\star) \neq 0$. Then the Newton iterates

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

satisfy the error recursion

$$x_{k+1} - x^\star = O\Big(|x_k - x^\star|^2\Big) \quad \text{for } x_k \to x^\star.$$

Proof. Subtracting the root $x^\star$ on both sides of the Newton iteration formula, Taylor-expanding $f(x)$ around $x^\star$ and using $f(x^\star) = 0$, we obtain

$$\begin{aligned}
x_{k+1} - x^\star &= x_k - x^\star - \tfrac{f(x_k)}{f'(x_k)} \\
&= x_k - x^\star - \frac{f(x^\star) + f'(x^\star)\,(x_k - x^\star) + \frac{1}{2}\,f''(x^\star)\,(x_k - x^\star)^2 + O\big(|\cdot|^3\big)}{f'(x^\star) + f''(x^\star)\,(x_k - x^\star) + O\big(|\cdot|^2\big)} \\
&= \frac{1}{2}\,\frac{f''(x^\star)}{f'(x^\star)}\,(x_k - x^\star)^2 + O\big(|\cdot|^3\big).
\end{aligned}$$

A rigorous argument for why we can drop the $O(|\cdot|)$ term in the denominator is beyond the scope of this module. You are not expected to be able to fill in this step.

Nonlinear Equations

Newton vs. bisection

The $x_{k+1} - x^\star = O\big((x_k - x^\star)^2\big)$ error recursion for Newton's method indicates extremely rapid convergence: we have

$$\begin{aligned} |x_1 - x^\star| &= O\big(|x_0 - x^\star|^2\big), & |x_2 - x^\star| &= O\big(|x_0 - x^\star|^4\big), \\ |x_3 - x^\star| &= O\big(|x_0 - x^\star|^8\big), & |x_4 - x^\star| &= O\big(|x_0 - x^\star|^{16}\big); \end{aligned}$$

thus if the initial error satisfies $|x_0 - x^\star| \approx 10^{-1}$, then after just four iterations we expect

$$|x_4 - x^\star| \approx 10^{-16} \approx \texttt{eps()}.$$

In comparison, the number of steps k required by the bisection method to achieve the same error reduction is given by

$$10^{-16} \leq 2^{-k}\, 10^{-1} \qquad \Longleftrightarrow \qquad k = \log_2(10^{15}) \approx 50.$$

We thus expect that Newton's method requires roughly 12x fewer iterations than bisection under the given circumstances.

Nonlinear Equations

Newton vs. bisection (continued)

We observed on the previous slide that Newton's method is expected to require 12x fewer iterations than the bisection method to achieve a 10^{-15} error reduction.

This does not necessarily mean that Newton's method is 12x faster than the bisection method. Each Newton's step

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

requires evaluating both $f(x)$ and $f'(x)$, while each bisection step

$$[a_{k+1}, b_{k+1}] = \begin{cases} [a_k, m_k] & \text{if } \operatorname{sign}\big(f(a_k)\big) \neq \operatorname{sign}\big(f(m_k)\big), \\ [m_k, b_k] & \text{if } \operatorname{sign}\big(f(a_k)\big) = \operatorname{sign}\big(f(m_k)\big), \end{cases}$$

only requires a single evaluation of $f(x)$.

For example, if we assume that derivative evaluations are about as costly as function evaluations, then we conclude that Newton's method is "only" about 6x faster than the bisection method.

Newton vs. bisection 2

Another way to get a feeling for the speed of convergence of Newton's method is to observe that $x_{k+1} - x^\star = O\big((x_k - x^\star)^2\big)$ roughly means that

$$x_k \text{ has } n \text{ correct digits} \quad \Longrightarrow \quad x_{k+1} \text{ has } 2n \text{ correct digits}$$

Terminology: x_k has n correct digits $\quad \Longleftrightarrow \quad |x_k - x^\star| \approx 10^{-n}$.

In comparison, the analogous statement for the bisection method is

$$m_k \text{ has } n \text{ correct digits} \quad \Longrightarrow \quad m_{k+3} \text{ has } n+1 \text{ correct digits},$$

since three bisection steps reduce the width of the bracketing interval by a factor $2^3 = 8 \approx 10$.

These observations are illustrated in `bisection_convergence()` and `newton_convergence()`.

Terminology: Linear and quadratic convergence

A special term is used in the literature to describe the type of convergence exhibited by Newton's method.

Def: Quadratic convergence

A sequence x_k such that $x_{k+1} - x^\star = O\big((x_k - x^\star)^2\big)$ is said to *converge quadratically*.

By analogy, the following term is sometimes used to describe the type of convergence exhibited by the bisection method.

Def: Linear convergence

A sequence x_k such that $|x_{k+1} - x^\star| \leq r\,|x_k - x^\star|$ for some $r < 1$ is said to *converge linearly with rate r*.

These terms should not be confused with quadratic and linear scaling, which refer to $x_k = O(k^2)$ and $x_k = O(k)$, respectively, cf. Lecture 1.

Linear convergence with rate r is the same as exponential convergence with rate r, and I will mostly use the latter term to avoid confusion.

There is no reasonable substitute for the term "quadratic convergence", however, so in this case we are stuck with the above terminology.

Nonlinear Equations

Newton's method and roots of multiplicity > 1

Recall from slide 24 that a more precise form of the error recursion formula $x_{k+1} - x^\star = O\big(|x_k - x^\star|^2\big)$ is given by

$$x_{k+1} - x_\star = \frac{1}{2}\,\frac{f''(x^\star)}{f'(x^\star)}\,(x_k - x^\star)^2 + O\big(|\cdot|^3\big).$$

This explains why we had to assume $f'(x^\star) \neq 0$ in the theorem on slide 24: if $f'(x^\star) = 0$, then the above formula would not make sense.

I will next discuss what happens if $f'(x^\star) = 0$. To do so, I will need the following terminology.

Def: Multiplicity of roots

Let $x^\star$ be a root of an infinitely differentiable function $f(x)$. The smallest integer $m \geq 1$ such that $f^{(m)}(x^\star) \neq 0$ is called the *multiplicity* of $x^\star$.

Example. $x^\star = 0$ is a root of multiplicity 2 of $f(x) = x^2$ since

$$f(x^\star) = (x^\star)^2 = 0, \qquad f'(x^\star) = 2x^\star = 0, \qquad f''(x^\star) = 2.$$

Nonlinear Equations

Newton's method and roots of multiplicity > 1 (continued)

If $x^\star$ is a root of multiplicity m, then the lowest-order term in the Newton error recursion formula becomes

$$\begin{aligned} x_{k+1} - x^\star &= x_k - x^\star - \frac{0 + \frac{1}{m!}\, f^{(m)}(x^\star)\,(x_k - x^\star)^m + O\big(|\cdot|^{m+1}\big)}{0 + \frac{1}{(m-1)!}\, f^{(m)}(x^\star)\,(x_k - x^\star)^{m-1} + O\big(|\cdot|^{m}\big)} \\ &= \big(1 - \tfrac{1}{m}\big)\,(x_k - x^\star) + O\big(|\cdot|^2\big). \end{aligned}$$

This shows that Newton's method converges only linearly when applied to roots of multiplicities > 1, and the rate of convergence $\big(1 - \frac{1}{m}\big)$ is as good as the bisection method for $m = 2$ and worse for $m > 2$.

This effect is illustrated in `newton_linear_convergence()`.

Guaranteed local convergence of Newton's method

The two Newton convergence estimates

$$x_{k+1} - x^\star = \begin{cases} O\big(|x_k - x^\star|^2\big) & \text{(multiplicity } m = 1) \\ \big(1 - \frac{1}{m}\big)\,(x_k - x^\star) + O(|\cdot|^2) & \text{(multiplicity } m > 1) \end{cases}$$

imply that if x_k is close to $x^\star$, then x_{k+1} will be even closer to $x^\star$ and hence Newton's method converges.

This property is sometimes called *guaranteed local convergence*.

"Local" here indicates that convergence is guaranteed only if x_0 is "close enough" to $x^\star$, where the precise definition of "close enough" depends on the details hidden by the big O notation.

The problem with beating bisection

We have seen above that the Newton convergence estimate

$$x_{k+1} - x^\star = O\big(|x_k - x^\star|^2\big) \qquad \text{(multiplicity}(x^\star) = 1)$$

indicates that if Newton's method converges, then it converges significantly faster than the bisection method.

On the other hand, we have also seen on slide 18 that any root-finding algorithm outperforming the bisection method must necessarily suffer from other drawbacks.

I will next present an argument which indicates that in the case of Newton's method, the drawback is that Newton's method must fail to converge for some input pairs $\big(f(x), x_0\big)$.

This argument is not a rigorous mathematical proof and should be taken with a good grain of salt. Its main purpose is to indicate that the occasional divergence of Newton's method is likely (but not provenly) due to fundamental mathematical limitations, not bad algorithm design.

Nonlinear Equations

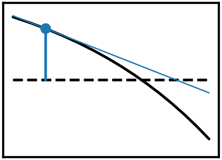
Why Newton's method must diverge for some inputs

According to our findings from slide 18, a root-finding algorithm can outperform the bisection method only if

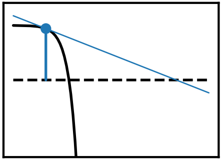
1. it exploits more information than only function values $f(x)$, or
2. it performs worse than bisection on some inputs.

Newton's method does exploit extra information about $f(x)$ in the form of derivatives, but this is not enough to explain why Newton's method beats bisection. For example, knowing that $f(x_k)$ is large and $f'(x_k)$ is small *suggests* that any roots of $f(x)$ must be far from x_k, but it does not *guarantee* that the root cannot be arbitrarily close to x_k.

Expection



Possible reality



Why Newton's method must diverge for some inputs (continued)

Newton's method therefore still suffers from the problem that any function evaluation gives us at most binary information regarding the location of the roots, and hence the decision tree associated with Newton's method must be at least as deep as the decision tree of the bisection method.

More precisely, Newton's method does not even extract binary information from the function evaluations since it is not based on bracketing intervals.

We therefore conclude that Newton's method must perform worse than bisection on some inputs. But the $x_{k+1} - x^\star = O\big((x_k - x^\star)^2\big)$ convergence estimate from slide 24 tells us that if Newton converges, then it converges much faster than bisection; thus Newton's method can perform worse than bisection only if it does not converge at all.

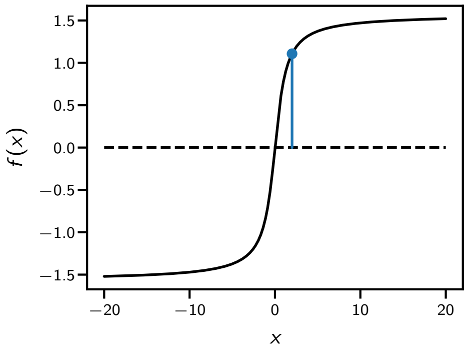
Outlook

The following slide illustrates the divergence of Newton's method at the example of the input $f(x) = \operatorname{atan}(x)$, $x_0 = 2$.

Nonlinear Equations

Algorithm Newton's method

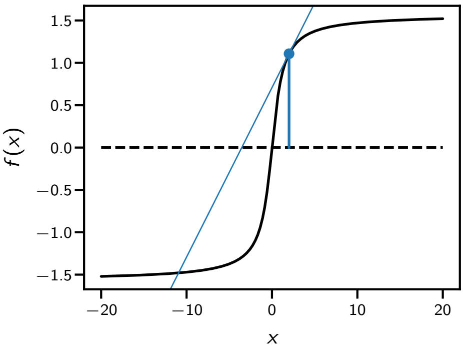
1: Start with any x_0.
2: **for** $k = 0, 1, 2, \ldots,$ **do**
3: Compute tangent of $f(x)$ at x_k.
4: Update $x_{k+1} =$ [root of tangent]
5: **end for**



Nonlinear Equations

Algorithm Newton's method

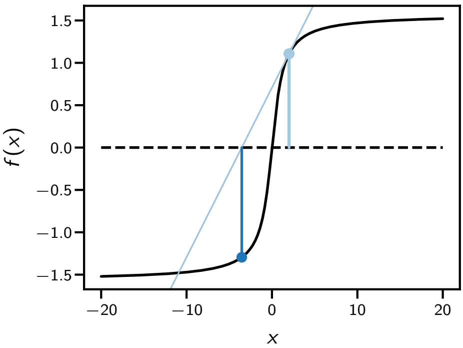
1: Start with any x_0.
2: **for** $k = 0, 1, 2, \ldots,$ **do**
3: Compute tangent of $f(x)$ at x_k.
4: Update $x_{k+1} =$ [root of tangent]
5: **end for**



Nonlinear Equations

Algorithm Newton's method

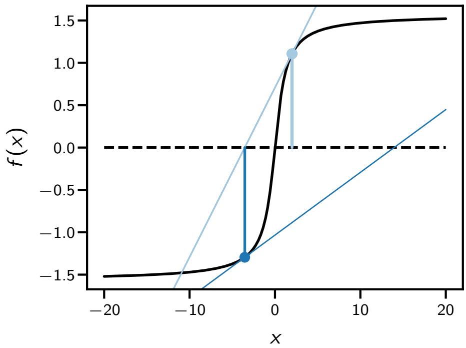
1: Start with any x_0.
2: **for** $k = 0, 1, 2, \ldots,$ **do**
3: Compute tangent of $f(x)$ at x_k.
4: Update $x_{k+1} = $ [root of tangent]
5: **end for**



Nonlinear Equations

Algorithm Newton's method

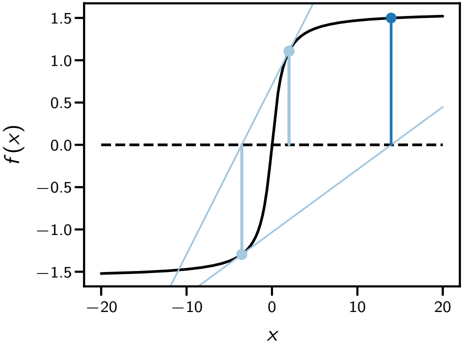
1: Start with any x_0.
2: **for** $k = 0, 1, 2, \ldots,$ **do**
3: Compute tangent of $f(x)$ at x_k.
4: Update $x_{k+1} =$ [root of tangent]
5: **end for**



Nonlinear Equations

Algorithm Newton's method

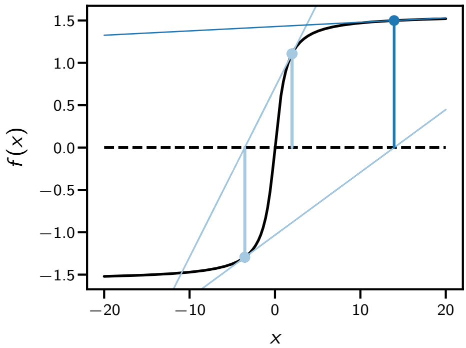
1: Start with any x_0.
2: **for** $k = 0, 1, 2, \ldots,$ **do**
3: Compute tangent of $f(x)$ at x_k.
4: Update $x_{k+1} =$ [root of tangent]
5: **end for**



Nonlinear Equations

Algorithm Newton's method

1: Start with any x_0.
2: **for** $k = 0, 1, 2, \ldots,$ **do**
3: Compute tangent of $f(x)$ at x_k.
4: Update $x_{k+1} =$ [root of tangent]
5: **end for**



Newton's method in practice

The fact that Newton's method may diverge for some inputs makes it significantly harder to use this method in practice. Specifically, at least one of the following statements must be true whenever we apply Newton's method to a particular problem.

1. We have a mathematical argument which shows that Newton's method is guaranteed to converge in our particular application.
2. We have a mechanism for detecting when Newton's method diverges, and we have another root-finding method (e.g. bisection) to finish the job when this happens.
3. We accept that Newton's method may fail to converge but keep our fingers crossed that this never actually happens, or that we can fix the situation by some other means if it does.

Nonlinear Equations

Newton's method in practice (continued)

We observe:

- There is not much to be said about Option 3 from a mathematical point of view. It clearly depends on the specific application whether hoping that Newton's method converges is reasonable (e.g. because we have observed Newton to work well in the past) and / or acceptable (i.e. the stakes are sufficiently low).
- Switching between root-finding algorithms (Option 2) can often be made to work reasonably well in practice, but it follows from the arguments on slide 18 that it might be difficult to prove that such a scheme is indeed better than bisection, and without such a proof Option 2 is essentially the same as Option 3.

These observations indicate that much headache can be avoided if can prove that Newton's method is guaranteed to converge in our particular application.

[To be continued]

Nonlinear Equations

Summary

- Bisection method:

$$[a_{k+1}, b_{k+1}] = \begin{cases} [a_k, m_k] & \text{if } \operatorname{sign}\big(f(a_k)\big) \neq \operatorname{sign}\big(f(m_k)\big), \\ [m_k, b_k] & \text{if } \operatorname{sign}\big(f(a_k)\big) = \operatorname{sign}\big(f(m_k)\big). \end{cases}$$

  Error recursion: $|b_{k+1} - a_{k+1}| = \frac{1}{2}\,|b_k - a_k|.$

  Good: Guaranteed convergence. Optimal in some sense.

  Bad: Only applies to scalar root finding.

- Newton's method: $x_{k+1} = x_k - \dfrac{f(x_k)}{f'(x_k)}$

  Error recursion: $x_{k+1} - x^\star = O\big((x_k - x^\star)^2\big)$ if $f'(x^\star) \neq 0.$

  Good: Quadratic convergence. Works in any dimension.

  Bad: May fail to converge.